

Contents

[Azure Machine Learning Documentation](#)

[Switch to SDK & CLI v1 documentation](#)

[Overview](#)

[What is Azure Machine Learning?](#)

[What is Azure Machine Learning studio?](#)

[How it works](#)

[Quickstart](#)

[Create ML resources to get started](#)

[Tutorials](#)

[Azure ML in a day](#)

- [1. Run a Python script](#)
- [2. Train your model](#)
- [3. Use your own data](#)

[Build models](#)

[Create production ML pipelines \(preview\)](#)

[Train a TensorFlow image classification model](#)

[Automated ML](#)

[Object detection with AutoML \(SDK\)](#)

[Create automated ML experiments](#)

[Forecast demand \(Bike share data\)](#)

[Designer \(drag-n-drop\)](#)

- [1. Train a regression model](#)
- [2. Deploy the model](#)

[Samples](#)

[Jupyter Notebooks](#)

[Examples repository](#)

[Designer examples & datasets](#)

[End-to-end MLOps examples](#)

[Concepts](#)

What is v2?

Train models

Model training

Distributed training

Deep learning

MLflow models

AutoML

Automated ML overview

Overfitting & imbalanced data

Designer (drag-n-drop ML)

Designer overview

Algorithm cheat sheet

How to select algorithms

Deploy models

Endpoints

Model portability (ONNX)

Prebuilt docker images for inference

Manage the ML lifecycle (MLOps)

MLOps capabilities

ML pipelines

ML components

MLflow

Responsible AI

Responsible AI overview

Implement Responsible AI in Practice

Responsible AI dashboard

Model interpretability

Fairness in Machine Learning

Causal analysis

Assess errors in ML models

Understand your datasets

Counterfactual analysis and what-if

Workspace & compute resources

Workspace

Environments

Compute instance

Compute target

Work with Data

Access data

Sourcing human data responsibly

Security

Enterprise security overview

Secured workspace traffic flow

Security baseline

Security controls by Azure Policy

Data encryption

Customer-managed keys

Vulnerability management

Infrastructure & security

Tutorials & templates

Tutorial: Create a secure workspace in Azure portal

Tutorial: Create a secure workspace with templates

Identity & access management

Set up authentication

Manage users and roles

Use managed identities for access control

Network security

Virtual network overview

Secure workspace resources

Secure training environment

Use studio in a virtual network

Use private endpoint

Use custom DNS

Configure required network traffic

Data exfiltration prevention

Configure network isolation with v2

Data protection

Failover & disaster recovery

Regenerate storage access keys

Create & manage workspaces

Use Azure portal or Python SDK

Use Azure CLI

Use Resource Manager template

Use Terraform

Use REST

How to move a workspace

Link to Azure Synapse Analytics workspace

Securely integrate Azure Synapse & Azure Machine Learning

Workspace Diagnostics

Customer-managed keys

Create & manage compute resources

Compute instance

Compute cluster

Manage compute resources

AKS and Azure Arc-enabled Kubernetes

What is Kubernetes compute target

Step 1 - Deploy AzureML extension

Step 2 - Attach cluster to workspace

Create and manage instance types

Manage AzureML inference router

Secure inferencing environment

Plan and manage costs

Monitor Azure Machine Learning

Secure coding

Audit and manage

How-to guides

Search for assets

[Set up an environment](#)

[Set up dev environments](#)

[Install and set up the CLI \(v2\)](#)

[Set up studio environment](#)

[Manage environments in studio](#)

[Create and manage files](#)

[Run Jupyter Notebooks](#)

[Use a terminal](#)

[Set up software environments](#)

[Use private Python packages](#)

[Set input & output directories](#)

[Set up VS Code extension](#)

[Connect to compute instance VS Code](#)

[Git integration](#)

[Manage environments with the CLI \(v2\)](#)

[How to troubleshoot environments](#)

[Work with data](#)

[Create datastores](#)

[Create data assets](#)

[Read & write data in jobs](#)

[Data administration](#)

[Label data](#)

[Set up image labeling](#)

[Set up text labeling](#)

[Label images and text](#)

[Add users](#)

[Outsource labeling tasks](#)

[Train models](#)

[Train with the job creation UI](#)

[Train with the Python SDK](#)

[Train with SDK v2 \(preview\)](#)

[Tune hyperparameters](#)

[Distributed GPU guide](#)

[Scikit-learn](#)

[TensorFlow](#)

[Keras](#)

[PyTorch](#)

[Train with custom Docker image](#)

[Use Key Vault when training](#)

[Train with the CLI v2](#)

[Train with the REST API](#)

[Track & monitor training](#)

[Monitor training jobs](#)

[Log metrics, parameters and files](#)

[Log MLflow models](#)

[Query & compare experiments and runs](#)

[Visualize runs with TensorBoard](#)

[Migrate from SDK v1 logging to MLflow](#)

[Train and track with MLflow](#)

[Track experiments with MLflow](#)

[Track Azure Databricks runs with MLflow](#)

[Track Azure Synapse Analytics runs with MLflow](#)

[Train experiments with MLflow Projects](#)

[Query & compare experiments and runs with MLflow](#)

[Automated machine learning](#)

[Use automated ML \(Python\)](#)

[Use automated ML \(interface\)](#)

[Use automated ML with Databricks](#)

[Auto-train a forecast model](#)

[Prep image data for computer vision models \(Python\)](#)

[Auto-train computer vision models \(Python\)](#)

[Auto-train a small object detection model](#)

[Auto-train a natural language processing model](#)

[Data splits & cross-validation \(Python\)](#)

- Featurization in automated ML (Python)
 - Understand charts and metrics
 - Generate AutoML training code
 - Use ONNX model in .NET application
 - Inference image models with ONNX model
 - Explain automated ML models
 - Troubleshoot automated ML
- Deploy models
 - Online endpoints (real-time)
 - Deploy an ML model with an online endpoint (CLI)
 - Deploy an ML model with an online endpoint (SDK preview)
 - Safe rollout for online endpoints (CLI)
 - Safe rollout for online endpoints (SDK preview)
 - Deployment scenarios
 - Deploy a MLflow model with an online endpoint
 - Deploy a custom container with an online endpoint
 - Use online endpoints in studio
 - High-performance serving with Triton
 - Use REST to deploy a model as an online endpoint
 - Deploy an AutoML model with an online endpoint
 - Security
 - Authenticate to endpoints
 - Network isolation with managed online endpoints
 - Access Azure resources from online endpoints
 - Autoscale online endpoints
 - Managed online endpoints VM SKU list
 - Viewing managed online endpoint costs
 - Monitoring online endpoints
 - Debug online endpoints locally VS Code
 - Troubleshoot online endpoints
 - Batch endpoints
 - Batch scoring with batch endpoints (CLI)

- [Batch scoring with batch endpoints \(SDK preview\)](#)
- [Batch endpoints in studio](#)
- [Use REST to deploy a model as a batch endpoint](#)
- [Troubleshoot batch endpoints](#)
- [Deploying MLflow models](#)
 - [Deploy MLflow models](#)
 - [Using MLflow models for no-code deployment](#)
 - [Convert custom model to MLflow model](#)
- [Inference HTTP server](#)
- [Manage the ML lifecycle](#)
 - [Manage models](#)
 - [Manage models with MLflow](#)
- [Automation](#)
 - [Azure Pipelines for CI/CD](#)
 - [GitHub Actions for CI/CD](#)
 - [Create event-driven workflows](#)
 - [Schedule a pipeline job](#)
- [Build & use ML pipelines](#)
 - [Create ML pipelines using components \(CLI v2\)](#)
 - [Create ML pipelines using components \(Python SDK v2\)](#)
 - [Create ML pipelines using components \(UI\)](#)
 - [How to use sweep in pipelines \(v2\)](#)
 - [Build and debug pipelines \(UI\)](#)
 - [Designer \(drag-n-drop\)](#)
 - [Log metrics](#)
 - [Transform data](#)
 - [Use pipeline parameters](#)
 - [Retrain using published pipelines](#)
 - [Batch predictions](#)
 - [Execute Python code](#)
 - [Manage resource quotas](#)
 - [Manage and optimize cost](#)

- [Manage resources VS Code](#)
- [Implement Responsible AI in Practice](#)
 - [Generate Responsible AI dashboard](#)
 - [Responsible AI dashboard \(V2\)](#)
 - [Responsible AI dashboard \(UI\)](#)
 - [Using the Responsible AI dashboard in studio](#)
 - [Responsible AI dashboard scorecard](#)
- [Migrate from v1](#)
- [Troubleshoot & debug](#)
 - [Troubleshoot secure workspace connectivity](#)
 - [VS Code interactive debugging](#)
 - [Troubleshoot SerializationError](#)
 - [Troubleshoot descriptor error](#)
 - [How to troubleshoot data access](#)
- [Reference](#)
 - [Python SDK v1](#)
 - [Python SDK v2 \(preview\)](#)
 - [REST API](#)
 - [CLI \(v2\)](#)
 - [CLI \(v2\) YAML schemas](#)
 - [Overview](#)
 - [Core syntax](#)
 - [Workspace](#)
 - [Environment](#)
 - [Data](#)
 - [Model](#)
 - [Schedule](#)
 - [Compute](#)
 - [Compute cluster \(AmlCompute\)](#)
 - [Compute instance](#)
 - [Attached Virtual Machine](#)
 - [Attached Azure Arc-enabled Kubernetes \(KubernetesCompute\)](#)

- Job
 - Command
 - Sweep
 - Pipeline
- Datastore
 - Azure Blob
 - Azure Files
 - Azure Data Lake Gen1
 - Azure Data Lake Gen2
- Endpoint
 - Online (real-time)
 - Batch
- Deployment
 - Managed online (real-time)
 - Kubernetes online (real-time)
 - Batch
- Component
 - Command
- Image data schemas for AutoML
- Hyperparameters for AutoML computer vision tasks
- Kubernetes cluster configuration
- Designer component reference
- Monitor data reference
- Azure Policy built-ins
- Curated environments
- Azure CLI

Resources

- Python SDK release notes
- CLI (v2) release notes
- Azure roadmap
- Pricing
- Conda licensing

[Regional availability](#)

[Enable preview features](#)

[Feature availability across regions](#)

[User forum](#)

[Microsoft Learn](#)

[Stack Overflow](#)

[Compare our ML products](#)

[What happened to Workbench](#)

[Designer accessibility features](#)

Azure Machine Learning SDK & CLI (v1)

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v1](#) [Python SDK azureml v1](#)

All articles in this section document the use of the first version of Azure Machine Learning Python SDK (v1) or Azure CLI ml extension (v1).

SDK v1

The Azure SDK examples in articles in this section require the `azureml-core`, or Python SDK v1 for Azure Machine Learning. The Python SDK v2 is now available in preview.

The v1 and v2 Python SDK packages are incompatible, and v2 style of coding will not work for articles in this directory. However, machine learning workspaces and all underlying resources can be interacted with from either, meaning one user can create a workspace with the SDK v1 and another can submit jobs to the same workspace with the SDK v2.

We recommend not to install both versions of the SDK on the same environment, since it can cause clashes and confusion in the code.

How do I know which SDK version I have?

- To find out whether you have Azure ML Python SDK v1, run `pip show azureml-core`. (Or, in a Jupyter notebook, use `%pip show azureml-core`)
- To find out whether you have Azure ML Python SDK v2, run `pip show azure-ai-ml`. (Or, in a Jupyter notebook, use `%pip show azure-ai-ml`)

Based on the results of `pip show` you can determine which version of SDK you have.

CLI v1

The Azure CLI commands in articles in this section **require** the `azure-cli-ml`, or v1, extension for Azure Machine Learning. The enhanced v2 CLI using the `ml` extension is now available and recommended.

The extensions are incompatible, so v2 CLI commands will not work for articles in this directory. However, machine learning workspaces and all underlying resources can be interacted with from either, meaning one user can create a workspace with the v1 CLI and another can submit jobs to the same workspace with the v2 CLI.

How do I know which CLI extension I have?

To find which extensions you have installed, use `az extension list`.

- If the list of **Extensions** contains `azure-cli-ml`, you have the v1 extension.
- If the list contains `ml`, you have the v2 extension.

Next steps

For more information on installing and using the different extensions, see the following articles:

- `azure-cli-ml` - [Install, set up, and use the CLI \(v1\)](#)

- `ml` - [Install and set up the CLI \(v2\)](#)

For more information on installing and using the different SDK versions:

- `azureml-core` - [Install the Azure Machine Learning SDK \(v1\) for Python](#)
- `azure-ai-ml` - [Install the Azure Machine Learning SDK \(v2\) for Python](#)

What is Azure Machine Learning?

9/22/2022 • 6 minutes to read • [Edit Online](#)

Azure Machine Learning is a cloud service for accelerating and managing the machine learning project lifecycle. Machine learning professionals, data scientists, and engineers can use it in their day-to-day workflows: Train and deploy models, and manage MLOps.

You can create a model in Azure Machine Learning or use a model built from an open-source platform, such as Pytorch, TensorFlow, or scikit-learn. MLOps tools help you monitor, retrain, and redeploy models.

TIP

Free trial! If you don't have an Azure subscription, create a free account before you begin. [Try the free or paid version of Azure Machine Learning](#). You get credits to spend on Azure services. After they're used up, you can keep the account and use [free Azure services](#). Your credit card is never charged unless you explicitly change your settings and ask to be charged.

Who is Azure Machine Learning for?

Azure Machine Learning is for individuals and teams implementing MLOps within their organization to bring machine learning models into production in a secure and auditable production environment.

Data scientists and ML engineers will find tools to accelerate and automate their day-to-day workflows. Application developers will find tools for integrating models into applications or services. Platform developers will find a robust set of tools, backed by durable Azure Resource Manager APIs, for building advanced ML tooling.

Enterprises working in the Microsoft Azure cloud will find familiar security and role-based access control (RBAC) for infrastructure. You can set up a project to deny access to protected data and select operations.

Collaboration for machine learning teams

Machine learning projects often require a team with varied skillsets to build and maintain. Azure Machine Learning has tools that help enable collaboration, such as:

- Shared notebooks, compute resources, data, and environments
- Tracking and auditability that shows who made changes and when
- Asset versioning

Tools for developers

Developers find familiar interfaces in Azure Machine Learning, such as:

- [Python SDK](#)
- [Azure Resource Manager REST APIs \(preview\)](#)
- [CLI v2](#)

Studio UI

The [Azure Machine Learning studio](#) is a graphical user interface for a project workspace. In the studio, you can:

- View runs, metrics, logs, outputs, and so on.
- Author and edit notebooks and files.
- Manage common assets, such as
 - Data credentials

- Compute
- Environments
- Visualize run metrics, results, and reports.
- Visualize pipelines authored through developer interfaces.
- Author AutoML jobs.

Plus, the designer has a drag-and-drop interface where you can train and deploy models.

If you're a ML Studio (classic) user, [learn about Studio \(classic\) deprecation and the difference between it and Azure Machine Learning studio](#).

Enterprise-readiness and security

Azure Machine Learning integrates with the Azure cloud platform to add security to ML projects.

Security integrations include:

- Azure Virtual Networks (VNets) with network security groups
- Azure Key Vault where you can save security secrets, such as access information for storage accounts
- Azure Container Registry set up behind a VNet

See [Tutorial: Set up a secure workspace](#).

Azure integrations for complete solutions

Other integrations with Azure services support a machine learning project from end-to-end. They include:

- Azure Synapse Analytics to process and stream data with Spark
- Azure Arc, where you can run Azure services in a Kubernetes environment
- Storage and database options, such as Azure SQL Database, Azure Storage Blobs, and so on
- Azure App Service allowing you to deploy and manage ML-powered apps

IMPORTANT

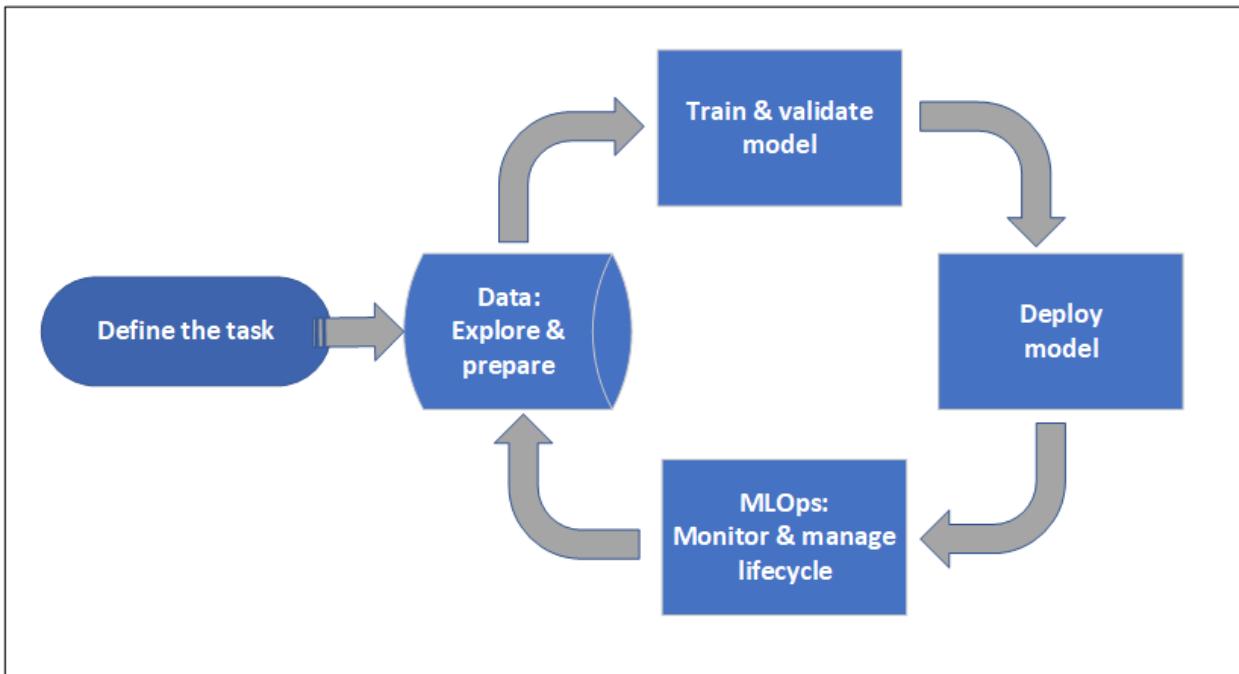
Azure Machine Learning doesn't store or process your data outside of the region where you deploy.

Machine learning project workflow

Typically models are developed as part of a project with an objective and goals. Projects often involve more than one person. When experimenting with data, algorithms, and models, development is iterative.

Project lifecycle

While the project lifecycle can vary by project, it will often look like this:



A workspace organizes a project and allows for collaboration for many users all working toward a common objective. Users in a workspace can easily share the results of their runs from experimentation in the studio user interface or use versioned assets for jobs like environments and storage references.

For more information, see [Manage Azure Machine Learning workspaces](#).

When a project is ready for operationalization, users' work can be automated in a machine learning pipeline and triggered on a schedule or HTTPS request.

Models can be deployed to the managed inferencing solution, for both real-time and batch deployments, abstracting away the infrastructure management typically required for deploying models.

Train models

In Azure Machine Learning, you can run your training script in the cloud or build a model from scratch. Customers often bring models they've built and trained in open-source frameworks, so they can operationalize them in the cloud.

Open and interoperable

Data scientists can use models in Azure Machine Learning that they've created in common Python frameworks, such as:

- PyTorch
- TensorFlow
- scikit-learn
- XGBoost
- LightGBM

Other languages and frameworks are supported as well, including:

- R
- .NET

See [Open-source integration with Azure Machine Learning](#).

Automated featurization and algorithm selection (AutoML)

In a repetitive, time-consuming process, in classical machine learning data scientists use prior experience and

intuition to select the right data featurization and algorithm for training. Automated ML (AutoML) speeds this process and can be used through the studio UI or Python SDK.

See [What is automated machine learning?](#)

Hyperparameter optimization

Hyperparameter optimization, or hyperparameter tuning, can be a tedious task. Azure Machine Learning can automate this task for arbitrary parameterized commands with little modification to your job definition. Results are visualized in the studio.

See [How to tune hyperparameters](#).

Multinode distributed training

Efficiency of training for deep learning and sometimes classical machine learning training jobs can be drastically improved via multinode distributed training. Azure Machine Learning compute clusters offer the latest GPU options.

Supported via Azure ML Kubernetes and Azure ML compute clusters:

- PyTorch
- TensorFlow
- MPI

The MPI distribution can be used for Horovod or custom multinode logic. Additionally, Apache Spark is supported via Azure Synapse Analytics Spark clusters (preview).

See [Distributed training with Azure Machine Learning](#).

Embarrassingly parallel training

Scaling a machine learning project may require scaling embarrassingly parallel model training. This pattern is common for scenarios like forecasting demand, where a model may be trained for many stores.

Deploy models

To bring a model into production, it is deployed. Azure Machine Learning's managed endpoints abstract the required infrastructure for both batch or real-time (online) model scoring (inferencing).

Real-time and batch scoring (inferencing)

Batch scoring, or *batch inferencing*, involves invoking an endpoint with a reference to data. The batch endpoint runs jobs asynchronously to process data in parallel on compute clusters and store the data for further analysis.

Real-time scoring, or *online inferencing*, involves invoking an endpoint with one or more model deployments and receiving a response in near-real-time via HTTPs. Traffic can be split across multiple deployments, allowing for testing new model versions by diverting some amount of traffic initially and increasing once confidence in the new model is established.

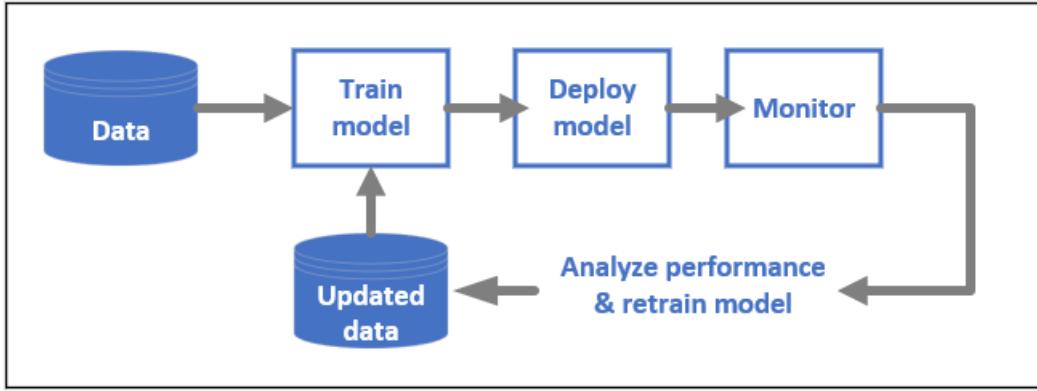
See:

- [Deploy a model with a real-time managed endpoint](#)
- [Use batch endpoints for scoring](#)

MLOps: DevOps for machine learning

DevOps for machine learning models, often called MLOps, is a process for developing models for production. A model's lifecycle from training to deployment must be auditable if not reproducible.

ML model lifecycle



Learn more about [MLOps in Azure Machine Learning](#).

Integrations enabling MLOPs

Azure Machine Learning is built with the model lifecycle in mind. You can audit the model lifecycle down to a specific commit and environment.

Some key features enabling MLOps include:

- `git` integration
- MLflow integration
- Machine learning pipeline scheduling
- Azure Event Grid integration for custom triggers
- Easy to use with CI/CD tools like GitHub Actions or Azure DevOps

Also, Azure Machine Learning includes features for monitoring and auditing:

- Job artifacts, such as code snapshots, logs, and other outputs
- Lineage between jobs and assets, such as containers, data, and compute resources

Next steps

Start using Azure Machine Learning:

- [Set up an Azure Machine Learning workspace](#)
- [Tutorial: Build a first machine learning project](#)
- [Preview: Run model training jobs with the v2 CLI](#)

What is Azure Machine Learning studio?

9/22/2022 • 4 minutes to read • [Edit Online](#)

In this article, you learn about Azure Machine Learning studio, the web portal for data scientist developers in [Azure Machine Learning](#). The studio combines no-code and code-first experiences for an inclusive data science platform.

In this article you learn:

- How to [author machine learning projects](#) in the studio.
- How to [manage assets and resources](#) in the studio.
- The differences between [Azure Machine Learning studio and ML Studio \(classic\)](#).

We recommend that you use the most up-to-date browser that's compatible with your operating system. The following browsers are supported:

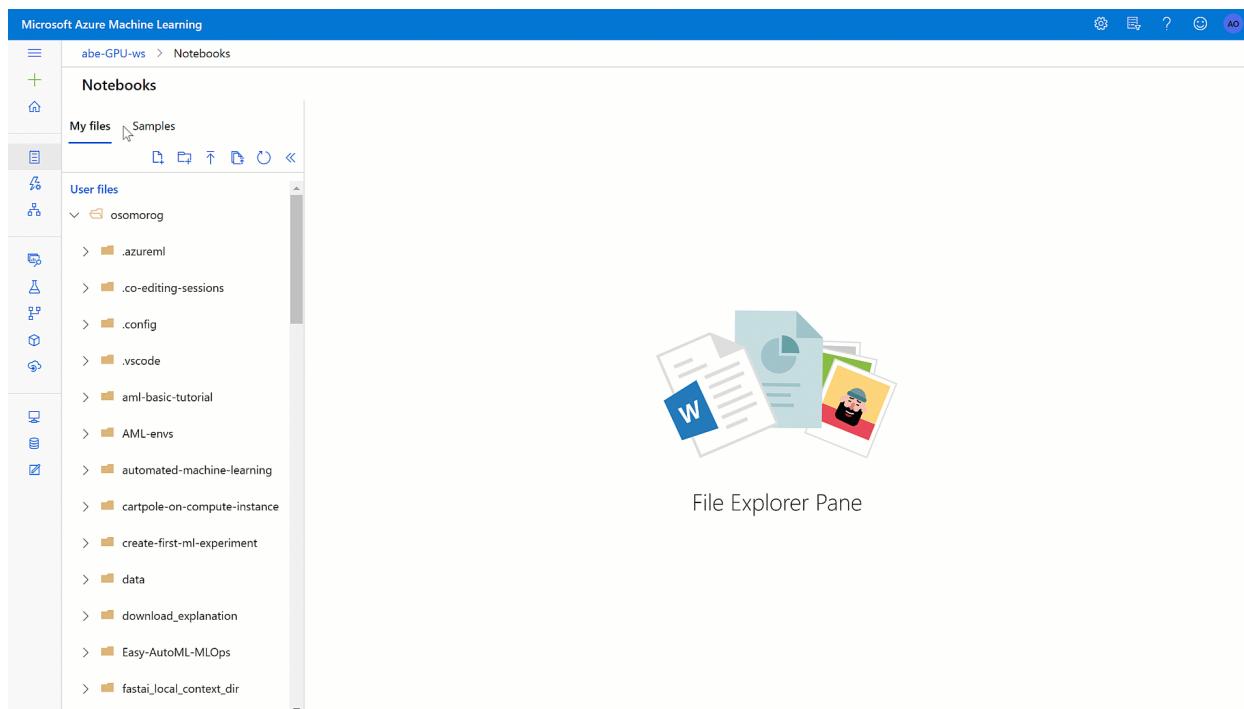
- Microsoft Edge (latest version)
- Safari (latest version, Mac only)
- Chrome (latest version)
- Firefox (latest version)

Author machine learning projects

The studio offers multiple authoring experiences depending on the type project and the level of user experience.

- **Notebooks**

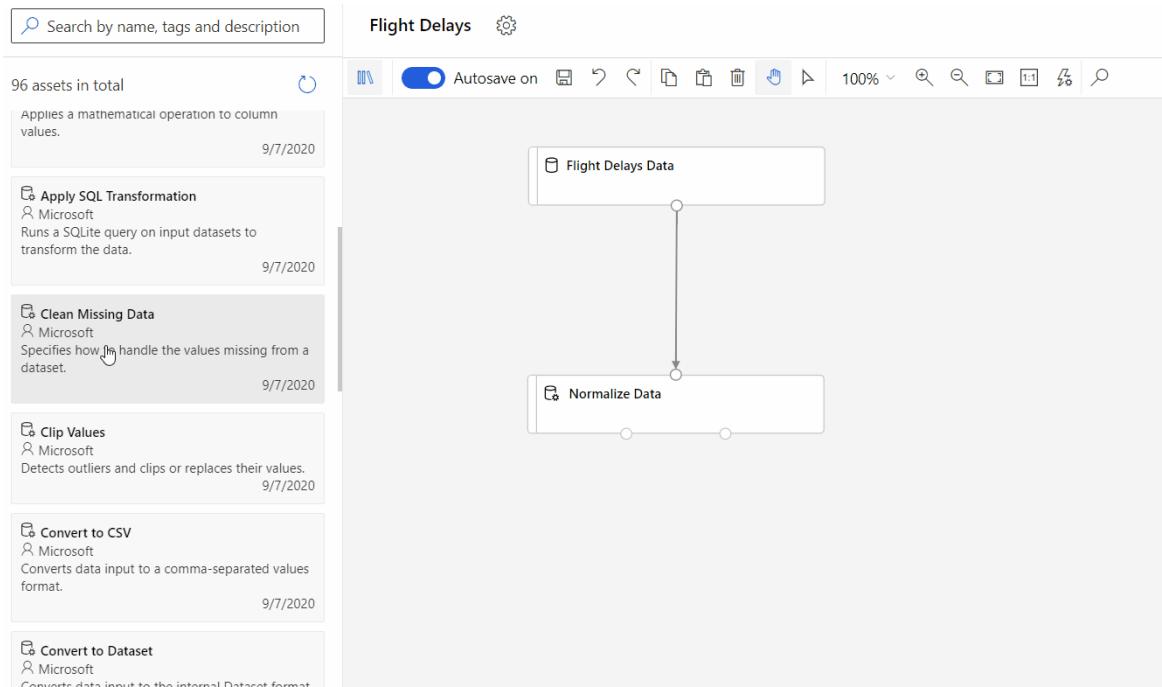
Write and run your own code in managed [Jupyter Notebook servers](#) that are directly integrated in the studio.



- **Azure Machine Learning designer**

Use the designer to train and deploy machine learning models without writing any code. Drag and drop

datasets and components to create ML pipelines. Try out the [designer tutorial](#).



- **Automated machine learning UI**

Learn how to create [automated ML experiments](#) with an easy-to-use interface.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The left sidebar has a tree view with 'Microsoft' expanded, showing 'New', 'Home', 'Author', 'Notebooks', 'Automated ML' (which is selected and highlighted in grey), 'Designer', 'Assets', 'Data', 'Jobs', 'Components', 'Pipelines', 'Environments', 'Models', and 'Endpoints'. Under 'Manage', there are 'Compute', 'Datastores', 'Linked Services', and 'Data Labeling'. The main content area is titled 'Automated ML' and contains the message 'Let Automated ML train and find the best model based on your data without writing a single line of code. [Learn more about Automated ML](#)'. Below this is a button '+ New Automated ML job' and a 'Refresh' button. A message 'No recent Automated ML jobs to display.' is shown, followed by 'Click "New Automated ML job" to create your first job' and a link 'Learn more about creating Automated ML jobs'. At the bottom, there is a 'Documentation' section with links to 'Concept: What is Automated ML?', 'Tutorial: Create your first classification model with Automated ML', and 'Blog: Build more accurate forecasts with new capabilities in Automated ML'. There is also a link 'View all documentation'.

- **Data labeling**

Use Azure Machine Learning data labeling to efficiently coordinate [image labeling](#) or [text labeling](#) projects.

Manage assets and resources

Manage your machine learning assets directly in your browser. Assets are shared in the same workspace between the SDK and the studio for a seamless experience. Use the studio to manage:

- **Models**

- Datasets
- Datastores
- Compute resources
- Notebooks
- Experiments
- Run logs
- Pipelines
- Pipeline endpoints

Even if you're an experienced developer, the studio can simplify how you manage workspace resources.

ML Studio (classic) vs Azure Machine Learning studio

IMPORTANT

Support for Machine Learning Studio (classic) will end on 31 August 2024. We recommend you transition to [Azure Machine Learning](#) by that date.

Beginning 1 December 2021, you will not be able to create new Machine Learning Studio (classic) resources (workspace and web service plan). Through 31 August 2024, you can continue to use the existing Machine Learning Studio (classic) experiments and web services.

- See [information on moving machine learning projects from ML Studio \(classic\) to Azure Machine Learning](#).
- Learn more about [Azure Machine Learning](#)

ML Studio (classic) documentation is being retired and may not be updated in the future.

Released in 2015, **ML Studio (classic)** was the first drag-and-drop machine learning model builder in Azure. **ML Studio (classic)** is a standalone service that only offers a visual experience. Studio (classic) does not interoperate with Azure Machine Learning.

Azure Machine Learning is a separate, and modernized, service that delivers a complete data science platform. It supports both code-first and low-code experiences.

Azure Machine Learning studio is a web portal *in* Azure Machine Learning that contains low-code and no-code options for project authoring and asset management.

If you're a new user, choose **Azure Machine Learning**, instead of ML Studio (classic). As a complete ML platform, Azure Machine Learning offers:

- Scalable compute clusters for large-scale training.
- Enterprise security and governance.
- Interoperable with popular open-source tools.
- End-to-end MLOps.

Feature comparison

The following table summarizes the key differences between ML Studio (classic) and Azure Machine Learning.

FEATURE	ML STUDIO (CLASSIC)	AZURE MACHINE LEARNING
Drag and drop interface	Classic experience	Updated experience - Azure Machine Learning designer
Code SDKs	Not supported	Fully integrated with Azure Machine Learning Python and R SDKs

FEATURE	ML STUDIO (CLASSIC)	AZURE MACHINE LEARNING
Experiment	Scalable (10-GB training data limit)	Scale with compute target
Training compute targets	Proprietary compute target, CPU support only	Wide range of customizable training compute targets . Includes GPU and CPU support
Deployment compute targets	Proprietary web service format, not customizable	Wide range of customizable deployment compute targets . Includes GPU and CPU support
ML Pipeline	Not supported	Build flexible, modular pipelines to automate workflows
MLOps	Basic model management and deployment; CPU only deployments	Entity versioning (model, data, workflows), workflow automation, integration with CICD tooling, CPU and GPU deployments and more
Model format	Proprietary format, Studio (classic) only	Multiple supported formats depending on training job type
Automated model training and hyperparameter tuning	Not supported	Supported . Code-first and no-code options.
Data drift detection	Not supported	Supported
Data labeling projects	Not supported	Supported
Role-Based Access Control (RBAC)	Only contributor and owner role	Flexible role definition and RBAC control
AI Gallery	Supported (https://gallery.azure.ai/)	Unsupported Learn with sample Python SDK notebooks .

Troubleshooting

- **Missing user interface items in studio** Azure role-based access control can be used to restrict actions that you can perform with Azure Machine Learning. These restrictions can prevent user interface items from appearing in the Azure Machine Learning studio. For example, if you are assigned a role that cannot create a compute instance, the option to create a compute instance will not appear in the studio. For more information, see [Manage users and roles](#).

Next steps

Visit the [studio](#), or explore the different authoring options with these tutorials:

Start with [Quickstart: Get started with Azure Machine Learning](#). Then use these resources to create your first experiment with your preferred method:

- [Run a "Hello world!" Python script \(part 1 of 3\)](#)
- [Use a Jupyter notebook to train image classification models](#)

- Use automated machine learning to train & deploy models
- Use the designer to train & deploy models
- Use studio in a secured virtual network

How Azure Machine Learning works: resources and assets (v2)

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#) [Python SDK azure-ai-ml v2 \(preview\)](#)

This article applies to the second version of the [Azure Machine Learning CLI & Python SDK \(v2\)](#). For version one (v1), see [How Azure Machine Learning works: Architecture and concepts \(v1\)](#)

Azure Machine Learning includes several resources and assets to enable you to perform your machine learning tasks. These resources and assets are needed to run any job.

- **Resources:** setup or infrastructural resources needed to run a machine learning workflow. Resources include:
 - [Workspace](#)
 - [Compute](#)
 - [Datastore](#)
- **Assets:** created using Azure ML commands or as part of a training/scoring run. Assets are versioned and can be registered in the Azure ML workspace. They include:
 - [Model](#)
 - [Environment](#)
 - [Data](#)
 - [Component](#)

This document provides a quick overview of these resources and assets.

Workspace

The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. The workspace keeps a history of all jobs, including logs, metrics, output, and a snapshot of your scripts. The workspace stores references to resources like datastores and compute. It also holds all assets like models, environments, components and data asset.

Create a workspace

- [Azure CLI](#)
- [Python SDK](#)

To create a workspace using CLI v2, use the following command:

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

```
az ml workspace create --file my_workspace.yml
```

For more information, see [workspace YAML schema](#).

Compute

A compute is a designated compute resource where you run your job or host your endpoint. Azure Machine learning supports the following types of compute:

- **Compute cluster** - a managed-compute infrastructure that allows you to easily create a cluster of CPU or GPU compute nodes in the cloud.
 - **Compute instance** - a fully configured and managed development environment in the cloud. You can use the instance as a training or inference compute for development and testing. It's similar to a virtual machine on the cloud.
 - **Inference cluster** - used to deploy trained machine learning models to Azure Kubernetes Service. You can create an Azure Kubernetes Service (AKS) cluster from your Azure ML workspace, or attach an existing AKS cluster.
 - **Attached compute** - You can attach your own compute resources to your workspace and use them for training and inference.
- [Azure CLI](#)
 - [Python SDK](#)

To create a compute using CLI v2, use the following command:

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml compute --file my_compute.yml
```

For more information, see [compute YAML schema](#).

Datastore

Azure Machine Learning datastores securely keep the connection information to your data storage on Azure, so you don't have to code it in your scripts. You can register and create a datastore to easily connect to your storage account, and access the data in your underlying storage service. The CLI v2 and SDK v2 support the following types of cloud-based storage services:

- Azure Blob Container
 - Azure File Share
 - Azure Data Lake
 - Azure Data Lake Gen2
- [Azure CLI](#)
 - [Python SDK](#)

To create a datastore using CLI v2, use the following command:

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml datastore create --file my_datastore.yml
```

For more information, see [datastore YAML schema](#).

Model

Azure machine learning models consist of the binary file(s) that represent a machine learning model and any corresponding metadata. Models can be created from a local or remote file or directory. For remote locations `https`, `wasbs` and `azureml` locations are supported. The created model will be tracked in the workspace under the specified name and version. Azure ML supports three types of storage format for models:

- `custom_model`

- [mlflow_model](#)
- [triton_model](#)

Creating a model

- [Azure CLI](#)
- [Python SDK](#)

To create a model using CLI v2, use the following command:

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml model create --file my_model.yml
```

For more information, see [model YAML schema](#).

Environment

Azure Machine Learning environments are an encapsulation of the environment where your machine learning task happens. They specify the software packages, environment variables, and software settings around your training and scoring scripts. The environments are managed and versioned entities within your Machine Learning workspace. Environments enable reproducible, auditable, and portable machine learning workflows across a variety of computes.

Types of environment

Azure ML supports two types of environments: curated and custom.

Curated environments are provided by Azure Machine Learning and are available in your workspace by default. Intended to be used as is, they contain collections of Python packages and settings to help you get started with various machine learning frameworks. These pre-created environments also allow for faster deployment time. For a full list, see the [curated environments article](#).

In custom environments, you're responsible for setting up your environment and installing packages or any other dependencies that your training or scoring script needs on the compute. Azure ML allows you to create your own environment using

- A docker image
- A base docker image with a conda YAML to customize further
- A docker build context

Create an Azure ML custom environment

- [Azure CLI](#)
- [Python SDK](#)

To create an environment using CLI v2, use the following command:

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml environment create --file my_environment.yml
```

For more information, see [environment YAML schema](#).

Data

Azure Machine Learning allows you to work with different types of data:

- URLs (a location in local/cloud storage)
 - `uri_folder`
 - `uri_file`
- Tables (a tabular data abstraction)
 - `mltable`
- Primitives
 - `string`
 - `boolean`
 - `number`

For most scenarios, you'll use URIs (`uri_folder` and `uri_file`) - a location in storage that can be easily mapped to the filesystem of a compute node in a job by either mounting or downloading the storage to the node.

`mltable` is an abstraction for tabular data that is to be used for AutoML Jobs, Parallel Jobs, and some advanced scenarios. If you're just starting to use Azure Machine Learning and aren't using AutoML, we strongly encourage you to begin with URIs.

Component

An Azure Machine Learning [component](#) is a self-contained piece of code that does one step in a machine learning pipeline. Components are the building blocks of advanced machine learning pipelines. Components can do tasks such as data processing, model training, model scoring, and so on. A component is analogous to a function - it has a name, parameters, expects input, and returns output.

Next steps

- [How to migrate from v1 to v2](#)
- [Train models with the CLI \(v2\)](#)
- [Train models with the Azure ML Python SDK v2 \(preview\)](#)

Quickstart: Create workspace resources you need to get started with Azure Machine Learning

9/22/2022 • 5 minutes to read • [Edit Online](#)

In this quickstart, you'll create a workspace and then add compute resources to the workspace. You'll then have everything you need to get started with Azure Machine Learning.

The workspace is the top-level resource for your machine learning activities, providing a centralized place to view and manage the artifacts you create when you use Azure Machine Learning. The compute resources provide a pre-configured cloud-based environment you can use to train, deploy, automate, manage, and track machine learning models.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).

Create the workspace

If you already have a workspace, skip this section and continue to [Create a compute instance](#).

If you don't yet have a workspace, create one now:

1. Sign in to [Azure Machine Learning studio](#)
2. Select **Create workspace**
3. Provide the following information to configure your new workspace:

FIELD	DESCRIPTION
Workspace name	Enter a unique name that identifies your workspace. Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others. The workspace name is case-insensitive.
Subscription	Select the Azure subscription that you want to use.
Resource group	Use an existing resource group in your subscription or enter a name to create a new resource group. A resource group holds related resources for an Azure solution. You need <i>contributor</i> or <i>owner</i> role to use an existing resource group. For more information about access, see Manage access to an Azure Machine Learning workspace .
Region	Select the Azure region closest to your users and the data resources to create your workspace.

4. Select **Create** to create the workspace

Create compute instance

You could install Azure Machine Learning on your own computer. But in this quickstart, you'll create an online compute resource that has a development environment already installed and ready to go. You'll use this online machine, a *compute instance*, for your development environment to write and run code in Python scripts and Jupyter notebooks.

Create a *compute instance* to use this development environment for the rest of the tutorials and quickstarts.

1. If you didn't just create a workspace in the previous section, sign in to [Azure Machine Learning studio](#) now, and select your workspace.
2. On the left side, select **Compute**.
3. Select **+ New** to create a new compute instance.
4. Supply a name, Keep all the defaults on the first page.
5. Select **Create**.

In about two minutes, you'll see the **State** of the compute instance change from *Creating* to *Running*. It's now ready to go.

Create compute clusters

Next you'll create a compute cluster. Clusters allow you to distribute a training or batch inference process across a cluster of CPU or GPU compute nodes in the cloud.

Create a compute cluster that will autoscale between zero and four nodes:

1. Still in the **Compute** section, in the top tab, select **Compute clusters**.
2. Select **+ New** to create a new compute cluster.
3. Keep all the defaults on the first page, select **Next**. If you don't see any available compute, you'll need to request a quota increase. Learn more about [managing and increasing quotas](#).
4. Name the cluster **cpu-cluster**. If this name already exists, add your initials to the name to make it unique.
5. Leave the **Minimum number of nodes** at 0.
6. Change the **Maximum number of nodes** to 4 if possible. Depending on your settings, you may have a smaller limit.
7. Change the **Idle seconds before scale down** to 2400.
8. Leave the rest of the defaults, and select **Create**.

In less than a minute, the **State** of the cluster will change from *Creating* to *Succeeded*. The list shows the provisioned compute cluster, along with the number of idle nodes, busy nodes, and unprovisioned nodes. Since you haven't used the cluster yet, all the nodes are currently unprovisioned.

NOTE

When the cluster is created, it will have 0 nodes provisioned. The cluster *does not* incur costs until you submit a job. This cluster will scale down when it has been idle for 2,400 seconds (40 minutes). This will give you time to use it in a few tutorials if you wish without waiting for it to scale back up.

Quick tour of the studio

The studio is your web portal for Azure Machine Learning. This portal combines no-code and code-first experiences for an inclusive data science platform.

Review the parts of the studio on the left-hand navigation bar:

- The **Author** section of the studio contains multiple ways to get started in creating machine learning models. You can:

- **Notebooks** section allows you to create Jupyter Notebooks, copy sample notebooks, and run notebooks and Python scripts.
- **Automated ML** steps you through creating a machine learning model without writing code.
- **Designer** gives you a drag-and-drop way to build models using prebuilt components.
- The **Assets** section of the studio helps you keep track of the assets you create as you run your jobs. If you have a new workspace, there's nothing in any of these sections yet.
- You already used the **Manage** section of the studio to create your compute resources. This section also lets you create and manage data and external services you link to your workspace.

Workspace diagnostics

You can run diagnostics on your workspace from Azure Machine Learning studio or the Python SDK. After diagnostics run, a list of any detected problems is returned. This list includes links to possible solutions. For more information, see [How to use workspace diagnostics](#).

Clean up resources

If you plan to continue now to the next tutorial, skip to [Next steps](#).

Stop compute instance

If you're not going to use it now, stop the compute instance:

1. In the studio, on the left, select **Compute**.
2. In the top tabs, select **Compute instances**
3. Select the compute instance in the list.
4. On the top toolbar, select **Stop**.

Delete all resources

IMPORTANT

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar includes 'Create a resource', 'Home', 'Dashboard', 'All services', and a 'FAVORITES' section with links to 'All resources', 'Resource groups', 'App Services', 'SQL databases', 'SQL data warehouses', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', 'Storage accounts', and 'Virtual networks'. The main content area is titled 'my-rg' (Resource group) and shows the 'Overview' tab selected. Other tabs include 'Activity log', 'Access control (IAM)', 'Tags', 'Events', 'Settings', 'Quickstart', 'Deployments', 'Policies', 'Properties', 'Locks', and 'Export template'. At the top right, there are buttons for '+ Add', 'Edit columns', 'Delete resource group' (which is highlighted with a red box), 'Refresh', and 'Move'. Below these are sections for 'Subscription (change)' (Visual Studio Ultimate with MSDN), 'Subscription ID' (XXXXXX-XXX-XXXX-XXXX), and 'Tags (change)' (with a link to 'Click here to add tags'). A resource list table follows, with columns for 'NAME' and 'TYPE'. It contains three items: 'myworkspace' (Machine Learning service workspace), 'my-workspace' (Machine Learning service workspace), and 'myworkspace0013141752' (Application Insights). Filter options at the top of the table allow filtering by name, type, and location.

4. Enter the resource group name. Then select **Delete**.

Next steps

You now have an Azure Machine Learning workspace that contains:

- A compute instance to use for your development environment.
- A compute cluster to use for submitting training runs.

Use these resources to learn more about Azure Machine Learning and train a model with Python scripts.

[Learn more with Python scripts](#)

Tutorial: Get started with a Python script in Azure Machine Learning (part 1 of 3)

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

In this tutorial, you run your first Python script in the cloud with Azure Machine Learning. This tutorial is *part 1 of a three-part tutorial series*.

This tutorial avoids the complexity of training a machine learning model. You will run a "Hello World" Python script in the cloud. You will learn how a control script is used to configure and create a run in Azure Machine Learning.

In this tutorial, you will:

- Create and run a "Hello world!" Python script.
- Create a Python control script to submit "Hello world!" to Azure Machine Learning.
- Understand the Azure Machine Learning concepts in the control script.
- Submit and run the "Hello world!" script.
- View your code output in the cloud.

Prerequisites

- Complete [Quickstart: Set up your workspace to get started with Azure Machine Learning](#) to create a workspace, compute instance, and compute cluster to use in this tutorial series.
- The Azure Machine Learning Python SDK v2 (preview) installed.

To install the SDK you can either,

- Create a compute instance, which automatically installs the SDK and is pre-configured for ML workflows. For more information, see [Create and manage an Azure Machine Learning compute instance](#).
- Use the following commands to install Azure ML Python SDK v2:
 - Uninstall previous preview version:

```
pip uninstall azure-ai-ml
```

- Install the Azure ML Python SDK v2:

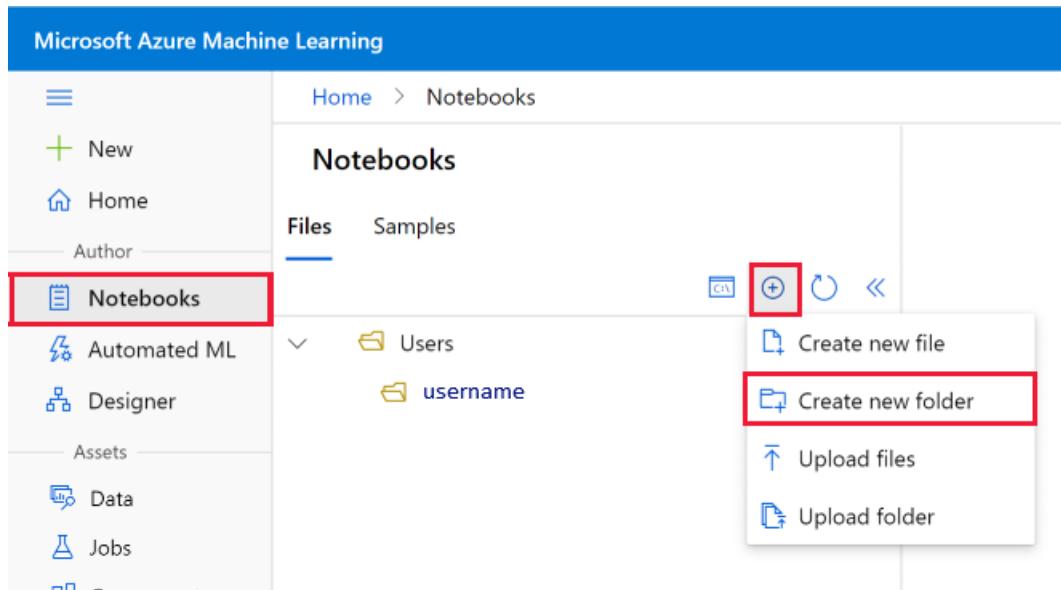
```
pip install azure-ai-ml
```

Create and run a Python script

This tutorial will use the compute instance as your development computer. First create a few folders and the script:

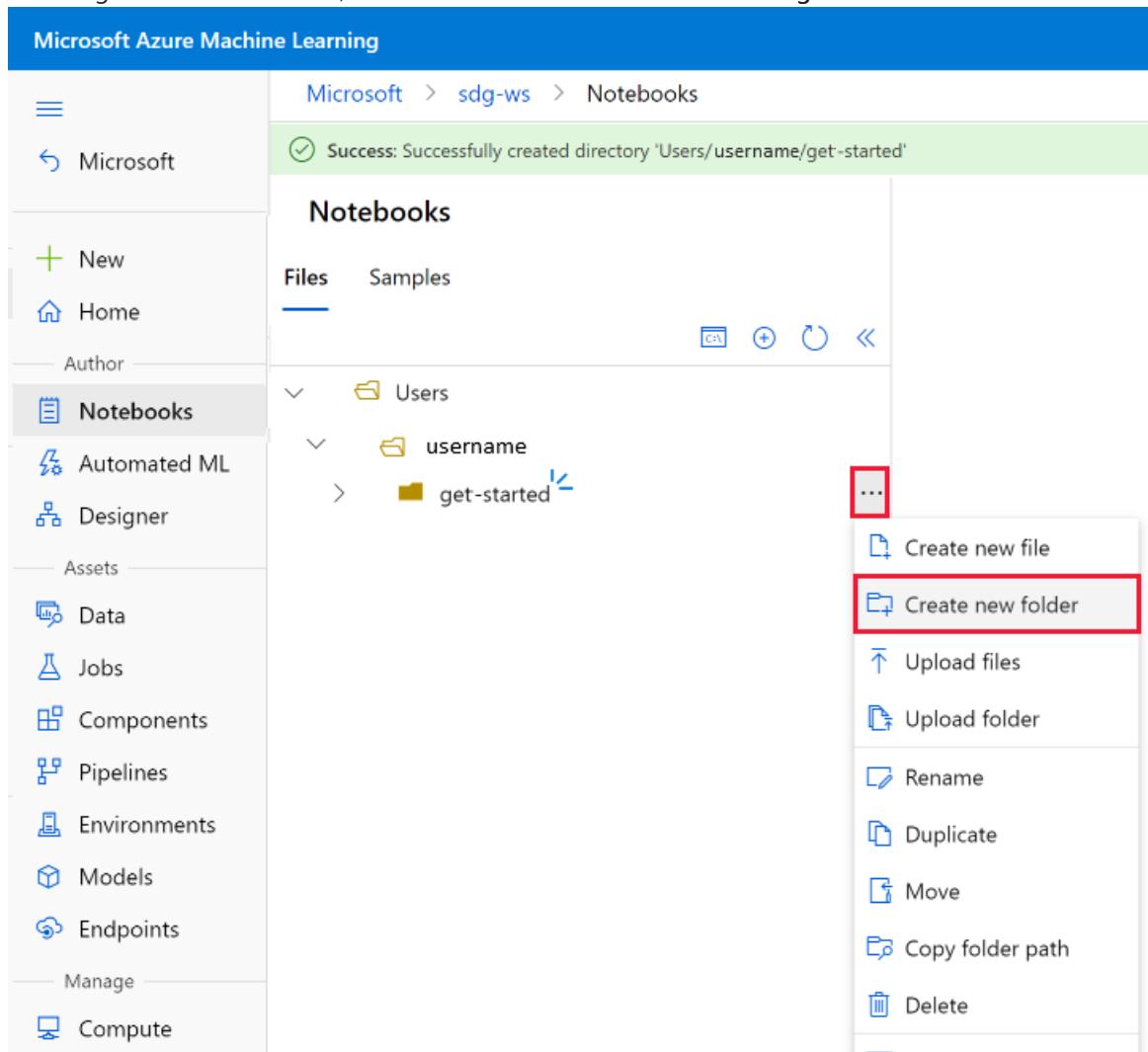
1. Sign in to the [Azure Machine Learning studio](#) and select your workspace if prompted.
2. On the left, select **Notebooks**

3. In the Files toolbar, select +, then select **Create new folder**.



4. Name the folder **get-started**.

5. To the right of the folder name, use the ... to create another folder under **get-started**.



6. Name the new folder **src**. Use the **Edit location** link if the file location is not correct.

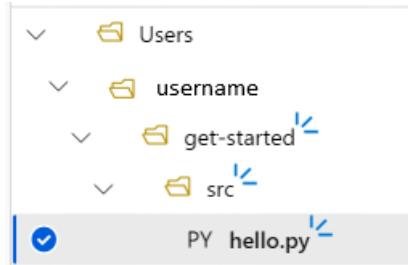
7. To the right of the **src** folder, use the ... to create a new file in the **src** folder.

8. Name your file **hello.py**. Switch the **File type** to **Python (.py)***.

Copy this code into your file:

```
# src/hello.py
print("Hello world!")
```

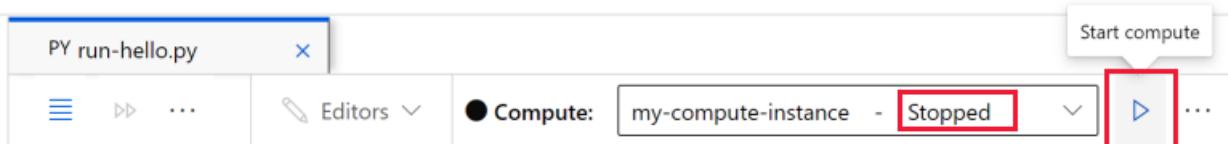
Your project folder structure will now look like:



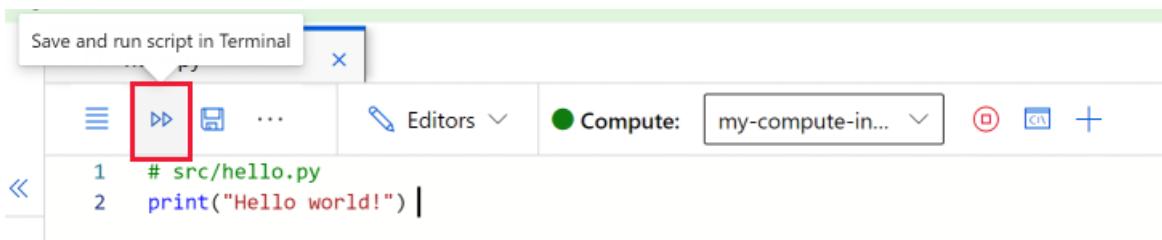
Test your script

You can run your code locally, which in this case means on the compute instance. Running code locally has the benefit of interactive debugging of code.

If you have previously stopped your compute instance, start it now with the **Start compute** tool to the right of the compute dropdown. Wait about a minute for state to change to *Running*.



Select **Save and run script in terminal** to run the script.



You'll see the output of the script in the terminal window that opens. Close the tab and select **Terminate** to close the session.

Create a control script

A *control script* allows you to run your `hello.py` script on different compute resources. You use the control script to control how and where your machine learning code is run.

Select the ... at the end of **get-started** folder to create a new file. Create a Python file called `run-hello.py` and copy/paste the following code into that file:

```

# get-started/run-hello.py
from azure.ai.ml import MLClient, command, Input
from azure.identity import DefaultAzureCredential
from azureml.core import Workspace

# get details of the current Azure ML workspace
ws = Workspace.from_config()

# default authentication flow for Azure applications
default_azure_credential = DefaultAzureCredential()
subscription_id = ws.subscription_id
resource_group = ws.resource_group
workspace = ws.name

# client class to interact with Azure ML services and resources, e.g. workspaces, jobs, models and so on.
ml_client = MLClient(
    default_azure_credential,
    subscription_id,
    resource_group,
    workspace)

# target name of compute where job will be executed
computeName="cpu-cluster"
job = command(
    code=".src",
    command="python hello.py",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute=computeName,
    display_name="hello-world-example",
)
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)

```

TIP

If you used a different name when you created your compute cluster, make sure to adjust the name in the code `computeName='cpu-cluster'` as well.

Understand the code

Here's a description of how the control script works:

```
ml_client = MLClient(DefaultAzureCredential(), subscription_id, resource_group, workspace)
```

MLClient manages your Azure Machine Learning workspace and its assets and resources.

```
job = command(...)
```

command provides a simple way to construct a standalone command job or one as part of a dsl.pipeline.

```
returned_job = ml_client.create_or_update(job)
```

Submits your script. This submission is called a **job**. A job encapsulates a single execution of your code. Use a job to monitor the script progress, capture the output, analyze the results, visualize metrics, and more.

```
aml_url = returned_job.studio_url
```

The `job` object provides a handle on the execution of your code. Monitor its progress from the Azure Machine Learning studio with the URL that's printed from the Python script.

Submit and run your code in the cloud

1. Select **Save and run script in terminal** to run your control script, which in turn runs `hello.py` on the compute cluster that you created in the [setup tutorial](#).
2. In the terminal, you may be asked to sign in to authenticate. Copy the code and follow the link to complete this step.
3. Once you're authenticated, you'll see a link in the terminal. Select the link to view the job.

NOTE

You may see some warnings starting with *Failure while loading azureml_run_type_providers....* You can ignore these warnings. Use the link at the bottom of these warnings to view your output.

View the output

1. In the page that opens, you'll see the job status.
2. When the status of the job is **Completed**, select **Output + logs** at the top of the page.
3. Select `std_log.txt` to view the output of your job.

Monitor your code in the cloud in the studio

The output from your script will contain a link to the studio that looks something like this:

```
https://ml.azure.com/runs/<run-id>?wsid=/subscriptions/<subscription-id>/resourcegroups/<resource-group>/workspaces/<workspace-name>
```

Follow the link. At first, you'll see a status of **Queued** or **Preparing**. The very first run will take 5-10 minutes to complete. This is because the following occurs:

- A docker image is built in the cloud
- The compute cluster is resized from 0 to 1 node
- The docker image is downloaded to the compute.

Subsequent jobs are much quicker (~15 seconds) as the docker image is cached on the compute. You can test this by resubmitting the code below after the first job has completed.

Wait about 10 minutes. You'll see a message that the run has completed. Then use **Refresh** to see the status change to **Completed**. Once the job completes, go to the **Outputs + logs** tab.

The `std_log.txt` file contains the standard output from a run. This file can be useful when you're debugging remote runs in the cloud.

```
hello world
```

Next steps

In this tutorial, you took a simple "Hello world!" script and ran it on Azure. You saw how to connect to your Azure Machine Learning workspace, create an experiment, and submit your `hello.py` code to the cloud.

In the next tutorial, you build on these learnings by running something more interesting than
`print("Hello world!")`.

[Tutorial: Train a model](#)

NOTE

If you want to finish the tutorial series here and not progress to the next step, remember to [clean up your resources](#).

Tutorial: Train your first machine learning model (part 2 of 3)

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

This tutorial shows you how to train a machine learning model in Azure Machine Learning. This tutorial is *part 2 of a three-part tutorial series*.

In [Part 1: Run "Hello world!"](#) of the series, you learned how to use a control script to run a job in the cloud.

In this tutorial, you take the next step by submitting a script that trains a machine learning model. This example will help you understand how Azure Machine Learning eases consistent behavior between local debugging and remote runs.

In this tutorial, you:

- Create a training script.
- Use Conda to define an Azure Machine Learning environment.
- Create a control script.
- Understand Azure Machine Learning classes ([Environment](#), [Run](#), [Metrics](#)).
- Submit and run your training script.
- View your code output in the cloud.
- Log metrics to Azure Machine Learning.
- View your metrics in the cloud.

Prerequisites

- Completion of [part 1](#) of the series.

Create training scripts

First you define the neural network architecture in a *model.py* file. All your training code will go into the [src](#) subdirectory, including *model.py*.

The training code is taken from [this introductory example](#) from PyTorch. Note that the Azure Machine Learning concepts apply to any machine learning code, not just PyTorch.

1. Create a *model.py* file in the [src](#) subfolder. Copy this code into the file:

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

2. On the toolbar, select **Save** to save the file. Close the tab if you wish.
3. Next, define the training script, also in the `src` subfolder. This script downloads the CIFAR10 dataset by using PyTorch `torchvision.dataset` APIs, sets up the network defined in `model.py`, and trains it for two epochs by using standard SGD and cross-entropy loss.

Create a `train.py` script in the `src` subfolder:

```

import torch
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

from model import Net

# download CIFAR 10 data
trainset = torchvision.datasets.CIFAR10(
    root="../data",
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor(),
)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=4, shuffle=True, num_workers=2
)

if __name__ == "__main__":
    # define convolutional network
    net = Net()

    # set up pytorch loss / optimizer
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

    # train the network
    for epoch in range(2):

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # unpack the data
            inputs, labels = data

            # zero the parameter gradients
            optimizer.zero_grad()

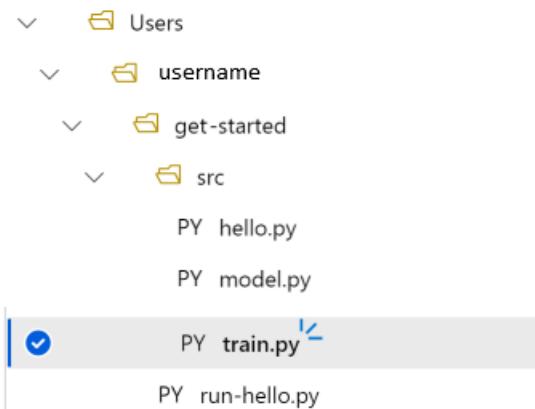
            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999:
                loss = running_loss / 2000
                print(f"epoch={epoch + 1}, batch={i + 1:5}: loss {loss:.2f}")
                running_loss = 0.0

    print("Finished Training")

```

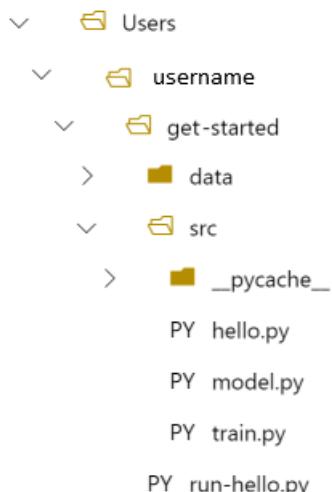
4. You now have the following folder structure:



Test locally

Select **Save and run script in terminal** to run the *train.py* script directly on the compute instance.

After the script completes, select **Refresh** above the file folders. You'll see the new data folder called **get-started/data**. Expand this folder to view the downloaded data.



Create a Python environment

Azure Machine Learning provides the concept of an [environment](#) to represent a reproducible, versioned Python environment for running experiments. It's easy to create an environment from a local Conda or pip environment.

First you'll create a file with the package dependencies.

1. Create a new file in the **get-started** folder called `pytorch-env.yml`:

```
name: pytorch-env
channels:
  - defaults
  - pytorch
dependencies:
  - python=3.8.5
  - pytorch
  - torchvision
```

2. On the toolbar, select **Save** to save the file. Close the tab if you wish.

Create the control script

The difference between the following control script and the one that you used to submit "Hello world!" is that you add a couple of extra lines to set the environment.

Create a new Python file in the **get-started** folder called `run-pytorch.py`:

```
# run-pytorch.py
from azure.ai.ml import MLClient, command, Input
from azure.identity import DefaultAzureCredential
from azure.ai.ml.entities import Environment
from azureml.core import Workspace

if __name__ == "__main__":
    # get details of the current Azure ML workspace
    ws = Workspace.from_config()

    # default authentication flow for Azure applications
    default_azure_credential = DefaultAzureCredential()
    subscription_id = ws.subscription_id
    resource_group = ws.resource_group
    workspace = ws.name

    # client class to interact with Azure ML services and resources, e.g. workspaces, jobs, models and so
    # on.
    ml_client = MLClient(
        default_azure_credential,
        subscription_id,
        resource_group,
        workspace)

    env_name = "pytorch-env"
    env_docker_image = Environment(
        image="pytorch/pytorch:latest",
        name=env_name,
        conda_file="pytorch-env.yml",
    )
    ml_client.environments.create_or_update(env_docker_image)

    # target name of compute where job will be executed
    computeName="cpu-cluster"
    job = command(
        code="./src",
        command="python train.py",
        environment=f"{env_name}@latest",
        compute=computeName,
        display_name="day1-experiment-train",
    )

    returned_job = ml_client.create_or_update(job)
    aml_url = returned_job.studio_url
    print("Monitor your job at", aml_url)
```

TIP

If you used a different name when you created your compute cluster, make sure to adjust the name in the code

```
computeName='cpu-cluster'
```

as well.

Understand the code changes

```
env_docker_image = ...
```

Creates the custom environment against the pytorch base image, with additional conda file to install.

```
environment=f"{env_name}@latest"
```

Adds the environment to [command](#).

Submit the job to Azure Machine Learning

1. Select **Save and run script in terminal** to run the *run-pytorch.py* script.
2. You'll see a link in the terminal window that opens. Select the link to view the job.

NOTE

You may see some warnings starting with *Failure while loading azurerm/run_type_providers...*. You can ignore these warnings. Use the link at the bottom of these warnings to view your output.

View the output

1. In the page that opens, you'll see the job status. The first time you run this script, Azure Machine Learning will build a new Docker image from your PyTorch environment. The whole job might take around 10 minutes to complete. This image will be reused in future jobs to make them job much quicker.
2. You can see view Docker build logs in the Azure Machine Learning studio. Select the **Outputs + logs** tab, and then select **20_image_build_log.txt**.
3. When the status of the job is **Completed**, select **Output + logs**.
4. Select **std_log.txt** to view the output of your job.

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ../data/cifar-10-python.tar.gz
Extracting ../data/cifar-10-python.tar.gz to ../data
epoch=1, batch= 2000: loss 2.30
epoch=1, batch= 4000: loss 2.17
epoch=1, batch= 6000: loss 1.99
epoch=1, batch= 8000: loss 1.87
epoch=1, batch=10000: loss 1.72
epoch=1, batch=12000: loss 1.63
epoch=2, batch= 2000: loss 1.54
epoch=2, batch= 4000: loss 1.53
epoch=2, batch= 6000: loss 1.50
epoch=2, batch= 8000: loss 1.46
epoch=2, batch=10000: loss 1.44
epoch=2, batch=12000: loss 1.41
Finished Training
```

If you see an error `Your total snapshot size exceeds the limit`, the **data** folder is located in the `source_directory` value used in `ScriptRunConfig`.

Select the ... at the end of the folder, then select **Move** to move **data** to the **get-started** folder.

Log training metrics

Now that you have a model training script in Azure Machine Learning, let's start tracking some performance metrics.

The current training script prints metrics to the terminal. Azure Machine Learning supports logging and tracking experiments using [MLflow tracking](#). By adding a few lines of code, you gain the ability to visualize metrics in the studio and to compare metrics between multiple jobs.

Modify *train.py* to include logging

1. Modify your *train.py* script to include two more lines of code:

```

import torch
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from model import Net
import mlflow

# ADDITIONAL CODE: OPTIONAL: turn on autolog
# mlflow.autolog()

# download CIFAR 10 data
trainset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor()
)
trainloader = torch.utils.data.DataLoader(
    trainset,
    batch_size=4,
    shuffle=True,
    num_workers=2
)

if __name__ == "__main__":
    # define convolutional network
    net = Net()
    # set up pytorch loss / optimizer
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
    # train the network
    for epoch in range(2):
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # unpack the data
            inputs, labels = data
            # zero the parameter gradients
            optimizer.zero_grad()
            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999:
                loss = running_loss / 2000
                # ADDITIONAL CODE: log loss metric to AML
                mlflow.log_metric('loss', loss)
                print(f'epoch={epoch + 1}, batch={i + 1:5}: loss {loss:.2f}')
                running_loss = 0.0
        print('Finished Training')

```

2. Save this file, then close the tab if you wish.

Understand the additional two lines of code

```

# ADDITIONAL CODE: OPTIONAL: turn on autolog
mlflow.autolog()

```

With Azure Machine Learning and MLFlow, users can log metrics, model parameters and model artifacts automatically when training a model.

```
# ADDITIONAL CODE: log loss metric to AML
mlflow.log_metric('loss', loss)
```

You can log individual metrics as well.

Metrics in Azure Machine Learning are:

- Organized by experiment and job, so it's easy to keep track of and compare metrics.
- Equipped with a UI so you can visualize training performance in the studio.
- Designed to scale, so you keep these benefits even as you run hundreds of experiments.

Update the Conda environment file

The `train.py` script just took a new dependency on `azureml.core`. Update `pytorch-env.yml` to reflect this change:

```
name: pytorch-env
channels:
  - defaults
  - pytorch
dependencies:
  - python=3.8.5
  - pytorch
  - torchvision
  - pip
  - pip:
    - mlflow
    - azureml-mlflow
```

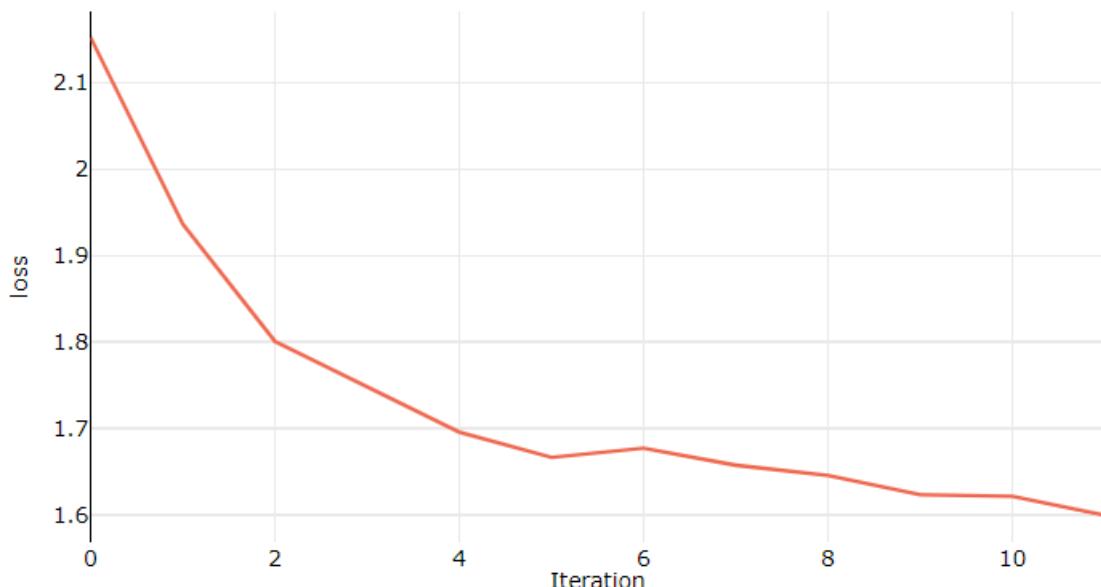
Make sure you save this file before you submit the job.

Submit the job to Azure Machine Learning

Select the tab for the `run-pytorch.py` script, then select **Save and run script in terminal** to re-run the `run-pytorch.py` script. Make sure you've saved your changes to `pytorch-env.yml` first.

This time when you visit the studio, go to the **Metrics** tab where you can now see live updates on the model training loss! It may take 1 to 2 minutes before the training begins.

loss



Next steps

In this session, you upgraded from a basic "Hello world!" script to a more realistic training script that required a specific Python environment to run. You saw how to use curated Azure Machine Learning environments. Finally, you saw how in a few lines of code you can log metrics to Azure Machine Learning.

There are other ways to create Azure Machine Learning environments, including [from a pip requirements.txt file](#) or [from an existing local Conda environment](#).

In the next session, you'll see how to work with data in Azure Machine Learning by uploading the CIFAR10 dataset to Azure.

[Tutorial: Bring your own data](#)

NOTE

If you want to finish the tutorial series here and not progress to the next step, remember to [clean up your resources](#).

Tutorial: Upload data and train a model (part 3 of 3)

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

This tutorial shows you how to upload and use your own data to train machine learning models in Azure Machine Learning. This tutorial is *part 3 of a three-part tutorial series*.

In [Part 2: Train a model](#), you trained a model in the cloud, using sample data from `PyTorch`. You also downloaded that data through the `torchvision.datasets.CIFAR10` method in the PyTorch API. In this tutorial, you'll use the downloaded data to learn the workflow for working with your own data in Azure Machine Learning.

In this tutorial, you:

- Upload data to Azure.
- Create a control script.
- Understand the new Azure Machine Learning concepts (passing parameters, data inputs).
- Submit and run your training script.
- View your code output in the cloud.

Prerequisites

You'll need the data that was downloaded in the previous tutorial. Make sure you have completed these steps:

1. [Create the training script](#).
2. [Test locally](#).

Adjust the training script

By now you have your training script (`get-started/src/train.py`) running in Azure Machine Learning, and you can monitor the model performance. Let's parameterize the training script by introducing arguments. Using arguments will allow you to easily compare different hyperparameters.

Our training script is currently set to download the CIFAR10 dataset on each run. The following Python code has been adjusted to read the data from a directory.

NOTE

The use of `argparse` parameterizes the script.

1. Open `train.py` and replace it with this code:

```
import os
import argparse
import torch
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from model import Net
import mlflow

if __name__ == "__main__":
```

```

parser = argparse.ArgumentParser()
parser.add_argument(
    '--data_path',
    type=str,
    help='Path to the training data'
)
parser.add_argument(
    '--learning_rate',
    type=float,
    default=0.001,
    help='Learning rate for SGD'
)
parser.add_argument(
    '--momentum',
    type=float,
    default=0.9,
    help='Momentum for SGD'
)
args = parser.parse_args()
print("===== DATA =====")
print("DATA PATH: " + args.data_path)
print("LIST FILES IN DATA PATH...")
print(os.listdir(args.data_path))
print("=====")
# prepare DataLoader for CIFAR10 data
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
trainset = torchvision.datasets.CIFAR10(
    root=args.data_path,
    train=True,
    download=False,
    transform=transform,
)
trainloader = torch.utils.data.DataLoader(
    trainset,
    batch_size=4,
    shuffle=True,
    num_workers=2
)
# define convolutional network
net = Net()
# set up pytorch loss / optimizer
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(
    net.parameters(),
    lr=args.learning_rate,
    momentum=args.momentum,
)
# train the network
for epoch in range(2):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # unpack the data
        inputs, labels = data
        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:
            loss = running_loss / 2000
            mlflow.log_metric('loss', loss)
            print(f'epoch={epoch + 1}, batch={i + 1:5}: loss {loss:.2f}')

```

```
    running_loss = 0.0
    print('Finished Training')
```

2. Save the file. Close the tab if you wish.

Understanding the code changes

The code in `train.py` has used the `argparse` library to set up `data_path`, `learning_rate`, and `momentum`.

```
# .... other code
parser = argparse.ArgumentParser()
parser.add_argument('--data_path', type=str, help='Path to the training data')
parser.add_argument('--learning_rate', type=float, default=0.001, help='Learning rate for SGD')
parser.add_argument('--momentum', type=float, default=0.9, help='Momentum for SGD')
args = parser.parse_args()
# ... other code
```

Also, the `train.py` script was adapted to update the optimizer to use the user-defined parameters:

```
optimizer = optim.SGD(
    net.parameters(),
    lr=args.learning_rate,      # get learning rate from command-line argument
    momentum=args.momentum,     # get momentum from command-line argument
)
```

Upload the data to Azure

To run this script in Azure Machine Learning, you need to make your training data available in Azure. Your Azure Machine Learning workspace comes equipped with a *default* datastore. This is an Azure Blob Storage account where you can store your training data.

NOTE

Azure Machine Learning allows you to connect other cloud-based storages that store your data. For more details, see the [data documentation](#).

TIP

There is no additional step needed for uploading data, the control script will define and upload the CIFAR10 training data.

Create a control script

As you've done previously, create a new Python control script called `run-pytorch-data.py` in the `get-started` folder:

```

# run-pytorch-data.py
from azure.ai.ml import MLClient, command, Input
from azure.identity import DefaultAzureCredential
from azure.ai.ml.entities import Environment
from azure.ai.ml import command, Input
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes
from azureml.core import Workspace

if __name__ == "__main__":
    # get details of the current Azure ML workspace
    ws = Workspace.from_config()

    # default authentication flow for Azure applications
    default_azure_credential = DefaultAzureCredential()
    subscription_id = ws.subscription_id
    resource_group = ws.resource_group
    workspace = ws.name

    # client class to interact with Azure ML services and resources, e.g. workspaces, jobs, models and so on.
    ml_client = MLClient(
        default_azure_credential,
        subscription_id,
        resource_group,
        workspace)

    # the key here should match the key passed to the command
    my_job_inputs = {
        "data_path": Input(type=AssetTypes.URI_FOLDER, path="./data")
    }

    env_name = "pytorch-env"
    env_docker_image = Environment(
        image="pytorch/pytorch:latest",
        name=env_name,
        conda_file="pytorch-env.yml",
    )
    ml_client.environments.create_or_update(env_docker_image)

    # target name of compute where job will be executed
    computeName="cpu-cluster"
    job = command(
        code="./src",
        # the parameter will match the training script argument name
        # inputs.data_path key should match the dictionary key
        command="python train.py --data_path ${{inputs.data_path}}",
        inputs=my_job_inputs,
        environment=f"{env_name}@latest",
        compute=computeName,
        display_name="day1-experiment-data",
    )

    returned_job = ml_client.create_or_update(job)
    aml_url = returned_job.studio_url
    print("Monitor your job at", aml_url)

```

TIP

If you used a different name when you created your compute cluster, make sure to adjust the name in the code `computeName='cpu-cluster'` as well.

Understand the code changes

The control script is similar to the one from [part 2 of this series](#), with the following new lines:

```
my_job_inputs = { "data_path": Input(type=AssetTypes.URI_FOLDER, path=".//data") }
```

An **Input** is used to reference inputs to your job. These can encompass data, either uploaded as part of the job or references to previously registered data assets. URI*FOLDER tells that the reference points to a folder of data. The data will be mounted by default to the compute for the job.

```
command="python train.py --data_path ${inputs.data_path}"
```

--data_path matches the argument defined in the updated training script. \${inputs.data_path} passes the input defined by the input dictionary, and the keys must match.

Submit the job to Azure Machine Learning

Select **Save and run script in terminal** to run the *run-pytorch-data.py* script. This job will train the model on the compute cluster using the data you uploaded.

This code will print a URL to the experiment in the Azure Machine Learning studio. If you go to that link, you'll be able to see your code running.

NOTE

You may see some warnings starting with *Failure while loading azureml_run_type_providers...*. You can ignore these warnings. Use the link at the bottom of these warnings to view your output.

Inspect the log file

In the studio, go to the experiment job (by selecting the previous URL output) followed by **Outputs + logs**.

Select the `std_log.txt` file. Scroll down through the log file until you see the following output:

```
===== DATA =====
DATA PATH: /mnt/azureml/cr/j/xxxxxxxxxxxxxxxxxxxxxxxxxxxxx/cap/data-capability/wd/INPUT_data_path
LIST FILES IN DATA PATH...
['.amlignore', 'cifar-10-batches-py', 'cifar-10-python.tar.gz']
=====
epoch=1, batch= 2000: loss 2.20
epoch=1, batch= 4000: loss 1.90
epoch=1, batch= 6000: loss 1.70
epoch=1, batch= 8000: loss 1.58
epoch=1, batch=10000: loss 1.54
epoch=1, batch=12000: loss 1.48
epoch=2, batch= 2000: loss 1.41
epoch=2, batch= 4000: loss 1.38
epoch=2, batch= 6000: loss 1.33
epoch=2, batch= 8000: loss 1.30
epoch=2, batch=10000: loss 1.29
epoch=2, batch=12000: loss 1.25
Finished Training
```

Notice:

- Azure Machine Learning has mounted Blob Storage to the compute cluster automatically for you, passing the mount point into `--data_path`. Compared to the previous job, there is no on the fly data download.
- The `inputs=my_job_inputs` used in the control script resolves to the mount point.

Clean up resources

If you plan to continue now to another tutorial, or to start your own training jobs, skip to [Next steps](#).

Stop compute instance

If you're not going to use it now, stop the compute instance:

1. In the studio, on the left, select **Compute**.
2. In the top tabs, select **Compute instances**
3. Select the compute instance in the list.
4. On the top toolbar, select **Stop**.

Delete all resources

IMPORTANT

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.

The screenshot shows the Azure portal interface for managing a resource group named 'my-rg'. The 'Overview' tab is active. At the top right, there is a red box around the 'Delete resource group' button. The main area displays subscription details ('Visual Studio Ultimate with MSDN'), deployment status ('1 Succeeded'), and a list of resources. The list includes three items: 'myworkspace' (Machine Learning service workspace), 'my-workspace' (Machine Learning service workspace), and 'myworkspace0013141752' (Application Insights). There are filters at the bottom for 'Filter by name...', 'All types', and 'All locations'.

4. Enter the resource group name. Then select **Delete**.

You can also keep the resource group but delete a single workspace. Display the workspace properties and select **Delete**.

Next steps

In this tutorial, we saw how to upload data to Azure by using **Datastore**. The datastore served as cloud storage for your workspace, giving you a persistent and flexible place to keep your data.

You saw how to modify your training script to accept a data path via the command line. By using **Dataset**, you were able to mount a directory to the remote job.

Now that you have a model, learn:

[How to deploy models with Azure Machine Learning.](#)

Tutorial: Create production ML pipelines with Python SDK v2 (preview) in a Jupyter notebook

9/22/2022 • 17 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

NOTE

For a tutorial that uses SDK v1 to build a pipeline, see [Tutorial: Build an Azure Machine Learning pipeline for image classification](#)

In this tutorial, you'll use Azure Machine Learning (Azure ML) to create a production ready machine learning (ML) project, using AzureML Python SDK v2 (preview).

You'll learn how to use the AzureML Python SDK v2 to:

- Connect to your Azure ML workspace
- Create Azure ML data assets
- Create reusable Azure ML components
- Create, validate and run Azure ML pipelines
- Deploy the newly-trained model as an endpoint
- Call the Azure ML endpoint for inferencing

Prerequisites

- Complete the [Quickstart: Get started with Azure Machine Learning](#) to:
 - Create a workspace.
 - Create a cloud-based compute instance to use for your development environment.
 - Create a cloud-based compute cluster to use for training your model.

Install the SDK

You'll complete the following experiment setup and run steps in Azure Machine Learning studio. This consolidated interface includes machine learning tools to perform data science scenarios for data science practitioners of all skill levels.

First you'll install the v2 SDK on your compute instance:

1. Sign in to [Azure Machine Learning studio](#).
2. Select the subscription and the workspace you created as part of the [Prerequisites](#).
3. On the left, select **Compute**.

4. From the list of **Compute Instances**, find the one you created.
5. Select "Terminal", to open the terminal session on the compute instance.
6. In the terminal window, install Python SDK v2 (preview) with this command:

```
pip install --pre azure-ai-ml
```

For more information, see [Install the Python SDK v2](#).

Clone the azureml-examples repo

1. Now on the terminal, run the command:

```
git clone --depth 1 https://github.com/Azure/azureml-examples
```

2. On the left, select **Notebooks**.

3. Now, on the left, Select the **Files**

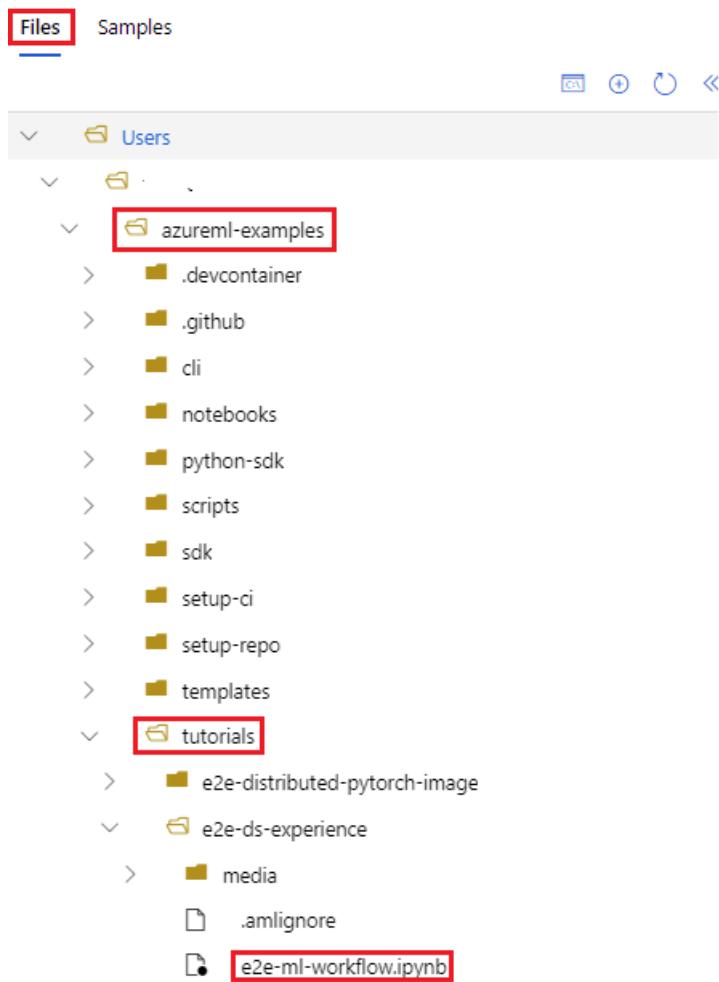


4. A list of folders shows each user who accesses the workspace. Select your folder, you'll find **azureml-examples** is cloned.

Open the cloned notebook

1. Open the **tutorials** folder that was cloned into your **User files** section.
2. Select the **e2e-ml-workflow.ipynb** file from your **azureml-examples/tutorials/e2e-ds-experience/** folder.

Notebooks



3. On the top bar, select the compute instance you created during the [Quickstart: Get started with Azure Machine Learning](#) to use for running the notebook.

IMPORTANT

The rest of this article contains the same content as you see in the notebook.

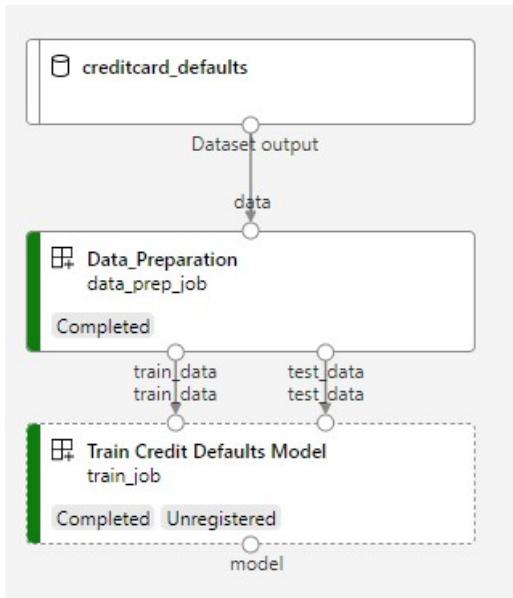
Switch to the Jupyter Notebook now if you want to run the code while you read along. To run a single code cell in a notebook, click the code cell and hit **Shift+Enter**. Or, run the entire notebook by choosing **Run all** from the top toolbar

Introduction

In this tutorial, you'll create an Azure ML pipeline to train a model for credit default prediction. The pipeline handles the data preparation, training and registering the trained model. You'll then run the pipeline, deploy the model and use it.

The image below shows the pipeline as you'll see it in the AzureML portal once submitted. It's a rather simple pipeline we'll use to walk you through the AzureML SDK v2.

The two steps are first data preparation and second training.



Set up the pipeline resources

The Azure ML framework can be used from CLI, Python SDK, or studio interface. In this example, you'll use the AzureML Python SDK v2 to create a pipeline.

Before creating the pipeline, you'll set up the resources the pipeline will use:

- The data asset for training
- The software environment to run the pipeline
- A compute resource to where the job will run

Connect to the workspace

Before we dive in the code, you'll need to connect to your Azure ML workspace. The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning.

```

# Handle to the workspace
from azure.ai.ml import MLClient

# Authentication package
from azure.identity import DefaultAzureCredential, InteractiveBrowserCredential

try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()

```

In the next cell, enter your Subscription ID, Resource Group name and Workspace name. To find your Subscription ID:

1. In the upper right Azure Machine Learning studio toolbar, select your workspace name.
2. You'll see the values you need for <SUBSCRIPTION_ID>, <RESOURCE_GROUP>, and <AML_WORKSPACE_NAME>.
3. Copy a value, then close the window and paste that into your code. Open the tool again to get the next value.

Directory + Subscription +
Workspace

Current directory
Microsoft

Current subscription
ML-docs

Current workspace
my-workspace

Resource Group
sgilley-rg

Subscription ID
xxxx-xxxx-xxxxxx-xxxxxx-xxxxxx-xxxxxx-xxxxxx-xxxxxx

Location
eastus2

Download config file

View all properties in Azure Portal

```
# Get a handle to the workspace
ml_client = MLClient(
    credential=credential,
    subscription_id=<SUBSCRIPTION_ID>,
    resource_group_name=<RESOURCE_GROUP>,
    workspace_name=<AML_WORKSPACE_NAME>,
)
```

The result is a handler to the workspace that you'll use to manage other resources and jobs.

IMPORTANT

Creating `MLClient` will not connect to the workspace. The client initialization is lazy, it will wait for the first time it needs to make a call (in the notebook below, that will happen during dataset registration).

Register data from an external url

The data you use for training is usually in one of the locations below:

- Local machine
- Web
- Big Data Storage services (for example, Azure Blob, Azure Data Lake Storage, SQL)

Azure ML uses a `Data` object to register a reusable definition of data, and consume data within a pipeline. In the section below, you'll consume some data from web url as one example. Data from other sources can be created as well. `Data` assets from other sources can be created as well.

```

from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

web_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%20clients.xls"

credit_data = Data(
    name="creditcard_defaults",
    path=web_path,
    type=AssetTypes.URI_FILE,
    description="Dataset for credit card defaults",
    tags={"source_type": "web", "source": "UCI ML Repo"},
    version="1.0.0",
)

```

This code just created a `Data` asset, ready to be consumed as an input by the pipeline that you'll define in the next sections. In addition, you can register the data to your workspace so it becomes reusable across pipelines.

Registering the data asset will enable you to:

- Reuse and share the data asset in future pipelines
- Use versions to track the modification to the data asset
- Use the data asset from Azure ML designer, which is Azure ML's GUI for pipeline authoring

Since this is the first time that you're making a call to the workspace, you may be asked to authenticate. Once the authentication is complete, you'll then see the dataset registration completion message.

```

credit_data = ml_client.data.create_or_update(credit_data)
print(
    f"Dataset with name {credit_data.name} was registered to workspace, the dataset version is {credit_data.version}"
)

```

In the future, you can fetch the same dataset from the workspace using

```
credit_dataset = ml_client.data.get("<DATA ASSET NAME>", version='<VERSION>').
```

Create a compute resource to run your pipeline

Each step of an Azure ML pipeline can use a different compute resource for running the specific job of that step. It can be single or multi-node machines with Linux or Windows OS, or a specific compute fabric like Spark.

In this section, you'll provision a Linux [compute cluster](#). See the [full list on VM sizes and prices](#).

For this tutorial you only need a basic cluster, so we'll use a Standard_DS3_v2 model with 2 vCPU cores, 7 GB RAM and create an Azure ML Compute.

TIP

If you already have a compute cluster, replace "cpu-cluster" in the code below with the name of your cluster. This will keep you from creating another one.

```

from azure.ai.ml.entities import AmlCompute

cpu_compute_target = "cpu-cluster"

try:
    # let's see if the compute target already exists
    cpu_cluster = ml_client.compute.get(cpu_compute_target)
    print(
        f"You already have a cluster named {cpu_compute_target}, we'll reuse it as is."
    )

except Exception:
    print("Creating a new cpu compute target...")

    # Let's create the Azure ML compute object with the intended parameters
    cpu_cluster = AmlCompute(
        # Name assigned to the compute cluster
        name="cpu-cluster",
        # Azure ML Compute is the on-demand VM service
        type="amlcompute",
        # VM Family
        size="STANDARD_DS3_V2",
        # Minimum running nodes when there is no job running
        min_instances=0,
        # Nodes in cluster
        max_instances=4,
        # How many seconds will the node running after the job termination
        idle_time_before_scale_down=180,
        # Dedicated or LowPriority. The latter is cheaper but there is a chance of job termination
        tier="Dedicated",
    )

    # Now, we pass the object to MLClient's create_or_update method
    cpu_cluster = ml_client.begin_create_or_update(cpu_cluster)

print(
    f"AMLCompute with name {cpu_cluster.name} is created, the compute size is {cpu_cluster.size}"
)

```

Create a job environment for pipeline steps

So far, you've created a development environment on the compute instance, your development machine. You'll also need an environment to use for each step of the pipeline. Each step can have its own environment, or you can use some common environments for multiple steps.

In this example, you'll create a conda environment for your jobs, using a conda yaml file. First, create a directory to store the file in.

```

import os

dependencies_dir = "./dependencies"
os.makedirs(dependencies_dir, exist_ok=True)

```

Now, create the file in the dependencies directory.

```

%%writefile {dependencies_dir}/conda.yml
name: model-env
channels:
- conda-forge
dependencies:
- python=3.8
- numpy=1.21.2
- pip=21.2.4
- scikit-learn=0.24.2
- scipy=1.7.1
- pandas>=1.1,<1.2
- pip:
- inference-schema[numpy-support]==1.3.0
- xlrd==2.0.1
- mlflow== 1.26.1
- azureml-mlflow==1.42.0

```

The specification contains some usual packages, that you'll use in your pipeline (numpy, pip), together with some Azure ML specific packages (azureml-defaults, azureml-mlflow).

The Azure ML packages aren't mandatory to run Azure ML jobs. However, adding these packages will let you interact with Azure ML for logging metrics and registering models, all inside the Azure ML job. You'll use them in the training script later in this tutorial.

Use the *yaml* file to create and register this custom environment in your workspace:

```

from azure.ai.ml.entities import Environment

custom_env_name = "aml-scikit-learn"

pipeline_job_env = Environment(
    name=custom_env_name,
    description="Custom environment for Credit Card Defaults pipeline",
    tags={"scikit-learn": "0.24.2"},
    conda_file=os.path.join(dependencies_dir, "conda.yml"),
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:latest",
    version="1.0",
)
pipeline_job_env = ml_client.environments.create_or_update(pipeline_job_env)

print(
    f"Environment with name {pipeline_job_env.name} is registered to workspace, the environment version is
{pipeline_job_env.version}"
)

```

Build the training pipeline

Now that you have all assets required to run your pipeline, it's time to build the pipeline itself, using the Azure ML Python SDK v2.

Azure ML pipelines are reusable ML workflows that usually consist of several components. The typical life of a component is:

- Write the yaml specification of the component, or create it programmatically using [ComponentMethod](#).
- Optionally, register the component with a name and version in your workspace, to make it reusable and shareable.
- Load that component from the pipeline code.
- Implement the pipeline using the component's inputs, outputs and parameters
- Submit the pipeline.

Create component 1: data prep (using programmatic definition)

Let's start by creating the first component. This component handles the preprocessing of the data. The preprocessing task is performed in the `data_prep.py` python file.

First create a source folder for the data_prep component:

```
import os

data_prep_src_dir = "./components/data_prep"
os.makedirs(data_prep_src_dir, exist_ok=True)
```

This script performs the simple task of splitting the data into train and test datasets. Azure ML mounts datasets as folders to the computes, therefore, we created an auxiliary `select_first_file` function to access the data file inside the mounted input folder.

[MLFlow](#) will be used to log the parameters and metrics during our pipeline run.

```

%%writefile {data_prep_src_dir}/data_prep.py
import os
import argparse
import pandas as pd
from sklearn.model_selection import train_test_split
import logging
import mlflow

def main():
    """Main function of the script."""

    # input and output arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--data", type=str, help="path to input data")
    parser.add_argument("--test_train_ratio", type=float, required=False, default=0.25)
    parser.add_argument("--train_data", type=str, help="path to train data")
    parser.add_argument("--test_data", type=str, help="path to test data")
    args = parser.parse_args()

    # Start Logging
    mlflow.start_run()

    print(" ".join(f"{k}={v}" for k, v in vars(args).items()))

    print("input data:", args.data)

    credit_df = pd.read_excel(args.data, header=1, index_col=0)

    mlflow.log_metric("num_samples", credit_df.shape[0])
    mlflow.log_metric("num_features", credit_df.shape[1] - 1)

    credit_train_df, credit_test_df = train_test_split(
        credit_df,
        test_size=args.test_train_ratio,
    )

    # output paths are mounted as folder, therefore, we are adding a filename to the path
    credit_train_df.to_csv(os.path.join(args.train_data, "data.csv"), index=False)

    credit_test_df.to_csv(os.path.join(args.test_data, "data.csv"), index=False)

    # Stop Logging
    mlflow.end_run()

if __name__ == "__main__":
    main()

```

Now that you have a script that can perform the desired task, create an Azure ML Component from it.

You'll use the general purpose **CommandComponent** that can run command line actions. This command line action can directly call system commands or run a script. The inputs/outputs are specified on the command line via the `${{ ... }}` notation.

```

from azure.ai.ml import command
from azure.ai.ml import Input, Output

data_prep_component = command(
    name="data_prep_credit_defaults",
    display_name="Data preparation for training",
    description="reads a .xl input, split the input to train and test",
    inputs={
        "data": Input(type="uri_folder"),
        "test_train_ratio": Input(type="number"),
    },
    outputs=dict(
        train_data=Output(type="uri_folder", mode="rw_mount"),
        test_data=Output(type="uri_folder", mode="rw_mount"),
    ),
    # The source folder of the component
    code=data_prep_src_dir,
    command="""python data_prep.py \
        --data ${{inputs.data}} --test_train_ratio ${{inputs.test_train_ratio}} \
        --train_data ${{outputs.train_data}} --test_data ${{outputs.test_data}} \
        """,
    environment=f"{pipeline_job_env.name}:{pipeline_job_env.version}",
)

```

Optionally, register the component in the workspace for future re-use.

Create component 2: training (using yaml definition)

The second component that you'll create will consume the training and test data, train a tree based model and return the output model. You'll use Azure ML logging capabilities to record and visualize the learning progress.

You used the `CommandComponent` class to create your first component. This time you'll use the yaml definition to define the second component. Each method has its own advantages. A yaml definition can actually be checked-in along the code, and would provide a readable history tracking. The programmatic method using `CommandComponent` can be easier with built-in class documentation and code completion.

Create the directory for this component:

```

import os

train_src_dir = "./components/train"
os.makedirs(train_src_dir, exist_ok=True)

```

As you can see in this training script, once the model is trained, the model file is saved and registered to the workspace. Now you can use the registered model in inferencing endpoints.

For the environment of this step, you'll use one of the built-in (curated) Azure ML environments. The tag `azureml`, tells the system to use look for the name in curated environments.

First, create the `yaml` file describing the component:

```

%%writefile {train_src_dir}/train.yml
# <component>
name: train_credit_defaults_model
display_name: Train Credit Defaults Model
# version: 1 # Not specifying a version will automatically update the version
type: command
inputs:
  train_data:
    type: uri_folder
  test_data:
    type: uri_folder
  learning_rate:
    type: number
  registered_model_name:
    type: string
outputs:
  model:
    type: uri_folder
code: .
environment:
  # for this step, we'll use an AzureML curate environment
  azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:21
command: >-
  python train.py
  --train_data ${{inputs.train_data}}
  --test_data ${{inputs.test_data}}
  --learning_rate ${{inputs.learning_rate}}
  --registered_model_name ${{inputs.registered_model_name}}
  --model ${{outputs.model}}
# </component>

```

Now create and register the component:

```

# importing the Component Package
from azure.ai.ml import load_component

# Loading the component from the yml file
train_component = load_component(path=os.path.join(train_src_dir, "train.yml"))

```

```

# Now we register the component to the workspace
train_component = ml_client.create_or_update(train_component)

# Create (register) the component in your workspace
print(
    f"Component {train_component.name} with Version {train_component.version} is registered"
)

```

Create the pipeline from components

Now that both your components are defined and registered, you can start implementing the pipeline.

Here, you'll use *input data*, *split ratio* and *registered model name* as input variables. Then call the components and connect them via their inputs/outputs identifiers. The outputs of each step can be accessed via the `.outputs` property.

The python functions returned by `load_component()` work as any regular python function that we'll use within a pipeline to call each step.

To code the pipeline, you use a specific `@dsl.pipeline` decorator that identifies the Azure ML pipelines. In the decorator, we can specify the pipeline description and default resources like compute and storage. Like a python

function, pipelines can have inputs. You can then create multiple instances of a single pipeline with different inputs.

Here, we used *input data*, *split ratio* and *registered model name* as input variables. We then call the components and connect them via their inputs/outputs identifiers. The outputs of each step can be accessed via the `.outputs` property.

```
# the dsl decorator tells the sdk that we are defining an Azure ML pipeline
from azure.ai.ml import dsl, Input, Output

@dsl.pipeline(
    compute=cpu_compute_target,
    description="E2E data_perp-train pipeline",
)
def credit_defaults_pipeline(
    pipeline_job_data_input,
    pipeline_job_test_train_ratio,
    pipeline_job_learning_rate,
    pipeline_job_registered_model_name,
):
    # using data_prep_function like a python call with its own inputs
    data_prep_job = data_prep_component(
        data=pipeline_job_data_input,
        test_train_ratio=pipeline_job_test_train_ratio,
    )

    # using train_func like a python call with its own inputs
    train_job = train_component(
        train_data=data_prep_job.outputs.train_data, # note: using outputs from previous step
        test_data=data_prep_job.outputs.test_data, # note: using outputs from previous step
        learning_rate=pipeline_job_learning_rate, # note: using a pipeline input as parameter
        registered_model_name=pipeline_job_registered_model_name,
    )

    # a pipeline returns a dictionary of outputs
    # keys will code for the pipeline output identifier
    return {
        "pipeline_job_train_data": data_prep_job.outputs.train_data,
        "pipeline_job_test_data": data_prep_job.outputs.test_data,
    }
```

Now use your pipeline definition to instantiate a pipeline with your dataset, split rate of choice and the name you picked for your model.

```
registered_model_name = "credit_defaults_model"

# Let's instantiate the pipeline with the parameters of our choice
pipeline = credit_defaults_pipeline(
    # pipeline_job_data_input=credit_data,
    pipeline_job_data_input=Input(type="uri_file", path=web_path),
    pipeline_job_test_train_ratio=0.2,
    pipeline_job_learning_rate=0.25,
    pipeline_job_registered_model_name=registered_model_name,
)
```

Submit the job

It's now time to submit the job to run in Azure ML. This time you'll use `create_or_update` on `ml_client.jobs`.

Here you'll also pass an experiment name. An experiment is a container for all the iterations one does on a certain project. All the jobs submitted under the same experiment name would be listed next to each other in

Azure ML studio.

Once completed, the pipeline will register a model in your workspace as a result of training.

```
import webbrowser

# submit the pipeline job
pipeline_job = ml_client.jobs.create_or_update(
    pipeline,
    # Project's name
    experiment_name="e2e_registered_components",
)
# open the pipeline in web browser
webbrowser.open(pipeline_job.services["Studio"].endpoint)
```

An output of "False" is expected from the above cell. You can track the progress of your pipeline, by using the link generated in the cell above.

When you select on each component, you'll see more information about the results of that component. There are two important parts to look for at this stage:

- `Outputs+logs` > `user_logs` > `std_log.txt` This section shows the script run sdtout.

The screenshot shows the Azure ML studio interface with a pipeline titled "Train Credit Defaults Model". The pipeline consists of three main components: "creditcard_defaults" (Dataset output), "Data Preparation data_prep_job" (Completed), and "Train Credit Defaults Model train_job" (Completed, Unregistered). The "Outputs + logs" tab is selected, and the "user_logs" section is expanded, showing the "std_log.txt" file. The log file content is as follows:

```
1 Training with data of shape (24000, 23)
2 precision recall f1-score support
3 0 0.84 0.86 0.89 4674
4 1 0.69 0.35 0.46 1326
5
6 accuracy 0.82 6000
7 macro avg 0.76 0.65 0.68 6000
8 weighted avg 0.80 0.82 0.80 6000
9
10 Registering model credit_defaults_model
11
12
```

- `Outputs+logs` > `Metric` This section shows different logged metrics. In this example, `mlflow autologging`, has automatically logged the training metrics.

The screenshot shows the Azure ML studio interface with the same pipeline structure. The "Metrics" tab is selected. The "View as:" dropdown is set to "Chart". The chart displays the following metrics:

metric	value
training_accuracy_score	0.826
training_f1_score	0.806
training_log_loss	0.412
training_precision_score	0.811
training_recall_score	0.826
training_roc_auc_score	0.806
training_score	0.826

Deploy the model as an online endpoint

Now deploy your machine learning model as a web service in the Azure cloud, an [online endpoint](#). To deploy a machine learning service, you usually need:

- The model assets (filed, metadata) that you want to deploy. You've already registered these assets in your

training component.

- Some code to run as a service. The code executes the model on a given input request. This entry script receives data submitted to a deployed web service and passes it to the model, then returns the model's response to the client. The script is specific to your model. The entry script must understand the data that the model expects and returns. When using a MLFlow model, as in this tutorial, this script is automatically created for you

Create a new online endpoint

Now that you have a registered model and an inference script, it's time to create your online endpoint. The endpoint name needs to be unique in the entire Azure region. For this tutorial, you'll create a unique name using [UUID](#).

```
import uuid

# Creating a unique name for the endpoint
online_endpoint_name = "credit-endpoint-" + str(uuid.uuid4())[:8]
```

```
from azure.ai.ml.entities import (
    ManagedOnlineEndpoint,
    ManagedOnlineDeployment,
    Model,
    Environment,
)

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is an online endpoint",
    auth_mode="key",
    tags={
        "training_dataset": "credit_defaults",
        "model_type": "sklearn.GradientBoostingClassifier",
    },
)

endpoint = ml_client.begin_create_or_update(endpoint)

print(f"Endpoint {endpoint.name} provisioning state: {endpoint.provisioning_state}")
```

Once you've created an endpoint, you can retrieve it as below:

```
endpoint = ml_client.online_endpoints.get(name=online_endpoint_name)

print(
    f'Endpoint "{endpoint.name}" with provisioning state "{endpoint.provisioning_state}" is retrieved'
)
```

Deploy the model to the endpoint

Once the endpoint is created, deploy the model with the entry script. Each endpoint can have multiple deployments and direct traffic to these deployments can be specified using rules. Here you'll create a single deployment that handles 100% of the incoming traffic. We have chosen a color name for the deployment, for example, *blue*, *green*, *red* deployments, which is arbitrary.

You can check the *Models* page on the Azure ML studio, to identify the latest version of your registered model. Alternatively, the code below will retrieve the latest version number for you to use.

```
# Let's pick the latest version of the model
latest_model_version = max(
    [int(m.version) for m in ml_client.models.list(name=registered_model_name)]
)
```

Deploy the latest version of the model.

NOTE

Expect this deployment to take approximately 6 to 8 minutes.

```
# picking the model to deploy. Here we use the latest version of our registered model
model = ml_client.models.get(name=registered_model_name, version=latest_model_version)

# create an online deployment.
blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    instance_type="Standard_DS3_v2",
    instance_count=1,
)

blue_deployment = ml_client.begin_create_or_update(blue_deployment)
```

Test with a sample query

Now that the model is deployed to the endpoint, you can run inference with it.

Create a sample request file following the design expected in the run method in the score script.

```
deploy_dir = "./deploy"
os.makedirs(deploy_dir, exist_ok=True)
```

```
%%writefile {deploy_dir}/sample-request.json
{
    "input_data": {
        "columns": [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22],
        "index": [0, 1],
        "data": [
            [20000,2,2,1,24,2,2,-1,-1,-2,-2,3913,3102,689,0,0,0,0,689,0,0,0,0],
            [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8]
        ]
    }
}
```

```
# test the blue deployment with some sample data
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    request_file="./deploy/sample-request.json",
    deployment_name="blue",
)
```

Clean up resources

If you're not going to use the endpoint, delete it to stop using the resource. Make sure no other deployments are

using an endpoint before you delete it.

NOTE

Expect this step to take approximately 6 to 8 minutes.

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Next steps

Learn more about [Azure ML logging](#).

Train an image classification TensorFlow model using the Azure Machine Learning Visual Studio Code Extension (preview)

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Learn how to train an image classification model to recognize hand-written numbers using TensorFlow and the Azure Machine Learning Visual Studio Code Extension.

In this tutorial, you learn the following tasks:

- Understand the code
- Create a workspace
- Create a GPU cluster for training
- Train a model

Prerequisites

- Azure subscription. If you don't have one, sign up to try the [free or paid version of Azure Machine Learning](#). If you're using the free subscription, only CPU clusters are supported.
- Install [Visual Studio Code](#), a lightweight, cross-platform code editor.
- Azure Machine Learning Studio Visual Studio Code extension. For install instructions see the [Setup Azure Machine Learning Visual Studio Code extension guide](#)
- CLI (v2). For installation instructions, see [Install, set up, and use the CLI \(v2\)](#)
- Clone the community driven repository

```
git clone https://github.com/Azure/azureml-examples.git
```

Understand the code

The code for this tutorial uses TensorFlow to train an image classification machine learning model that categorizes handwritten digits from 0-9. It does so by creating a neural network that takes the pixel values of 28 px x 28 px image as input and outputs a list of 10 probabilities, one for each of the digits being classified. Below is a sample of what the data looks like.

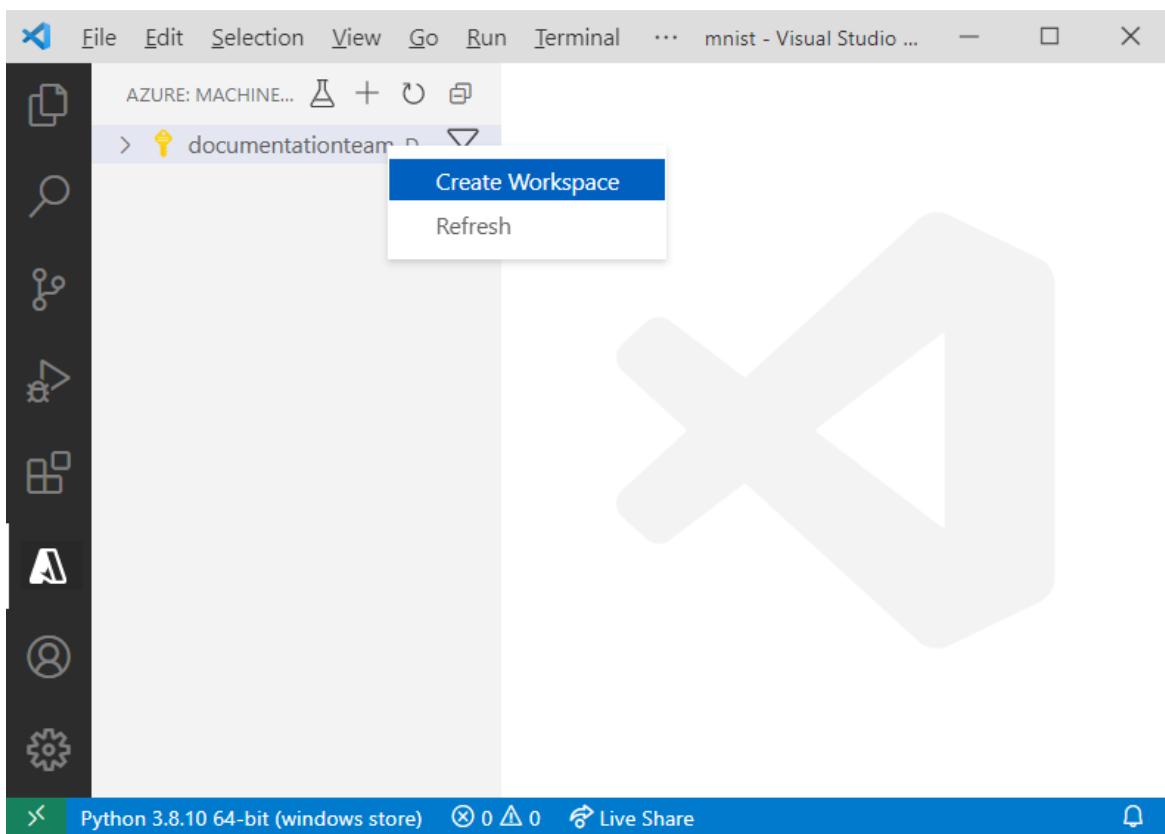
8	9	4	7	9	2	9	4	7	8	4	8	8	2	3	2	4	6	1	1	3	8	8	1	3	1	8	4	1	8
8	9	4	7	9	2	9	4	7	8	4	8	8	2	3	2	4	6	1	1	3	8	8	1	3	1	8	4	1	8

Create a workspace

The first thing you have to do to build an application in Azure Machine Learning is to create a workspace. A workspace contains the resources to train models as well as the trained models themselves. For more information, see [what is a workspace](#).

1. Open the `azureml-examples/cli/jobs/single-step/tensorflow/mnist` directory from the community driven repository in Visual Studio Code.

2. On the Visual Studio Code activity bar, select the **Azure** icon to open the Azure Machine Learning view.
3. In the Azure Machine Learning view, right-click your subscription node and select **Create Workspace**.



4. A specification file appears. Configure the specification file with the following options.

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: TeamWorkspace
location: WestUS2
friendly_name: team-ml-workspace
description: A workspace for training machine learning models
tags:
  purpose: training
  team: ml-team
```

The specification file creates a workspace called `TeamWorkspace` in the `WestUS2` region. The rest of the options defined in the specification file provide friendly naming, descriptions, and tags for the workspace.

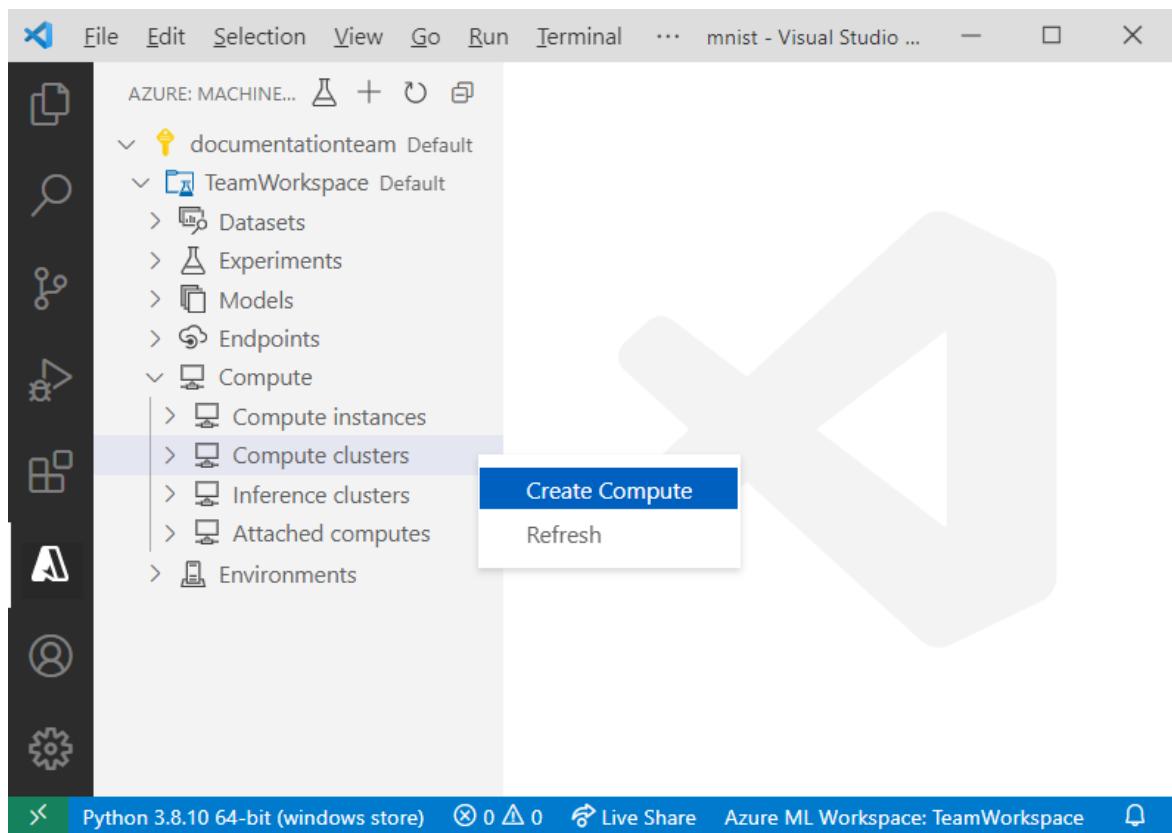
5. Right-click the specification file and select **Azure ML: Execute YAML**. Creating a resource uses the configuration options defined in the YAML specification file and submits a job using the CLI (v2). At this point, a request to Azure is made to create a new workspace and dependent resources in your account. After a few minutes, the new workspace appears in your subscription node.
6. Set `TeamWorkspace` as your default workspace. Doing so places resources and jobs you create in the workspace by default. Select the **Set Azure ML Workspace** button on the Visual Studio Code status bar and follow the prompts to set `TeamWorkspace` as your default workspace.

For more information on workspaces, see [how to manage resources in VS Code](#).

Create a GPU cluster for training

A compute target is the computing resource or environment where you run training jobs. For more information, see the [Azure Machine Learning compute targets documentation](#).

1. In the Azure Machine Learning view, expand your workspace node.
2. Right-click the **Compute clusters** node inside your workspace's **Compute** node and select **Create Compute**



3. A specification file appears. Configure the specification file with the following options.

```
$schema: https://azureschemas.azureedge.net/latest/compute.schema.json
name: gpu-cluster
type: amlcompute
size: Standard_NC12

min_instances: 0
max_instances: 3
idle_time_before_scale_down: 120
```

The specification file creates a GPU cluster called `gpu-cluster` with at most 3 Standard_NC12 VM nodes that automatically scales down to 0 nodes after 120 seconds of inactivity.

For more information on VM sizes, see [sizes for Linux virtual machines in Azure](#).

4. Right-click the specification file and select **Azure ML: Execute YAML**.

After a few minutes, the new compute target appears in the *Compute > Compute clusters* node of your workspace.

Train image classification model

During the training process, a TensorFlow model is trained by processing the training data and learning patterns embedded within it for each of the respective digits being classified.

Like workspaces and compute targets, training jobs are defined using resource templates. For this sample, the specification is defined in the `job.yml` file which looks like the following:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
code:
  local_path: src
command: >
  python train.py
environment: azureml:AzureML-TensorFlow2.4-Cuda11-OpenMp4.1.0-py36:1
compute:
  target: azureml:gpu-cluster
experiment_name: tensorflow-mnist-example
description: Train a basic neural network with TensorFlow on the MNIST dataset.
```

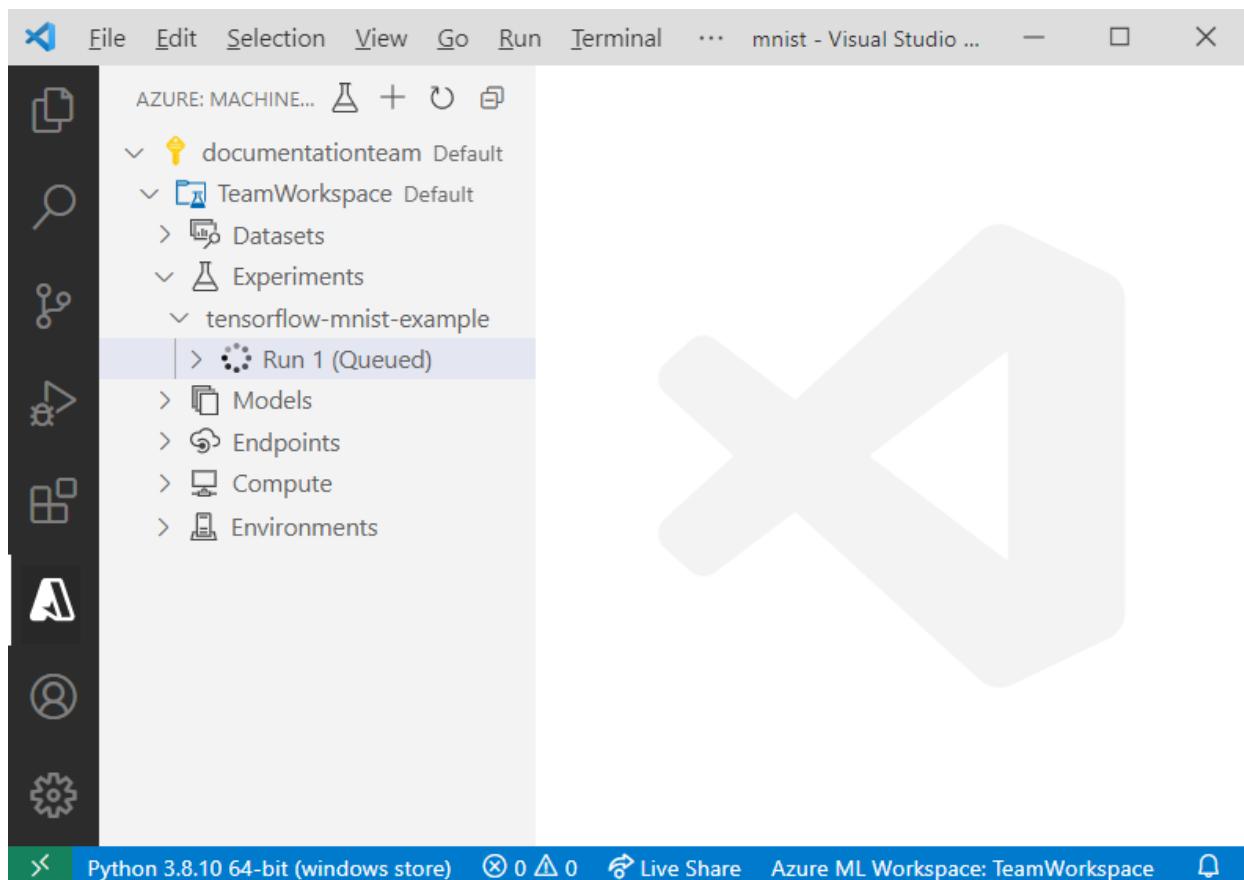
This specification file submits a training job called `tensorflow-mnist-example` to the recently created `gpu-cluster` computer target that runs the code in the `train.py` Python script. The environment used is one of the curated environments provided by Azure Machine Learning which contains TensorFlow and other software dependencies required to run the training script. For more information on curated environments, see [Azure Machine Learning curated environments](#).

To submit the training job:

1. Open the `job.yml` file.
2. Right-click the file in the text editor and select **Azure ML: Execute YAML**.

At this point, a request is sent to Azure to run your experiment on the selected compute target in your workspace. This process takes several minutes. The amount of time to run the training job is impacted by several factors like the compute type and training data size. To track the progress of your experiment, right-click the current run node and select **View Job in Azure portal**.

When the dialog requesting to open an external website appears, select **Open**.



When the model is done training, the status label next to the run node updates to "Completed".

Next steps

In this tutorial, you learn the following tasks:

- Understand the code
- Create a workspace
- Create a GPU cluster for training
- Train a model

For next steps, see:

- [Create and manage Azure Machine Learning resources using Visual Studio Code](#).
- [Connect Visual Studio Code to a compute instance](#) for a full development experience.
- For a walkthrough of how to edit, run, and debug code locally, see the [Python hello-world tutorial](#).
- [Run Jupyter Notebooks in Visual Studio Code](#) using a remote Jupyter server.
- For a walkthrough of how to train with Azure Machine Learning outside of Visual Studio Code, see [Tutorial: Train and deploy a model with Azure Machine Learning](#).

Tutorial: Train an object detection model (preview) with AutoML and Python

9/22/2022 • 15 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

IMPORTANT

The features presented in this article are in preview. They should be considered [experimental](#) preview features that might change at any time.

In this tutorial, you learn how to train an object detection model using Azure Machine Learning automated ML with the Azure Machine Learning CLI extension v2 or the Azure Machine Learning Python SDK v2 (preview). This object detection model identifies whether the image contains objects, such as a can, carton, milk bottle, or water bottle.

Automated ML accepts training data and configuration settings, and automatically iterates through combinations of different feature normalization/standardization methods, models, and hyperparameter settings to arrive at the best model.

You'll write code using the Python SDK in this tutorial and learn the following tasks:

- Download and transform data
- Train an automated machine learning object detection model
- Specify hyperparameter values for your model
- Perform a hyperparameter sweep
- Deploy your model
- Visualize detections

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version](#) of Azure Machine Learning today.
- Python 3.6 or 3.7 are supported for this feature
- Complete the [Quickstart: Get started with Azure Machine Learning](#) if you don't already have an Azure Machine Learning workspace.
- Download and unzip the [*odFridgeObjects.zip](#) data file. The dataset is annotated in Pascal VOC format, where each image corresponds to an xml file. Each xml file contains information on where its corresponding image file is located and also contains information about the bounding boxes and the object labels. In order to use this data, you first need to convert it to the required JSONL format as seen in the [Convert the downloaded data to JSONL](#) section of the notebook.
- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO: Azure CLI ml extension v2 (current)

This tutorial is also available in the [azureml-examples repository on GitHub](#). If you wish to run it in your own local environment, setup using the following instructions

- Install and [set up CLI \(v2\)](#) and make sure you install the `ml` extension.

Compute target setup

You first need to set up a compute target to use for your automated ML model training. Automated ML models for image tasks require GPU SKUs.

This tutorial uses the NCsv3-series (with V100 GPUs) as this type of compute target leverages multiple GPUs to speed up training. Additionally, you can set up multiple nodes to take advantage of parallelism when tuning hyperparameters for your model.

The following code creates a GPU compute of size `Standard_NC24s_v3` with four nodes.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Create a .yml file with the following configuration.

```
$schema: https://azurermschemas.azureedge.net/latest/amlCompute.schema.json
name: gpu-cluster
type: amlcompute
size: Standard_NC24s_v3
min_instances: 0
max_instances: 4
idle_time_before_scale_down: 120
```

To create the compute, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

```
az ml compute create -f [PATH_TO_YML_FILE] --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

The created compute can be provided using `compute` key in the `automl` task configuration yaml:

```
compute: azureml:gpu-cluster
```

Experiment setup

You can use an Experiment to track your model training runs.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Experiment name can be provided using `experiment_name` key as follows:

```
experiment_name: dpv2-cli-automl-image-object-detection-experiment
```

Visualize input data

Once you have the input image data prepared in [JSONL](#) (JSON Lines) format, you can visualize the ground truth bounding boxes for an image. To do so, be sure you have `matplotlib` installed.

```
%pip install --upgrade matplotlib
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.patches as patches
from PIL import Image as pil_image
import numpy as np
import json
import os

def plot_ground_truth_boxes(image_file, ground_truth_boxes):
    # Display the image
    plt.figure()
    img_np = mpimg.imread(image_file)
    img = pil_image.fromarray(img_np.astype("uint8"), "RGB")
    img_w, img_h = img.size

    fig,ax = plt.subplots(figsize=(12, 16))
    ax.imshow(img_np)
    ax.axis("off")

    label_to_color_mapping = {}

    for gt in ground_truth_boxes:
        label = gt["label"]

        xmin, ymin, xmax, ymax = gt["topX"], gt["topY"], gt["bottomX"], gt["bottomY"]
        topleft_x, topleft_y = img_w * xmin, img_h * ymin
        width, height = img_w * (xmax - xmin), img_h * (ymax - ymin)

        if label in label_to_color_mapping:
            color = label_to_color_mapping[label]
        else:
            # Generate a random color. If you want to use a specific color, you can use something like
            "red".
            color = np.random.rand(3)
            label_to_color_mapping[label] = color

        # Display bounding box
        rect = patches.Rectangle((topleft_x, topleft_y), width, height,
                               linewidth=2, edgecolor=color, facecolor="none")
        ax.add_patch(rect)

        # Display label
        ax.text(topleft_x, topleft_y - 10, label, color=color, fontsize=20)

    plt.show()

def plot_ground_truth_boxes_jsonl(image_file, jsonl_file):
    image_base_name = os.path.basename(image_file)
    ground_truth_data_found = False
    with open(jsonl_file) as fp:
        for line in fp.readlines():
            line_json = json.loads(line)
            filename = line_json["image_url"]
            if image_base_name in filename:
                ground_truth_data_found = True
                plot_ground_truth_boxes(image_file, line_json["label"])
                break
    if not ground_truth_data_found:
        print("Unable to find ground truth information for image: {}".format(image_file))
```

Using the above helper functions, for any given image, you can run the following code to display the bounding boxes.

```
image_file = "./odFridgeObjects/images/31.jpg"
jsonl_file = "./odFridgeObjects/train_annotations.jsonl"

plot_ground_truth_boxes_jsonl(image_file, jsonl_file)
```

Upload data and create MLTable

In order to use the data for training, upload data to default Blob Storage of your Azure ML Workspace and register it as an asset. The benefits of registering data are:

- Easy to share with other members of the team
 - Versioning of the metadata (location, description, etc)
 - Lineage tracking
- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

Create a .yml file with the following configuration.

```
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: fridge-items-images-object-detection
description: Fridge-items images Object detection
path: ./data/odFridgeObjects
type: uri_folder
```

To upload the images as a data asset, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

```
az ml data create -f [PATH_TO_YML_FILE] --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Next step is to create `MLTable` from your data in jsonl format as shown below. MLTable package your data into a consumable object for training.

```
paths:
  - file: ./train_annotations.jsonl
transformations:
  - read_json_lines:
      encoding: utf8
      invalid_lines: error
      include_path_column: false
  - convert_column_types:
      - columns: image_url
        column_type: stream_info
```

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

The following configuration creates training and validation data from the MLTable.

```
target_column_name: label
training_data:
  path: data/training-mltable-folder
  type: mltable
validation_data:
  path: data/validation-mltable-folder
  type: mltable
```

Configure your object detection experiment

To configure automated ML runs for image-related tasks, create a task specific AutoML job.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
task: image_object_detection
primary_metric: mean_average_precision
```

In your AutoML job, you can specify the model algorithms by using `model_name` parameter and configure the settings to perform a hyperparameter sweep over a defined search space to find the optimal model.

In this example, we will train an object detection model with `yolov5` and `fasterrcnn_resnet50_fpn`, both of which are pretrained on COCO, a large-scale object detection, segmentation, and captioning dataset that contains over thousands of labeled images with over 80 label categories.

Hyperparameter sweeping for image tasks

You can perform a hyperparameter sweep over a defined search space to find the optimal model.

The following code, defines the search space in preparation for the hyperparameter sweep for each defined algorithm, `yolov5` and `fasterrcnn_resnet50_fpn`. In the search space, specify the range of values for `learning_rate`, `optimizer`, `lr_scheduler`, etc., for AutoML to choose from as it attempts to generate a model with the optimal primary metric. If hyperparameter values are not specified, then default values are used for each algorithm.

For the tuning settings, use random sampling to pick samples from this parameter space by using the `random` sampling_algorithm. Doing so, tells automated ML to try a total of 10 trials with these different samples, running two trials at a time on our compute target, which was set up using four nodes. The more parameters the search space has, the more trials you need to find optimal models.

The Bandit early termination policy is also used. This policy terminates poor performing configurations; that is, those configurations that are not within 20% slack of the best performing configuration, which significantly saves compute resources.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
sweep:  
  limits:  
    max_trials: 10  
    max_concurrent_trials: 2  
    sampling_algorithm: random  
  early_termination:  
    type: bandit  
    evaluation_interval: 2  
    slack_factor: 0.2  
    delay_evaluation: 6
```

```
search_space:  
  - model_name: "yolov5"  
    learning_rate: "uniform(0.0001, 0.01)"  
    model_size: "choice('small', 'medium')"  
  - model_name: "fasterrcnn_resnet50_fpn"  
    learning_rate: "uniform(0.0001, 0.001)"  
    optimizer: "choice('sgd', 'adam', 'adamw')"  
    min_size: "choice(600, 800)"
```

Once the search space and sweep settings are defined, you can then submit the job to train an image model using your training dataset.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

To submit your AutoML job, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

```
az ml job create --file ./hello-automl-job-basic.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

When doing a hyperparameter sweep, it can be useful to visualize the different configurations that were tried using the HyperDrive UI. You can navigate to this UI by going to the 'Child runs' tab in the UI of the main automl_image_run from above, which is the HyperDrive parent run. Then you can go into the 'Child runs' tab of this one.

Alternatively, here below you can see directly the HyperDrive parent run and navigate to its 'Child runs' tab:

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

CLI example not available, please use Python SDK.

Register and deploy model

Once the run completes, you can register the model that was created from the best run (configuration that resulted in the best primary metric). You can either register the model after downloading or by specifying the azureml path with corresponding jobid.

Get the best run

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

```
CLI example not available, please use Python SDK.
```

Register the model

Register the model either using the `azureml` path or your locally downloaded path.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

```
az ml model create --name od-fridge-items-mlflow-model --version 1 --path  
azureml://jobs/$best_run/outputs/artifacts/outputs/mlflow-model/ --type mlflow_model --workspace-name  
[YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

After you register the model you want to use, you can deploy it using the managed online endpoint [deploy-managed-online-endpoint](#)

Configure online endpoint

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json  
name: od-fridge-items-endpoint  
auth_mode: key
```

Create the endpoint

Using the `MLClient` created earlier, we'll now create the Endpoint in the workspace. This command will start the endpoint creation and return a confirmation response while the endpoint creation continues.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

```
az ml online-endpoint create --file .\create_endpoint.yml --workspace-name [YOUR_AZURE_WORKSPACE] --  
resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Configure online deployment

A deployment is a set of resources required for hosting the model that does the actual inferencing. We'll create a deployment for our endpoint using the `ManagedOnlineDeployment` class. You can use either GPU or CPU VM SKUs for your deployment cluster.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

```
name: od-fridge-items-mlflow-deploy
endpoint_name: od-fridge-items-endpoint
model: azureml:od-fridge-items-mlflow-model@latest
instance_type: Standard_DS3_v2
instance_count: 1
liveness_probe:
  failure_threshold: 30
  success_threshold: 1
  timeout: 2
  period: 10
  initial_delay: 2000
readiness_probe:
  failure_threshold: 10
  success_threshold: 1
  timeout: 10
  period: 10
  initial_delay: 2000
```

Create the deployment

Using the `MLClient` created earlier, we'll now create the deployment in the workspace. This command will start the deployment creation and return a confirmation response while the deployment creation continues.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

```
az ml online-deployment create --file .\create_deployment.yml --workspace-name [YOUR_AZURE_WORKSPACE] --
resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Update traffic:

By default the current deployment is set to receive 0% traffic. you can set the traffic percentage current deployment should receive. Sum of traffic percentages of all the deployments with one end point should not exceed 100%.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

```
az ml online-endpoint update --name 'od-fridge-items-endpoint' --traffic 'od-fridge-items-mlflow-deploy=100'
--workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription
[YOUR_AZURE_SUBSCRIPTION]
```

Test the deployment

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

CLI example not available, please use Python SDK.

Visualize detections

Now that you have scored a test image, you can visualize the bounding boxes for this image. To do so, be sure you have matplotlib installed.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

CLI example not available, please use Python SDK.

Clean up resources

Do not complete this section if you plan on running other Azure Machine Learning tutorials.

If you don't plan to use the resources you created, delete them, so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name. Then select **Delete**.

You can also keep the resource group but delete a single workspace. Display the workspace properties and select **Delete**.

Next steps

In this automated machine learning tutorial, you did the following tasks:

- Configured a workspace and prepared data for an experiment.
- Trained an automated object detection model
- Specified hyperparameter values for your model
- Performed a hyperparameter sweep
- Deployed your model
- Visualized detections
- [Learn more about computer vision in automated ML \(preview\)](#).
- [Learn how to set up AutoML to train computer vision models with Python \(preview\)](#).
- [Learn how to configure incremental training on computer vision models](#).
- See [what hyperparameters are available for computer vision tasks](#).
- Code examples:
 - [Azure CLI](#)
 - [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

- Review detailed code examples and use cases in the [azureml-examples repository for automated machine learning samples](#). Please check the folders with 'cli-automl-image-' prefix for samples specific to building computer vision models.

NOTE

Use of the fridge objects dataset is available through the license under the [MIT License](#).

Tutorial: Train a classification model with no-code AutoML in the Azure Machine Learning studio

9/22/2022 • 13 minutes to read • [Edit Online](#)

Learn how to train a classification model with no-code AutoML using Azure Machine Learning automated ML in the Azure Machine Learning studio. This classification model predicts if a client will subscribe to a fixed term deposit with a financial institution.

With automated ML, you can automate away time intensive tasks. Automated machine learning rapidly iterates over many combinations of algorithms and hyperparameters to help you find the best model based on a success metric of your choosing.

You won't write any code in this tutorial, you'll use the studio interface to perform training. You'll learn how to do the following tasks:

- Create an Azure Machine Learning workspace.
- Run an automated machine learning experiment.
- Explore model details.
- Deploy the recommended model.

Also try automated machine learning for these other model types:

- For a no-code example of forecasting, see [Tutorial: Demand forecasting & AutoML](#).
- For a code first example of an object detection model, see the [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#),

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#).
- Download the [bankmarketing_train.csv](#) data file. The `y` column indicates if a customer subscribed to a fixed term deposit, which is later identified as the target column for predictions in this tutorial.

Create a workspace

An Azure Machine Learning workspace is a foundational resource in the cloud that you use to experiment, train, and deploy machine learning models. It ties your Azure subscription and resource group to an easily consumed object in the service.

There are many [ways to create a workspace](#). In this tutorial, you create a workspace via the Azure portal, a web-based console for managing your Azure resources.

1. Sign in to the [Azure portal](#) by using the credentials for your Azure subscription.
2. In the upper-left corner of the Azure portal, select the three bars, then + [Create a resource](#).



+ Create a resource

Home

Dashboard

All services

3. Use the search bar to find **Azure Machine Learning**.

4. Select **Azure Machine Learning**.

Azure Machine Learning X Category : All Pricing : All All

Azure benefit eligible only Publisher Type : All All X Product

Showing 1 to 20 of 362 results for 'Azure Machine Learning'. [Clear search](#)



Azure Machine Learning

Microsoft

Azure Service

Enterprise-grade machine learning to build and deploy models faster

Create ▼ Heart



Jupyter Hub for Machine Learning using Python

Data Science Dojo

Virtual Machine

Our Jupyter Instance provides easy to use environment for Machine Learning applications.

Starts at
Free

Create ▼ Heart

5. In the **Machine Learning** pane, select **Create** to begin.

6. Provide the following information to configure your new workspace:

FIELD	DESCRIPTION
Workspace name	Enter a unique name that identifies your workspace. In this example, we use <code>docs-ws</code> . Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others.

FIELD	DESCRIPTION
Subscription	Select the Azure subscription that you want to use.
Resource group	Use an existing resource group in your subscription, or enter a name to create a new resource group. A resource group holds related resources for an Azure solution. In this example, we use docs-aml .
Region	Select the location closest to your users and the data resources to create your workspace.
Storage account	A storage account is used as the default datastore for the workspace. You may create a new Azure Storage resource or select an existing one in your subscription.
Key vault	A key vault is used to store secrets and other sensitive information that is needed by the workspace. You may create a new Azure Key Vault resource or select an existing one in your subscription.
Application insights	The workspace uses Azure Application Insights to store monitoring information about your deployed models. You may create a new Azure Application Insights resource or select an existing one in your subscription.
Container registry	A container registry is used to register docker images used in training and deployments. You may choose to create a resource or select an existing one in your subscription.

7. After you're finished configuring the workspace, select **Review + Create**.

8. Select **Create** to create the workspace.

WARNING

It can take several minutes to create your workspace in the cloud.

When the process is finished, a deployment success message appears.

9. To view the new workspace, select **Go to resource**.

10. From the portal view of your workspace, select **Launch studio** to go to the Azure Machine Learning studio.

IMPORTANT

Take note of your **workspace** and **subscription**. You'll need these to ensure you create your experiment in the right place.

Sign in to the studio

You complete the following experiment set-up and run steps via the Azure Machine Learning studio at <https://ml.azure.com>, a consolidated web interface that includes machine learning tools to perform data science scenarios for data science practitioners of all skill levels. The studio is not supported on Internet Explorer

browsers.

1. Sign in to [Azure Machine Learning studio](#).
2. Select your subscription and the workspace you created.
3. Select **Get started**.
4. In the left pane, select **Automated ML** under the **Author** section.

Since this is your first automated ML experiment, you'll see an empty list and links to documentation.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The left sidebar has a red box around the 'Automated ML' option under the 'Author' section. The main content area shows a message: 'No recent Automated ML jobs to display.' It includes a link to 'New Automated ML job'. Below this, there's a 'Documentation' section with links to 'Concept: What is Automated ML?', 'Tutorial: Create your first classification model with Automated ML', and 'Blog: Build more accurate forecasts with new capabilities in Automated ML'. A 'View all documentation' link is also present.

5. Select **+ New automated ML job**.

Create and load dataset

Before you configure your experiment, upload your data file to your workspace in the form of an Azure Machine Learning dataset. Doing so, allows you to ensure that your data is formatted appropriately for your experiment.

1. Create a new dataset by selecting **From local files** from the **+Create dataset** drop-down.
 - a. On the **Basic info** form, give your dataset a name and provide an optional description. The automated ML interface currently only supports TabularDatasets, so the dataset type should default to *Tabular*.
 - b. Select **Next** on the bottom left
 - c. On the **Datastore and file selection** form, select the default datastore that was automatically set up during your workspace creation, **workspaceblobstore (Azure Blob Storage)**. This is where you'll upload your data file to make it available to your workspace.
 - d. Select **Upload files** from the **Upload** drop-down.
 - e. Choose the **bankmarketing_train.csv** file on your local computer. This is the file you downloaded as a [prerequisite](#).
 - f. Select **Next** on the bottom left, to upload it to the default container that was automatically set up during your workspace creation.

When the upload is complete, the **Settings and preview** form is pre-populated based on the file type.

- g. Verify that the **Settings and preview** form is populated as follows and select **Next**.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
File format	Defines the layout and type of data stored in a file.	Delimited
Delimiter	One or more characters for specifying the boundary between separate, independent regions in plain text or other data streams.	Comma
Encoding	Identifies what bit to character schema table to use to read your dataset.	UTF-8
Column headers	Indicates how the headers of the dataset, if any, will be treated.	All files have same headers
Skip rows	Indicates how many, if any, rows are skipped in the dataset.	None

- h. The **Schema** form allows for further configuration of your data for this experiment. For this example, select the toggle switch for the **day_of_week**, so as to not include it. Select **Next**.

Create dataset from local files

Schema			
Include	Column name	Properties	Type
Off	Path	Not applicable to selected type	String
Off	age	Not applicable to selected type	Integer
Off	job	Not applicable to selected type	String
Off	marital	Not applicable to selected type	String
Off	education	Not applicable to selected type	String
Off	default	Not applicable to selected type	String
Off	housing	Not applicable to selected type	String
Off	loan	Not applicable to selected type	String

Back Next Cancel

- i. On the **Confirm details** form, verify the information matches what was previously populated on the **Basic info**, **Datastore and file selection** and **Settings and preview** forms.
- j. Select **Create** to complete the creation of your dataset.
- k. Select your dataset once it appears in the list.
- l. Review the **Data preview** to ensure you didn't include **day_of_week** then, select **Close**.
- m. Select **Next**.

Configure job

After you load and configure your data, you can set up your experiment. This setup includes experiment design tasks such as, selecting the size of your compute environment and specifying what column you want to predict.

1. Select the **Create new** radio button.

2. Populate the **Configure Job** form as follows:

- a. Enter this experiment name: `my-1st-automl-experiment`
- b. Select **y** as the target column, what you want to predict. This column indicates whether the client subscribed to a term deposit or not.
- c. Select **compute cluster** as your compute type.
- d. **+ New** to configure your compute target. A compute target is a local or cloud-based resource environment used to run your training script or host your service deployment. For this experiment, we use a cloud-based compute.
 - a. Populate the **Select virtual machine** form to set up your compute.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
Location	Your region that you'd like to run the machine from	West US 2
Virtual machine tier	Select what priority your experiment should have	Dedicated
Virtual machine type	Select the virtual machine type for your compute.	CPU (Central Processing Unit)
Virtual machine size	Select the virtual machine size for your compute. A list of recommended sizes is provided based on your data and experiment type.	Standard_DS12_V2

b. Select **Next** to populate the **Configure settings** form.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
Compute name	A unique name that identifies your compute context.	automl-compute
Min / Max nodes	To profile data, you must specify 1 or more nodes.	Min nodes: 1 Max nodes: 6
Idle seconds before scale down	Idle time before the cluster is automatically scaled down to the minimum node count.	120 (default)
Advanced settings	Settings to configure and authorize a virtual network for your experiment.	None

c. Select **Create** to create your compute target.

This takes a couple minutes to complete.

Create compute cluster [?](#)

Virtual Machine

Settings

Configure Settings
Configure compute cluster settings for your selected virtual machine size.

Name	Category	Cores	Available quota	RAM	Storage	Cost/Hour
Standard_DS12_v2	Memory optimized	4	94 cores	28 GB	56 GB	\$0.37/hr

Compute name * [?](#)

Minimum number of nodes * [?](#)

To avoid charges when no jobs are running, set the minimum nodes to 0. This setting allows Azure Machine Learning to de-allocate the compute nodes when idle. Any higher value will result in charges for the number of nodes allocated.

Maximum number of nodes * [?](#)

Idle seconds before scale down * [?](#)

Enable SSH access [?](#)

[Advanced settings](#)

Enable virtual network [?](#)

Assign a managed identity [?](#)

[Back](#) [Create](#) [Download a template for automation](#) [Cancel](#)

d. After creation, select your new compute target from the drop-down list.

e. Select **Next**.

3. On the **Select task and settings** form, complete the setup for your automated ML experiment by specifying the machine learning task type and configuration settings.

a. Select **Classification** as the machine learning task type.

b. Select **View additional configuration settings** and populate the fields as follows. These settings are to better control the training job. Otherwise, defaults are applied based on experiment selection and data.

ADDITIONAL CONFIGURATIONS	DESCRIPTION	VALUE FOR TUTORIAL
Primary metric	Evaluation metric that the machine learning algorithm will be measured by.	AUC_weighted
Explain best model	Automatically shows explainability on the best model created by automated ML.	Enable
Blocked algorithms	Algorithms you want to exclude from the training job	None
Additional classification settings	These settings help improve the accuracy of your model	Positive class label: None
Exit criterion	If a criteria is met, the training job is stopped.	Training job time (hours): 1 Metric score threshold: None

ADDITIONAL CONFIGURATIONS	DESCRIPTION	VALUE FOR TUTORIAL
Concurrency	The maximum number of parallel iterations executed per iteration	Max concurrent iterations: 5

Select **Save**.

c. Select **Next**.

4. On the **[Optional] Validate and test** form,

- a. Select k-fold cross-validation as your **Validation type**.
- b. Select 2 as your **Number of cross validations**.

5. Select **Finish** to run the experiment. The **Job Detail** screen opens with the **Job status** at the top as the experiment preparation begins. This status updates as the experiment progresses. Notifications also appear in the top right corner of the studio to inform you of the status of your experiment.

IMPORTANT

Preparation takes 10-15 minutes to prepare the experiment run. Once running, it takes 2-3 minutes more for each iteration.

In production, you'd likely walk away for a bit. But for this tutorial, we suggest you start exploring the tested algorithms on the **Models** tab as they complete while the others are still running.

Explore models

Navigate to the **Models** tab to see the algorithms (models) tested. By default, the models are ordered by metric score as they complete. For this tutorial, the model that scores the highest based on the chosen **AUC_weighted** metric is at the top of the list.

While you wait for all of the experiment models to finish, select the **Algorithm name** of a completed model to explore its performance details.

The following navigates through the **Details** and the **Metrics** tabs to view the selected model's properties, metrics, and performance charts.

Properties

Status
Completed

Created
Oct 27, 2021 3:40 PM

Started
Oct 27, 2021 3:40 PM

Duration
33m 18.08s

Compute duration
33m 18.08s

Compute target
[test-compute](#)

Run ID
AutoML_e1d071ab-5336-46b1-9cbc-038932134f46

Script name
--

Created by
Nina Baccam

Input datasets
Input name: training_data, Dataset: [bank marketing: Version 1](#)

Output datasets
None

Best model summary

Algorithm name
[VotingEnsemble](#)

Ensemble details
[View ensemble details](#)

AUC weighted
0.94836 [View all other metrics](#)

Sampling
100.00 % [\(i\)](#)

Registered models
No registration yet

Deploy status
No deployment yet

Run summary

Task type
Classification [View configuration settings](#)

Featurization
Auto

Primary metric
AUC weighted

Experiment name
new-test

Model explanations

While you wait for the models to complete, you can also take a look at model explanations and see which data features (raw or engineered) influenced a particular model's predictions.

These model explanations can be generated on demand, and are summarized in the model explanations dashboard that's part of the [Explanations \(preview\)](#) tab.

To generate model explanations,

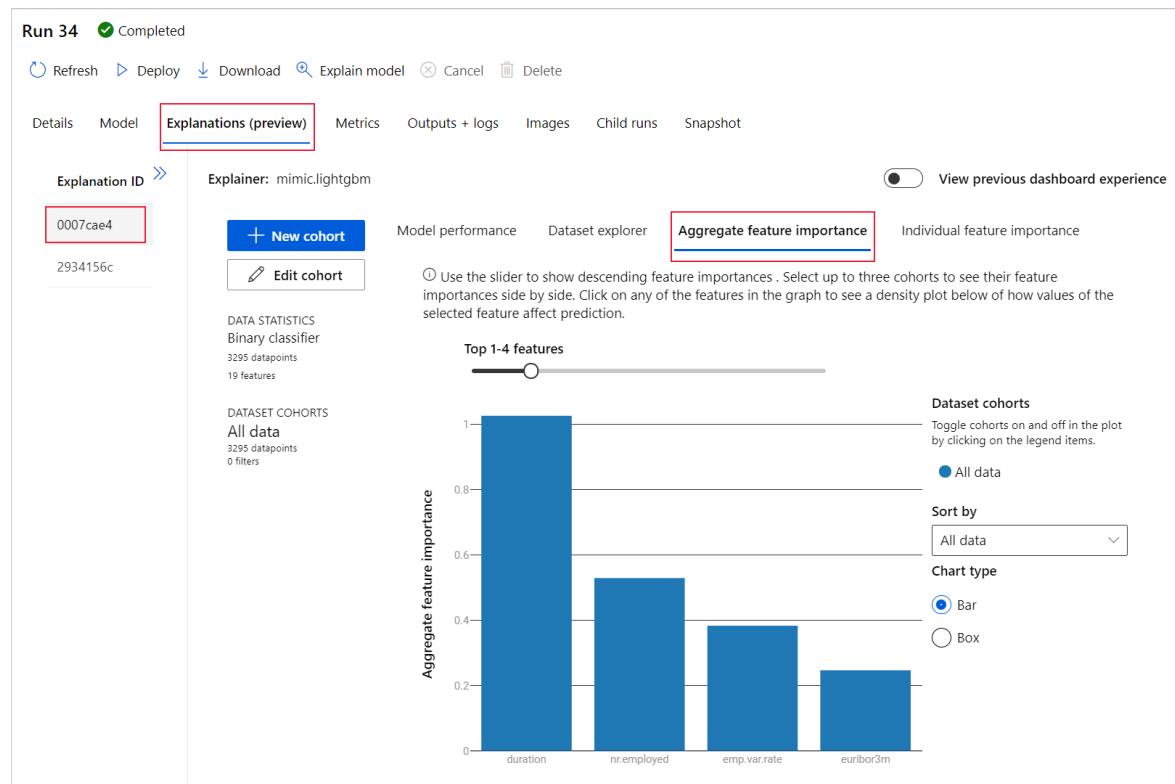
1. Select **Job 1** at the top to navigate back to the **Models** screen.
2. Select the **Models** tab.
3. For this tutorial, select the first **MaxAbsScaler, LightGBM** model.
4. Select the **Explain model** button at the top. On the right, the **Explain model** pane appears.
5. Select the **automl-compute** that you created previously. This compute cluster initiates a child job to generate the model explanations.
6. Select **Create** at the bottom. A green success message appears towards the top of your screen.

NOTE

The explainability job takes about 2-5 minutes to complete.

7. Select the **Explanations (preview)** button. This tab populates once the explainability run completes.
8. On the left hand side, expand the pane and select the row that says **raw** under **Features**.
9. Select the **Aggregate feature importance** tab on the right. This chart shows which data features influenced the predictions of the selected model.

In this example, the *duration* appears to have the most influence on the predictions of this model.



Deploy the best model

The automated machine learning interface allows you to deploy the best model as a web service in a few steps. Deployment is the integration of the model so it can predict on new data and identify potential areas of opportunity.

For this experiment, deployment to a web service means that the financial institution now has an iterative and scalable web solution for identifying potential fixed term deposit customers.

Check to see if your experiment run is complete. To do so, navigate back to the parent job page by selecting **Job 1** at the top of your screen. A **Completed** status is shown on the top left of the screen.

Once the experiment run is complete, the **Details** page is populated with a **Best model summary** section. In this experiment context, **VotingEnsemble** is considered the best model, based on the **AUC_weighted** metric.

We deploy this model, but be advised, deployment takes about 20 minutes to complete. The deployment process entails several steps including registering the model, generating resources, and configuring them for the web service.

1. Select **VotingEnsemble** to open the model-specific page.
2. Select the **Deploy** menu in the top-left and select **Deploy to web service**.
3. Populate the **Deploy a model** pane as follows:

FIELD	VALUE
Deployment name	my-automl-deploy
Deployment description	My first automated machine learning experiment deployment
Compute type	Select Azure Container Instance (ACI)

FIELD	VALUE
Enable authentication	Disable.
Use custom deployments	Disable. Allows for the default driver file (scoring script) and environment file to be auto-generated.

For this example, we use the defaults provided in the *Advanced* menu.

4. Select Deploy.

A green success message appears at the top of the **Job** screen, and in the **Model summary** pane, a status message appears under **Deploy status**. Select **Refresh** periodically to check the deployment status.

Now you have an operational web service to generate predictions.

Proceed to the [Next Steps](#) to learn more about how to consume your new web service, and test your predictions using Power BI's built in Azure Machine Learning support.

Clean up resources

Deployment files are larger than data and experiment files, so they cost more to store. Delete only the deployment files to minimize costs to your account, or if you want to keep your workspace and experiment files. Otherwise, delete the entire resource group, if you don't plan to use any of the files.

Delete the deployment instance

Delete just the deployment instance from Azure Machine Learning at <https://ml.azure.com/>, if you want to keep the resource group and workspace for other tutorials and exploration.

1. Go to [Azure Machine Learning](#). Navigate to your workspace and on the left under the **Assets** pane, select **Endpoints**.
2. Select the deployment you want to delete and select **Delete**.
3. Select **Proceed**.

Delete the resource group

IMPORTANT

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'FAVORITES' section with links to Home, Dashboard, All services, and various Azure services like App Services, SQL databases, and Storage accounts. The main content area is titled 'my-rg' and shows the 'Overview' tab selected. At the top right, there are buttons for Add, Edit columns, Delete resource group (which is highlighted with a red box), Refresh, and Move. Below these are sections for Subscription (change), Visual Studio Ultimate with MSDN, Subscription ID (XXXXXX-XXX-XXXX-XXXX), and Tags (change). A link to Click here to add tags is provided. A table lists 14 items, filtered by name, type, and location. The table includes columns for NAME, TYPE, and STATUS. The resources listed are: myworkspace (Machine Learning service workspace), my-workspace (Machine Learning service workspace), and myworkspace0013141752 (Application Insights).

4. Enter the resource group name. Then select **Delete**.

Next steps

In this automated machine learning tutorial, you used Azure Machine Learning's automated ML interface to create and deploy a classification model. See these articles for more information and next steps:

Consume a web service

- Learn more about [automated machine learning](#).
- For more information on classification metrics and charts, see the [Understand automated machine learning results](#) article.
- Learn more about [featurization](#).
- Learn more about [data profiling](#).

NOTE

This Bank Marketing dataset is made available under the [Creative Commons \(CC0: Public Domain\) License](#). Any rights in individual contents of the database are licensed under the [Database Contents License](#) and available on [Kaggle](#). This dataset was originally available within the [UCI Machine Learning Database](#).

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014.

Tutorial: Forecast demand with no-code automated machine learning in the Azure Machine Learning studio

9/22/2022 • 10 minutes to read • [Edit Online](#)

Learn how to create a [time-series forecasting model](#) without writing a single line of code using automated machine learning in the Azure Machine Learning studio. This model will predict rental demand for a bike sharing service.

You won't write any code in this tutorial, you'll use the studio interface to perform training. You'll learn how to do the following tasks:

- Create and load a dataset.
- Configure and run an automated ML experiment.
- Specify forecasting settings.
- Explore the experiment results.
- Deploy the best model.

Also try automated machine learning for these other model types:

- For a no-code example of a classification model, see [Tutorial: Create a classification model with automated ML in Azure Machine Learning](#).
- For a code first example of an object detection model, see the [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#).

Prerequisites

- An Azure Machine Learning workspace. See [Create workspace resources](#).
- Download the [bike-no.csv](#) data file

Sign in to the studio

For this tutorial, you create your automated ML experiment run in Azure Machine Learning studio, a consolidated web interface that includes machine learning tools to perform data science scenarios for data science practitioners of all skill levels. The studio is not supported on Internet Explorer browsers.

1. Sign in to [Azure Machine Learning studio](#).
2. Select your subscription and the workspace you created.
3. Select **Get started**.
4. In the left pane, select **Automated ML** under the **Author** section.
5. Select **+New automated ML job**.

Create and load dataset

Before you configure your experiment, upload your data file to your workspace in the form of an Azure Machine Learning dataset. Doing so, allows you to ensure that your data is formatted appropriately for your experiment.

1. On the **Select dataset** form, select **From local files** from the **+Create dataset** drop-down.
- a. On the **Basic info** form, give your dataset a name and provide an optional description. The dataset type should default to **Tabular**, since automated ML in Azure Machine Learning studio currently only supports tabular datasets.
- b. Select **Next** on the bottom left
- c. On the **Datastore and file selection** form, select the default datastore that was automatically set up during your workspace creation, **workspaceblobstore (Azure Blob Storage)**. This is the storage location where you'll upload your data file.
- d. Select **Upload files** from the **Upload** drop-down..
- e. Choose the **bike-no.csv** file on your local computer. This is the file you downloaded as a [prerequisite](#).

f. Select **Next**

When the upload is complete, the Settings and preview form is pre-populated based on the file type.

- g. Verify that the **Settings and preview** form is populated as follows and select **Next**.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
File format	Defines the layout and type of data stored in a file.	Delimited
Delimiter	One or more characters for specifying the boundary between separate, independent regions in plain text or other data streams.	Comma
Encoding	Identifies what bit to character schema table to use to read your dataset.	UTF-8
Column headers	Indicates how the headers of the dataset, if any, will be treated.	Only first file has headers
Skip rows	Indicates how many, if any, rows are skipped in the dataset.	None

- h. The **Schema** form allows for further configuration of your data for this experiment.
- a. For this example, choose to ignore the **casual** and **registered** columns. These columns are a breakdown of the **cnt** column so, therefore we don't include them.
- b. Also for this example, leave the defaults for the **Properties** and **Type**.
- c. Select **Next**.
- i. On the **Confirm details** form, verify the information matches what was previously populated on the **Basic info** and **Settings and preview** forms.
- j. Select **Create** to complete the creation of your dataset.
- k. Select your dataset once it appears in the list.

I. Select **Next**.

Configure job

After you load and configure your data, set up your remote compute target and select which column in your data you want to predict.

1. Populate the **Configure job** form as follows:

- a. Enter an experiment name: **automl-bikeshare**
- b. Select **cnt** as the target column, what you want to predict. This column indicates the number of total bike share rentals.
- c. Select **compute cluster** as your compute type.
- d. Select **+ New** to configure your compute target. Automated ML only supports Azure Machine Learning compute.
 - a. Populate the **Select virtual machine** form to set up your compute.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
Virtual machine tier	Select what priority your experiment should have	Dedicated
Virtual machine type	Select the virtual machine type for your compute.	CPU (Central Processing Unit)
Virtual machine size	Select the virtual machine size for your compute. A list of recommended sizes is provided based on your data and experiment type.	Standard_DS12_V2

- b. Select **Next** to populate the **Configure settings** form.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
Compute name	A unique name that identifies your compute context.	bike-compute
Min / Max nodes	To profile data, you must specify 1 or more nodes.	Min nodes: 1 Max nodes: 6
Idle seconds before scale down	Idle time before the cluster is automatically scaled down to the minimum node count.	120 (default)
Advanced settings	Settings to configure and authorize a virtual network for your experiment.	None

- c. Select **Create** to get the compute target.

This takes a couple minutes to complete.

- d. After creation, select your new compute target from the drop-down list.
- e. Select **Next**.

Select forecast settings

Complete the setup for your automated ML experiment by specifying the machine learning task type and configuration settings.

1. On the **Task type and settings** form, select **Time series forecasting** as the machine learning task type.
2. Select **date** as your **Time column** and leave **Time series identifiers** blank.
3. The **Frequency** is how often your historic data is collected. Keep **Autodetect** selected.
- 4.
5. The **forecast horizon** is the length of time into the future you want to predict. Deselect **Autodetect** and type 14 in the field.
6. Select **View additional configuration settings** and populate the fields as follows. These settings are to better control the training job and specify settings for your forecast. Otherwise, defaults are applied based on experiment selection and data.

ADDITIONAL CONFIGURATIONS	DESCRIPTION	VALUE FOR TUTORIAL
Primary metric	Evaluation metric that the machine learning algorithm will be measured by.	Normalized root mean squared error
Explain best model	Automatically shows explainability on the best model created by automated ML.	Enable
Blocked algorithms	Algorithms you want to exclude from the training job	Extreme Random Trees
Additional forecasting settings	<p>These settings help improve the accuracy of your model.</p> <p>Forecast target lags: how far back you want to construct the lags of the target variable</p> <p>Target rolling window: specifies the size of the rolling window over which features, such as the <i>max</i>, <i>min</i> and <i>sum</i>, will be generated.</p>	Forecast target lags: None Target rolling window size: None
Exit criterion	If a criteria is met, the training job is stopped.	Training job time (hours): 3 Metric score threshold: None
Concurrency	The maximum number of parallel iterations executed per iteration	Max concurrent iterations: 6

Select **Save**.

7. Select **Next**.
8. On the **[Optional] Validate and test** form,
 - a. Select k-fold cross-validation as your **Validation type**.
 - b. Select 5 as your **Number of cross validations**.

Run experiment

To run your experiment, select **Finish**. The **Job details** screen opens with the **Job status** at the top next to the job number. This status updates as the experiment progresses. Notifications also appear in the top right corner of the studio, to inform you of the status of your experiment.

IMPORTANT

Preparation takes **10-15 minutes** to prepare the experiment job. Once running, it takes **2-3 minutes more for each iteration**.

In production, you'd likely walk away for a bit as this process takes time. While you wait, we suggest you start exploring the tested algorithms on the **Models** tab as they complete.

Explore models

Navigate to the **Models** tab to see the algorithms (models) tested. By default, the models are ordered by metric score as they complete. For this tutorial, the model that scores the highest based on the chosen **Normalized root mean squared error** metric is at the top of the list.

While you wait for all of the experiment models to finish, select the **Algorithm name** of a completed model to explore its performance details.

The following example navigates through the **Details** and the **Metrics** tabs to view the selected model's properties, metrics and performance charts.

The screenshot shows the Azure Machine Learning studio interface with the following details:

- Header:** Microsoft > testautoml > Experiments > new-test > Bank class (parent)
- Top Bar:** Bank class (parent) ⚙ ☆
Refresh Cancel Delete
- Actions:** Details Data guardrails Models Outputs + logs Child runs Snapshot
- Properties:**
 - Status:** Completed (with a warning message about early stopping)
 - Created:** Oct 27, 2021 3:40 PM
 - Started:** Oct 27, 2021 3:40 PM
 - Duration:** 33m 18.08s
 - Compute duration:** 33m 18.08s
 - Compute target:** test-compute
 - Run ID:** AutoML_e1d071ab-5336-46b1-9cbc-038932134f46
 - Script name:** --
 - Created by:** Nina Baccam
 - Input datasets:** Input name: training_data, Dataset: bank marketing: Version 1
 - Output datasets:** None
- Best model summary:**
 - Algorithm name: VotingEnsemble
 - Ensemble details: View ensemble details
 - AUC weighted: 0.94836 (View all other metrics)
 - Sampling: 100.00 %
 - Registered models: No registration yet
 - Deploy status: No deployment yet
- Run summary:**
 - Task type: Classification (View configuration settings)
 - Featurization: Auto
 - Primary metric: AUC weighted
 - Experiment name: new-test

Deploy the model

Automated machine learning in Azure Machine Learning studio allows you to deploy the best model as a web service in a few steps. Deployment is the integration of the model so it can predict on new data and identify potential areas of opportunity.

For this experiment, deployment to a web service means that the bike share company now has an iterative and scalable web solution for forecasting bike share rental demand.

Once the job is complete, navigate back to parent job page by selecting **Job 1** at the top of your screen.

In the **Best model summary** section, the best model in the context of this experiment, is selected based on the **Normalized root mean squared error metric**.

We deploy this model, but be advised, deployment takes about 20 minutes to complete. The deployment process entails several steps including registering the model, generating resources, and configuring them for the web service.

1. Select **the best model** to open the model-specific page.
2. Select the **Deploy** button located in the top-left area of the screen.
3. Populate the **Deploy a model** pane as follows:

FIELD	VALUE
Deployment name	bikeshare-deploy
Deployment description	bike share demand deployment
Compute type	Select Azure Compute Instance (ACI)
Enable authentication	Disable.
Use custom deployment assets	Disable. Disabling allows for the default driver file (scoring script) and environment file to be autogenerated.

For this example, we use the defaults provided in the *Advanced* menu.

4. Select **Deploy**.

A green success message appears at the top of the **Job** screen stating that the deployment was started successfully. The progress of the deployment can be found in the **Model summary** pane under **Deploy status**.

Once deployment succeeds, you have an operational web service to generate predictions.

Proceed to the **Next steps** to learn more about how to consume your new web service, and test your predictions using Power BI's built in Azure Machine Learning support.

Clean up resources

Deployment files are larger than data and experiment files, so they cost more to store. Delete only the deployment files to minimize costs to your account, or if you want to keep your workspace and experiment files. Otherwise, delete the entire resource group, if you don't plan to use any of the files.

Delete the deployment instance

Delete just the deployment instance from the Azure Machine Learning studio, if you want to keep the resource group and workspace for other tutorials and exploration.

1. Go to the [Azure Machine Learning studio](#). Navigate to your workspace and on the left under the **Assets** pane, select **Endpoints**.

2. Select the deployment you want to delete and select **Delete**.

3. Select **Proceed**.

Delete the resource group

IMPORTANT

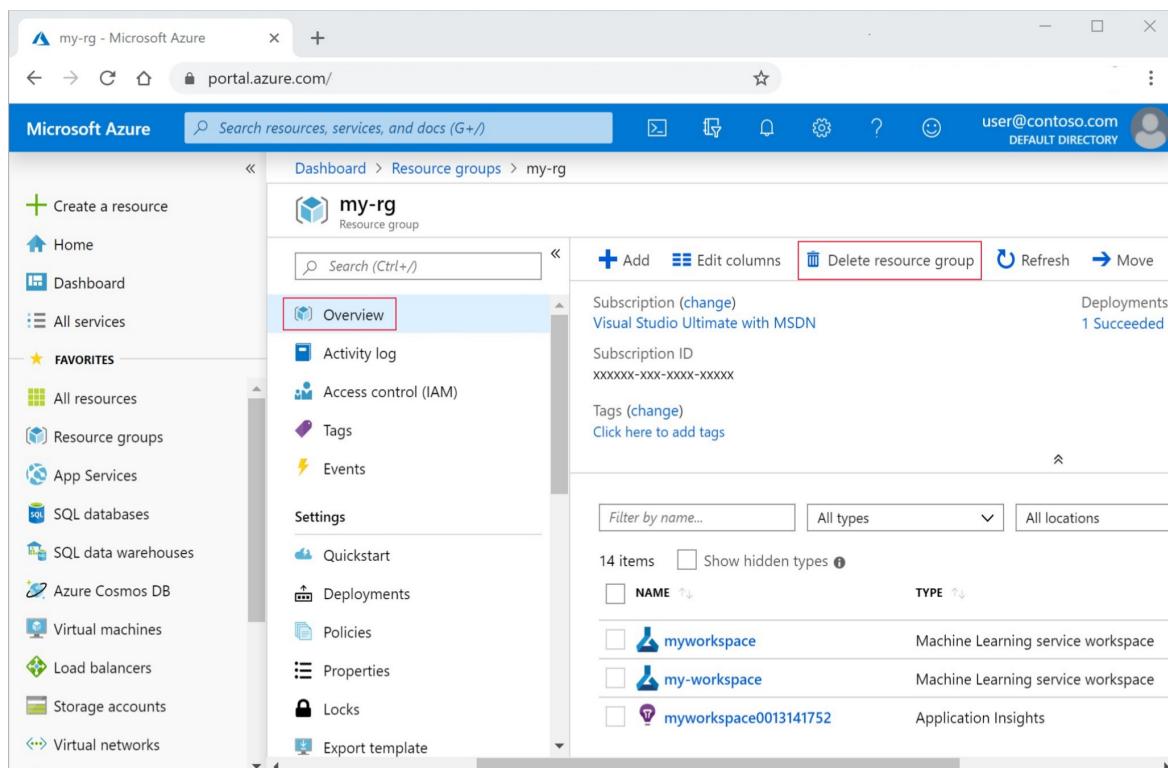
The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.

2. From the list, select the resource group that you created.

3. Select **Delete resource group**.



The screenshot shows the Microsoft Azure portal interface. The left sidebar is open, showing various service categories like Home, Dashboard, All services, Favorites, and Resource groups. The 'Resource groups' section is expanded, and the 'my-rg' resource group is selected. The main content area displays the 'Overview' tab for 'my-rg'. At the top right of this section, there are buttons for 'Add', 'Edit columns', 'Delete resource group' (which is highlighted with a red box), 'Refresh', and 'Move'. Below these buttons, it shows 'Subscription (change)' as 'Visual Studio Ultimate with MSDN', 'Subscription ID' as 'xxxxxxxx-xxxx-xxxx-xxxx', and 'Tags (change)' with a link to 'Click here to add tags'. A table below lists 14 items, filtered by 'All types' and 'All locations'. The table includes columns for 'NAME' (with a sorting arrow) and 'TYPE' (also with a sorting arrow). The listed items are: 'myworkspace' (Machine Learning service workspace), 'my-workspace' (Machine Learning service workspace), and 'myworkspace0013141752' (Application Insights).

4. Enter the resource group name. Then select **Delete**.

Next steps

In this tutorial, you used automated ML in the Azure Machine Learning studio to create and deploy a time series forecasting model that predicts bike share rental demand.

See this article for steps on how to create a Power BI supported schema to facilitate consumption of your newly deployed web service:

Consume a web service

- Learn more about [automated machine learning](#).
- For more information on classification metrics and charts, see the [Understand automated machine learning results](#) article.
- Learn more about [featurization](#).
- Learn more about [data profiling](#).

NOTE

This bike share dataset has been modified for this tutorial. This dataset was made available as part of a [Kaggle competition](#) and was originally available via [Capital Bikeshare](#). It can also be found within the [UCI Machine Learning Database](#).

Source: Fanaee-T, Hadi, and Gama, Joao, Event labeling combining ensemble detectors and background knowledge, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.

Tutorial: Designer - train a no-code regression model

9/22/2022 • 13 minutes to read • [Edit Online](#)

Train a linear regression model that predicts car prices using the Azure Machine Learning designer. This tutorial is part one of a two-part series.

This tutorial uses the Azure Machine Learning designer, for more information, see [What is Azure Machine Learning designer?](#)

In part one of the tutorial, you learn how to:

- Create a new pipeline.
- Import data.
- Prepare data.
- Train a machine learning model.
- Evaluate a machine learning model.

In [part two](#) of the tutorial, you deploy your model as a real-time inferencing endpoint to predict the price of any car based on technical specifications you send it.

NOTE

A completed version of this tutorial is available as a sample pipeline.

To find it, go to the designer in your workspace. In the **New pipeline** section, select **Sample 1 - Regression: Automobile Price Prediction(Basic)**.

IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Create a new pipeline

Azure Machine Learning pipelines organize multiple machine learning and data processing steps into a single resource. Pipelines let you organize, manage, and reuse complex machine learning workflows across projects and users.

To create an Azure Machine Learning pipeline, you need an Azure Machine Learning workspace. In this section, you learn how to create both these resources.

Create a new workspace

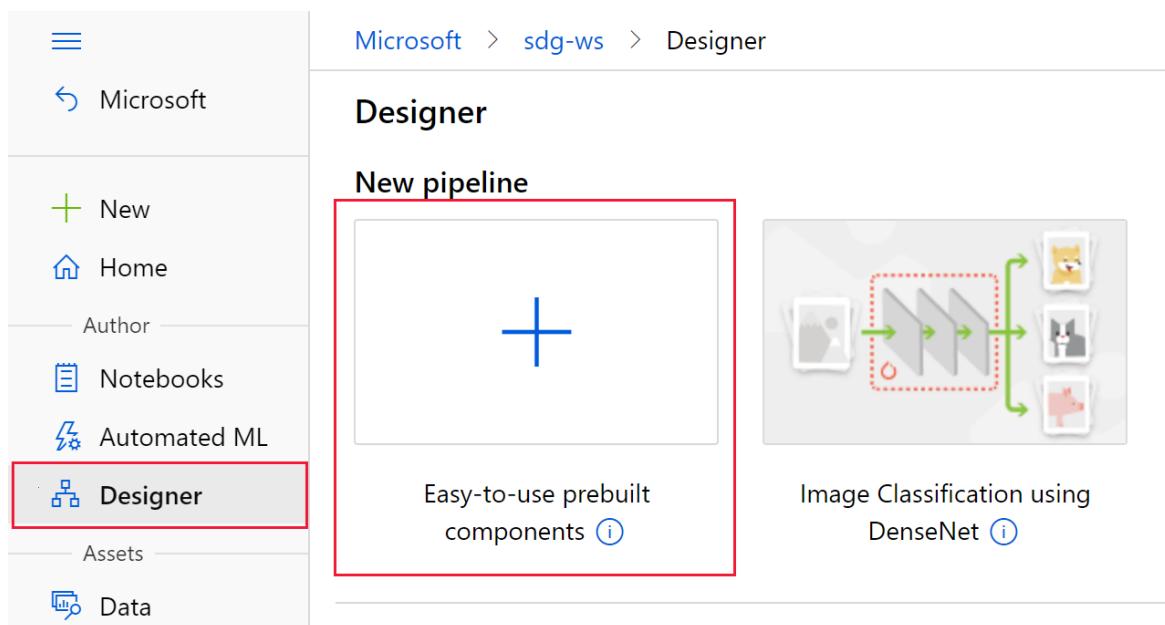
You need an Azure Machine Learning workspace to use the designer. The workspace is the top-level resource for Azure Machine Learning, it provides a centralized place to work with all the artifacts you create in Azure Machine Learning. For instruction on creating a workspace, see [Create workspace resources](#).

NOTE

If your workspace uses a Virtual network, there are additional configuration steps you must use to use the designer. For more information, see [Use Azure Machine Learning studio in an Azure virtual network](#)

Create the pipeline

1. Sign in to [ml.azure.com](#), and select the workspace you want to work with.
2. Select **Designer**.



3. Select **Easy-to-use prebuilt components**.

4. Open the **Settings** pane to the right of the canvas, and scroll to **Draft name** text box. Rename it to *Automobile price prediction*. The name doesn't need to be unique.

Set the default compute target

A pipeline jobs on a compute target, which is a compute resource that's attached to your workspace. After you create a compute target, you can reuse it for future jobs.

IMPORTANT

Attached compute is not supported, use [compute instances or clusters](#) instead.

You can set a **Default compute target** for the entire pipeline, which will tell every component to use the same compute target by default. However, you can specify compute targets on a per-module basis.

1. Select **Settings** to the right of the canvas to open the **Settings** pane.
2. Select **Create Azure ML compute instance**.
If you already have an available compute target, you can select it from the **Select Azure ML compute instance** drop-down to run this pipeline.
3. Enter a name for the compute resource.
4. Select **Create**.

NOTE

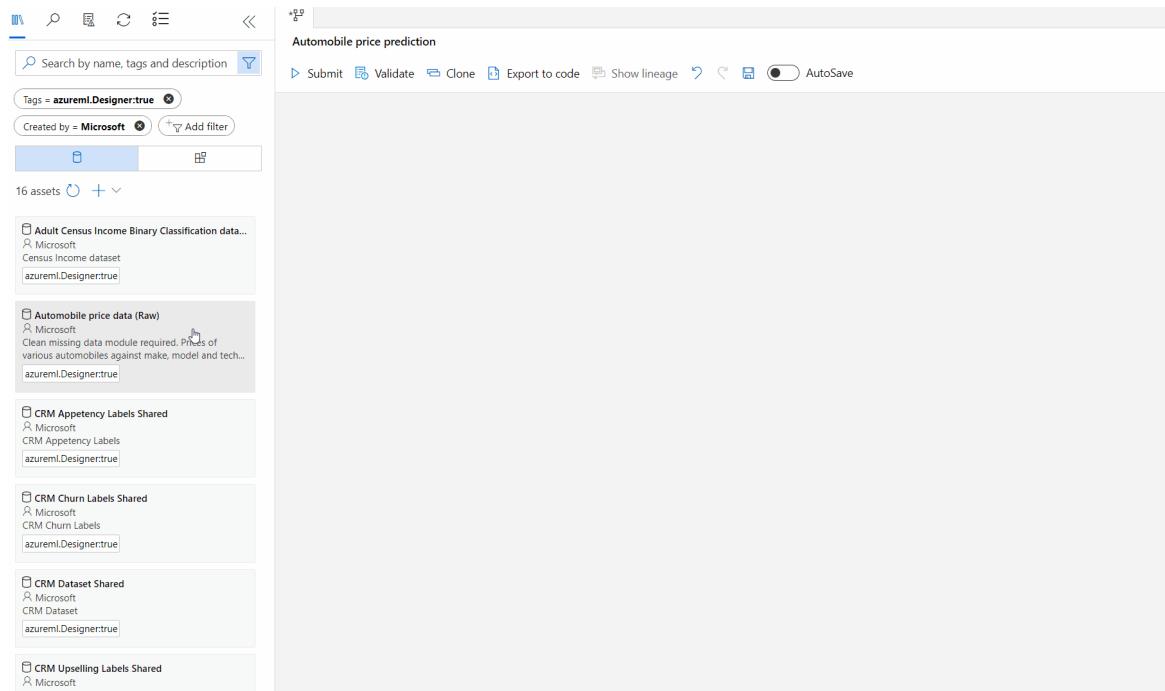
It takes approximately five minutes to create a compute resource. After the resource is created, you can reuse it and skip this wait time for future jobs.

The compute resource autoscales to zero nodes when it's idle to save cost. When you use it again after a delay, you might experience approximately five minutes of wait time while it scales back up.

Import data

There are several sample datasets included in the designer for you to experiment with. For this tutorial, use **Automobile price data (Raw)**.

1. To the left of the pipeline canvas is a palette of datasets and components. Select **Data**.
2. Select the dataset **Automobile price data (Raw)**, and drag it onto the canvas.



Visualize the data

You can visualize the data to understand the dataset that you'll use.

1. Right-click the **Automobile price data (Raw)** and select **Preview Data**.
2. Select the different columns in the data window to view information about each one.

Each row represents an automobile, and the variables associated with each automobile appear as columns. There are 205 rows and 26 columns in this dataset.

Prepare data

Datasets typically require some preprocessing before analysis. You might have noticed some missing values when you inspected the dataset. These missing values must be cleaned so that the model can analyze the data correctly.

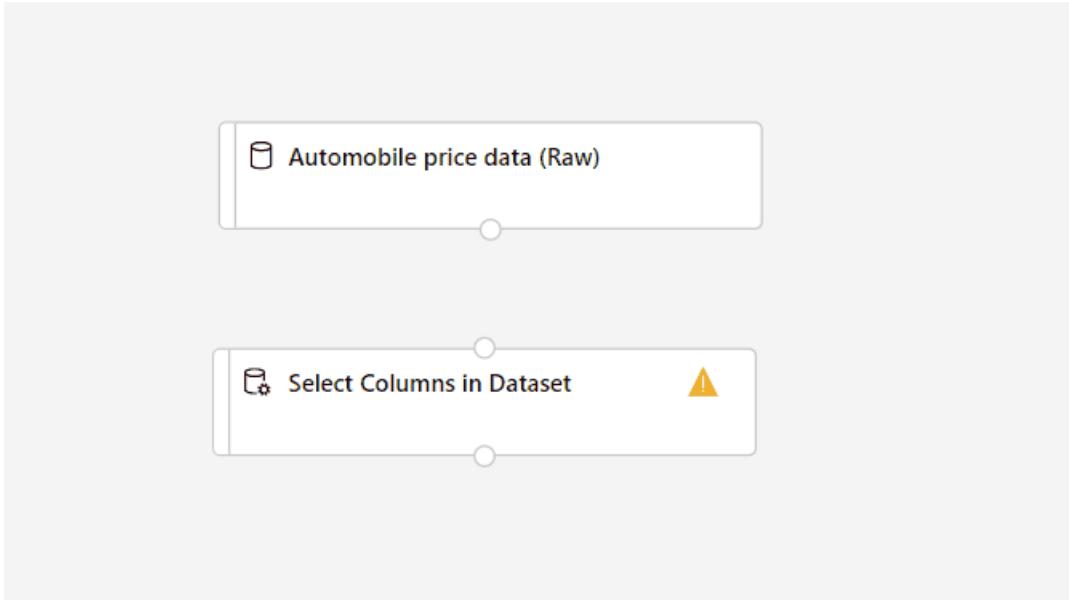
Remove a column

When you train a model, you have to do something about the data that's missing. In this dataset, the **normalized-losses** column is missing many values, so you'll exclude that column from the model altogether.

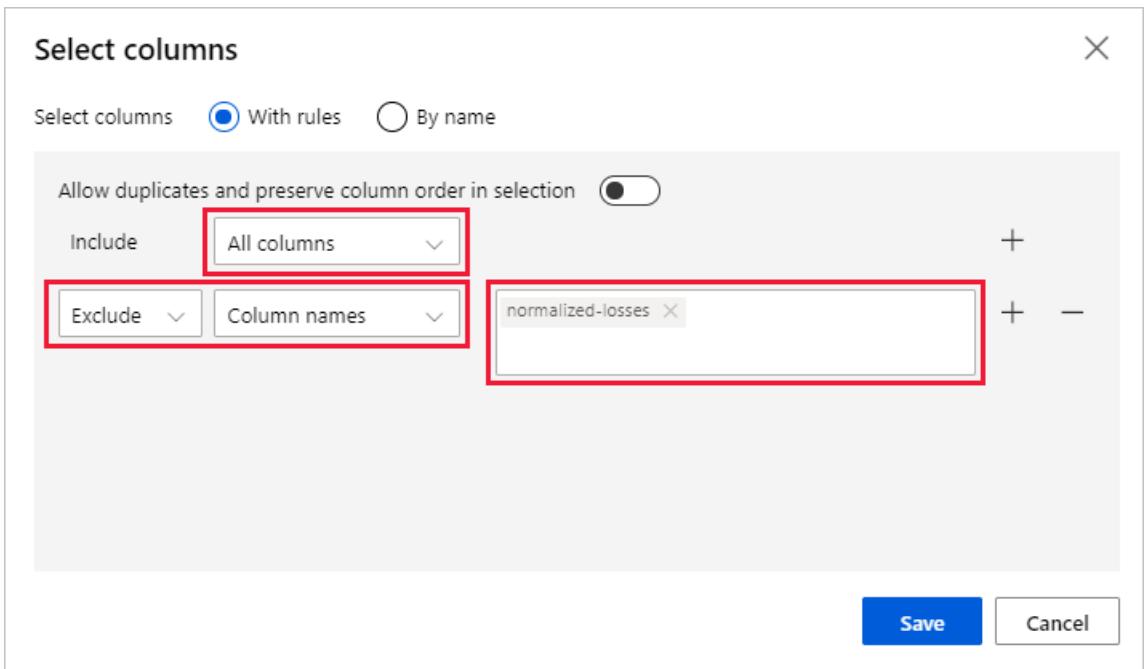
1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Select Columns in Dataset** component.
2. Drag the **Select Columns in Dataset** component onto the canvas. Drop the component below the dataset component.
3. Connect the **Automobile price data (Raw)** dataset to the **Select Columns in Dataset** component. Drag from the dataset's output port, which is the small circle at the bottom of the dataset on the canvas, to the input port of **Select Columns in Dataset**, which is the small circle at the top of the component.

TIP

You create a flow of data through your pipeline when you connect the output port of one component to an input port of another.



4. Select the **Select Columns in Dataset** component.
5. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Select Columns in Dataset** component to open the details pane.
6. Select **Edit column** to the right of the pane.
7. Expand the **Column names** drop down next to **Include**, and select **All columns**.
8. Select the **+** to add a new rule.
9. From the drop-down menus, select **Exclude** and **Column names**.
10. Enter *normalized-losses* in the text box.
11. In the lower right, select **Save** to close the column selector.



12. In the **Select Columns in Dataset** component details pane, expand **Node info**.

13. Select the **Comment** text box and enter *Exclude normalized losses*.

Comments will appear on the graph to help you organize your pipeline.

Clean missing data

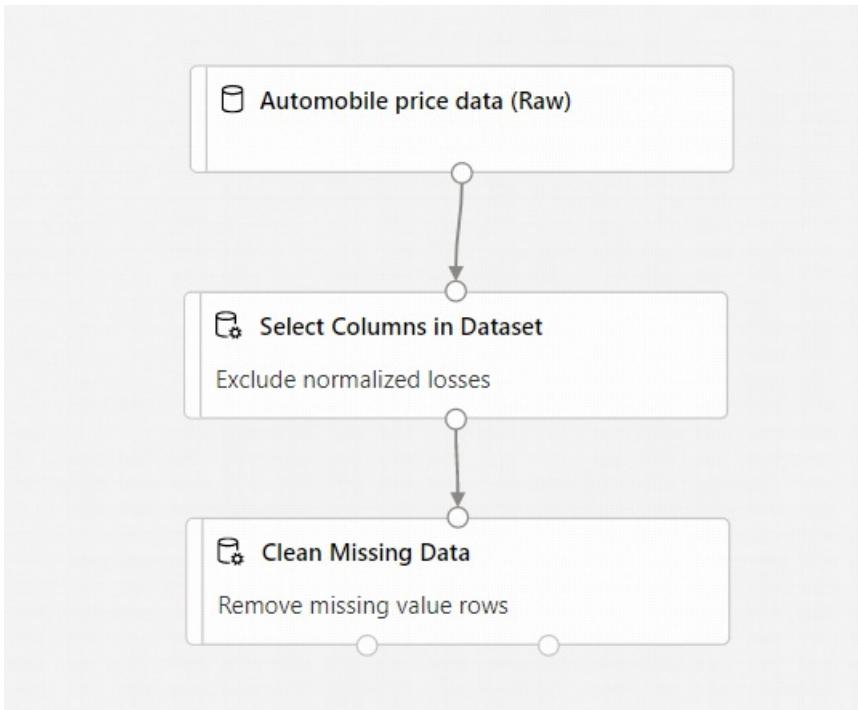
Your dataset still has missing values after you remove the **normalized-losses** column. You can remove the remaining missing data by using the **Clean Missing Data** component.

TIP

Cleaning the missing values from input data is a prerequisite for using most of the components in the designer.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Clean Missing Data** component.
2. Drag the **Clean Missing Data** component to the pipeline canvas. Connect it to the **Select Columns in Dataset** component.
3. Select the **Clean Missing Data** component.
4. Click on the arrow icon under **Settings** to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Clean Missing Data** component to open the details pane.
5. Select **Edit column** to the right of the pane.
6. In the **Columns to be cleaned** window that appears, expand the drop-down menu next to **Include**. Select **All columns**
7. Select **Save**
8. In the **Clean Missing Data** component details pane, under **Cleaning mode**, select **Remove entire row**.
9. In the **Clean Missing Data** component details pane, expand **Node info**.
10. Select the **Comment** text box and enter *Remove missing value rows*.

Your pipeline should now look something like this:



Train a machine learning model

Now that you have the components in place to process the data, you can set up the training components.

Because you want to predict price, which is a number, you can use a regression algorithm. For this example, you use a linear regression model.

Split the data

Splitting data is a common task in machine learning. You'll split your data into two separate datasets. One dataset will train the model and the other will test how well the model performed.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Split Data** component.
2. Drag the **Split Data** component to the pipeline canvas.
3. Connect the left port of the **Clean Missing Data** component to the **Split Data** component.

IMPORTANT

Make sure that the left output port of **Clean Missing Data** connects to **Split Data**. The left port contains the cleaned data. The right port contains the discarded data.

4. Select the **Split Data** component.
5. Click on the arrow icon under Settings to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Split Data** component to open the details pane.
6. In the **Split Data** details pane, set the **Fraction of rows in the first output dataset** to 0.7.

This option splits 70 percent of the data to train the model and 30 percent for testing it. The 70 percent dataset will be accessible through the left output port. The remaining data will be available through the right output port.

7. In the **Split Data** details pane, expand **Node info**.
8. Select the **Comment** text box and enter *Split the dataset into training set (0.7) and test set (0.3)*.

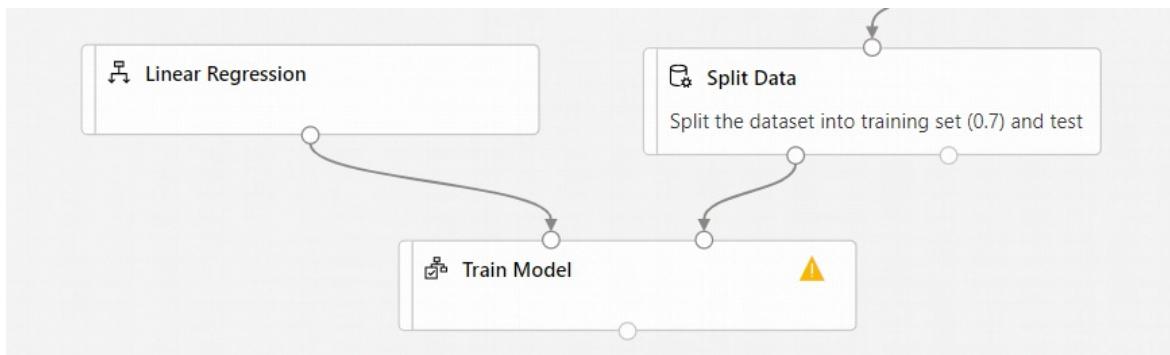
Train the model

Train the model by giving it a dataset that includes the price. The algorithm constructs a model that explains the relationship between the features and the price as presented by the training data.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Linear Regression** component.
2. Drag the **Linear Regression** component to the pipeline canvas.
3. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Train Model** component.
4. Drag the **Train Model** component to the pipeline canvas.
5. Connect the output of the **Linear Regression** component to the left input of the **Train Model** component.
6. Connect the training data output (left port) of the **Split Data** component to the right input of the **Train Model** component.

IMPORTANT

Make sure that the left output port of **Split Data** connects to **Train Model**. The left port contains the training set. The right port contains the test set.

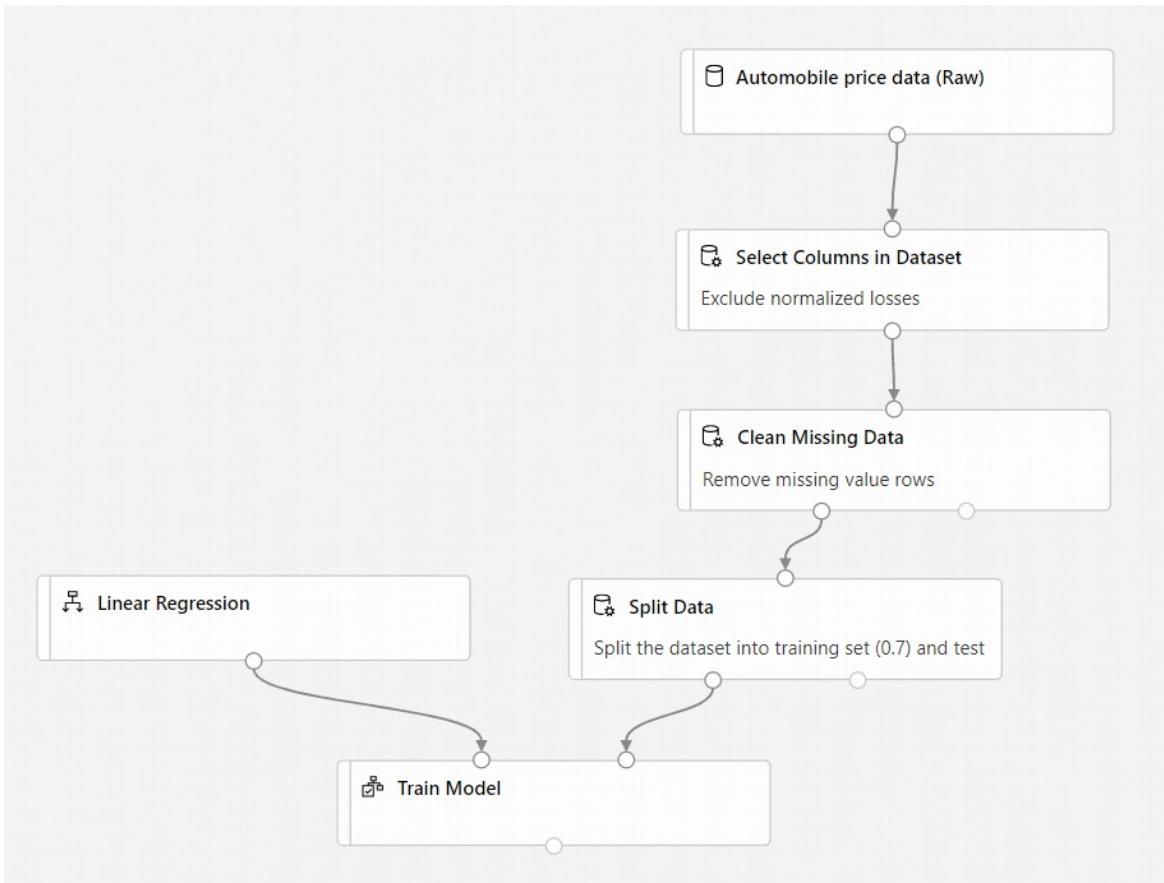


7. Select the **Train Model** component.
8. Click on the arrow icon under **Settings** to the right of the canvas to open the component details pane. Alternatively, you can double-click the **Train Model** component to open the details pane.
9. Select **Edit column** to the right of the pane.
10. In the **Label column** window that appears, expand the drop-down menu and select **Column names**.
11. In the text box, enter *price* to specify the value that your model is going to predict.

IMPORTANT

Make sure you enter the column name exactly. Do not capitalize **price**.

Your pipeline should look like this:



Add the Score Model component

After you train your model by using 70 percent of the data, you can use it to score the other 30 percent to see how well your model functions.

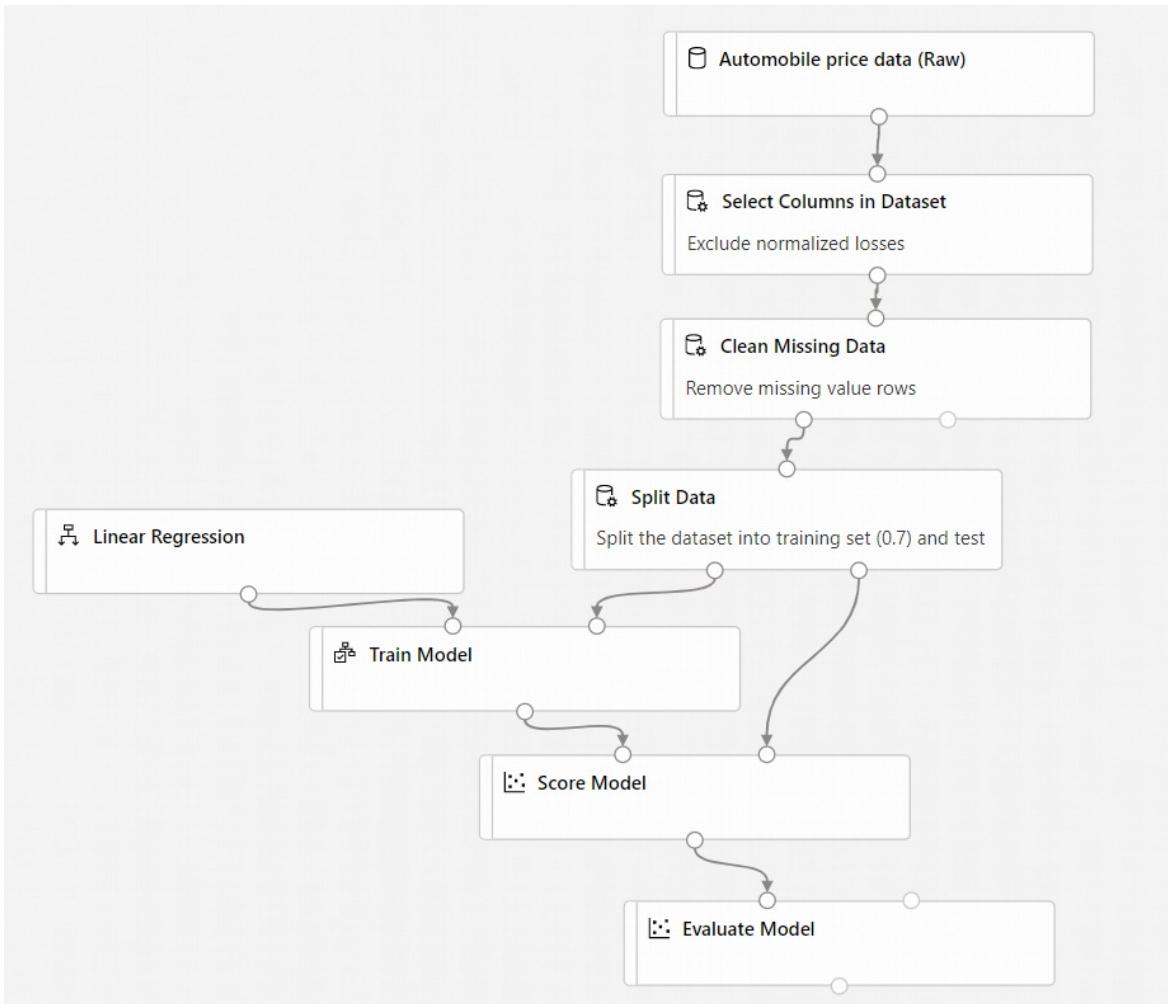
1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Score Model** component.
2. Drag the **Score Model** component to the pipeline canvas.
3. Connect the output of the **Train Model** component to the left input port of **Score Model**. Connect the test data output (right port) of the **Split Data** component to the right input port of **Score Model**.

Add the Evaluate Model component

Use the **Evaluate Model** component to evaluate how well your model scored the test dataset.

1. In the datasets and component palette to the left of the canvas, click **Component** and search for the **Evaluate Model** component.
2. Drag the **Evaluate Model** component to the pipeline canvas.
3. Connect the output of the **Score Model** component to the left input of **Evaluate Model**.

The final pipeline should look something like this:



Submit the pipeline

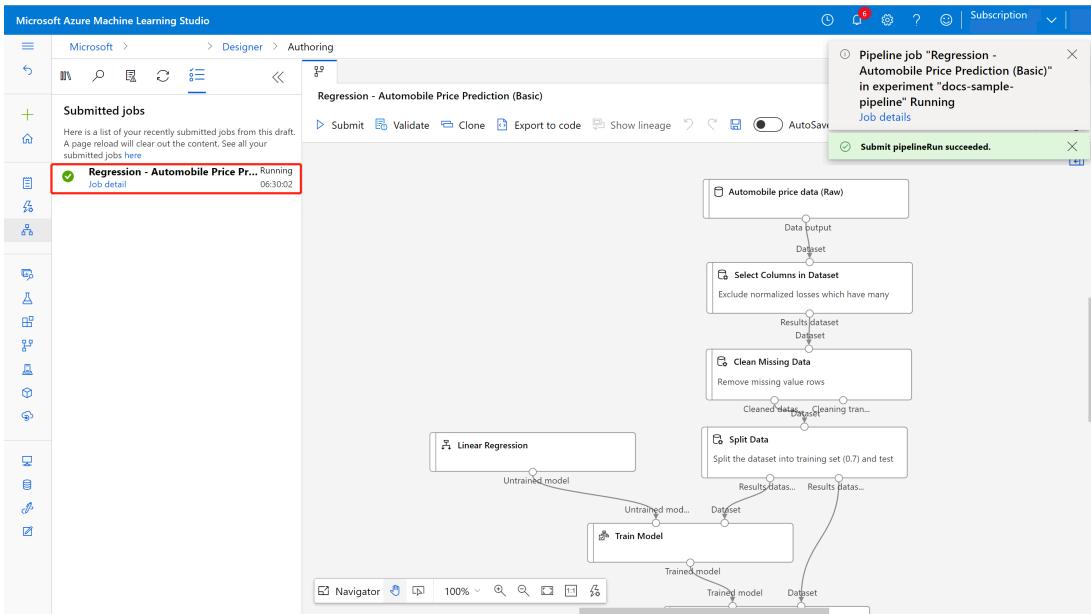
Now that your pipeline is all setup, you can submit a pipeline job to train your machine learning model. You can submit a valid pipeline job at any point, which can be used to review changes to your pipeline during development.

1. At the top of the canvas, select **Submit**.
2. In the **Set up pipeline job** dialog box, select **Create new**.

NOTE

Experiments group similar pipeline jobs together. If you run a pipeline multiple times, you can select the same experiment for successive jobs.

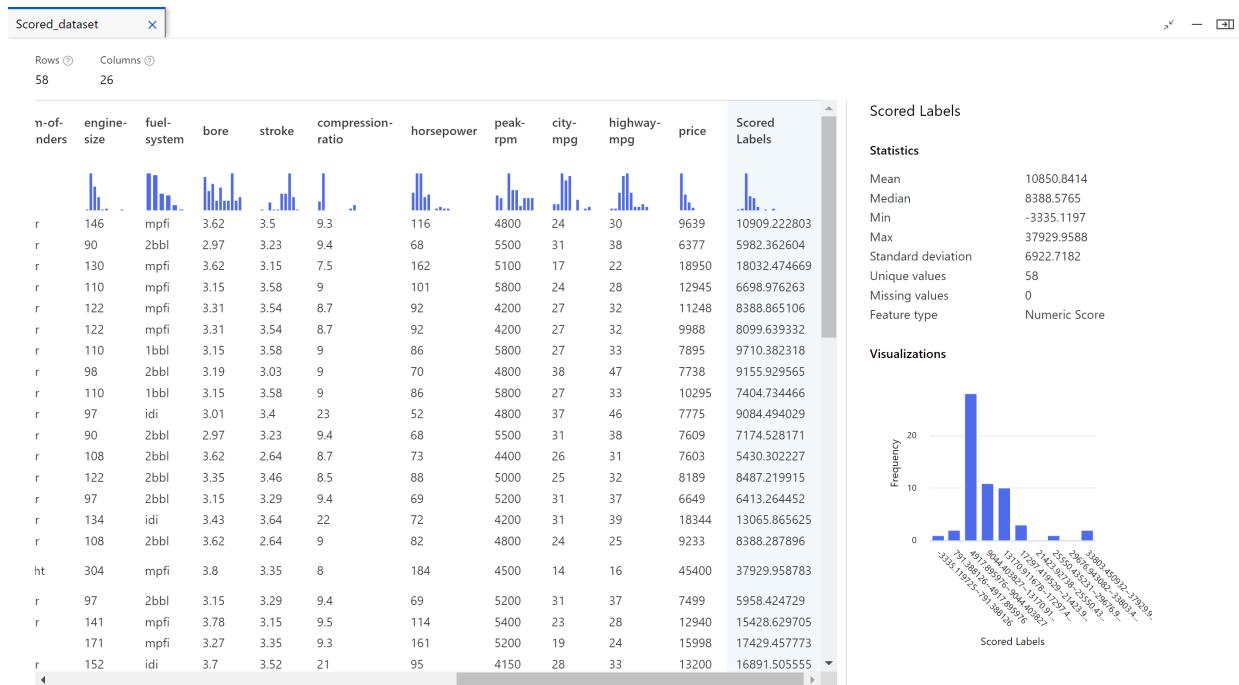
- a. For **New experiment Name**, enter **Tutorial-CarPrices**.
- b. Select **Submit**.
- c. You'll see a submission list in the left pane of the canvas, and a notification will pop up at the top right corner of the page. You can select the **Job detail** link to go to job detail page for debugging.



If this is the first job, it may take up to 20 minutes for your pipeline to finish running. The default compute settings have a minimum node size of 0, which means that the designer must allocate resources after being idle. Repeated pipeline jobs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each component to further improve efficiency.

View scored labels

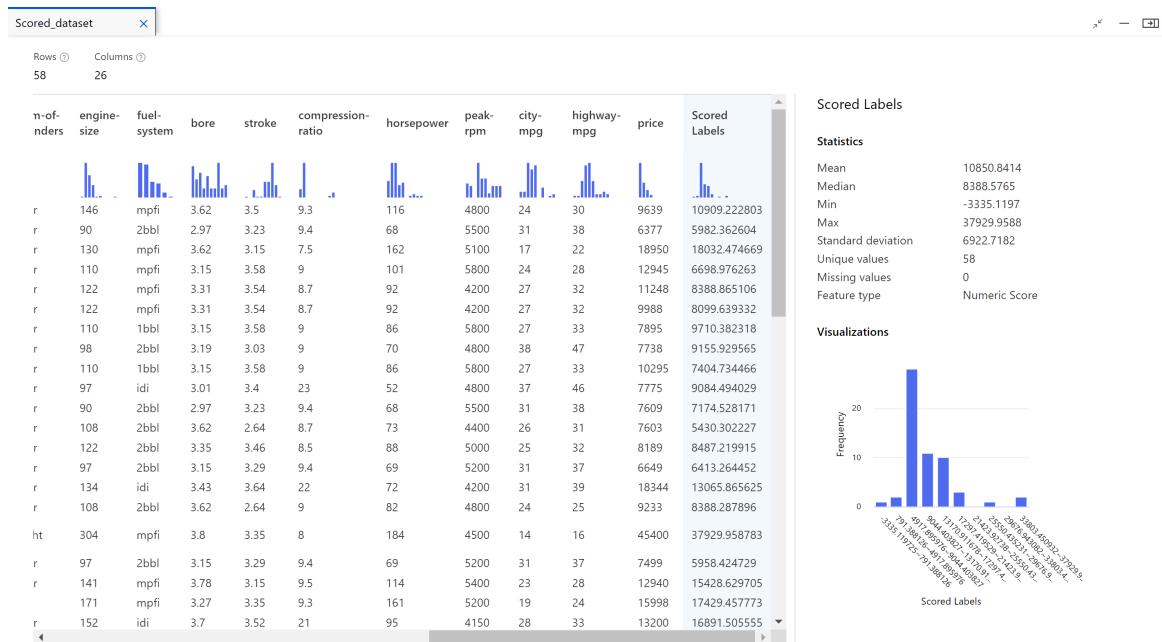
In the job detail page, you can check the pipeline job status, results and logs.



After the job completes, you can view the results of the pipeline job. First, look at the predictions generated by the regression model.

- Right-click the **Score Model** component, and select **Preview data > Scored dataset** to view its output.

Here you can see the predicted prices and the actual prices from the testing data.



Evaluate models

Use the **Evaluate Model** component to see how well the trained model performed on the test dataset.

1. Right-click the **Evaluate Model** component and select **Preview data > Evaluation results** to view its output.

The following statistics are shown for your model:

- **Mean Absolute Error (MAE)**: The average of absolute errors. An error is the difference between the predicted value and the actual value.
- **Root Mean Squared Error (RMSE)**: The square root of the average of squared errors of predictions made on the test dataset.
- **Relative Absolute Error**: The average of absolute errors relative to the absolute difference between actual values and the average of all actual values.
- **Relative Squared Error**: The average of squared errors relative to the squared difference between the actual values and the average of all actual values.
- **Coefficient of Determination**: Also known as the R squared value, this statistical metric indicates how well a model fits the data.

For each of the error statistics, smaller is better. A smaller value indicates that the predictions are closer to the actual values. For the coefficient of determination, the closer its value is to one (1.0), the better the predictions.

Clean up resources

Skip this section if you want to continue on with part 2 of the tutorial, [deploying models](#).

IMPORTANT

You can use the resources that you created as prerequisites for other Azure Machine Learning tutorials and how-to articles.

Delete everything

If you don't plan to use anything that you created, delete the entire resource group so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the left side of the window.

The screenshot shows the Microsoft Azure Resource Groups blade. On the left, there's a list of resource groups with one named 'my-rg' highlighted with a red box. The main pane shows details for 'my-rg' under the 'Overview' tab. It includes a search bar, subscription information ('Subscription (change) : documentationteam'), and a list of resources like 'my-ws', 'myws22', 'myws70', and 'myws74'. A sidebar on the right provides navigation links for Overview, Activity log, Access control (IAM), Tags, Events, Quickstart, Deployments, Policies, Properties, and Locks.

2. In the list, select the resource group that you created.

3. Select **Delete resource group**.

Deleting the resource group also deletes all resources that you created in the designer.

Delete individual assets

In the designer where you created your experiment, delete individual assets by selecting them and then selecting the Delete button.

The compute target that you created here *automatically autoscales* to zero nodes when it's not being used. This action is taken to minimize charges. If you want to delete the compute target, take these steps:

The screenshot shows the Azure Machine Learning Compute blade. The left sidebar lists 'Home', 'Author', 'Automated ML', 'Designer', 'Notebooks', 'Assets', 'Datasets', 'Experiments', 'Pipelines', 'Models', and 'Endpoints'. The 'Compute' section is highlighted with a red box. The main area shows the 'Compute' tab selected, with sub-tabs for 'Compute Instances', 'Training Clusters', 'Inference Clusters' (which is underlined and highlighted with a red box), and 'Attached Compute'. Below these are buttons for '+ New', 'Refresh', 'Delete' (highlighted with a red box), and '...'. A search bar says 'Search to filter items...'. A table lists an inference cluster named 'aks-compute' with type 'Kubernetes Serv...', status 'Success...', creation date '10/17/...', and status 'STAN...'. Navigation arrows at the bottom indicate 'Prev' and 'Next'.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a sidebar with various options: New, Home, Author, Notebooks, Automated ML, Designer, Assets, and Datasets (which is highlighted with a red box). The main area displays a dataset named "TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1". The top navigation bar includes Details, Explore, Models, Datasheet, Refresh, Generate profile, Unregister (which is also highlighted with a red box), and New version. The dataset details page shows sections for Attributes, Tags, and Sample usage. The Attributes section includes Properties (File), Description (a note about automatic promotion), Datastore (workspaceblobstore), Relative path (the specific workspace path), Profile (no profile generated), Files in dataset (4), Current version (1), and Latest version (1). The Tags section lists CreatedByAMLStudio and true. The Sample usage section contains Python code for interacting with the dataset using the azureml.core library.

```
# azureml-core of version 1.0.72 or higher is required
from azureml.core import Workspace, Dataset

subscription_id = 'ee85ed72-2b26-48f6-a0e8-cb5bcf98fb9'
resource_group = 'test-like'
workspace_name = 'like_test'

workspace = Workspace(subscription_id, resource_group, workspace_name)

dataset = Dataset.get_by_name(workspace, name="TD-Sample_1:_Regression_-_Aut
dataset.download(target_path='.', overwrite=False)
```

To delete a dataset, go to the storage account by using the Azure portal or Azure Storage Explorer and manually delete those assets.

Next steps

In part two, you'll learn how to deploy your model as a real-time endpoint.

[Continue to deploying models](#)

Tutorial: Designer - deploy a machine learning model

9/22/2022 • 7 minutes to read • [Edit Online](#)

Use the designer to deploy a machine learning model to predict the price of cars. This tutorial is part two of a two-part series.

In [part one of the tutorial](#) you trained a linear regression model on car prices. In part two, you deploy the model to give others a chance to use it. In this tutorial, you:

- Create a real-time inference pipeline.
- Create an inferencing cluster.
- Deploy the real-time endpoint.
- Test the real-time endpoint.

Prerequisites

Complete [part one of the tutorial](#) to learn how to train and score a machine learning model in the designer.

IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Create a real-time inference pipeline

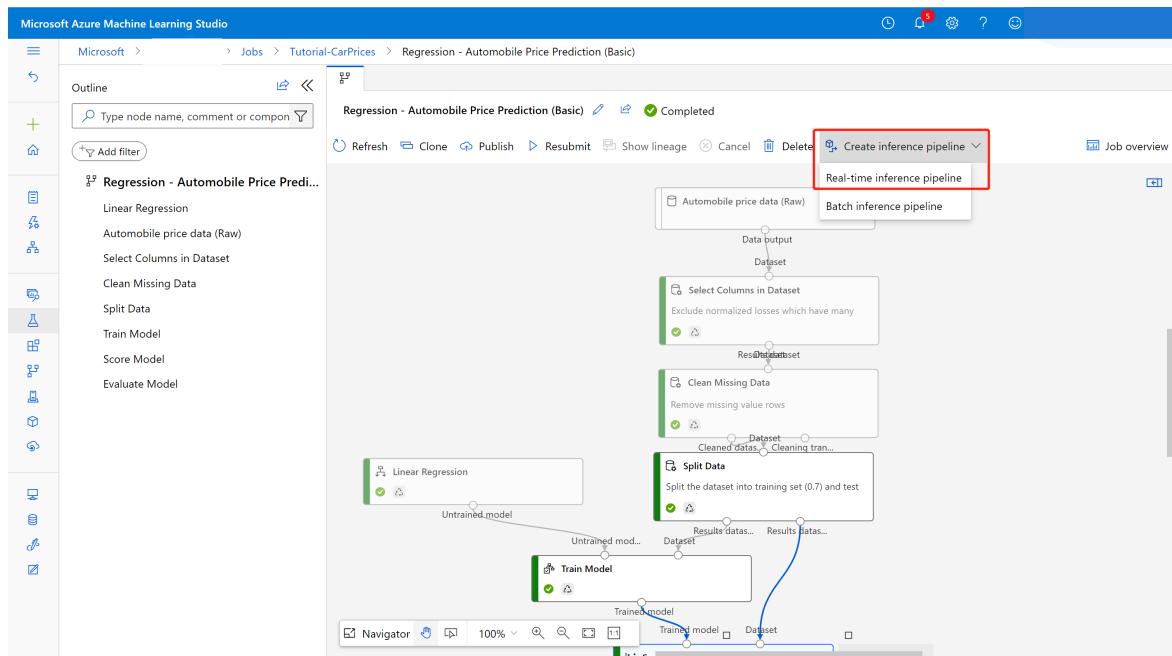
To deploy your pipeline, you must first convert the training pipeline into a real-time inference pipeline. This process removes training components and adds web service inputs and outputs to handle requests.

NOTE

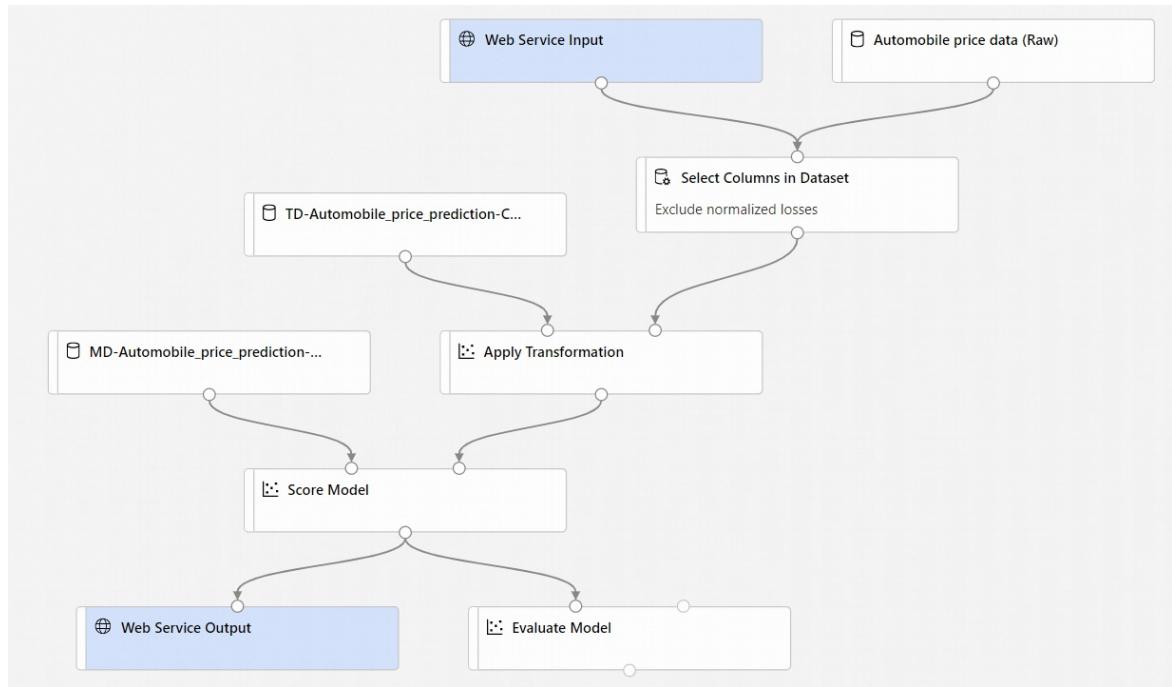
Create inference pipeline only supports training pipelines which contain only the designer built-in components and must have a component like **Train Model** which outputs the trained model.

Create a real-time inference pipeline

1. On pipeline job detail page, above the pipeline canvas, select **Create inference pipeline > Real-time inference pipeline**.



Your new pipeline will now look like this:



When you select **Create inference pipeline**, several things happen:

- The trained model is stored as a **Dataset** component in the component palette. You can find it under **My Datasets**.
- Training components like **Train Model** and **Split Data** are removed.
- The saved trained model is added back into the pipeline.
- **Web Service Input** and **Web Service Output** components are added. These components show where user data enters the pipeline and where data is returned.

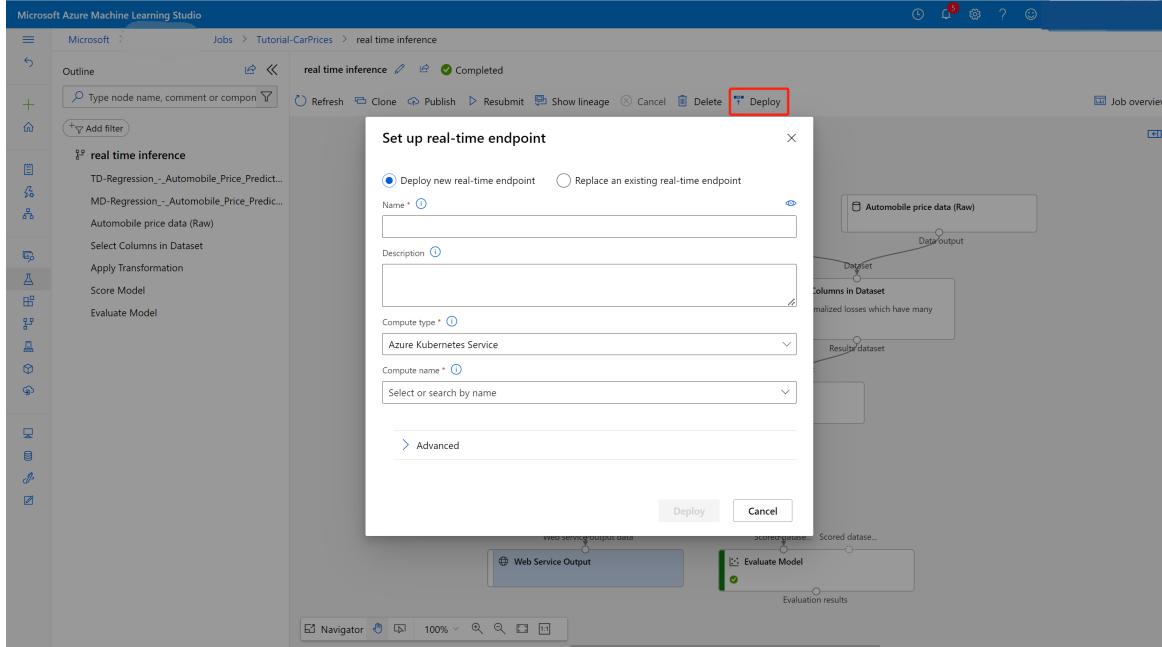
NOTE

By default, the **Web Service Input** will expect the same data schema as the component output data which connects to the same downstream port as it. In this sample, **Web Service Input** and **Automobile price data (Raw)** connect to the same downstream component, hence **Web Service Input** expect the same data schema as **Automobile price data (Raw)** and target variable column `price` is included in the schema. However, usually When you score the data, you won't know the target variable values. For such case, you can remove the target variable column in the inference pipeline using **Select Columns in Dataset** component. Make sure that the output of **Select Columns in Dataset** removing target variable column is connected to the same port as the output of the **Web Service Input** component.

2. Select **Submit**, and use the same compute target and experiment that you used in part one.

If this is the first job, it may take up to 20 minutes for your pipeline to finish running. The default compute settings have a minimum node size of 0, which means that the designer must allocate resources after being idle. Repeated pipeline jobs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each component to further improve efficiency.

3. Go to the real-time inference pipeline job detail by selecting **Job detail** link in the left pane.
4. Select **Deploy** in the job detail page.



Create an inferencing cluster

In the dialog box that appears, you can select from any existing Azure Kubernetes Service (AKS) clusters to deploy your model to. If you don't have an AKS cluster, use the following steps to create one.

1. Select **Compute** in the dialog box that appears to go to the **Compute** page.
2. On the navigation ribbon, select **Inference Clusters > + New**.

The screenshot shows the Microsoft Compute interface. On the left, there's a navigation sidebar with sections like Microsoft, New, Home, Author, Notebooks, Automated ML, Designer, Assets, Data, Jobs, Components, Pipelines, Environments, Models, Endpoints, Manage, and Compute. The Compute section is highlighted with a red box. At the top right, there are tabs for Compute instances, Compute clusters, Inference clusters (which is also highlighted with a red box), and Attached computes. Below these are buttons for New, Refresh, Delete, Detach, Edit columns, and Reset view. A search bar is present. The main area displays a table with columns Name, State, and Type. One entry is listed: aks-cluster, State: Creating, Type: Kubernetes service.

3. In the inference cluster pane, configure a new Kubernetes Service.
4. Enter *aks-compute* for the **Compute name**.
5. Select a nearby region that's available for the **Region**.
6. Select **Create**.

NOTE

It takes approximately 15 minutes to create a new AKS service. You can check the provisioning state on the [Inference Clusters](#) page.

Deploy the real-time endpoint

After your AKS service has finished provisioning, return to the real-time inferencing pipeline to complete deployment.

1. Select **Deploy** above the canvas.
2. Select **Deploy new real-time endpoint**.
3. Select the AKS cluster you created.

Set up real-time endpoint

X

- Deploy new real-time endpoint Replace an existing real-time endpoint

Name * (i)



Description (i)

Compute type * (i)

Azure Kubernetes Service



Compute name * (i)

Select or search by name



> Advanced

Deploy

Cancel

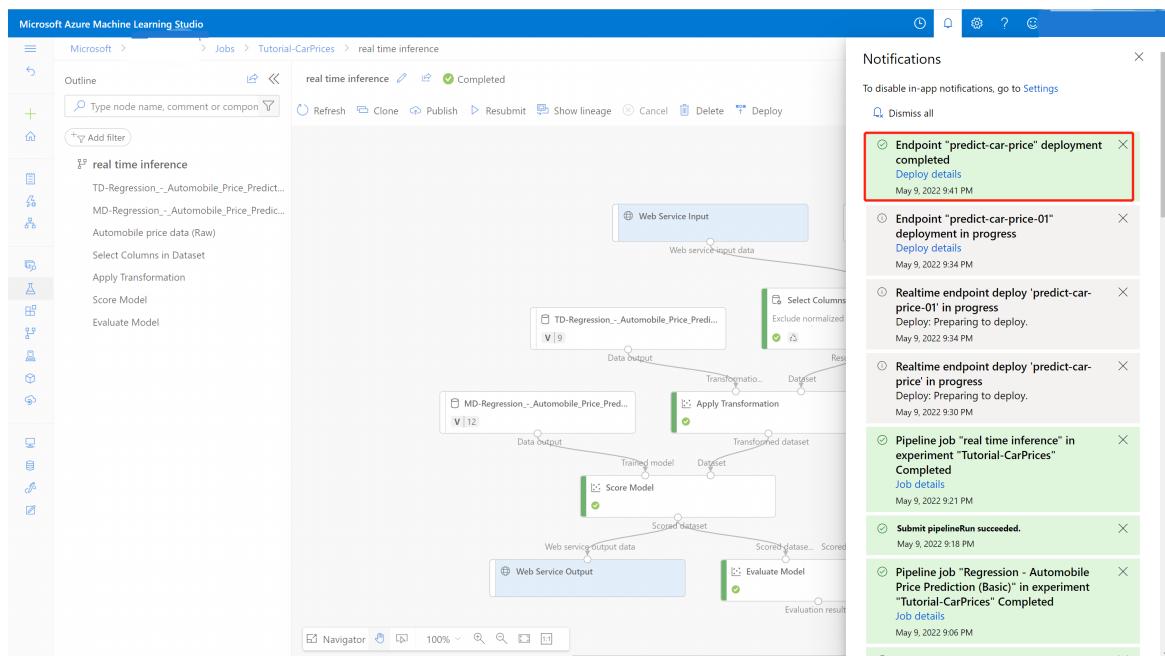
You can also change **Advanced** setting for your real-time endpoint.

ADVANCED SETTING	DESCRIPTION
Enable Application Insights diagnostics and data collection	Whether to enable Azure Application Insights to collect data from the deployed endpoints. By default: false.
Scoring timeout	A timeout in milliseconds to enforce for scoring calls to the web service. By default: 60000.
Auto scale enabled	Whether to enable autoscaling for the web service. By default: true.
Min replicas	The minimum number of containers to use when autoscaling this web service. By default: 1.
Max replicas	The maximum number of containers to use when autoscaling this web service. By default: 10.

ADVANCED SETTING	DESCRIPTION
Target utilization	The target utilization (in percent out of 100) that the autoscaler should attempt to maintain for this web service. By default: 70.
Refresh period	How often (in seconds) the autoscaler attempts to scale this web service. By default: 1.
CPU reserve capacity	The number of CPU cores to allocate for this web service. By default: 0.1.
Memory reserve capacity	The amount of memory (in GB) to allocate for this web service. By default: 0.5.

4. Select Deploy.

A success notification from the notification center appears after deployment finishes. It might take a few minutes.



TIP

You can also deploy to **Azure Container Instance (ACI)** if you select **Azure Container Instance** for **Compute type** in the real-time endpoint setting box. Azure Container Instance is used for testing or development. Use ACI for low-scale CPU-based workloads that require less than 48 GB of RAM.

Test the real-time endpoint

After deployment finishes, you can view your real-time endpoint by going to the **Endpoints** page.

1. On the **Endpoints** page, select the endpoint you deployed.

In the **Details** tab, you can see more information such as the REST URI, Swagger definition, status, and tags.

In the **Consume** tab, you can find sample consumption code, security keys, and set authentication methods.

In the **Deployment logs** tab, you can find the detailed deployment logs of your real-time endpoint.

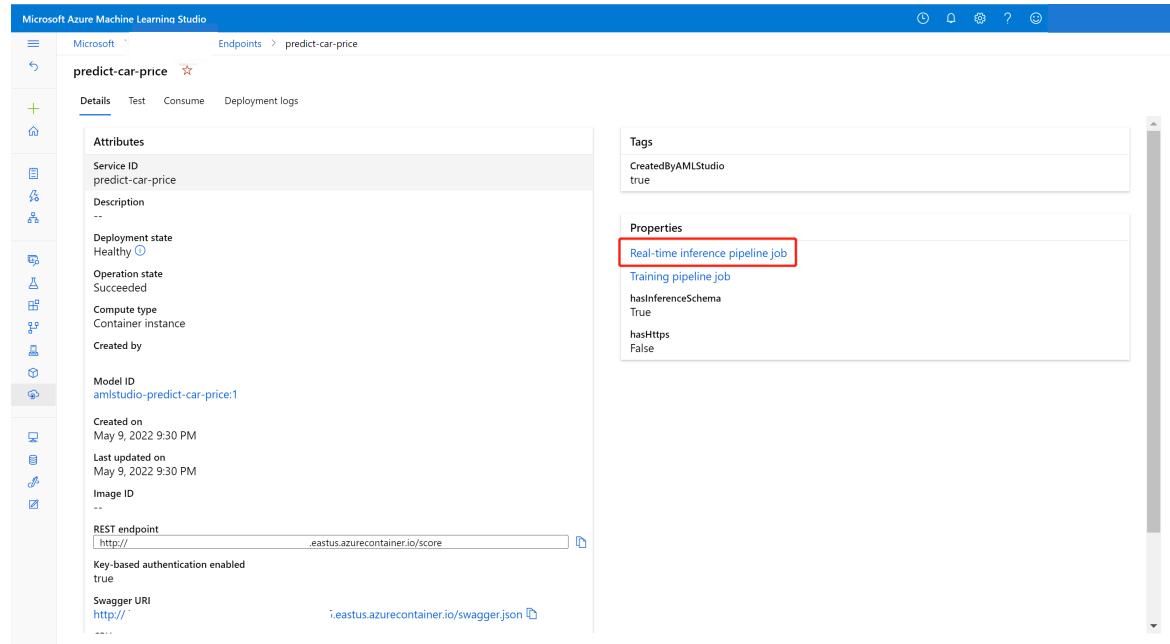
2. To test your endpoint, go to the **Test** tab. From here, you can enter test data and select **Test** verify the output of your endpoint.

Update the real-time endpoint

You can update the online endpoint with new model trained in the designer. On the online endpoint detail page, find your previous training pipeline job and inference pipeline job.

1. You can directly find and modify your training pipeline draft in the designer homepage.

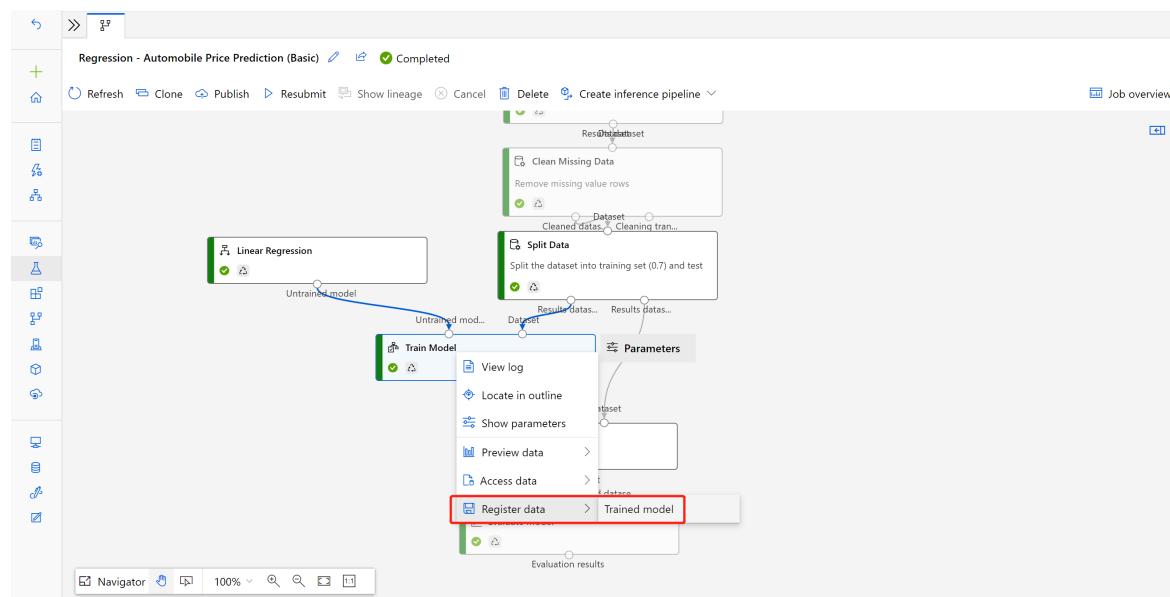
Or you can open the training pipeline job link and then clone it into a new pipeline draft to continue editing.



The screenshot shows the Microsoft Azure Machine Learning Studio interface. In the top navigation bar, 'Microsoft' is selected, and the path 'Endpoints > predict-car-price' is shown. Below the navigation, there are tabs: Details, Test, Consume, and Deployment logs. The 'Details' tab is active. The main content area displays the endpoint's attributes, deployment state (Healthy), and properties. A red box highlights the 'Properties' section, specifically the 'Real-time inference pipeline job' entry. Other properties listed include 'Training pipeline job', 'hasInferenceSchema' (True), and 'hasHttps' (False). The 'Tags' section also lists 'CreatedByAMLStudio' and 'true'.

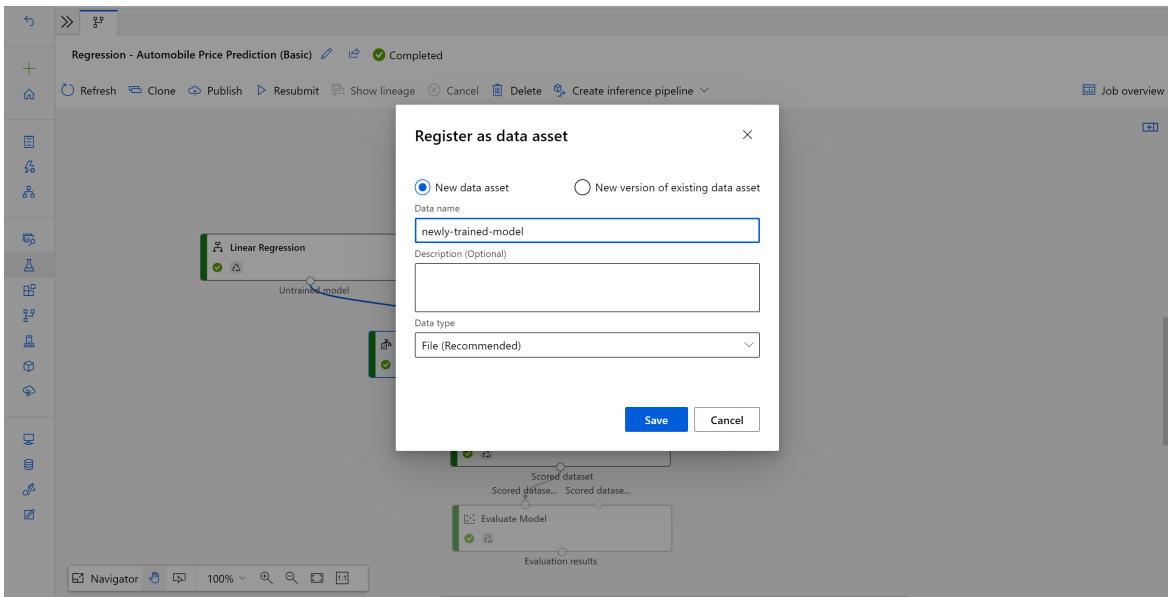
2. After you submit the modified training pipeline, go to the job detail page.

3. When the job completes, right click **Train Model** and select **Register data**.

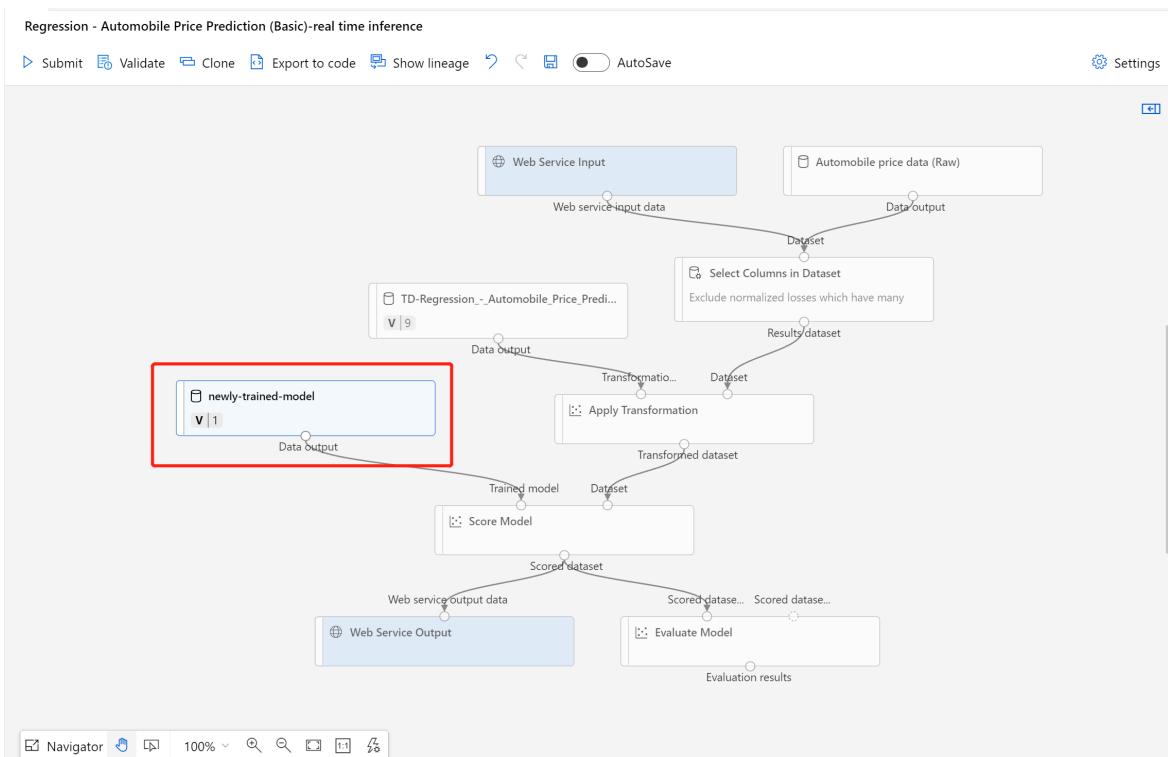


The screenshot shows the Microsoft Azure Machine Learning Studio designer interface. A completed training pipeline titled 'Regression - Automobile Price Prediction (Basic)' is displayed. The pipeline consists of several components: 'Linear Regression', 'Clean Missing Data', 'Split Data', and 'Train Model'. The 'Train Model' component has a blue arrow pointing to it from the 'Linear Regression' component. A context menu is open over the 'Train Model' component, with the 'Register data' option highlighted by a red box. Other options in the menu include 'View log', 'Locate in outline', 'Show parameters', 'Preview data', 'Access data', and 'Trained model'.

Input name and select File type.



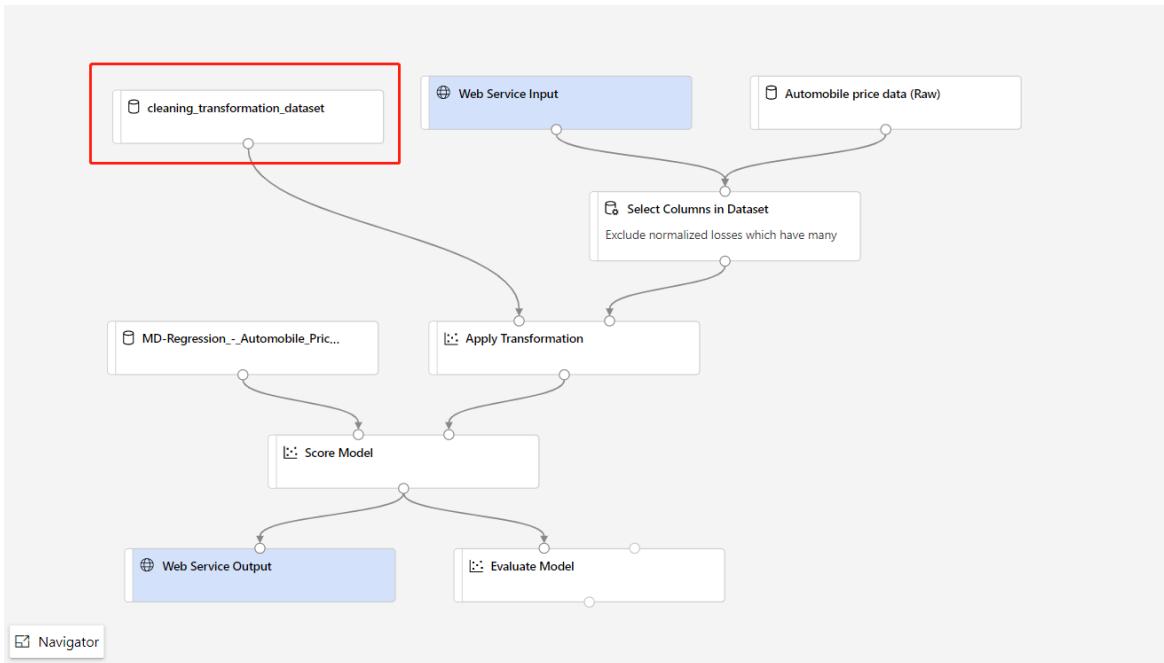
4. After the dataset registers successfully, open your inference pipeline draft, or clone the previous inference pipeline job into a new draft. In the inference pipeline draft, replace the previous trained model shown as **MD-XXXX** node connected to the **Score Model** component with the newly registered dataset.



5. If you need to update the data preprocessing part in your training pipeline, and would like to update that into the inference pipeline, the processing is similar as steps above.

You just need to register the transformation output of the transformation component as dataset.

Then manually replace the TD- component in inference pipeline with the registered dataset.



6. After modifying your inference pipeline with the newly trained model or transformation, submit it. When the job is completed, deploy it to the existing online endpoint deployed previously.

Set up real-time endpoint

X

Deploy new real-time endpoint

Replace an existing real-time endpoint

Existing real-time endpoint *

No existing real-time endpoint to select

Description i

Deploy

Cancel

Limitations

Due to datastore access limitation, if your inference pipeline contains **Import Data** or **Export Data** component, they'll be auto-removed when deploy to real-time endpoint.

Clean up resources

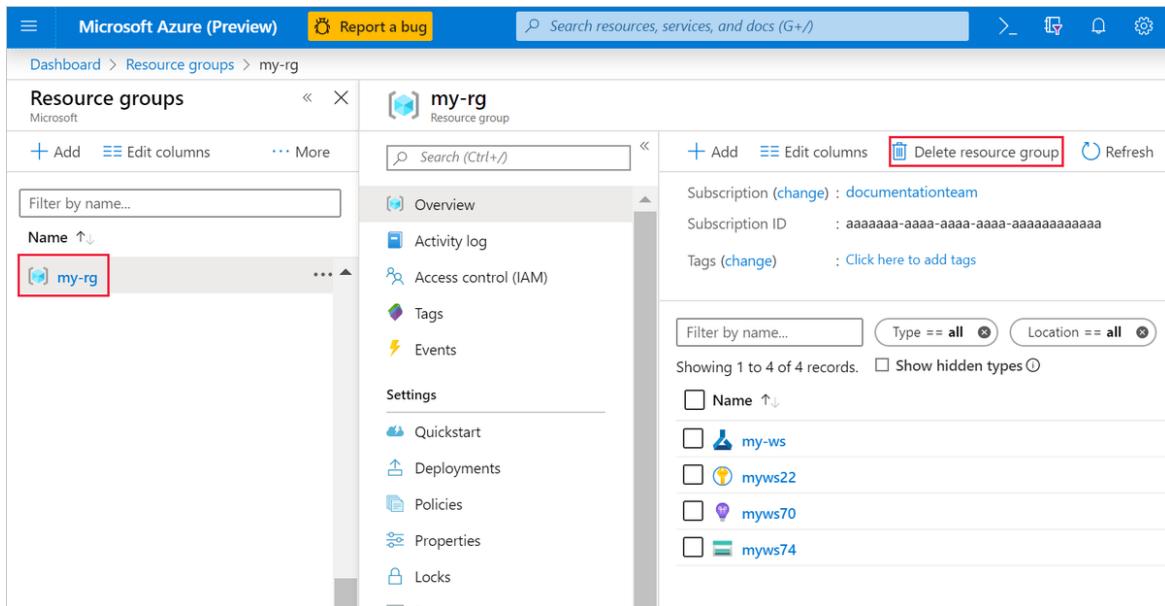
IMPORTANT

You can use the resources that you created as prerequisites for other Azure Machine Learning tutorials and how-to articles.

Delete everything

If you don't plan to use anything that you created, delete the entire resource group so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the left side of the window.



The screenshot shows the Azure portal's Resource groups blade. On the left, there's a list of resource groups with a red box around the entry for 'my-rg'. On the right, the details for 'my-rg' are shown, including its subscription information ('Subscription (change) : documentationteam'), a list of resources ('my-ws', 'myws22', 'myws70', 'myws74'), and a 'Delete resource group' button at the top right which is also highlighted with a red box.

2. In the list, select the resource group that you created.

3. Select **Delete resource group**.

Deleting the resource group also deletes all resources that you created in the designer.

Delete individual assets

In the designer where you created your experiment, delete individual assets by selecting them and then selecting the **Delete** button.

The compute target that you created here *automatically autoscales* to zero nodes when it's not being used. This action is taken to minimize charges. If you want to delete the compute target, take these steps:

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a sidebar with sections like Home, Author, Automated ML, Designer, Notebooks, Assets, Datasets, Experiments, Pipelines, Models, Endpoints, Manage, and Compute. The 'Compute' button is highlighted with a red box. The main area is titled 'Compute' and shows tabs for Compute Instances, Training Clusters, Inference Clusters (which is selected), and Attached Compute. Below these tabs is a toolbar with New, Refresh, Delete (which is also highlighted with a red box), and a search bar. A table lists datasets, with one entry for 'aks-compute' (Kubernetes Service, Success, 10/17/..., Standard). Navigation arrows for 'Prev' and 'Next' are shown at the bottom of the table.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the details page for a dataset named 'TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1'. The sidebar on the left has buttons for New, Home, Author, Notebooks, Automated ML, Designer, Assets, Datasets (which is highlighted with a red box), Experiments, Pipelines, Models, Endpoints, Manage, Compute, Environments, Datastores, and Data labeling. The main content area shows the dataset's attributes, tags, and sample usage. The 'Unregister' button in the top navigation bar is highlighted with a red box.

To delete a dataset, go to the storage account by using the Azure portal or Azure Storage Explorer and manually delete those assets.

Next steps

In this tutorial, you learned the key steps in how to create, deploy, and consume a machine learning model in the designer. To learn more about how you can use the designer see the following links:

- [Designer samples](#): Learn how to use the designer to solve other types of problems.
- [Use Azure Machine Learning studio in an Azure virtual network](#).

Explore Azure Machine Learning with Jupyter Notebooks

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

The [AzureML-Examples](#) repository includes the latest (v2) Azure Machine Learning Python CLI and SDK samples. For information on the various example types, see the [readme](#).

This article shows you how to access the repository from the following environments:

- Azure Machine Learning compute instance
- Your own compute resource
- Data Science Virtual Machine

Option 1: Access on Azure Machine Learning compute instance (recommended)

The easiest way to get started with the samples is to complete the [Quickstart: Get started with Azure Machine Learning](#). Once completed, you'll have a dedicated notebook server pre-loaded with the SDK and the Azure Machine Learning Notebooks repository. No downloads or installation necessary.

To add the community-driven repository, [use a compute instance terminal](#). In the terminal window, clone the repository:

```
git clone https://github.com/Azure/azureml-examples.git --depth 1
```

Option 2: Access on your own notebook server

If you'd like to bring your own notebook server for local development, follow these steps on your computer.

1. Use the instructions at [Azure Machine Learning SDK](#) to install the Azure Machine Learning SDK (v2) for Python
2. Create an [Azure Machine Learning workspace](#).
3. Write a [configuration file](#) file (`aml_config/config.json`).
4. Clone [the AzureML-Examples repository](#).

```
git clone https://github.com/Azure/azureml-examples.git --depth 1
```

5. Start the notebook server from the directory containing your clone.

```
jupyter notebook
```

These instructions install the base SDK packages necessary for the quickstart and tutorial notebooks. Other sample notebooks may require you to install extra components. For more information, see [Install the Azure Machine Learning SDK for Python](#).

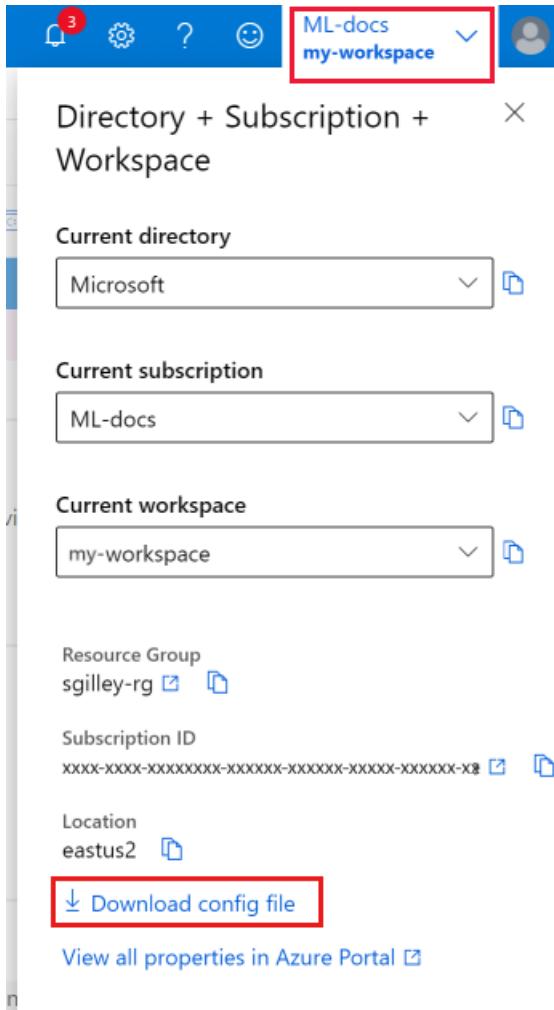
Option 3: Access on a DSVM

The Data Science Virtual Machine (DSVM) is a customized VM image built specifically for doing data science. If you [create a DSVM](#), the SDK and notebook server are installed and configured for you. However, you'll still need to create a workspace and clone the sample repository.

1. [Create an Azure Machine Learning workspace](#).

2. Download a workspace configuration file:

- Sign in to [Azure Machine Learning studio](#)
- Select your workspace settings in the upper right
- Select **Download config file**



3. From the directory where you added the configuration file, clone the [the AzureML-Examples repository](#).

```
git clone https://github.com/Azure/azureml-examples.git --depth 1
```

4. Start the notebook server from the directory, which now contains the clone and the config file.

```
jupyter notebook
```

Next steps

Explore the [MachineLearningNotebooks](#) and [AzureML-Examples](#) repositories to discover what Azure Machine Learning can do.

For more GitHub sample projects and examples, see these repos:

- [Microsoft/MLOps](#)
- [Microsoft/MLOpsPython](#)

Try these tutorials:

- [Train and deploy an image classification model with MNIST](#)
- [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#)

Example pipelines & datasets for Azure Machine Learning designer

9/22/2022 • 10 minutes to read • [Edit Online](#)

Use the built-in examples in Azure Machine Learning designer to quickly get started building your own machine learning pipelines. The Azure Machine Learning designer [GitHub repository](#) contains detailed documentation to help you understand some common machine learning scenarios.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#)
- An Azure Machine Learning workspace

IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Use sample pipelines

The designer saves a copy of the sample pipelines to your studio workspace. You can edit the pipeline to adapt it to your needs and save it as your own. Use them as a starting point to jumpstart your projects.

Here's how to use a designer sample:

1. Sign in to [ml.azure.com](#), and select the workspace you want to work with.
2. Select **Designer**.
3. Select a sample pipeline under the **New pipeline** section.
Select **Show more samples** for a complete list of samples.
4. To run a pipeline, you first have to set default compute target to run the pipeline on.
 - a. In the **Settings** pane to the right of the canvas, select **Select compute target**.
 - b. In the dialog that appears, select an existing compute target or create a new one. Select **Save**.
 - c. Select **Submit** at the top of the canvas to submit a pipeline job.

Depending on the sample pipeline and compute settings, jobs may take some time to complete. The default compute settings have a minimum node size of 0, which means that the designer must allocate resources after being idle. Repeated pipeline jobs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each component to further improve efficiency.

5. After the pipeline finishes running, you can review the pipeline and view the output for each component to learn more. Use the following steps to view component outputs:
 - a. Right-click the component in the canvas whose output you'd like to see.
 - b. Select **Visualize**.

Use the samples as starting points for some of the most common machine learning scenarios.

Regression

Explore these built-in regression samples.

SAMPLE TITLE	DESCRIPTION
Regression - Automobile Price Prediction (Basic)	Predict car prices using linear regression.
Regression - Automobile Price Prediction (Advanced)	Predict car prices using decision forest and boosted decision tree regressors. Compare models to find the best algorithm.

Classification

Explore these built-in classification samples. You can learn more about the samples by opening the samples and viewing the component comments in the designer.

SAMPLE TITLE	DESCRIPTION
Binary Classification with Feature Selection - Income Prediction	Predict income as high or low, using a two-class boosted decision tree. Use Pearson correlation to select features.
Binary Classification with custom Python script - Credit Risk Prediction	Classify credit applications as high or low risk. Use the Execute Python Script component to weight your data.
Binary Classification - Customer Relationship Prediction	Predict customer churn using two-class boosted decision trees. Use SMOTE to sample biased data.
Text Classification - Wikipedia SP 500 Dataset	Classify company types from Wikipedia articles with multiclass logistic regression.
Multiclass Classification - Letter Recognition	Create an ensemble of binary classifiers to classify written letters.

Computer vision

Explore these built-in computer vision samples. You can learn more about the samples by opening the samples and viewing the component comments in the designer.

SAMPLE TITLE	DESCRIPTION
Image Classification using DenseNet	Use computer vision components to build image classification model based on PyTorch DenseNet.

Recommender

Explore these built-in recommender samples. You can learn more about the samples by opening the samples and viewing the component comments in the designer.

SAMPLE TITLE	DESCRIPTION
Wide & Deep based Recommendation - Restaurant Rating Prediction	Build a restaurant recommender engine from restaurant/user features and ratings.
Recommendation - Movie Rating Tweets	Build a movie recommender engine from movie/user features and ratings.

Utility

Learn more about the samples that demonstrate machine learning utilities and features. You can learn more about the samples by opening the samples and viewing the component comments in the designer.

SAMPLE TITLE	DESCRIPTION
Binary Classification using Vowpal Wabbit Model - Adult Income Prediction	Vowpal Wabbit is a machine learning system which pushes the frontier of machine learning with techniques such as online, hashing, allreduce, reductions, learning2search, active, and interactive learning. This sample shows how to use Vowpal Wabbit model to build binary classification model.
Use custom R script - Flight Delay Prediction	Use customized R script to predict if a scheduled passenger flight will be delayed by more than 15 minutes.
Cross Validation for Binary Classification - Adult Income Prediction	Use cross validation to build a binary classifier for adult income.
Permutation Feature Importance	Use permutation feature importance to compute importance scores for the test dataset.
Tune Parameters for Binary Classification - Adult Income Prediction	Use Tune Model Hyperparameters to find optimal hyperparameters to build a binary classifier.

Datasets

When you create a new pipeline in Azure Machine Learning designer, a number of sample datasets are included by default. These sample datasets are used by the sample pipelines in the designer homepage.

The sample datasets are available under **Datasets-Samples** category. You can find this in the component palette to the left of the canvas in the designer. You can use any of these datasets in your own pipeline by dragging it to the canvas.

DATASET NAME	DATASET DESCRIPTION
Adult Census Income Binary Classification dataset	A subset of the 1994 Census database, using working adults over the age of 16 with an adjusted income index of > 100. Usage: Classify people using demographics to predict whether a person earns over 50K a year. Related Research: Kohavi, R., Becker, B., (1996). UCI Machine Learning Repository . Irvine, CA: University of California, School of Information and Computer Science

Dataset Name	Dataset Description
Automobile price data (Raw)	<p>Information about automobiles by make and model, including the price, features such as the number of cylinders and MPG, as well as an insurance risk score.</p> <p>The risk score is initially associated with auto price. It is then adjusted for actual risk in a process known to actuaries as symboling. A value of +3 indicates that the auto is risky, and a value of -3 that it is probably safe.</p> <p>Usage: Predict the risk score by features, using regression or multivariate classification.</p> <p>Related Research: Schlimmer, J.C. (1987). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>
CRM Appetency Labels Shared	Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train_appetency.labels).
CRM Churn Labels Shared	Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train_churn.labels).
CRM Dataset Shared	<p>This data comes from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train.data.zip).</p> <p>The dataset contains 50K customers from the French Telecom company Orange. Each customer has 230 anonymized features, 190 of which are numeric and 40 are categorical. The features are very sparse.</p>
CRM Upselling Labels Shared	Labels from the KDD Cup 2009 customer relationship prediction challenge (orange_large_train_upselling.labels)
Flight Delays Data	<p>Passenger flight on-time performance data taken from the TranStats data collection of the U.S. Department of Transportation (On-Time).</p> <p>The dataset covers the time period April–October 2013. Before uploading to the designer, the dataset was processed as follows:</p> <ul style="list-style-type: none"> - The dataset was filtered to cover only the 70 busiest airports in the continental US - Canceled flights were labeled as delayed by more than 15 minutes - Diverted flights were filtered out - The following columns were selected: Year, Month, DayofMonth, DayOfWeek, Carrier, OriginAirportID, DestAirportID, CRSDepTime, DepDelay, DepDel15, CRSArrTime, ArrDelay, ArrDel15, Canceled
German Credit Card UCI dataset	<p>The UCI Statlog (German Credit Card) dataset (Statlog+German+Credit+Data), using the german.data file. The dataset classifies people, described by a set of attributes, as low or high credit risks. Each example represents a person. There are 20 features, both numerical and categorical, and a binary label (the credit risk value). High credit risk entries have label = 2, low credit risk entries have label = 1. The cost of misclassifying a low risk example as high is 1, whereas the cost of misclassifying a high risk example as low is 5.</p>

DATASET NAME	DATASET DESCRIPTION
IMDB Movie Titles	<p>The dataset contains information about movies that were rated in Twitter tweets: IMDB movie ID, movie name, genre, and production year. There are 17K movies in the dataset. The dataset was introduced in the paper "S. Dooms, T. De Pessemier and L. Martens. MovieTweetings: a Movie Rating Dataset Collected From Twitter. Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013."</p>
Movie Ratings	<p>The dataset is an extended version of the Movie Tweetings dataset. The dataset has 170K ratings for movies, extracted from well-structured tweets on Twitter. Each instance represents a tweet and is a tuple: user ID, IMDB movie ID, rating, timestamp, number of favorites for this tweet, and number of retweets of this tweet. The dataset was made available by A. Said, S. Dooms, B. Loni and D. Tikk for Recommender Systems Challenge 2014.</p>
Weather Dataset	<p>Hourly land-based weather observations from NOAA (merged data from 201304 to 201310). The weather data covers observations made from airport weather stations, covering the time period April-October 2013. Before uploading to the designer, the dataset was processed as follows:</p> <ul style="list-style-type: none"> - Weather station IDs were mapped to corresponding airport IDs - Weather stations not associated with the 70 busiest airports were filtered out - The Date column was split into separate Year, Month, and Day columns - The following columns were selected: AirportID, Year, Month, Day, Time, TimeZone, SkyCondition, Visibility, WeatherType, DryBulbFarenheit, DryBulbCelsius, WetBulbFarenheit, WetBulbCelsius, DewPointFarenheit, DewPointCelsius, RelativeHumidity, WindSpeed, WindDirection, ValueForWindCharacter, StationPressure, PressureTendency, PressureChange, SeaLevelPressure, RecordType, HourlyPrecip, Altimeter
Wikipedia SP 500 Dataset	<p>Data is derived from Wikipedia (https://www.wikipedia.org/) based on articles of each S&P 500 company, stored as XML data. Before uploading to the designer, the dataset was processed as follows:</p> <ul style="list-style-type: none"> - Extract text content for each specific company - Remove wiki formatting - Remove non-alphanumeric characters - Convert all text to lowercase - Known company categories were added <p>Note that for some companies an article could not be found, so the number of records is less than 500.</p>

Dataset Name	Dataset Description
Restaurant Feature Data	<p>A set of metadata about restaurants and their features, such as food type, dining style, and location.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>
Restaurant Ratings	<p>Contains ratings given by users to restaurants on a scale from 0 to 2.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>
Restaurant Customer Data	<p>A set of metadata about customers, including demographics and preferences.</p> <p>Usage: Use this dataset, in combination with the other two restaurant datasets, to train and test a recommender system.</p> <p>Related Research: Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.</p>

Clean up resources

IMPORTANT

You can use the resources that you created as prerequisites for other Azure Machine Learning tutorials and how-to articles.

Delete everything

If you don't plan to use anything that you created, delete the entire resource group so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the left side of the window.

The screenshot shows the Microsoft Azure Resource Groups blade. On the left, there's a list of resource groups with one named 'my-rg' selected and highlighted with a red box. On the right, there's a detailed view for 'my-rg'. At the top of this view, there's a 'Delete resource group' button, which is also highlighted with a red box.

2. In the list, select the resource group that you created.

3. Select **Delete resource group**.

Deleting the resource group also deletes all resources that you created in the designer.

Delete individual assets

In the designer where you created your experiment, delete individual assets by selecting them and then selecting the Delete button.

The compute target that you created here *automatically autoscales* to zero nodes when it's not being used. This action is taken to minimize charges. If you want to delete the compute target, take these steps:

The screenshot shows the Azure Machine Learning Compute blade. On the left, there's a sidebar with various options like 'Automated ML', 'Designer', 'Notebooks', 'Datasets', 'Experiments', 'Pipelines', 'Models', and 'Endpoints'. The 'Compute' option is selected and highlighted with a red box. On the right, there are tabs for 'Compute Instances', 'Training Clusters', 'Inference Clusters' (which is selected and highlighted with a red box), and 'Attached Compute'. Below these tabs, there's a 'Delete' button, which is also highlighted with a red box. A table below lists an 'Inference Cluster' named 'aks-compute'.

Name	Type	Provis...	Cre...	Virtua...
aks-compute	Kubernetes Serv...	Succe...	10/17/...	STAN...

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the Azure Machine Learning studio interface. On the left, a sidebar lists various assets: New, Home, Author, Notebooks, Automated ML, Designer, Assets, Datasets (which is selected and highlighted with a red box), Experiments, Pipelines, Models, Endpoints, Compute, Environments, Datastores, and Data labeling. The main content area displays details for a dataset named "TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1". The "Attributes" section includes fields for Properties (File), Description (This is a dataset promoted by inference graph generation automatically on 11/12/2018.), Datastore (workspaceblobstore), Relative path (azureml://4393076b-19ff-4e41-81d9-a1146d905696/Cleaning_transformation), Profile (No profile generated), Files in dataset (4), Current version (1), and Latest version (1). The "Tags" section contains CreatedByAMLStudio and true. The "Sample usage" section provides a Python code snippet:

```
# azureml-core of version 1.0.72 or higher is required
from azureml.core import Workspace, Dataset

subscription_id = 'ee85ed72-2b26-48f6-a0e8-cb5bcf98fb9'
resource_group = 'test-like'
workspace_name = 'like_test'

workspace = Workspace(subscription_id, resource_group, workspace_name)

dataset = Dataset.get_by_name(workspace, name='TD-Sample_1:_Regression_-_Aut
dataset.download(target_path='.', overwrite=False)
```

To delete a dataset, go to the storage account by using the Azure portal or Azure Storage Explorer and manually delete those assets.

Next steps

Learn the fundamentals of predictive analytics and machine learning with [Tutorial: Predict automobile price with the designer](#)

What is Azure Machine Learning CLI & Python SDK v2?

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)  [Python SDK azure-ai-ml v2 \(preview\)](#)

Azure Machine Learning CLI v2 and Azure Machine Learning Python SDK v2 (preview) introduce a consistency of features and terminology across the interfaces. In order to create this consistency, the syntax of commands differs, in some cases significantly, from the first versions (v1).

Azure Machine Learning CLI v2

The Azure Machine Learning CLI v2 (CLI v2) is the latest extension for the [Azure CLI](#). The CLI v2 provides commands in the format `az ml <noun> <verb> <options>` to create and maintain Azure ML assets and workflows. The assets or workflows themselves are defined using a YAML file. The YAML file defines the configuration of the asset or workflow – what is it, where should it run, and so on.

A few examples of CLI v2 commands:

- `az ml job create --file my_job_definition.yaml`
- `az ml environment update --name my-env --file my_updated_env_definition.yaml`
- `az ml model list`
- `az ml compute show --name my_compute`

Use cases for CLI v2

The CLI v2 is useful in the following scenarios:

- On board to Azure ML without the need to learn a specific programming language

The YAML file defines the configuration of the asset or workflow – what is it, where should it run, and so on. Any custom logic/IP used, say data preparation, model training, model scoring can remain in script files, which are referred to in the YAML, but not part of the YAML itself. Azure ML supports script files in python, R, Java, Julia or C#. All you need to learn is YAML format and command lines to use Azure ML. You can stick with script files of your choice.

- Ease of deployment and automation

The use of command-line for execution makes deployment and automation simpler, since workflows can be invoked from any offering/platform, which allows users to call the command line.

- Managed inference deployments

Azure ML offers [endpoints](#) to streamline model deployments for both real-time and batch inference deployments. This functionality is available only via CLI v2 and SDK v2 (preview).

- Reusable components in pipelines

Azure ML introduces [components](#) for managing and reusing common logic across pipelines. This functionality is available only via CLI v2 and SDK v2.

Azure Machine Learning Python SDK v2 (preview)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Azure ML Python SDK v2 is an updated Python SDK package, which allows users to:

- Submit training jobs
- Manage data, models, environments
- Perform managed inferencing (real time and batch)
- Stitch together multiple tasks and production workflows using Azure ML pipelines

The SDK v2 is on par with CLI v2 functionality and is consistent in how assets (nouns) and actions (verbs) are used between SDK and CLI. For example, to list an asset, the `list` action can be used in both CLI and SDK. The same `list` action can be used to list a compute, model, environment, and so on.

Use cases for SDK v2

The SDK v2 is useful in the following scenarios:

- Use Python functions to build a single step or a complex workflow

SDK v2 allows you to build a single command or a chain of commands like python functions - the command has a name, parameters, expects input, and returns output.

- Move from simple to complex concepts incrementally

SDK v2 allows you to:

- Construct a single command.
- Add a hyperparameter sweep on top of that command,
- Add the command with various others into a pipeline one after the other.

This construction is useful, given the iterative nature of machine learning.

- Reusable components in pipelines

Azure ML introduces [components](#) for managing and reusing common logic across pipelines. This functionality is available only via CLI v2 and SDK v2.

- Managed inferencing

Azure ML offers [endpoints](#) to streamline model deployments for both real-time and batch inference deployments. This functionality is available only via CLI v2 and SDK v2.

Should I use v1 or v2?

CLI v2

The Azure Machine Learning CLI v1 has been deprecated. We recommend you to use CLI v2 if:

- You were a CLI v1 user
- You want to use new features like - reusable components, managed inferencing
- You don't want to use a Python SDK - CLI v2 allows you to use YAML with scripts in python, R, Java, Julia or C#
- You were a user of R SDK previously - Azure ML won't support an SDK in `R`. However, the CLI v2 has support for `R` scripts.

- You want to use command line based automation/deployments

SDK v2

The Azure Machine Learning Python SDK v1 doesn't have a planned deprecation date. If you have significant investments in Python SDK v1 and don't need any new features offered by SDK v2, you can continue to use SDK v1. However, you should consider using SDK v2 if:

- You want to use new features like - reusable components, managed inferencing
- You're starting a new workflow or pipeline - all new features and future investments will be introduced in v2
- You want to take advantage of the improved usability of the Python SDK v2 - ability to compose jobs and pipelines using Python functions, easy evolution from simple to complex tasks etc.
- You don't need features like AutoML in pipelines, Parallel Run Steps, Scheduling Pipelines and Spark Jobs. These features are not yet available in SDK v2.

Next steps

- [How to migrate from v1 to v2](#)
- Get started with CLI v2
 - [Install and set up CLI \(v2\)](#)
 - [Train models with the CLI \(v2\)](#)
 - [Deploy and score models with managed online endpoint](#)
- Get started with SDK v2
 - [Install and set up SDK \(v2\)](#)
 - [Train models with the Azure ML Python SDK v2 \(preview\)](#)
 - [Tutorial: Create production ML pipelines with Python SDK v2 \(preview\) in a Jupyter notebook](#)

Train models with Azure Machine Learning

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

Azure Machine Learning provides several ways to train your models, from code-first solutions using the SDK to low-code solutions such as automated machine learning and the visual designer. Use the following list to determine which training method is right for you:

- [Azure Machine Learning SDK for Python](#): The Python SDK provides several ways to train models, each with different capabilities.

TRAINING METHOD	DESCRIPTION
command()	A typical way to train models is to submit a command() that includes a training script, environment, and compute information.
Automated machine learning	Automated machine learning allows you to train models without extensive data science or programming knowledge . For people with a data science and programming background, it provides a way to save time and resources by automating algorithm selection and hyperparameter tuning. You don't have to worry about defining a job configuration when using automated machine learning.
Machine learning pipeline	Pipelines are not a different training method, but a way of defining a workflow using modular, reusable steps that can include training as part of the workflow. Machine learning pipelines support using automated machine learning and run configuration to train models. Since pipelines are not focused specifically on training, the reasons for using a pipeline are more varied than the other training methods. Generally, you might use a pipeline when: <ul style="list-style-type: none">* You want to schedule unattended processes such as long running training jobs or data preparation.* Use multiple steps that are coordinated across heterogeneous compute resources and storage locations.* Use the pipeline as a reusable template for specific scenarios, such as retraining or batch scoring.* Track and version data sources, inputs, and outputs for your workflow.* Your workflow is implemented by different teams that work on specific steps independently. Steps can then be joined together in a pipeline to implement the workflow.

- **Designer**: Azure Machine Learning designer provides an easy entry-point into machine learning for building proof of concepts, or for users with little coding experience. It allows you to train models using a drag and drop web-based UI. You can use Python code as part of the design, or train models without writing any code.
- **Azure CLI**: The machine learning CLI provides commands for common tasks with Azure Machine

Learning, and is often used for **scripting and automating tasks**. For example, once you've created a training script or pipeline, you might use the Azure CLI to start a training job on a schedule or when the data files used for training are updated. For training models, it provides commands that submit training jobs. It can submit jobs using run configurations or pipelines.

Each of these training methods can use different types of compute resources for training. Collectively, these resources are referred to as **compute targets**. A compute target can be a local machine or a cloud resource, such as an Azure Machine Learning Compute, Azure HDInsight, or a remote virtual machine.

Python SDK

The Azure Machine Learning SDK for Python allows you to build and run machine learning workflows with Azure Machine Learning. You can interact with the service from an interactive Python session, Jupyter Notebooks, Visual Studio Code, or other IDE.

- [Install/update the SDK](#)
- [Configure a development environment for Azure Machine Learning](#)

Submit a command

A generic training job with Azure Machine Learning can be defined using the [command\(\)](#). The command is then used, along with your training script(s) to train a model on the specified compute target.

You may start with a command for your local computer, and then switch to one for a cloud-based compute target as needed. When changing the compute target, you only change the compute parameter in the command that you use. A run also logs information about the training job, such as the inputs, outputs, and logs.

- [Tutorial: Train your first ML model](#)
- [Examples: Jupyter Notebook and Python examples of training models](#)

Automated Machine Learning

Define the iterations, hyperparameter settings, featurization, and other settings. During training, Azure Machine Learning tries different algorithms and parameters in parallel. Training stops once it hits the exit criteria you defined.

TIP

In addition to the Python SDK, you can also use Automated ML through [Azure Machine Learning studio](#).

- [What is automated machine learning?](#)
- [Tutorial: Create your first classification model with automated machine learning](#)
- [How to: Configure automated ML experiments in Python](#)
- [How to: Create, explore, and deploy automated machine learning experiments with Azure Machine Learning studio](#)

Machine learning pipeline

Machine learning pipelines can use the previously mentioned training methods. Pipelines are more about creating a workflow, so they encompass more than just the training of models.

- [What are ML pipelines in Azure Machine Learning?](#)
- [Tutorial: Create production ML pipelines with Python SDK v2 \(preview\) in a Jupyter notebook](#)

Understand what happens when you submit a training job

The Azure training lifecycle consists of:

1. Zipping the files in your project folder, ignoring those specified in `.amlignore` or `.gitignore`

2. Scaling up your compute cluster
3. Building or downloading the dockerfile to the compute node
 - a. The system calculates a hash of:
 - The base image
 - Custom docker steps (see [Deploy a model using a custom Docker base image](#))
 - The conda definition YAML (see [Manage Azure Machine Learning environments with the CLI \(v2\)](#))
 - b. The system uses this hash as the key in a lookup of the workspace Azure Container Registry (ACR)
 - c. If it is not found, it looks for a match in the global ACR
 - d. If it is not found, the system builds a new image (which will be cached and registered with the workspace ACR)
4. Downloading your zipped project file to temporary storage on the compute node
5. Unzipping the project file
6. The compute node executing `python <entry script> <arguments>`
7. Saving logs, model files, and other files written to `./outputs` to the storage account associated with the workspace
8. Scaling down compute, including removing temporary storage

Azure Machine Learning designer

The designer lets you train models using a drag and drop interface in your web browser.

- [What is the designer?](#)
- [Tutorial: Predict automobile price](#)

Azure CLI

The machine learning CLI is an extension for the Azure CLI. It provides cross-platform CLI commands for working with Azure Machine Learning. Typically, you use the CLI to automate tasks, such as training a machine learning model.

- [Use the CLI extension for Azure Machine Learning](#)
- [MLOps on Azure](#)
- [Train models with the CLI \(v2\)](#)

VS Code

You can use the VS Code extension to run and manage your training jobs. See the [VS Code resource management how-to guide](#) to learn more.

Next steps

Learn how to [Tutorial: Create production ML pipelines with Python SDK v2 \(preview\) in a Jupyter notebook](#).

Distributed training with Azure Machine Learning

9/22/2022 • 2 minutes to read • [Edit Online](#)

In this article, you learn about distributed training and how Azure Machine Learning supports it for deep learning models.

In distributed training the workload to train a model is split up and shared among multiple mini processors, called worker nodes. These worker nodes work in parallel to speed up model training. Distributed training can be used for traditional ML models, but is better suited for compute and time intensive tasks, like [deep learning](#) for training deep neural networks.

Deep learning and distributed training

There are two main types of distributed training: [data parallelism](#) and [model parallelism](#). For distributed training on deep learning models, the [Azure Machine Learning SDK in Python](#) supports integrations with popular frameworks, PyTorch and TensorFlow. Both frameworks employ data parallelism for distributed training, and can leverage [horovod](#) for optimizing compute speeds.

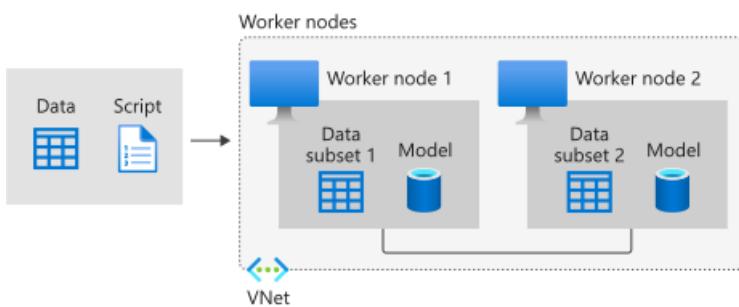
- [Distributed training with PyTorch](#)
- [Distributed training with TensorFlow](#)

For ML models that don't require distributed training, see [train models with Azure Machine Learning](#) for the different ways to train models using the Python SDK.

Data parallelism

Data parallelism is the easiest to implement of the two distributed training approaches, and is sufficient for most use cases.

In this approach, the data is divided into partitions, where the number of partitions is equal to the total number of available nodes, in the compute cluster. The model is copied in each of these worker nodes, and each worker operates on its own subset of the data. Keep in mind that each node has to have the capacity to support the model that's being trained, that is the model has to entirely fit on each node. The following diagram provides a visual demonstration of this approach.



Each node independently computes the errors between its predictions for its training samples and the labeled outputs. In turn, each node updates its model based on the errors and must communicate all of its changes to the other nodes to update their corresponding models. This means that the worker nodes need to synchronize the model parameters, or gradients, at the end of the batch computation to ensure they are training a consistent model.

Model parallelism

In model parallelism, also known as network parallelism, the model is segmented into different parts that can run concurrently in different nodes, and each one will run on the same data. The scalability of this method depends on the degree of task parallelization of the algorithm, and it is more complex to implement than data parallelism.

In model parallelism, worker nodes only need to synchronize the shared parameters, usually once for each forward or backward-propagation step. Also, larger models aren't a concern since each node operates on a subsection of the model on the same training data.

Next steps

- For a technical example, see the [reference architecture scenario](#).
- Find tips for MPI, TensorFlow, and PyTorch in the [Distributed GPU training guide](#)

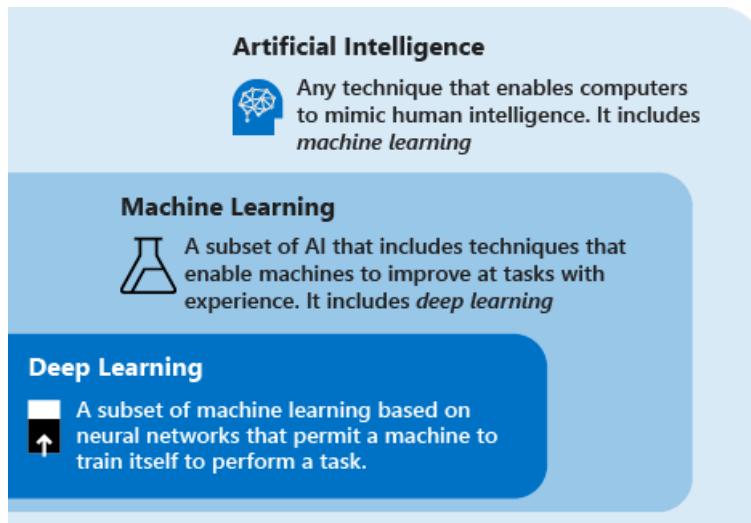
Deep learning vs. machine learning in Azure Machine Learning

9/22/2022 • 9 minutes to read • [Edit Online](#)

This article explains deep learning vs. machine learning and how they fit into the broader category of artificial intelligence. Learn about deep learning solutions you can build on Azure Machine Learning, such as fraud detection, voice and facial recognition, sentiment analysis, and time series forecasting.

For guidance on choosing algorithms for your solutions, see the [Machine Learning Algorithm Cheat Sheet](#).

Deep learning, machine learning, and AI



Consider the following definitions to understand deep learning vs. machine learning vs. AI:

- **Deep learning** is a subset of machine learning that's based on artificial neural networks. The *learning process* is *deep* because the structure of artificial neural networks consists of multiple input, output, and hidden layers. Each layer contains units that transform the input data into information that the next layer can use for a certain predictive task. Thanks to this structure, a machine can learn through its own data processing.
- **Machine learning** is a subset of artificial intelligence that uses techniques (such as deep learning) that enable machines to use experience to improve at tasks. The *learning process* is based on the following steps:
 1. Feed data into an algorithm. (In this step you can provide additional information to the model, for example, by performing feature extraction.)
 2. Use this data to train a model.
 3. Test and deploy the model.
 4. Consume the deployed model to do an automated predictive task. (In other words, call and use the deployed model to receive the predictions returned by the model.)
- **Artificial intelligence (AI)** is a technique that enables computers to mimic human intelligence. It includes machine learning.

By using machine learning and deep learning techniques, you can build computer systems and applications that do tasks that are commonly associated with human intelligence. These tasks include image recognition, speech recognition, and language translation.

Techniques of deep learning vs. machine learning

Now that you have the overview of machine learning vs. deep learning, let's compare the two techniques. In machine learning, the algorithm needs to be told how to make an accurate prediction by consuming more information (for example, by performing feature extraction). In deep learning, the algorithm can learn how to make an accurate prediction through its own data processing, thanks to the artificial neural network structure.

The following table compares the two techniques in more detail:

	ALL MACHINE LEARNING	ONLY DEEP LEARNING
Number of data points	Can use small amounts of data to make predictions.	Needs to use large amounts of training data to make predictions.
Hardware dependencies	Can work on low-end machines. It doesn't need a large amount of computational power.	Depends on high-end machines. It inherently does a large number of matrix multiplication operations. A GPU can efficiently optimize these operations.
Featurization process	Requires features to be accurately identified and created by users.	Learns high-level features from data and creates new features by itself.
Learning approach	Divides the learning process into smaller steps. It then combines the results from each step into one output.	Moves through the learning process by resolving the problem on an end-to-end basis.
Execution time	Takes comparatively little time to train, ranging from a few seconds to a few hours.	Usually takes a long time to train because a deep learning algorithm involves many layers.
Output	The output is usually a numerical value, like a score or a classification.	The output can have multiple formats, like a text, a score or a sound.

What is transfer learning?

Training deep learning models often requires large amounts of training data, high-end compute resources (GPU, TPU), and a longer training time. In scenarios when you don't have any of these available to you, you can shortcut the training process using a technique known as *transfer learning*.

Transfer learning is a technique that applies knowledge gained from solving one problem to a different but related problem.

Due to the structure of neural networks, the first set of layers usually contains lower-level features, whereas the final set of layers contains higher-level features that are closer to the domain in question. By repurposing the final layers for use in a new domain or problem, you can significantly reduce the amount of time, data, and compute resources needed to train the new model. For example, if you already have a model that recognizes cars, you can repurpose that model using transfer learning to also recognize trucks, motorcycles, and other kinds of vehicles.

Learn how to apply transfer learning for image classification using an open-source framework in Azure Machine Learning : [Train a deep learning PyTorch model using transfer learning](#).

Deep learning use cases

Because of the artificial neural network structure, deep learning excels at identifying patterns in unstructured

data such as images, sound, video, and text. For this reason, deep learning is rapidly transforming many industries, including healthcare, energy, finance, and transportation. These industries are now rethinking traditional business processes.

Some of the most common applications for deep learning are described in the following paragraphs. In Azure Machine Learning, you can use a model from you build from an open-source framework or build the model using the tools provided.

Named-entity recognition

Named-entity recognition is a deep learning method that takes a piece of text as input and transforms it into a pre-specified class. This new information could be a postal code, a date, a product ID. The information can then be stored in a structured schema to build a list of addresses or serve as a benchmark for an identity validation engine.

Object detection

Deep learning has been applied in many object detection use cases. Object detection comprises two parts: image classification and then image localization. Image *classification* identifies the image's objects, such as cars or people. Image *localization* provides the specific location of these objects.

Object detection is already used in industries such as gaming, retail, tourism, and self-driving cars.

Image caption generation

Like image recognition, in image captioning, for a given image, the system must generate a caption that describes the contents of the image. When you can detect and label objects in photographs, the next step is to turn those labels into descriptive sentences.

Usually, image captioning applications use convolutional neural networks to identify objects in an image and then use a recurrent neural network to turn the labels into consistent sentences.

Machine translation

Machine translation takes words or sentences from one language and automatically translates them into another language. Machine translation has been around for a long time, but deep learning achieves impressive results in two specific areas: automatic translation of text (and translation of speech to text) and automatic translation of images.

With the appropriate data transformation, a neural network can understand text, audio, and visual signals. Machine translation can be used to identify snippets of sound in larger audio files and transcribe the spoken word or image as text.

Text analytics

Text analytics based on deep learning methods involves analyzing large quantities of text data (for example, medical documents or expenses receipts), recognizing patterns, and creating organized and concise information out of it.

Companies use deep learning to perform text analysis to detect insider trading and compliance with government regulations. Another common example is insurance fraud: text analytics has often been used to analyze large amounts of documents to recognize the chances of an insurance claim being fraud.

Artificial neural networks

Artificial neural networks are formed by layers of connected nodes. Deep learning models use neural networks that have a large number of layers.

The following sections explore most popular artificial neural network typologies.

Feedforward neural network

The feedforward neural network is the most simple type of artificial neural network. In a feedforward network, information moves in only one direction from input layer to output layer. Feedforward neural networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, and each layer is fully connected to all neurons in the layer before. The last fully connected layer (the output layer) represents the generated predictions.

Recurrent neural network (RNN)

Recurrent neural networks are a widely used artificial neural network. These networks save the output of a layer and feed it back to the input layer to help predict the layer's outcome. Recurrent neural networks have great learning abilities. They're widely used for complex tasks such as time series forecasting, learning handwriting, and recognizing language.

Convolutional neural network (CNN)

A convolutional neural network is a particularly effective artificial neural network, and it presents a unique architecture. Layers are organized in three dimensions: width, height, and depth. The neurons in one layer connect not to all the neurons in the next layer, but only to a small region of the layer's neurons. The final output is reduced to a single vector of probability scores, organized along the depth dimension.

Convolutional neural networks have been used in areas such as video recognition, image recognition, and recommender systems.

Generative adversarial network (GAN)

Generative adversarial networks are generative models trained to create realistic content such as images. It is made up of two networks known as generator and discriminator. Both networks are trained simultaneously. During training, the generator uses random noise to create new synthetic data that closely resembles real data. The discriminator takes the output from the generator as input and uses real data to determine whether the generated content is real or synthetic. Each network is competing with each other. The generator is trying to generate synthetic content that is indistinguishable from real content and the discriminator is trying to correctly classify inputs as real or synthetic. The output is then used to update the weights of both networks to help them better achieve their respective goals.

Generative adversarial networks are used to solve problems like image to image translation and age progression.

Transformers

Transformers are a model architecture that is suited for solving problems containing sequences such as text or time-series data. They consist of [encoder and decoder layers](#). The encoder takes an input and maps it to a numerical representation containing information such as context. The decoder uses information from the encoder to produce an output such as translated text. What makes transformers different from other architectures containing encoders and decoders are the attention sub-layers. Attention is the idea of focusing on specific parts of an input based on the importance of their context in relation to other inputs in a sequence. For example, when summarizing a news article, not all sentences are relevant to describe the main idea. By focusing on key words throughout the article, summarization can be done in a single sentence, the headline.

Transformers have been used to solve natural language processing problems such as translation, text generation, question answering, and text summarization.

Some well-known implementations of transformers are:

- Bidirectional Encoder Representations from Transformers (BERT)
- Generative Pre-trained Transformer 2 (GPT-2)
- Generative Pre-trained Transformer 3 (GPT-3)

Next steps

The following articles show you more options for using open-source deep learning models in [Azure Machine Learning](#):

- [Classify handwritten digits by using a TensorFlow model](#)
- [Classify handwritten digits by using a TensorFlow estimator and Keras](#)

From artifacts to models in MLflow

9/22/2022 • 7 minutes to read • [Edit Online](#)

The following article explains the differences between an artifact and a model in MLflow and how to transition from one to the other. It also explains how Azure Machine Learning uses the MLflow model's concept to enable streamlined deployment workflows.

What's the difference between an artifact and a model?

If you are not familiar with MLflow, you may not be aware of the difference between logging artifacts or files vs. logging MLflow models. There are some fundamental differences between the two:

Artifacts

Any file generated (and captured) from an experiment's run or job is an artifact. It may represent a model serialized as a Pickle file, the weights of a PyTorch or TensorFlow model, or even a text file containing the coefficients of a linear regression. Other artifacts can have nothing to do with the model itself, but they can contain configuration to run the model, pre-processing information, sample data, etc. As you can see, an artifact can come in any format.

You may have been logging artifacts already:

- [Using MLflow SDK](#)
- [Using Azure ML SDK v1](#)
- [Using the outputs folder](#)

```
filename = 'model.pkl'
with open(filename, 'wb') as f:
    pickle.dump(model, f)

mlflow.log_artifact(filename)
```

Models

A model in MLflow is also an artifact. However, we make stronger assumptions about this type of artifacts. Such assumptions provide a clear contract between the saved files and what they mean. When you log your models as artifacts (simple files), you need to know what the model builder meant for each of them in order to know how to load the model for inference. On the contrary, MLflow models can be loaded using the contract specified in the [The MLModel format](#).

In Azure Machine Learning, logging models has the following advantages:

- You can deploy them on real-time or batch endpoints without providing an scoring script nor an environment.
- When deployed, Model's deployments have a Swagger generated automatically and the **Test** feature can be used in Azure ML studio.
- Models can be used as pipelines inputs directly.
- You can use the [Responsible AI dashboard \(preview\)](#).

Models can get logged by:

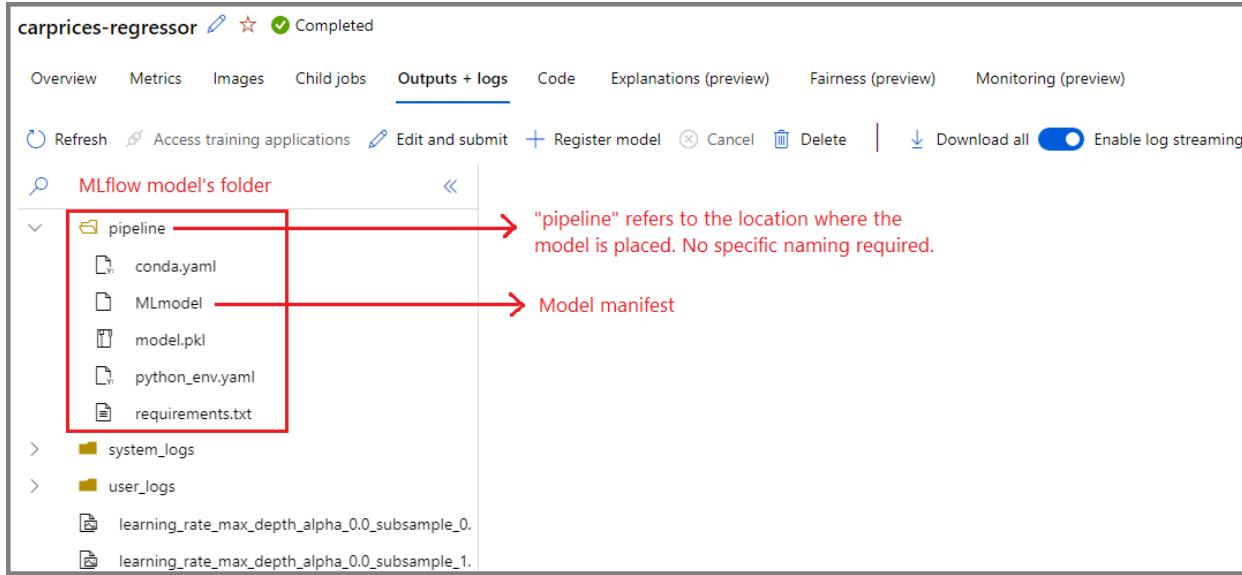
- [Using MLflow SDK](#)

- Using Azure ML SDK v1
- Using the outputs folder

```
import mlflow
mlflow.log_model(sklearn_estimator, "classifier")
```

The MLmodel format

MLflow adopts the MLmodel format as a way to create a contract between the artifacts and what they represent. The MLmodel format stores assets in a folder. Among them, there is a particular file named MLmodel. This file is the single source of truth about how a model can be loaded and used.



The following example shows how the `MLmodel` file for a computer version model trained with `fastai` may look like:

MLmodel

```
artifact_path: classifier
flavors:
  fastai:
    data: model.fastai
    fastai_version: 2.4.1
  python_function:
    data: model.fastai
    env: conda.yaml
    loader_module: mlflow.fastai
    python_version: 3.8.12
model_uuid: e694c68eba484299976b06ab9058f636
run_id: e13da8ac-b1e6-45d4-a9b2-6a0a5cfac537
signature:
  inputs: '[{"type": "tensor",
            "tensor-spec":
              {"dtype": "uint8", "shape": [-1, 300, 300, 3]}
            }'
  outputs: '[{"type": "tensor",
             "tensor-spec":
               {"dtype": "float32", "shape": [-1, 2]}
             }]'
```

The model's flavors

Considering the variety of machine learning frameworks available to use, MLflow introduced the concept of

flavor as a way to provide a unique contract to work across all of them. A flavor indicates what to expect for a given model created with a specific framework. For instance, TensorFlow has its own flavor, which specifies how a TensorFlow model should be persisted and loaded. Because each model flavor indicates how they want to persist and load models, the MLModel format doesn't enforce a single serialization mechanism that all the models need to support. Such decision allows each flavor to use the methods that provide the best performance or best support according to their best practices - without compromising compatibility with the MLModel standard.

The following is an example of the `flavors` section for an `fastai` model.

```
flavors:  
  fastai:  
    data: model.fastai  
    fastai_version: 2.4.1  
  python_function:  
    data: model.fastai  
    env: conda.yaml  
    loader_module: mlflow.fastai  
    python_version: 3.8.12
```

Signatures

Model signatures in MLflow are an important part of the model specification, as they serve as a data contract between the model and the server running our models. They are also important for parsing and enforcing model's input's types at deployment time. [MLflow enforces types when data is submitted to your model if a signature is available.](#)

Signatures are indicated when the model gets logged and persisted in the `MLmodel` file, in the `signature` section. Autolog's feature in MLflow automatically infers signatures in a best effort way. However, it may be required to [log the models manually if the signatures inferred are not the ones you need.](#)

There are two types of signatures:

- **Column-based signature** corresponding to signatures that operate to tabular data. For models with this signature, MLflow supplies `pandas.DataFrame` objects as inputs.
- **Tensor-based signature:** corresponding to signatures that operate with n-dimensional arrays or tensors. For models with this signature, MLflow supplies `numpy.ndarray` as inputs (or a dictionary of `numpy.ndarray` in the case of named-tensors).

The following example corresponds to a computer vision model trained with `fastai`. This model receives a batch of images represented as tensors of shape `(300, 300, 3)` with the RGB representation of them (unsigned integers). It outputs batches of predictions (probabilities) for two classes.

MLmodel

```
signature:  
  inputs: '[{"type": "tensor",  
            "tensor-spec":  
              {"dtype": "uint8", "shape": [-1, 300, 300, 3]}  
          }]'  
  outputs: '[{"type": "tensor",  
             "tensor-spec":  
               {"dtype": "float32", "shape": [-1, 2]}  
           }]'
```

TIP

Azure Machine Learning generates Swagger for model's deployment in MLflow format with a signature available. This makes easier to test deployed endpoints using the Azure ML studio.

Model's environment

Requirements for the model to run are specified in the `conda.yaml` file. Dependencies can be automatically detected by MLflow or they can be manually indicated when you call `mlflow.<flavor>.log_model()` method. The latter can be needed in cases that the libraries included in your environment are not the ones you intended to use.

The following is an example of an environment used for a model created with `fastai` framework:

`conda.yaml`

```
channels:
- conda-forge
dependencies:
- python=3.8.5
- pip
- pip:
  - mlflow
  - astunparse==1.6.3
  - cffi==1.15.0
  - configparser==3.7.4
  - defusedxml==0.7.1
  - fastai==2.4.1
  - google-api-core==2.7.1
  - ipython==8.2.0
  - psutil==5.9.0
name: mlflow-env
```

NOTE

MLflow environments and Azure Machine Learning environments are different concepts. While the former operates at the level of the model, the latter operates at the level of the workspace (for registered environments) or jobs/deployments (for anonymous environments). When you deploy MLflow models in Azure Machine Learning, the model's environment is built and used for deployment. Alternatively, you can override this behaviour with the [Azure ML CLI v2](#) and deploy MLflow models using a specific Azure Machine Learning environments.

Model's predict function

All MLflow models contain a `predict` function. **This function is the one that is called when a model is deployed using a no-code-deployment experience.** What the `predict` function returns (classes, probabilities, a forecast, etc.) depend on the framework (i.e. flavor) used for training. Read the documentation of each flavor to know what they return.

In some cases, you may need to customize this function to change the way inference is executed. On those cases, you will need to [log models with a different behavior in the predict method](#) or [log a custom model's flavor](#).

Loading MLflow models back

Models created as MLflow models can be loaded back directly from the run where they were logged, from the file system where they are saved or from the model registry where they are registered. MLflow provides a consistent way to load those models regardless of the location.

There are two workflows available for loading models:

- **Loading back the same object and types that were logged:** You can load models using MLflow SDK and obtain an instance of the model with types belonging to the training library. For instance, an ONNX model will return a `ModelProto` while a decision tree trained with Scikit-Learn model will return a `DecisionTreeClassifier` object. Use `mlflow.<flavor>.load_model()` to do so.
- **Loading back a model for running inference:** You can load models using MLflow SDK and obtain a wrapper where MLflow warranties there will be a `predict` function. It doesn't matter which flavor you are using, every MLflow model needs to implement this contract. Furthermore, MLflow warranties that this function can be called using arguments of type `pandas.DataFrame`, `numpy.ndarray` or `dict[string, numpyndarray]` (depending on the signature of the model). MLflow handles the type conversion to the input type the model actually expects. Use `mlflow.pyfunc.load_model()` to do so.

Start logging models

We recommend starting taking advantage of MLflow models in Azure Machine Learning. There are different ways to start using the model's concept with MLflow. Read [How to log MLFlow models](#) to a comprehensive guide.

What is automated machine learning (AutoML)?

9/22/2022 • 15 minutes to read • [Edit Online](#)

Automated machine learning, also referred to as automated ML or AutoML, is the process of automating the time-consuming, iterative tasks of machine learning model development. It allows data scientists, analysts, and developers to build ML models with high scale, efficiency, and productivity all while sustaining model quality. Automated ML in Azure Machine Learning is based on a breakthrough from our [Microsoft Research division](#).

Traditional machine learning model development is resource-intensive, requiring significant domain knowledge and time to produce and compare dozens of models. With automated machine learning, you'll accelerate the time it takes to get production-ready ML models with great ease and efficiency.

Ways to use AutoML in Azure Machine Learning

Azure Machine Learning offers the following two experiences for working with automated ML. See the following sections to understand feature availability in each experience.

- For code-experienced customers, [Azure Machine Learning Python SDK](#). Get started with [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#)
- For limited/no-code experience customers, Azure Machine Learning studio at <https://ml.azure.com>. Get started with these tutorials:
 - [Tutorial: Create a classification model with automated ML in Azure Machine Learning](#).
 - [Tutorial: Forecast demand with automated machine learning](#)

Experiment settings

The following settings allow you to configure your automated ML experiment.

	THE PYTHON SDK	THE STUDIO WEB EXPERIENCE
Split data into train/validation sets	✓	✓
Supports ML tasks: classification, regression, & forecasting	✓	✓
Supports computer vision tasks (preview): image classification, object detection & instance segmentation	✓	
NLP-Text	✓	✓
Optimizes based on primary metric	✓	✓
Supports Azure ML compute as compute target	✓	✓
Configure forecast horizon, target lags & rolling window	✓	✓

	THE PYTHON SDK	THE STUDIO WEB EXPERIENCE
Set exit criteria	✓	✓
Set concurrent iterations	✓	✓
Block algorithms	✓	✓
Cross validation	✓	✓
Supports training on Azure Databricks clusters	✓	
View engineered feature names	✓	
Featurization summary	✓	
Featurization for holidays	✓	
Log file verbosity levels	✓	

Model settings

These settings can be applied to the best model as a result of your automated ML experiment.

	THE PYTHON SDK	THE STUDIO WEB EXPERIENCE
Best model registration, deployment, explainability	✓	✓
Enable voting ensemble & stack ensemble models	✓	✓
Show best model based on non-primary metric	✓	
Enable/disable ONNX model compatibility	✓	
Test the model	✓	✓ (preview)

Job control settings

These settings allow you to review and control your experiment jobs and its child jobs.

	THE PYTHON SDK	THE STUDIO WEB EXPERIENCE
Job summary table	✓	✓
Cancel jobs & child jobs	✓	✓
Get guardrails	✓	✓

When to use AutoML: classification, regression, forecasting, computer

vision & NLP

Apply automated ML when you want Azure Machine Learning to train and tune a model for you using the target metric you specify. Automated ML democratizes the machine learning model development process, and empowers its users, no matter their data science expertise, to identify an end-to-end machine learning pipeline for any problem.

ML professionals and developers across industries can use automated ML to:

- Implement ML solutions without extensive programming knowledge
- Save time and resources
- Leverage data science best practices
- Provide agile problem-solving

Classification

Classification is a common machine learning task. Classification is a type of supervised learning in which models learn using training data, and apply those learnings to new data. Azure Machine Learning offers featurizations specifically for these tasks, such as deep neural network text featurizers for classification. Learn more about [featurization options](#).

The main goal of classification models is to predict which categories new data will fall into based on learnings from its training data. Common classification examples include fraud detection, handwriting recognition, and object detection. Learn more and see an example at [Create a classification model with automated ML](#).

See examples of classification and automated machine learning in these Python notebooks: [Fraud Detection](#), [Marketing Prediction](#), and [Newsgroup Data Classification](#)

Regression

Similar to classification, regression tasks are also a common supervised learning task. Azure Machine Learning offers [featurizations specifically for these tasks](#).

Different from classification where predicted output values are categorical, regression models predict numerical output values based on independent predictors. In regression, the objective is to help establish the relationship among those independent predictor variables by estimating how one variable impacts the others. For example, automobile price based on features like, gas mileage, safety rating, etc. Learn more and see an example of [regression with automated machine learning](#).

See examples of regression and automated machine learning for predictions in these Python notebooks: [CPU Performance Prediction](#),

Time-series forecasting

Building forecasts is an integral part of any business, whether it's revenue, inventory, sales, or customer demand. You can use automated ML to combine techniques and approaches and get a recommended, high-quality time-series forecast. Learn more with this how-to: [automated machine learning for time series forecasting](#).

An automated time-series experiment is treated as a multivariate regression problem. Past time-series values are "pivoted" to become additional dimensions for the regressor together with other predictors. This approach, unlike classical time series methods, has an advantage of naturally incorporating multiple contextual variables and their relationship to one another during training. Automated ML learns a single, but often internally branched model for all items in the dataset and prediction horizons. More data is thus available to estimate model parameters and generalization to unseen series becomes possible.

Advanced forecasting configuration includes:

- holiday detection and featurization

- time-series and DNN learners (Auto-ARIMA, Prophet, ForecastTCN)
- many models support through grouping
- rolling-origin cross validation
- configurable lags
- rolling window aggregate features

See examples of regression and automated machine learning for predictions in these Python notebooks: [Sales Forecasting](#), [Demand Forecasting](#), and [Forecasting GitHub's Daily Active Users](#).

Computer vision (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Support for computer vision tasks allows you to easily generate models trained on image data for scenarios like image classification and object detection.

With this capability you can:

- Seamlessly integrate with the [Azure Machine Learning data labeling](#) capability
- Use labeled data for generating image models
- Optimize model performance by specifying the model algorithm and tuning the hyperparameters.
- Download or deploy the resulting model as a web service in Azure Machine Learning.
- Operationalize at scale, leveraging Azure Machine Learning [MLOps](#) and [ML Pipelines](#) capabilities.

Authoring AutoML models for vision tasks is supported via the Azure ML Python SDK. The resulting experimentation jobs, models, and outputs can be accessed from the Azure Machine Learning studio UI.

Learn how to [set up AutoML training for computer vision models](#).

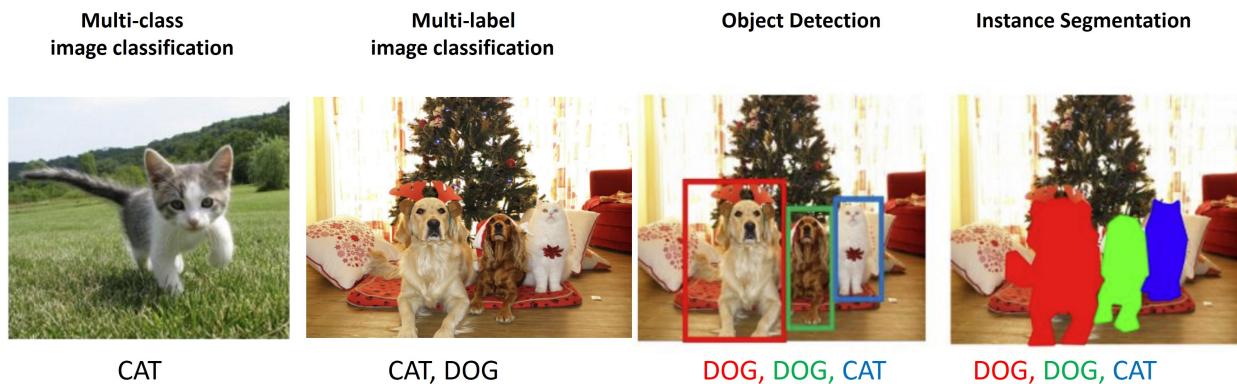


Image from: http://cs231n.stanford.edu/slides/2021/lecture_15.pdf

Automated ML for images supports the following computer vision tasks:

TASK	DESCRIPTION
Multi-class image classification	Tasks where an image is classified with only a single label from a set of classes - e.g. each image is classified as either an image of a 'cat' or a 'dog' or a 'duck'

TASK	DESCRIPTION
Multi-label image classification	Tasks where an image could have one or more labels from a set of labels - e.g. an image could be labeled with both 'cat' and 'dog'
Object detection	Tasks to identify objects in an image and locate each object with a bounding box e.g. locate all dogs and cats in an image and draw a bounding box around each.
Instance segmentation	Tasks to identify objects in an image at the pixel level, drawing a polygon around each object in the image.

Natural language processing: NLP (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Support for natural language processing (NLP) tasks in automated ML allows you to easily generate models trained on text data for text classification and named entity recognition scenarios. Authoring automated ML trained NLP models is supported via the Azure Machine Learning Python SDK. The resulting experimentation jobs, models, and outputs can be accessed from the Azure Machine Learning studio UI.

The NLP capability supports:

- End-to-end deep neural network NLP training with the latest pre-trained BERT models
- Seamless integration with [Azure Machine Learning data labeling](#)
- Use labeled data for generating NLP models
- Multi-lingual support with 104 languages
- Distributed training with Horovod

Learn how to [set up AutoML training for NLP models](#).

How automated ML works

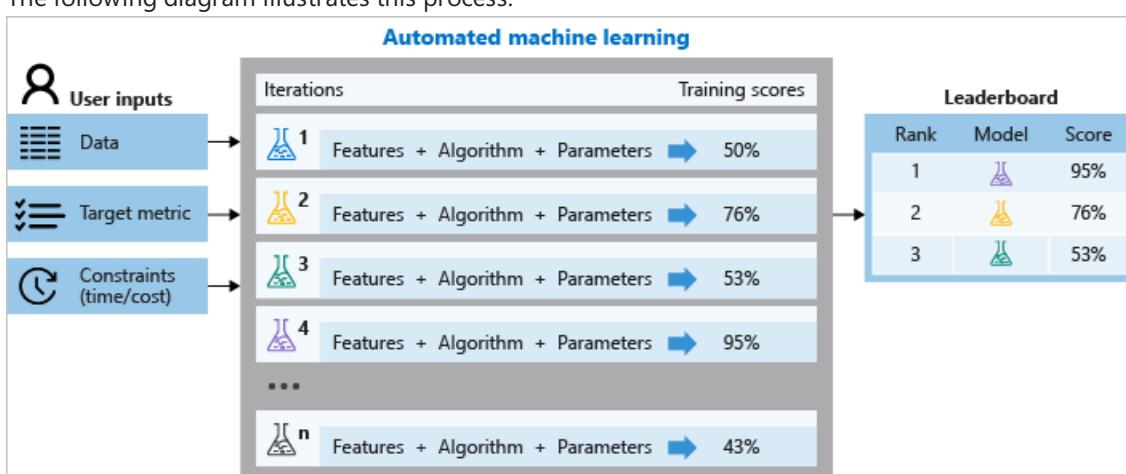
During training, Azure Machine Learning creates a number of pipelines in parallel that try different algorithms and parameters for you. The service iterates through ML algorithms paired with feature selections, where each iteration produces a model with a training score. The higher the score, the better the model is considered to "fit" your data. It will stop once it hits the exit criteria defined in the experiment.

Using [Azure Machine Learning](#), you can design and run your automated ML training experiments with these steps:

1. **Identify the ML problem** to be solved: classification, forecasting, regression or computer vision (preview).
2. **Choose whether you want to use the Python SDK or the studio web experience:** Learn about the parity between the [Python SDK and studio web experience](#).
 - For limited or no code experience, try the Azure Machine Learning studio web experience at <https://ml.azure.com>
 - For Python developers, check out the [Azure Machine Learning Python SDK](#)

3. Specify the source and format of the labeled training data: Numpy arrays or Pandas dataframe
4. Configure the automated machine learning parameters that determine how many iterations over different models, hyperparameter settings, advanced preprocessing/featurization, and what metrics to look at when determining the best model.
5. Submit the training job.
6. Review the results

The following diagram illustrates this process.



You can also inspect the logged job information, which [contains metrics](#) gathered during the job. The training job produces a Python serialized object (`.pk1` file) that contains the model and data preprocessing.

While model building is automated, you can also [learn how important or relevant features are](#) to the generated models.

Guidance on local vs. remote managed ML compute targets

The web interface for automated ML always uses a remote [compute target](#). But when you use the Python SDK, you will choose either a local compute or a remote compute target for automated ML training.

- **Local compute:** Training occurs on your local laptop or VM compute.
- **Remote compute:** Training occurs on Machine Learning compute clusters.

Choose compute target

Consider these factors when choosing your compute target:

- **Choose a local compute:** If your scenario is about initial explorations or demos using small data and short trains (i.e. seconds or a couple of minutes per child job), training on your local computer might be a better choice. There is no setup time, the infrastructure resources (your PC or VM) are directly available.
- **Choose a remote ML compute cluster:** If you are training with larger datasets like in production training creating models which need longer trains, remote compute will provide much better end-to-end time performance because [AutoML](#) will parallelize trains across the cluster's nodes. On a remote compute, the start-up time for the internal infrastructure will add around 1.5 minutes per child job, plus additional minutes for the cluster infrastructure if the VMs are not yet up and running.

Pros and cons

Consider these pros and cons when choosing to use local vs. remote.

	PROS (ADVANTAGES)	CONS (HANDICAPS)
Local compute target	<ul style="list-style-type: none"> No environment start-up time 	<ul style="list-style-type: none"> Subset of features Can't parallelize jobs Worse for large data. No data streaming while training No DNN-based featurization Python SDK only
Remote ML compute clusters	<ul style="list-style-type: none"> Full set of features Parallelize child jobs Large data support DNN-based featurization Dynamic scalability of compute cluster on demand No-code experience (web UI) also available 	<ul style="list-style-type: none"> Start-up time for cluster nodes Start-up time for each child job

Feature availability

More features are available when you use the remote compute, as shown in the table below.

FEATURE	REMOTE	LOCAL
Data streaming (Large data support, up to 100 GB)	✓	
DNN-BERT-based text featurization and training	✓	
Out-of-the-box GPU support (training and inference)	✓	
Image Classification and Labeling support	✓	
Auto-ARIMA, Prophet and ForecastTCN models for forecasting	✓	
Multiple jobs/iterations in parallel	✓	
Create models with interpretability in AutoML studio web experience UI	✓	
Feature engineering customization in studio web experience UI	✓	
Azure ML hyperparameter tuning	✓	
Azure ML Pipeline workflow support	✓	
Continue a job	✓	
Forecasting	✓	✓
Create and run experiments in notebooks	✓	✓

FEATURE	REMOTE	LOCAL
Register and visualize experiment's info and metrics in UI	✓	✓
Data guardrails	✓	✓

Training, validation and test data

With automated ML you provide the **training data** to train ML models, and you can specify what type of model validation to perform. Automated ML performs model validation as part of training. That is, automated ML uses **validation data** to tune model hyperparameters based on the applied algorithm to find the best combination that best fits the training data. However, the same validation data is used for each iteration of tuning, which introduces model evaluation bias since the model continues to improve and fit to the validation data.

To help confirm that such bias isn't applied to the final recommended model, automated ML supports the use of **test data** to evaluate the final model that automated ML recommends at the end of your experiment. When you provide test data as part of your AutoML experiment configuration, this recommended model is tested by default at the end of your experiment (preview).

IMPORTANT

Testing your models with a test dataset to evaluate generated models is a preview feature. This capability is an [experimental](#) preview feature, and may change at any time.

Learn how to [configure AutoML experiments to use test data \(preview\)](#) with the [SDK](#) or with the [Azure Machine Learning studio](#).

You can also [test any existing automated ML model \(preview\)](#), including models from child jobs, by providing your own test data or by setting aside a portion of your training data.

Feature engineering

Feature engineering is the process of using domain knowledge of the data to create features that help ML algorithms learn better. In Azure Machine Learning, scaling and normalization techniques are applied to facilitate feature engineering. Collectively, these techniques and feature engineering are referred to as featurization.

For automated machine learning experiments, featurization is applied automatically, but can also be customized based on your data. [Learn more about what featurization is included](#) and how AutoML helps [prevent over-fitting and imbalanced data](#) in your models.

NOTE

Automated machine learning featurization steps (feature normalization, handling missing data, converting text to numeric, etc.) become part of the underlying model. When using the model for predictions, the same featurization steps applied during training are applied to your input data automatically.

Customize featurization

Additional feature engineering techniques such as, encoding and transforms are also available.

Enable this setting with:

- Azure Machine Learning studio: Enable **Automatic featurization** in the [View additional configuration](#) section with these steps.

- Python SDK: Specify `"featurization": "auto" / "off" / FeaturizationConfig'` in your [AutoMLConfig](#) object. Learn more about [enabling featurization](#).

Ensemble models

Automated machine learning supports ensemble models, which are enabled by default. Ensemble learning improves machine learning results and predictive performance by combining multiple models as opposed to using single models. The ensemble iterations appear as the final iterations of your job. Automated machine learning uses both voting and stacking ensemble methods for combining models:

- **Voting**: predicts based on the weighted average of predicted class probabilities (for classification tasks) or predicted regression targets (for regression tasks).
- **Stacking**: stacking combines heterogenous models and trains a meta-model based on the output from the individual models. The current default meta-models are LogisticRegression for classification tasks and ElasticNet for regression/forecasting tasks.

The [Caruana ensemble selection algorithm](#) with sorted ensemble initialization is used to decide which models to use within the ensemble. At a high level, this algorithm initializes the ensemble with up to five models with the best individual scores, and verifies that these models are within 5% threshold of the best score to avoid a poor initial ensemble. Then for each ensemble iteration, a new model is added to the existing ensemble and the resulting score is calculated. If a new model improved the existing ensemble score, the ensemble is updated to include the new model.

See the [how-to](#) for changing default ensemble settings in automated machine learning.

AutoML & ONNX

With Azure Machine Learning, you can use automated ML to build a Python model and have it converted to the ONNX format. Once the models are in the ONNX format, they can be run on a variety of platforms and devices. Learn more about [accelerating ML models with ONNX](#).

See how to convert to ONNX format in [this Jupyter notebook example](#). Learn which [algorithms are supported in ONNX](#).

The ONNX runtime also supports C#, so you can use the model built automatically in your C# apps without any need for recoding or any of the network latencies that REST endpoints introduce. Learn more about [using an AutoML ONNX model in a .NET application with ML.NET](#) and [inferencing ONNX models with the ONNX runtime C# API](#).

Next steps

There are multiple resources to get you up and running with AutoML.

Tutorials/ how-tos

Tutorials are end-to-end introductory examples of AutoML scenarios.

- **For a code first experience**, follow the [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#)
- **For a low or no-code experience**, see the [Tutorial: Train a classification model with no-code AutoML in Azure Machine Learning studio](#).
- **For using AutoML to train computer vision models**, see the [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#).

How-to articles provide additional detail into what functionality automated ML offers. For example,

- Configure the settings for automatic training experiments
 - Without code in the Azure Machine Learning studio.
 - With the Python SDK.
- Learn how to train forecasting models with time series data.
- Learn how to train computer vision models with Python.
- Learn how to [view the generated code from your automated ML models](#).

Jupyter notebook samples

Review detailed code examples and use cases in the [GitHub notebook repository for automated machine learning samples](#).

Python SDK reference

Deepen your expertise of SDK design patterns and class specifications with the [AutoML class reference documentation](#).

NOTE

Automated machine learning capabilities are also available in other Microsoft solutions such as, [ML.NET](#), [HDInsight](#), [Power BI](#) and [SQL Server](#)

Prevent overfitting and imbalanced data with automated machine learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

Overfitting and imbalanced data are common pitfalls when you build machine learning models. By default, Azure Machine Learning's automated machine learning provides charts and metrics to help you identify these risks, and implements best practices to help mitigate them.

Identify overfitting

Overfitting in machine learning occurs when a model fits the training data too well, and as a result can't accurately predict on unseen test data. In other words, the model has simply memorized specific patterns and noise in the training data, but is not flexible enough to make predictions on real data.

Consider the following trained models and their corresponding train and test accuracies.

MODEL	TRAIN ACCURACY	TEST ACCURACY
A	99.9%	95%
B	87%	87%
C	99.9%	45%

Considering model A, there is a common misconception that if test accuracy on unseen data is lower than training accuracy, the model is overfitted. However, test accuracy should always be less than training accuracy, and the distinction for overfit vs. appropriately fit comes down to *how much* less accurate.

When comparing models A and B, model A is a better model because it has higher test accuracy, and although the test accuracy is slightly lower at 95%, it is not a significant difference that suggests overfitting is present. You wouldn't choose model B simply because the train and test accuracies are closer together.

Model C represents a clear case of overfitting; the training accuracy is very high but the test accuracy isn't anywhere near as high. This distinction is subjective, but comes from knowledge of your problem and data, and what magnitudes of error are acceptable.

Prevent overfitting

In the most egregious cases, an overfitted model assumes that the feature value combinations seen during training will always result in the exact same output for the target.

The best way to prevent overfitting is to follow ML best-practices including:

- Using more training data, and eliminating statistical bias
- Preventing target leakage
- Using fewer features
- **Regularization and hyperparameter optimization**
- **Model complexity limitations**
- **Cross-validation**

In the context of automated ML, the first three items above are **best-practices you implement**. The last three bolded items are **best-practices automated ML implements** by default to protect against overfitting. In settings other than automated ML, all six best-practices are worth following to avoid overfitting models.

Best practices you implement

Use more data

Using **more data** is the simplest and best possible way to prevent overfitting, and as an added bonus typically increases accuracy. When you use more data, it becomes harder for the model to memorize exact patterns, and it is forced to reach solutions that are more flexible to accommodate more conditions. It's also important to recognize **statistical bias**, to ensure your training data doesn't include isolated patterns that won't exist in live-prediction data. This scenario can be difficult to solve, because there may not be overfitting between your train and test sets, but there may be overfitting present when compared to live test data.

Prevent target leakage

Target leakage is a similar issue, where you may not see overfitting between train/test sets, but rather it appears at prediction-time. Target leakage occurs when your model "cheats" during training by having access to data that it shouldn't normally have at prediction-time. For example, if your problem is to predict on Monday what a commodity price will be on Friday, but one of your features accidentally included data from Thursdays, that would be data the model won't have at prediction-time since it cannot see into the future. Target leakage is an easy mistake to miss, but is often characterized by abnormally high accuracy for your problem. If you are attempting to predict stock price and trained a model at 95% accuracy, there is likely target leakage somewhere in your features.

Use fewer features

Removing features can also help with overfitting by preventing the model from having too many fields to use to memorize specific patterns, thus causing it to be more flexible. It can be difficult to measure quantitatively, but if you can remove features and retain the same accuracy, you have likely made the model more flexible and have reduced the risk of overfitting.

Best practices automated ML implements

Regularization and hyperparameter tuning

Regularization is the process of minimizing a cost function to penalize complex and overfitted models. There are different types of regularization functions, but in general they all penalize model coefficient size, variance, and complexity. Automated ML uses L1 (Lasso), L2 (Ridge), and ElasticNet (L1 and L2 simultaneously) in different combinations with different model hyperparameter settings that control overfitting. In simple terms, automated ML will vary how much a model is regulated and choose the best result.

Model complexity limitations

Automated ML also implements explicit **model complexity limitations** to prevent overfitting. In most cases this implementation is specifically for decision tree or forest algorithms, where individual tree max-depth is limited, and the total number of trees used in forest or ensemble techniques are limited.

Cross-validation

Cross-validation (CV) is the process of taking many subsets of your full training data and training a model on each subset. The idea is that a model could get "lucky" and have great accuracy with one subset, but by using many subsets the model won't achieve this high accuracy every time. When doing CV, you provide a validation holdout dataset, specify your CV folds (number of subsets) and automated ML will train your model and tune hyperparameters to minimize error on your validation set. One CV fold could be overfitted, but by using many of them it reduces the probability that your final model is overfitted. The tradeoff is that CV does result in longer training times and thus greater cost, because instead of training a model once, you train it once for each n CV subsets.

NOTE

Cross-validation is not enabled by default; it must be configured in automated ML settings. However, after cross-validation is configured and a validation data set has been provided, the process is automated for you. Learn more about [cross validation configuration in Auto ML](#)

Identify models with imbalanced data

Imbalanced data is commonly found in data for machine learning classification scenarios, and refers to data that contains a disproportionate ratio of observations in each class. This imbalance can lead to a falsely perceived positive effect of a model's accuracy, because the input data has bias towards one class, which results in the trained model to mimic that bias.

In addition, automated ML jobs generate the following charts automatically, which can help you understand the correctness of the classifications of your model, and identify models potentially impacted by imbalanced data.

CHART	DESCRIPTION
Confusion Matrix	Evaluates the correctly classified labels against the actual labels of the data.
Precision-recall	Evaluates the ratio of correct labels against the ratio of found label instances of the data
ROC Curves	Evaluates the ratio of correct labels against the ratio of false-positive labels.

Handle imbalanced data

As part of its goal of simplifying the machine learning workflow, **automated ML has built in capabilities** to help deal with imbalanced data such as,

- A **weight column**: automated ML supports a column of weights as input, causing rows in the data to be weighted up or down, which can be used to make a class more or less "important".
- The algorithms used by automated ML detect imbalance when the number of samples in the minority class is equal to or fewer than 20% of the number of samples in the majority class, where minority class refers to the one with fewest samples and majority class refers to the one with most samples. Subsequently, AutoML will run an experiment with sub-sampled data to check if using class weights would remedy this problem and improve performance. If it ascertains a better performance through this experiment, then this remedy is applied.
- Use a performance metric that deals better with imbalanced data. For example, the AUC_weighted is a primary metric that calculates the contribution of every class based on the relative number of samples representing that class, hence is more robust against imbalance.

The following techniques are additional options to handle imbalanced data **outside of automated ML**.

- Resampling to even the class imbalance, either by up-sampling the smaller classes or down-sampling the larger classes. These methods require expertise to process and analyze.
- Review performance metrics for imbalanced data. For example, the F1 score is the harmonic mean of precision and recall. Precision measures a classifier's exactness, where higher precision indicates fewer false positives, while recall measures a classifier's completeness, where higher recall indicates fewer false

negatives.

Next steps

See examples and learn how to build models using automated machine learning:

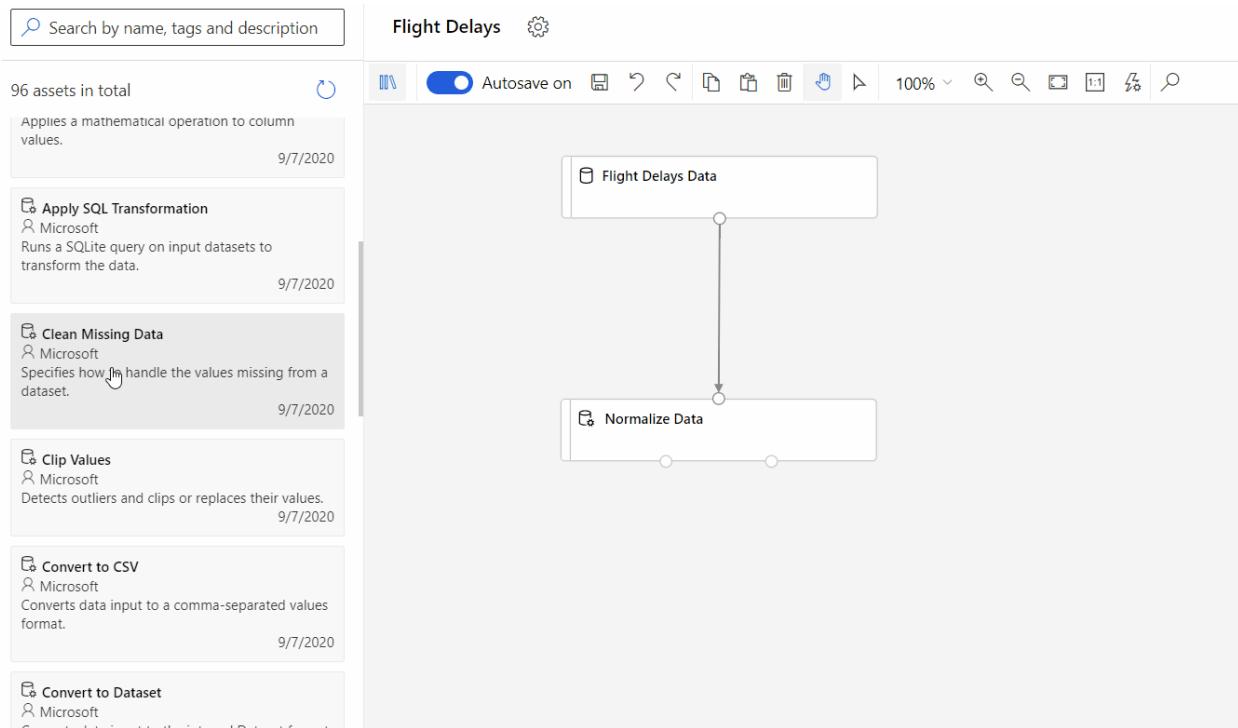
- Follow the [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#).
- Configure the settings for automatic training experiment:
 - In Azure Machine Learning studio, [use these steps](#).
 - With the Python SDK, [use these steps](#).

What is Azure Machine Learning designer?

9/22/2022 • 4 minutes to read • [Edit Online](#)

Azure Machine Learning designer is a drag-and-drop interface used to train and deploy models in Azure Machine Learning. This article describes the tasks you can do in the designer.

- To get started with the designer, see [Tutorial: Train a no-code regression model](#).
- To learn about the components available in the designer, see the [Algorithm and component reference](#).



The designer uses your Azure Machine Learning [workspace](#) to organize shared resources such as:

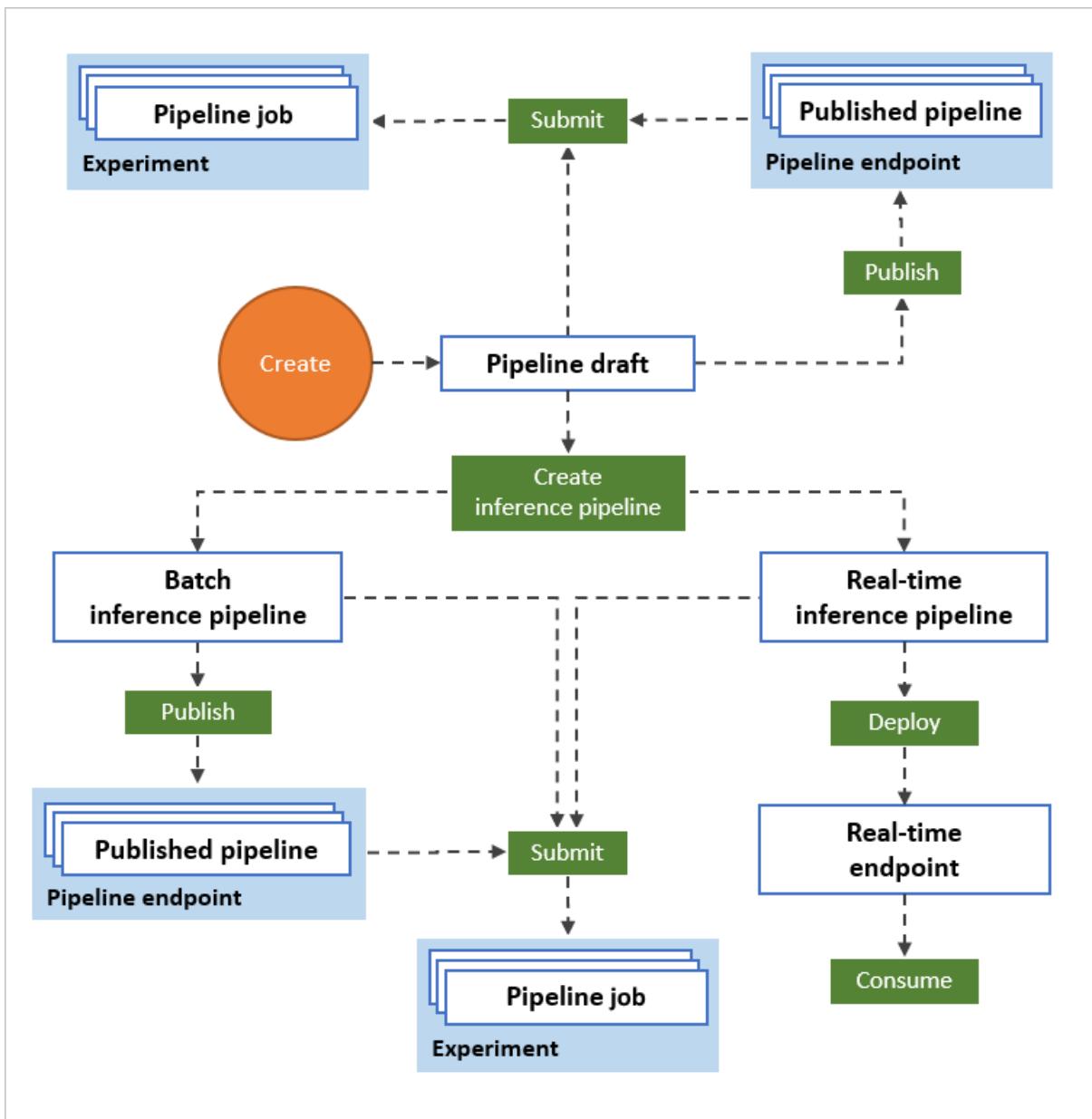
- [Pipelines](#)
- [Data](#)
- [Compute resources](#)
- [Registered models](#)
- [Published pipelines](#)
- [Real-time endpoints](#)

Model training and deployment

Use a visual canvas to build an end-to-end machine learning workflow. Train, test, and deploy models all in the designer:

- Drag-and-drop [data assets](#) and [components](#) onto the canvas.
- Connect the components to create a [pipeline draft](#).
- Submit a [pipeline run](#) using the compute resources in your Azure Machine Learning workspace.
- Convert your [training pipelines](#) to [inference pipelines](#).
- [Publish](#) your pipelines to a REST [pipeline endpoint](#) to submit a new pipeline that runs with different parameters and data assets.
 - Publish a [training pipeline](#) to reuse a single pipeline to train multiple models while changing

- parameters and data assets.
- Publish a **batch inference pipeline** to make predictions on new data by using a previously trained model.
 - [Deploy](#) a **real-time inference pipeline** to an online endpoint to make predictions on new data in real time.



Pipeline

A [pipeline](#) consists of data assets and analytical components, which you connect. Pipelines have many uses: you can make a pipeline that trains a single model, or one that trains multiple models. You can create a pipeline that makes predictions in real time or in batch, or make a pipeline that only cleans data. Pipelines let you reuse your work and organize your projects.

Pipeline draft

As you edit a pipeline in the designer, your progress is saved as a **pipeline draft**. You can edit a pipeline draft at any point by adding or removing components, configuring compute targets, creating parameters, and so on.

A valid pipeline has these characteristics:

- Data assets can only connect to components.
- Components can only connect to either data assets or other components.
- All input ports for components must have some connection to the data flow.

- All required parameters for each component must be set.

When you're ready to run your pipeline draft, you submit a pipeline job.

Pipeline job

Each time you run a pipeline, the configuration of the pipeline and its results are stored in your workspace as a **pipeline job**. You can go back to any pipeline job to inspect it for troubleshooting or auditing. **Clone** a pipeline job to create a new pipeline draft for you to edit.

Pipeline jobs are grouped into [experiments](#) to organize job history. You can set the experiment for every pipeline job.

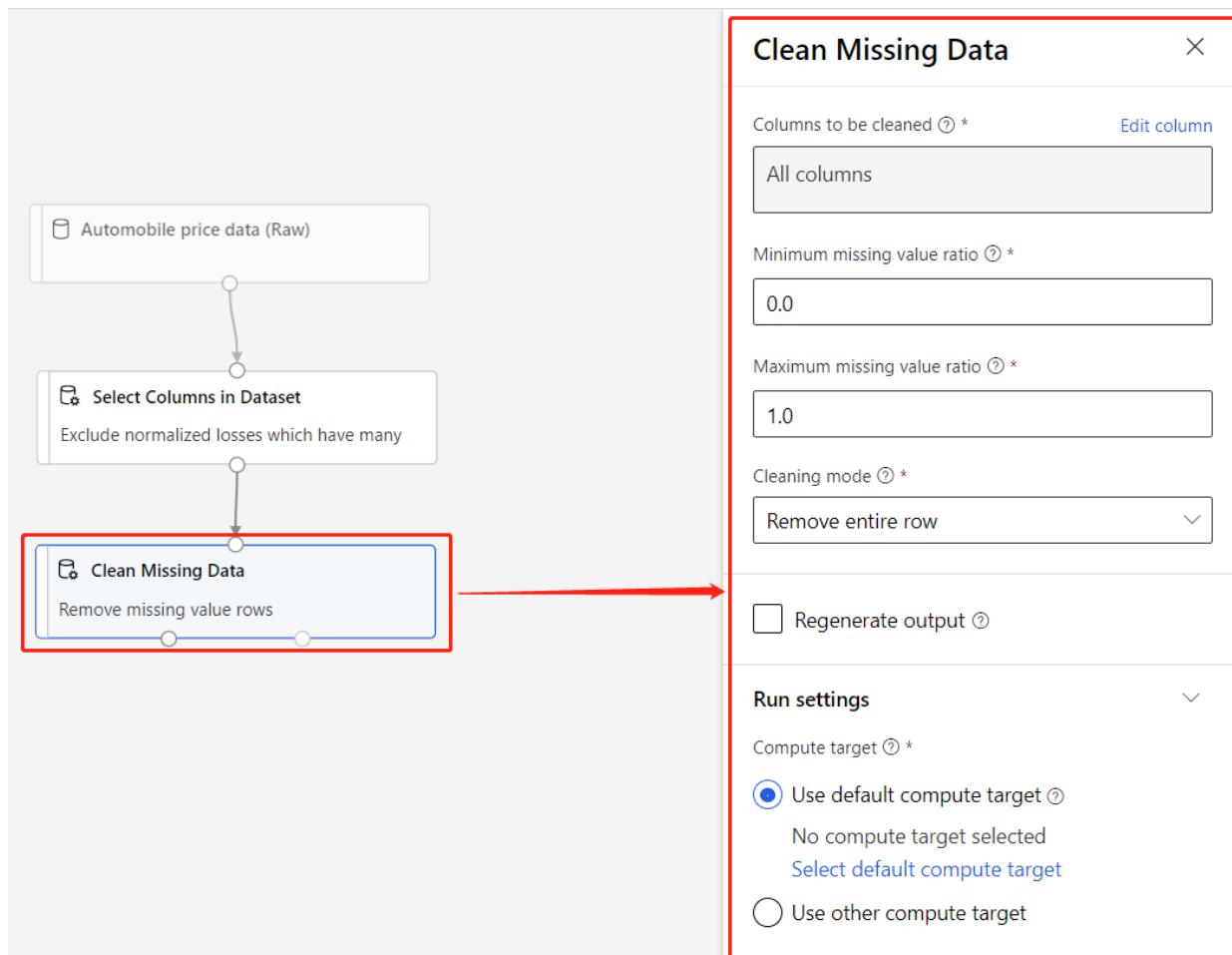
Data

A machine learning data asset makes it easy to access and work with your data. Several [sample data assets](#) are included in the designer for you to experiment with. You can [register](#) more data assets as you need them.

Component

A component is an algorithm that you can perform on your data. The designer has several components ranging from data ingress functions to training, scoring, and validation processes.

A component may have a set of parameters that you can use to configure the component's internal algorithms. When you select a component on the canvas, the component's parameters are displayed in the Properties pane to the right of the canvas. You can modify the parameters in that pane to tune your model. You can set the compute resources for individual components in the designer.



For some help navigating through the library of machine learning algorithms available, see [Algorithm & component reference overview](#). For help with choosing an algorithm, see the [Azure Machine Learning](#)

Compute resources

Use compute resources from your workspace to run your pipeline and host your deployed models as online endpoints or pipeline endpoints (for batch inference). The supported compute targets are:

COMPUTE TARGET	TRAINING	DEPLOYMENT
Azure Machine Learning compute	✓	
Azure Kubernetes Service		✓

Compute targets are attached to your [Azure Machine Learning workspace](#). You manage your compute targets in your workspace in the [Azure Machine Learning studio](#).

Deploy

To perform real-time inferencing, you must deploy a pipeline as a [online endpoint](#). The online endpoint creates an interface between an external application and your scoring model. A call to an online endpoint returns prediction results to the application in real time. To make a call to an online endpoint, you pass the API key that was created when you deployed the endpoint. The endpoint is based on REST, a popular architecture choice for web programming projects.

Online endpoints must be deployed to an Azure Kubernetes Service cluster.

To learn how to deploy your model, see [Tutorial: Deploy a machine learning model with the designer](#).

NOTE

Azure Machine Learning Endpoints (preview) provide an improved, simpler deployment experience. Endpoints support both real-time and batch inference scenarios. Endpoints provide a unified interface to invoke and manage model deployments across compute types. See [What are Azure Machine Learning endpoints \(preview\)?](#).

Publish

You can also publish a pipeline to a [pipeline endpoint](#). Similar to an online endpoint, a pipeline endpoint lets you submit new pipeline jobs from external applications using REST calls. However, you cannot send or receive data in real time using a pipeline endpoint.

Published pipelines are flexible, they can be used to train or retrain models, [perform batch inferencing](#), process new data, and much more. You can publish multiple pipelines to a single pipeline endpoint and specify which pipeline version to run.

A published pipeline runs on the compute resources you define in the pipeline draft for each component.

The designer creates the same [PublishedPipeline](#) object as the SDK.

Next steps

- Learn the fundamentals of predictive analytics and machine learning with [Tutorial: Predict automobile price with the designer](#)
- Learn how to modify existing [designer samples](#) to adapt them to your needs.

Machine Learning Algorithm Cheat Sheet for Azure Machine Learning designer

9/22/2022 • 2 minutes to read • [Edit Online](#)

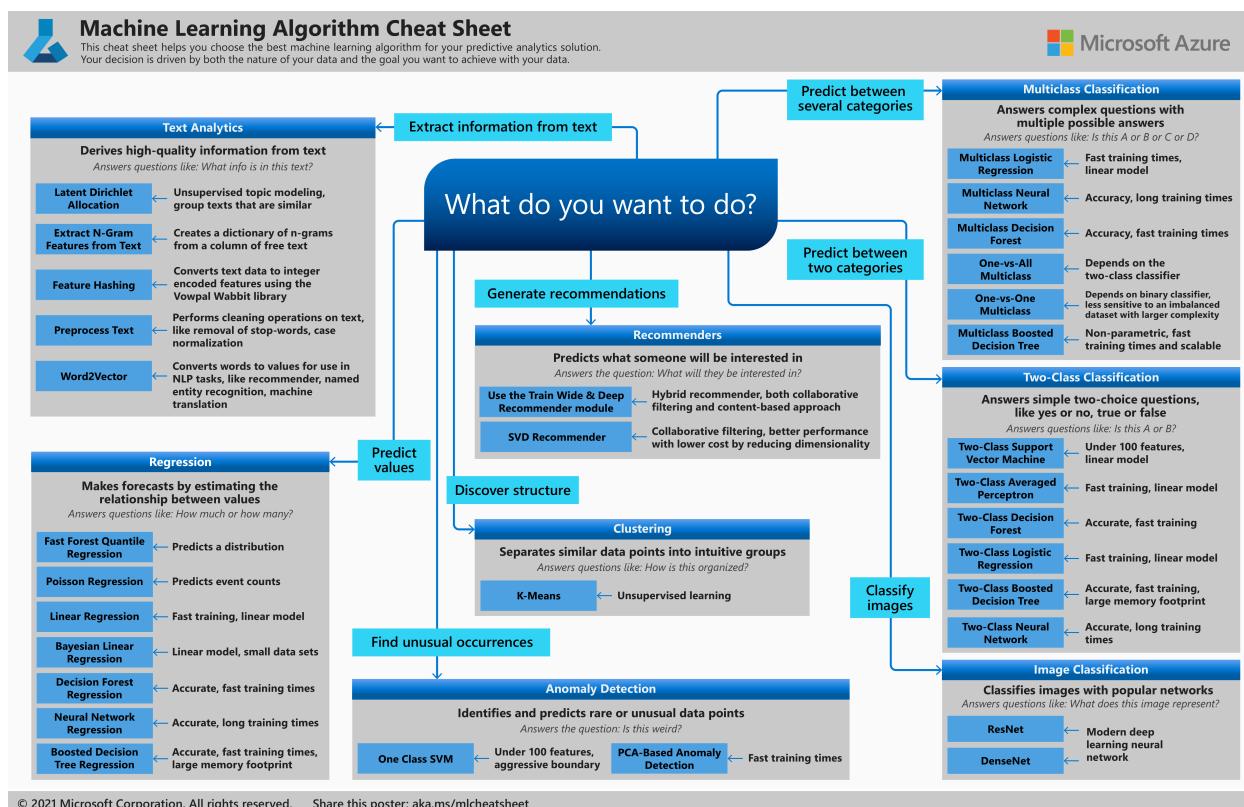
The Azure Machine Learning Algorithm Cheat Sheet helps you choose the right algorithm from the designer for a predictive analytics model.

Azure Machine Learning has a large library of algorithms from the *classification*, *recommender systems*, *clustering*, *anomaly detection*, *regression*, and *text analytics* families. Each is designed to address a different type of machine learning problem.

For more information, see [How to select algorithms](#).

Download: Machine Learning Algorithm Cheat Sheet

Download the cheat sheet here: [Machine Learning Algorithm Cheat Sheet \(11x17 in.\)](#)



Download and print the Machine Learning Algorithm Cheat Sheet in tabloid size to keep it handy and get help choosing an algorithm.

How to use the Machine Learning Algorithm Cheat Sheet

The suggestions offered in this algorithm cheat sheet are approximate rules-of-thumb. Some can be bent, and some can be flagrantly violated. This cheat sheet is intended to suggest a starting point. Don't be afraid to run a head-to-head competition between several algorithms on your data. There is simply no substitute for understanding the principles of each algorithm and the system that generated your data.

Every machine learning algorithm has its own style or inductive bias. For a specific problem, several algorithms may be appropriate, and one algorithm may be a better fit than others. But it's not always possible to know

beforehand, which is the best fit. In cases like these, several algorithms are listed together in the cheat sheet. An appropriate strategy would be to try one algorithm, and if the results are not yet satisfactory, try the others.

To learn more about the algorithms in Azure Machine Learning designer, go to the [Algorithm and component reference](#).

Kinds of machine learning

There are three main categories of machine learning: *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

Supervised learning

In supervised learning, each data point is labeled or associated with a category or value of interest. An example of a categorical label is assigning an image as either a 'cat' or a 'dog'. An example of a value label is the sale price associated with a used car. The goal of supervised learning is to study many labeled examples like these, and then to be able to make predictions about future data points. For example, identifying new photos with the correct animal or assigning accurate sale prices to other used cars. This is a popular and useful type of machine learning.

Unsupervised learning

In unsupervised learning, data points have no labels associated with them. Instead, the goal of an unsupervised learning algorithm is to organize the data in some way or to describe its structure. Unsupervised learning groups data into clusters, as K-means does, or finds different ways of looking at complex data so that it appears simpler.

Reinforcement learning

In reinforcement learning, the algorithm gets to choose an action in response to each data point. It is a common approach in robotics, where the set of sensor readings at one point in time is a data point, and the algorithm must choose the robot's next action. It's also a natural fit for Internet of Things applications. The learning algorithm also receives a reward signal a short time later, indicating how good the decision was. Based on this signal, the algorithm modifies its strategy in order to achieve the highest reward.

Next steps

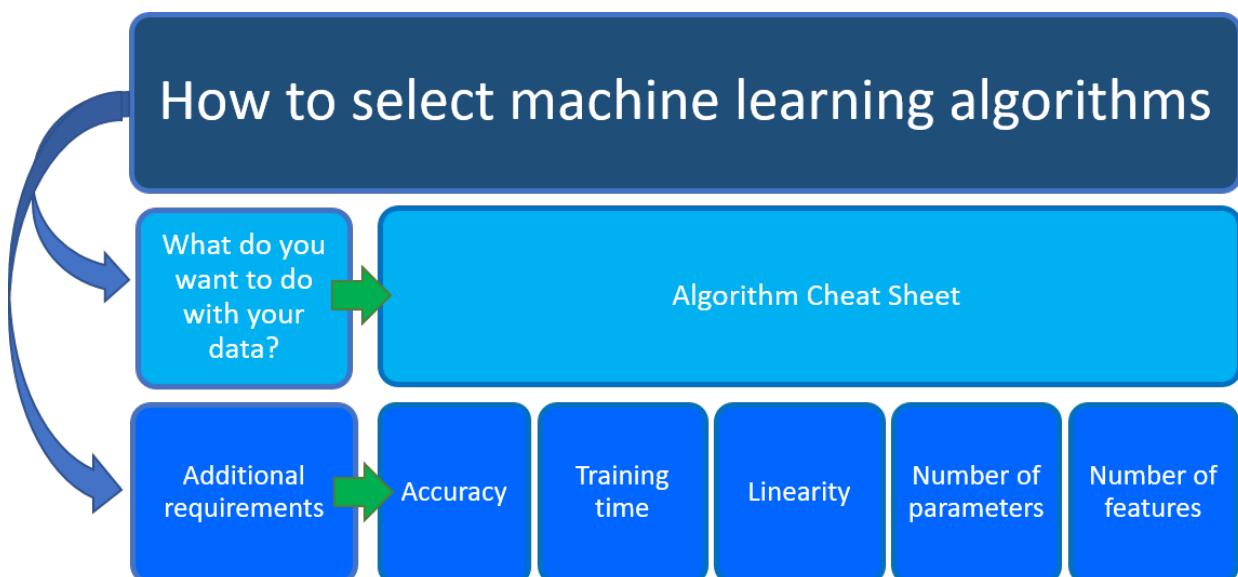
- See more information on [How to select algorithms](#)
- [Learn about studio in Azure Machine Learning and the Azure portal.](#)
- [Tutorial: Build a prediction model in Azure Machine Learning designer.](#)
- [Learn about deep learning vs. machine learning.](#)

How to select algorithms for Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

A common question is "Which machine learning algorithm should I use?" The algorithm you select depends primarily on two different aspects of your data science scenario:

- **What you want to do with your data?** Specifically, what is the business question you want to answer by learning from your past data?
- **What are the requirements of your data science scenario?** Specifically, what is the accuracy, training time, linearity, number of parameters, and number of features your solution supports?



Business scenarios and the Machine Learning Algorithm Cheat Sheet

The [Azure Machine Learning Algorithm Cheat Sheet](#) helps you with the first consideration: **What you want to do with your data?** On the Machine Learning Algorithm Cheat Sheet, look for task you want to do, and then find a [Azure Machine Learning designer](#) algorithm for the predictive analytics solution.

Machine Learning designer provides a comprehensive portfolio of algorithms, such as [Multiclass Decision Forest](#), [Recommendation systems](#), [Neural Network Regression](#), [Multiclass Neural Network](#), and [K-Means Clustering](#). Each algorithm is designed to address a different type of machine learning problem. See the [Machine Learning designer algorithm and component reference](#) for a complete list along with documentation about how each algorithm works and how to tune parameters to optimize the algorithm.

NOTE

Download the cheat sheet here: [Machine Learning Algorithm Cheat Sheet \(11x17 in.\)](#)

Along with guidance in the Azure Machine Learning Algorithm Cheat Sheet, keep in mind other requirements when choosing a machine learning algorithm for your solution. Following are additional factors to consider, such as the accuracy, training time, linearity, number of parameters and number of features.

Comparison of machine learning algorithms

Some learning algorithms make particular assumptions about the structure of the data or the desired results. If you can find one that fits your needs, it can give you more useful results, more accurate predictions, or faster training times.

The following table summarizes some of the most important characteristics of algorithms from the classification, regression, and clustering families:

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
Classification family					
Two-Class logistic regression	Good	Fast	Yes	4	
Two-class decision forest	Excellent	Moderate	No	5	Shows slower scoring times. Suggest not working with One-vs-All Multiclass, because of slower scoring times caused by tread locking in accumulating tree predictions
Two-class boosted decision tree	Excellent	Moderate	No	6	Large memory footprint
Two-class neural network	Good	Moderate	No	8	
Two-class averaged perceptron	Good	Moderate	Yes	4	
Two-class support vector machine	Good	Fast	Yes	5	Good for large feature sets
Multiclass logistic regression	Good	Fast	Yes	4	
Multiclass decision forest	Excellent	Moderate	No	5	Shows slower scoring times
Multiclass boosted decision tree	Excellent	Moderate	No	6	Tends to improve accuracy with some small risk of less coverage

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
Multiclass neural network	Good	Moderate	No	8	
One-vs-all multiclass	-	-	-	-	See properties of the two-class method selected
Regression family					
Linear regression	Good	Fast	Yes	4	
Decision forest regression	Excellent	Moderate	No	5	
Boosted decision tree regression	Excellent	Moderate	No	6	Large memory footprint
Neural network regression	Good	Moderate	No	8	
Clustering family					
K-means clustering	Excellent	Moderate	Yes	8	A clustering algorithm

Requirements for a data science scenario

Once you know what you want to do with your data, you need to determine additional requirements for your solution.

Make choices and possibly trade-offs for the following requirements:

- Accuracy
- Training time
- Linearity
- Number of parameters
- Number of features

Accuracy

Accuracy in machine learning measures the effectiveness of a model as the proportion of true results to total cases. In Machine Learning designer, the [Evaluate Model component](#) computes a set of industry-standard evaluation metrics. You can use this component to measure the accuracy of a trained model.

Getting the most accurate answer possible isn't always necessary. Sometimes an approximation is adequate, depending on what you want to use it for. If that is the case, you may be able to cut your processing time dramatically by sticking with more approximate methods. Approximate methods also naturally tend to avoid overfitting.

There are three ways to use the Evaluate Model component:

- Generate scores over your training data in order to evaluate the model
- Generate scores on the model, but compare those scores to scores on a reserved testing set
- Compare scores for two different but related models, using the same set of data

For a complete list of metrics and approaches you can use to evaluate the accuracy of machine learning models, see [Evaluate Model component](#).

Training time

In supervised learning, training means using historical data to build a machine learning model that minimizes errors. The number of minutes or hours necessary to train a model varies a great deal between algorithms. Training time is often closely tied to accuracy; one typically accompanies the other.

In addition, some algorithms are more sensitive to the number of data points than others. You might choose a specific algorithm because you have a time limitation, especially when the data set is large.

In Machine Learning designer, creating and using a machine learning model is typically a three-step process:

1. Configure a model, by choosing a particular type of algorithm, and then defining its parameters or hyperparameters.
2. Provide a dataset that is labeled and has data compatible with the algorithm. Connect both the data and the model to [Train Model component](#).
3. After training is completed, use the trained model with one of the [scoring components](#) to make predictions on new data.

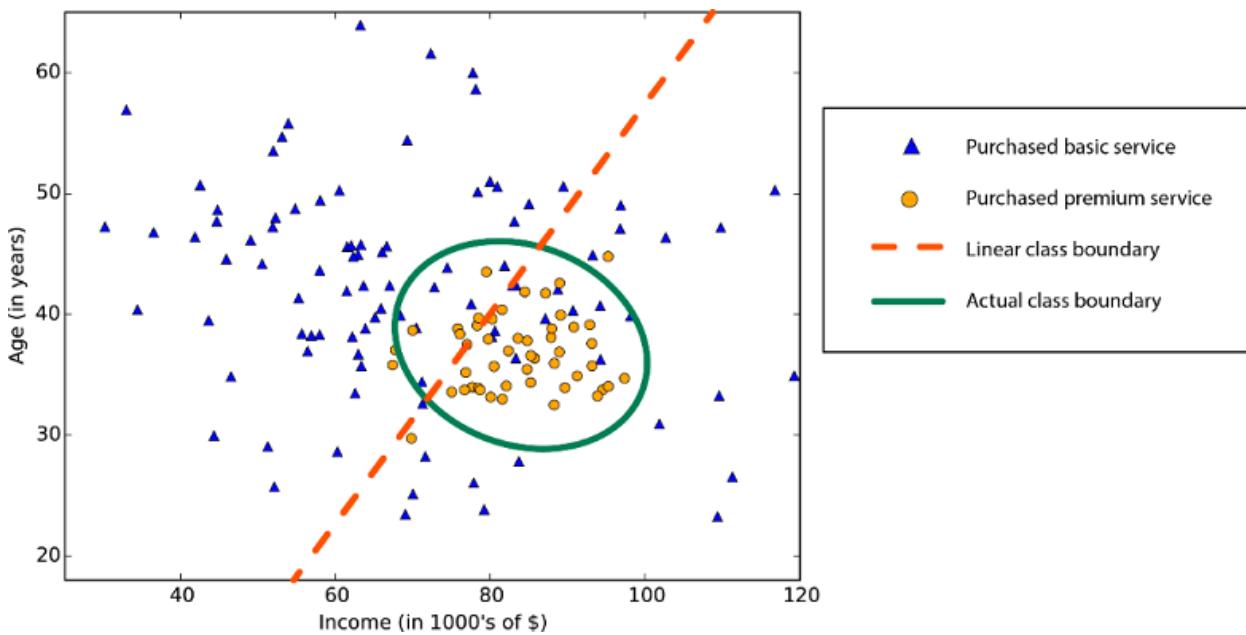
Linearity

Linearity in statistics and machine learning means that there is a linear relationship between a variable and a constant in your dataset. For example, linear classification algorithms assume that classes can be separated by a straight line (or its higher-dimensional analog).

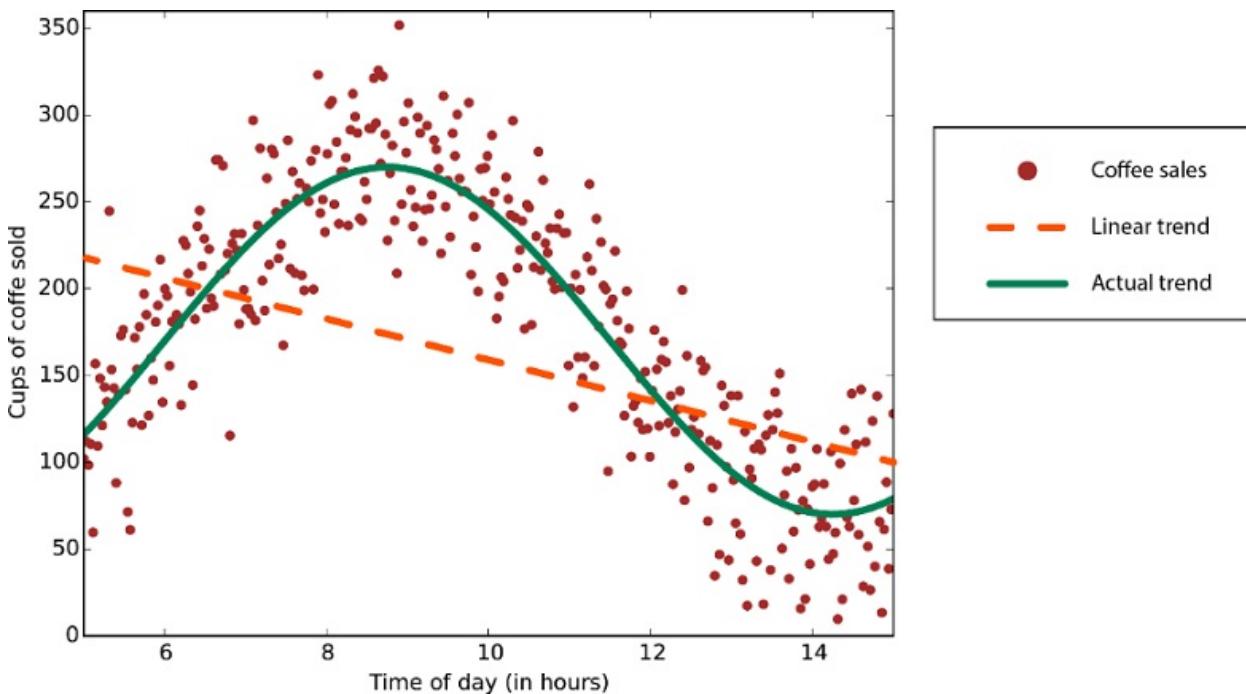
Lots of machine learning algorithms make use of linearity. In Azure Machine Learning designer, they include:

- [Multiclass logistic regression](#)
- [Two-class logistic regression](#)
- [Support vector machines](#)

Linear regression algorithms assume that data trends follow a straight line. This assumption isn't bad for some problems, but for others it reduces accuracy. Despite their drawbacks, linear algorithms are popular as a first strategy. They tend to be algorithmically simple and fast to train.



Nonlinear class boundary: Relying on a linear classification algorithm would result in low accuracy.



Data with a nonlinear trend: Using a linear regression method would generate much larger errors than necessary.

Number of parameters

Parameters are the knobs a data scientist gets to turn when setting up an algorithm. They are numbers that affect the algorithm's behavior, such as error tolerance or number of iterations, or options between variants of how the algorithm behaves. The training time and accuracy of the algorithm can sometimes be sensitive to getting just the right settings. Typically, algorithms with large numbers of parameters require the most trial and error to find a good combination.

Alternatively, there is the [Tune Model Hyperparameters component](#) in Machine Learning designer: The goal of this component is to determine the optimum hyperparameters for a machine learning model. The component builds and tests multiple models by using different combinations of settings. It compares metrics over all models to get the combinations of settings.

While this is a great way to make sure you've spanned the parameter space, the time required to train a model

increases exponentially with the number of parameters. The upside is that having many parameters typically indicates that an algorithm has greater flexibility. It can often achieve very good accuracy, provided you can find the right combination of parameter settings.

Number of features

In machine learning, a feature is a quantifiable variable of the phenomenon you are trying to analyze. For certain types of data, the number of features can be very large compared to the number of data points. This is often the case with genetics or textual data.

A large number of features can bog down some learning algorithms, making training time unfeasibly long.

[Support vector machines](#) are particularly well suited to scenarios with a high number of features. For this reason, they have been used in many applications from information retrieval to text and image classification. Support vector machines can be used for both classification and regression tasks.

Feature selection refers to the process of applying statistical tests to inputs, given a specified output. The goal is to determine which columns are more predictive of the output. The [Filter Based Feature Selection component](#) in Machine Learning designer provides multiple feature selection algorithms to choose from. The component includes correlation methods such as Pearson correlation and chi-squared values.

You can also use the [Permutation Feature Importance component](#) to compute a set of feature importance scores for your dataset. You can then leverage these scores to help you determine the best features to use in a model.

Next steps

- [Learn more about Azure Machine Learning designer](#)
- For descriptions of all the machine learning algorithms available in Azure Machine Learning designer, see [Machine Learning designer algorithm and component reference](#)
- To explore the relationship between deep learning, machine learning, and AI, see [Deep Learning vs. Machine Learning](#)

What are Azure Machine Learning endpoints?

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

Use Azure Machine Learning endpoints to streamline model deployments for both real-time and batch inference deployments. Endpoints provide a unified interface to invoke and manage model deployments across compute types.

In this article, you learn about:

- Endpoints
- Deployments
- Managed online endpoints
- Kubernetes online endpoints
- Batch inference endpoints

What are endpoints and deployments?

After you train a machine learning model, you need to deploy the model so that others can use it to do inferencing. In Azure Machine Learning, you can use **endpoints** and **deployments** to do so.

An **endpoint** is an HTTPS endpoint that clients can call to receive the inferencing (scoring) output of a trained model. It provides:

- Authentication using "key & token" based auth
- SSL termination
- A stable scoring URI (endpoint-name.region.inference.ml.azure.com)

A **deployment** is a set of resources required for hosting the model that does the actual inferencing.

A single endpoint can contain multiple deployments. Endpoints and deployments are independent Azure Resource Manager resources that appear in the Azure portal.

Azure Machine Learning uses the concept of endpoints and deployments to implement different types of endpoints: [online endpoints](#) and [batch endpoints](#).

Multiple developer interfaces

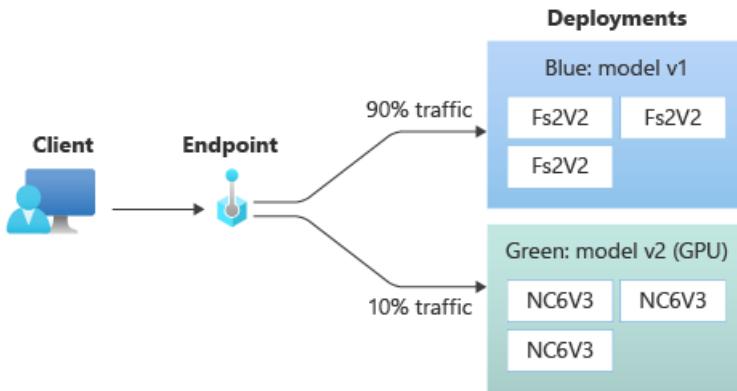
Create and manage batch and online endpoints with multiple developer tools:

- The Azure CLI
- Azure Resource Manager/REST API
- Azure Machine Learning studio web portal
- Azure portal (IT/Admin)
- Support for CI/CD MLOps pipelines using the Azure CLI interface & REST/ARM interfaces

What are online endpoints?

Online endpoints are endpoints that are used for online (real-time) inferencing. Compared to **batch endpoints**, **online endpoints** contain **deployments** that are ready to receive data from clients and can send responses back in real time.

The following diagram shows an online endpoint that has two deployments, 'blue' and 'green'. The blue deployment uses VMs with a CPU SKU, and runs v1 of a model. The green deployment uses VMs with a GPU SKU, and uses v2 of the model. The endpoint is configured to route 90% of incoming traffic to the blue deployment, while green receives the remaining 10%.



Online deployments requirements

To create an online endpoint, you need to specify the following elements:

- Model files (or specify a registered model in your workspace)
- Scoring script - code needed to do scoring/inferencing
- Environment - a Docker image with Conda dependencies, or a dockerfile
- Compute instance & scale settings

Learn how to deploy online endpoints from the [CLI](#) and the [studio web portal](#).

Test and deploy locally for faster debugging

Deploy locally to test your endpoints without deploying to the cloud. Azure Machine Learning creates a local Docker image that mimics the Azure ML image. Azure Machine Learning will build and run deployments for you locally, and cache the image for rapid iterations.

Native blue/green deployment

Recall, that a single endpoint can have multiple deployments. The online endpoint can do load balancing to give any percentage of traffic to each deployment.

Traffic allocation can be used to do safe rollout blue/green deployments by balancing requests between different instances.

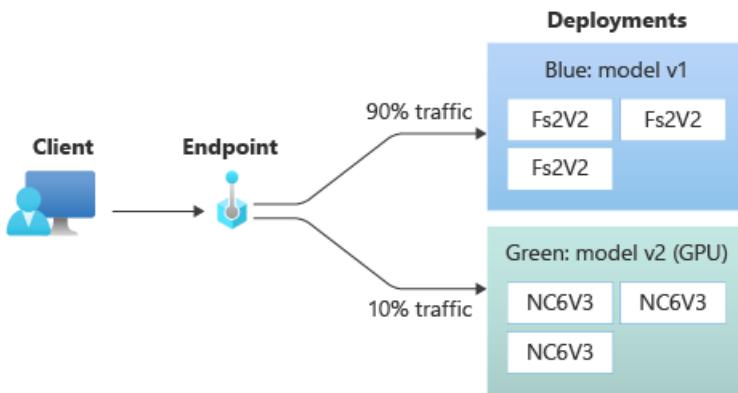
TIP

A request can bypass the configured traffic load balancing by including an HTTP header of `azureml-model-deployment`. Set the header value to the name of the deployment you want the request to route to.

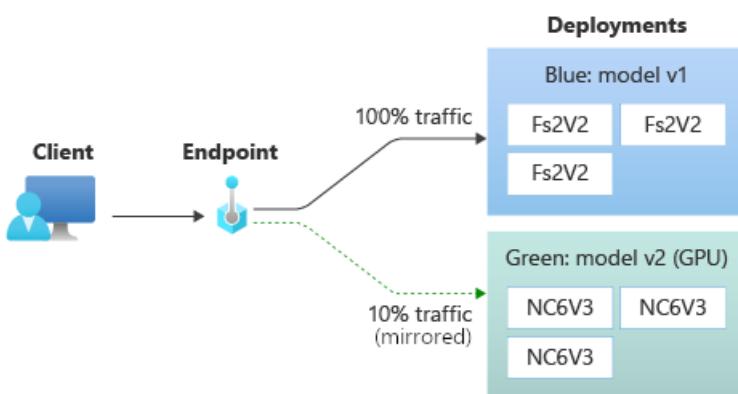
Update traffic allocation

Ensure that allocated traffic between all deployments adds up to 0% or 100%

Deployment name	Traffic allocation %
green	<input type="text" value="10"/> 10%
blue	<input type="text" value="90"/> 90%
Total traffic percentage	100% <input checked="" type="checkbox"/>



Traffic to one deployment can also be mirrored (copied) to another deployment. Mirroring is useful when you want to test for things like response latency or error conditions without impacting live clients. For example, a blue/green deployment where 100% of the traffic is routed to blue and a 10% is mirrored to the green deployment. With mirroring, the results of the traffic to the green deployment aren't returned to the clients but metrics and logs are collected. Mirror traffic functionality is a [preview feature](#).



Learn how to [safely rollout to online endpoints](#).

Application Insights integration

All online endpoints integrate with Application Insights to monitor SLAs and diagnose issues.

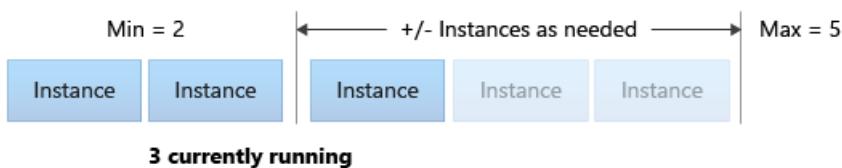
However [managed online endpoints](#) also include out-of-box integration with Azure Logs and Azure Metrics.

Security

- Authentication: Key and Azure ML Tokens
- Managed identity: User assigned and system assigned
- SSL by default for endpoint invocation

Autoscaling

Autoscale automatically runs the right amount of resources to handle the load on your application. Managed endpoints support autoscaling through integration with the [Azure monitor autoscale](#) feature. You can configure metrics-based scaling (for instance, CPU utilization >70%), schedule-based scaling (for example, scaling rules for peak business hours), or a combination.



Visual Studio Code debugging

Visual Studio Code enables you to interactively debug endpoints.

Private endpoint support (preview)

Optionally, you can secure communication with a managed online endpoint by using private endpoints. This functionality is currently in preview.

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

You can configure security for inbound scoring requests and outbound communications with the workspace and other services separately. Inbound communications use the private endpoint of the Azure Machine Learning workspace. Outbound communications use private endpoints created per deployment.

For more information, see [Secure online endpoints](#).

Managed online endpoints vs Kubernetes online endpoints

There are two types of online endpoints: **managed online endpoints** and **Kubernetes online endpoints**.

Managed online endpoints help to deploy your ML models in a turnkey manner. Managed online endpoints work with powerful CPU and GPU machines in Azure in a scalable, fully managed way. Managed online endpoints take care of serving, scaling, securing, and monitoring your models, freeing you from the overhead of setting up and managing the underlying infrastructure. The main example in this doc uses managed online endpoints for deployment.

Kubernetes online endpoint allows you to deploy models and serve online endpoints at your fully configured and managed [Kubernetes cluster anywhere](#), with CPUs or GPUs.

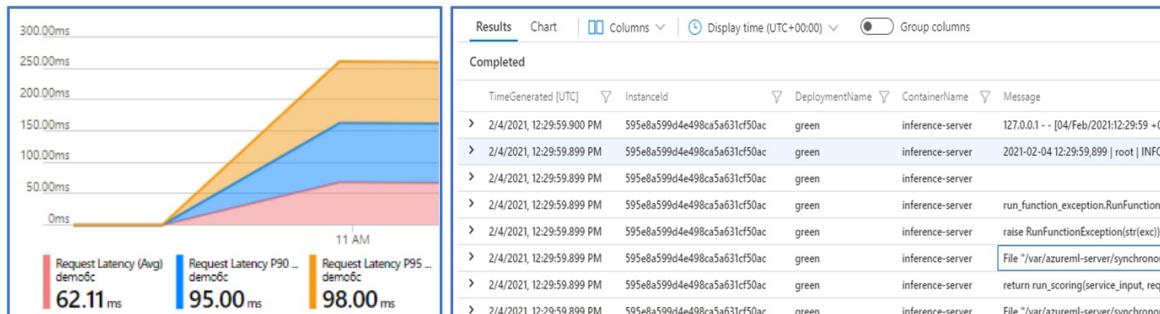
The following table highlights the key differences between managed online endpoints and Kubernetes online endpoints.

	MANAGED ONLINE ENDPOINTS	KUBERNETES ONLINE ENDPOINTS
Recommended users	Users who want a managed model deployment and enhanced MLOps experience	Users who prefer Kubernetes and can self-manage infrastructure requirements
Infrastructure management	Managed compute provisioning, scaling, host OS image updates, and security hardening	User responsibility
Compute type	Managed (AmlCompute)	Kubernetes cluster (Kubernetes)
Out-of-box monitoring	Azure Monitoring (includes key metrics like latency and throughput)	Supported
Out-of-box logging	Azure Logs and Log Analytics at endpoint level	Unsupported
Application Insights	Supported	Supported
Managed identity	Supported	Supported
Virtual Network (VNET)	Supported (preview)	Supported
View costs	Endpoint and deployment level	Cluster level
Mirrored traffic	Supported	Unsupported

Managed online endpoints

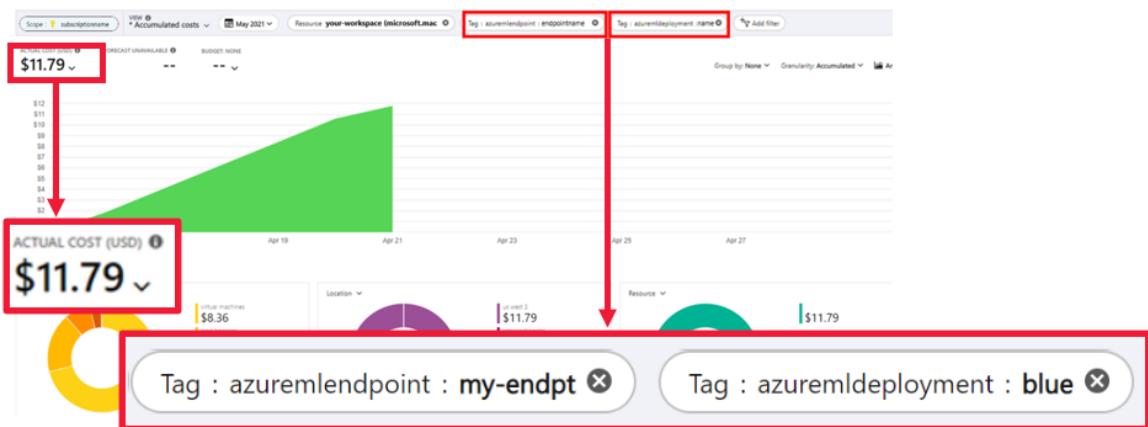
Managed online endpoints can help streamline your deployment process. Managed online endpoints provide the following benefits over Kubernetes online endpoints:

- Managed infrastructure
 - Automatically provisions the compute and hosts the model (you just need to specify the VM type and scale settings)
 - Automatically updates and patches the underlying host OS image
 - Automatic node recovery if there's a system failure
- Monitoring and logs
 - Monitor model availability, performance, and SLA using [native integration with Azure Monitor](#).
 - Debug deployments using the logs and native integration with Azure Log Analytics.



- View costs

- Managed online endpoints let you [monitor cost at the endpoint and deployment level](#)



NOTE

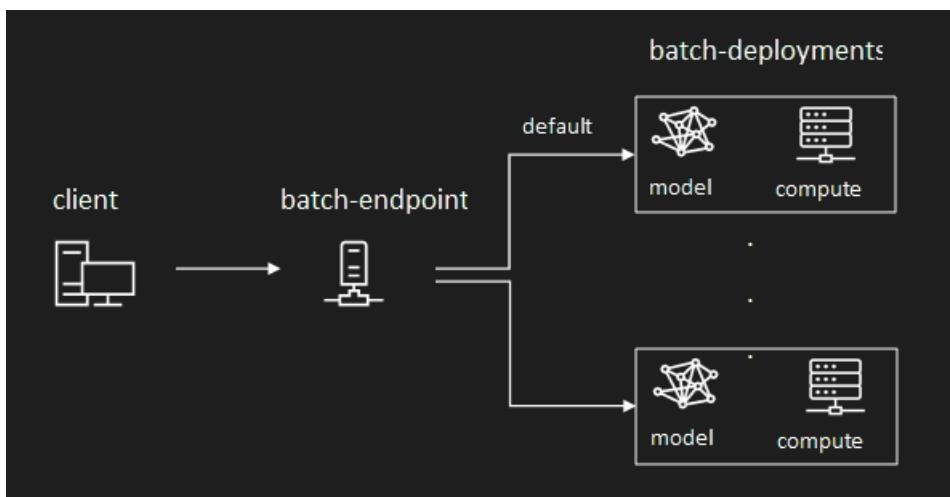
Managed online endpoints are based on Azure Machine Learning compute. When using a managed online endpoint, you pay for the compute and networking charges. There is no additional surcharge.

If you use a virtual network and secure outbound (egress) traffic from the managed online endpoint, there is an additional cost. For egress, three private endpoints are created *per deployment* for the managed online endpoint. These are used to communicate with the default storage account, Azure Container Registry, and workspace. Additional networking charges may apply. For more information on pricing, see the [Azure pricing calculator](#).

For a step-by-step tutorial, see [How to deploy online endpoints](#).

What are batch endpoints?

Batch endpoints are endpoints that are used to do batch inferencing on large volumes of data over a period of time. Batch endpoints receive pointers to data and run jobs asynchronously to process the data in parallel on compute clusters. Batch endpoints store outputs to a data store for further analysis.



Batch deployment requirements

To create a batch deployment, you need to specify the following elements:

- Model files (or specify a model registered in your workspace)
- Compute
- Scoring script - code needed to do the scoring/inferencing
- Environment - a Docker image with Conda dependencies

If you're deploying [MLFlow models](#), there's no need to provide a scoring script and execution environment, as

both are autogenerated.

Learn how to [deploy and use batch endpoints with the Azure CLI](#) and the [studio web portal](#)

Managed cost with autoscaling compute

Invoking a batch endpoint triggers an asynchronous batch inference job. Compute resources are automatically provisioned when the job starts, and automatically de-allocated as the job completes. So you only pay for compute when you use it.

You can [override compute resource settings](#) (like instance count) and advanced settings (like mini batch size, error threshold, and so on) for each individual batch inference job to speed up execution and reduce cost.

Flexible data sources and storage

You can use the following options for input data when invoking a batch endpoint:

- Cloud data - Either a path on Azure Machine Learning registered datastore, a reference to Azure Machine Learning registered V2 data asset, or a public URI. For more information, see [Connect to data with the Azure Machine Learning studio](#)
- Data stored locally - it will be automatically uploaded to the Azure ML registered datastore and passed to the batch endpoint.

NOTE

- If you are using existing V1 FileDataset for batch endpoint, we recommend migrating them to V2 data assets and refer to them directly when invoking batch endpoints. Currently only data assets of type `uri_folder` or `uri_file` are supported. Batch endpoints created with GA CLIV2 (2.4.0 and newer) or GA REST API (2022-05-01 and newer) will not support V1 Dataset.
- You can also extract the URI or path on datastore extracted from V1 FileDataset by using `az ml dataset show` command with `--query` parameter and use that information for invoke.
- While Batch endpoints created with earlier APIs will continue to support V1 FileDataset, we will be adding further V2 data assets support with the latest API versions for even more usability and flexibility. For more information on V2 data assets, see [Work with data using SDK v2 \(preview\)](#). For more information on the new V2 experience, see [What is v2](#).

For more information on supported input options, see [Batch scoring with batch endpoint](#).

Specify the storage output location to any datastore and path. By default, batch endpoints store their output to the workspace's default blob store, organized by the Job Name (a system-generated GUID).

Security

- Authentication: Azure Active Directory Tokens
- SSL: enabled by default for endpoint invocation
- VNET support: Batch endpoints support ingress protection. A batch endpoint with ingress protection will accept scoring requests only from hosts inside a virtual network but not from the public internet. A batch endpoint that is created in a private-link enabled workspace will have ingress protection. To create a private-link enabled workspace, see [Create a secure workspace](#).

NOTE

Creating batch endpoints in a private-link enabled workspace is only supported in the following versions.

- CLI - version 2.15.1 or higher.
- REST API - version 2022-05-01 or higher.

- SDK V2 - version 0.1.0b3 or higher.

Next steps

- [How to deploy online endpoints with the Azure CLI](#)
- [How to deploy batch endpoints with the Azure CLI](#)
- [How to use online endpoints with the studio](#)
- [Deploy models with REST](#)
- [How to monitor managed online endpoints](#)
- [How to view managed online endpoint costs](#)
- [Manage and increase quotas for resources with Azure Machine Learning](#)

ONNX and Azure Machine Learning: Create and accelerate ML models

9/22/2022 • 4 minutes to read • [Edit Online](#)

Learn how using the [Open Neural Network Exchange](#) (ONNX) can help optimize the inference of your machine learning model. Inference, or model scoring, is the phase where the deployed model is used for prediction, most commonly on production data.

Optimizing machine learning models for inference (or model scoring) is difficult since you need to tune the model and the inference library to make the most of the hardware capabilities. The problem becomes extremely hard if you want to get optimal performance on different kinds of platforms (cloud/edge, CPU/GPU, etc.), since each one has different capabilities and characteristics. The complexity increases if you have models from a variety of frameworks that need to run on a variety of platforms. It's very time consuming to optimize all the different combinations of frameworks and hardware. A solution to train once in your preferred framework and run anywhere on the cloud or edge is needed. This is where ONNX comes in.

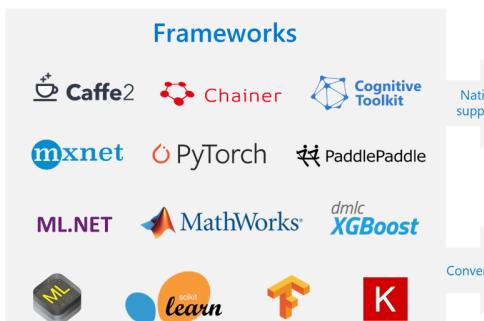
Microsoft and a community of partners created ONNX as an open standard for representing machine learning models. Models from [many frameworks](#) including TensorFlow, PyTorch, SciKit-Learn, Keras, Chainer, MXNet, MATLAB, and SparkML can be exported or converted to the standard ONNX format. Once the models are in the ONNX format, they can be run on a variety of platforms and devices.

[ONNX Runtime](#) is a high-performance inference engine for deploying ONNX models to production. It's optimized for both cloud and edge and works on Linux, Windows, and Mac. Written in C++, it also has C, Python, C#, Java, and JavaScript (Node.js) APIs for usage in a variety of environments. ONNX Runtime supports both DNN and traditional ML models and integrates with accelerators on different hardware such as TensorRT on Nvidia GPUs, OpenVINO on Intel processors, DirectML on Windows, and more. By using ONNX Runtime, you can benefit from the extensive production-grade optimizations, testing, and ongoing improvements.

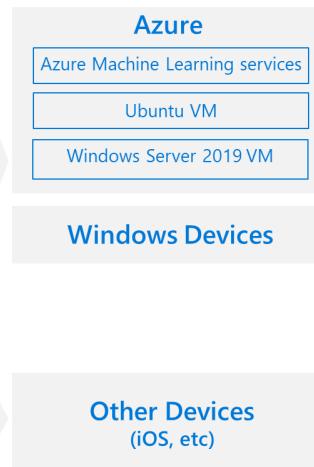
ONNX Runtime is used in high-scale Microsoft services such as Bing, Office, and Azure Cognitive Services. Performance gains are dependent on a number of factors, but these Microsoft services have seen an **average 2x performance gain on CPU**. In addition to Azure Machine Learning services, ONNX Runtime also runs in other products that support Machine Learning workloads, including:

- Windows: The runtime is built into Windows as part of [Windows Machine Learning](#) and runs on hundreds of millions of devices.
- Azure SQL product family: Run native scoring on data in [Azure SQL Edge](#) and [Azure SQL Managed Instance](#).
- ML.NET: [Run ONNX models in ML.NET](#).

Create



Deploy



Get ONNX models

You can obtain ONNX models in several ways:

- Train a new ONNX model in Azure Machine Learning (see examples at the bottom of this article) or by using [automated Machine Learning capabilities](#)
- Convert existing model from another format to ONNX (see the [tutorials](#))
- Get a pre-trained ONNX model from the [ONNX Model Zoo](#)
- Generate a customized ONNX model from [Azure Custom Vision service](#)

Many models including image classification, object detection, and text processing can be represented as ONNX models. If you run into an issue with a model that cannot be converted successfully, please file an issue in the GitHub of the respective converter that you used. You can continue using your existing format model until the issue is addressed.

Deploy ONNX models in Azure

With Azure Machine Learning, you can deploy, manage, and monitor your ONNX models. Using the standard [deployment workflow](#) and ONNX Runtime, you can create a REST endpoint hosted in the cloud. See example Jupyter notebooks at the end of this article to try it out for yourself.

Install and use ONNX Runtime with Python

Python packages for ONNX Runtime are available on [PyPi.org](#) ([CPU](#), [GPU](#)). Please read [system requirements](#) before installation.

To install ONNX Runtime for Python, use one of the following commands:

```
pip install onnxruntime      # CPU build
pip install onnxruntime-gpu  # GPU build
```

To call ONNX Runtime in your Python script, use:

```
import onnxruntime
session = onnxruntime.InferenceSession("path to model")
```

The documentation accompanying the model usually tells you the inputs and outputs for using the model. You can also use a visualization tool such as [Netron](#) to view the model. ONNX Runtime also lets you query the

model metadata, inputs, and outputs:

```
session.get_modelmeta()  
first_input_name = session.get_inputs()[0].name  
first_output_name = session.get_outputs()[0].name
```

To inference your model, use `run` and pass in the list of outputs you want returned (leave empty if you want all of them) and a map of the input values. The result is a list of the outputs.

```
results = session.run(["output1", "output2"], {  
    "input1": indata1, "input2": indata2})  
results = session.run([], {"input1": indata1, "input2": indata2})
```

For the complete Python API reference, see the [ONNX Runtime reference docs](#).

Examples

See [how-to-use-azureml/deployment/onnx](#) for example Python notebooks that create and deploy ONNX models.

Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service](#).

Samples for usage in other languages can be found in the [ONNX Runtime GitHub](#).

More info

Learn more about ONNX or contribute to the project:

- [ONNX project website](#)
- [ONNX code on GitHub](#)

Learn more about ONNX Runtime or contribute to the project:

- [ONNX Runtime project website](#)
- [ONNX Runtime GitHub Repo](#)

Prebuilt Docker images for inference

9/22/2022 • 2 minutes to read • [Edit Online](#)

Prebuilt Docker container images for inference are used when deploying a model with Azure Machine Learning. The images are prebuilt with popular machine learning frameworks and Python packages. You can also extend the packages to add other packages by using one of the following methods:

Why should I use prebuilt images?

- Reduces model deployment latency.
- Improves model deployment success rate.
- Avoid unnecessary image build during model deployment.
- Only have required dependencies and access right in the image/container.

List of prebuilt Docker images for inference

IMPORTANT

The list provided below includes only currently supported inference docker images by Azure Machine Learning.

- All the docker images run as non-root user.
- We recommend using `latest` tag for docker images. Prebuilt docker images for inference are published to Microsoft container registry (MCR), to query list of tags available, follow [instructions on the GitHub repository](#).
- If you want to use a specific tag for any inference docker image, we support from `latest` to the tag that is *6 months* old from the `latest`.

Inference minimal base images

FRAMEWORK VERSION	CPU/GPU	PRE-INSTALLED PACKAGES	MCR PATH
NA	CPU	NA	<code>mcr.microsoft.com/azureml/minimal-ubuntu18.04-py37-cpu-inference:latest</code>
NA	GPU	NA	<code>mcr.microsoft.com/azureml/minimal-ubuntu18.04-py37-cuda11.0.3-gpu-inference:latest</code>
NA	CPU	NA	<code>mcr.microsoft.com/azureml/minimal-ubuntu20.04-py38-cpu-inference:latest</code>

How to use inference prebuilt docker images?

[Check examples in the Azure machine learning GitHub repository](#)

Next steps

- [Deploy and score a machine learning model by using an online endpoint](#)
- [Learn more about custom containers](#)

- [azureml-examples GitHub repository](#)

MLOps: Model management, deployment, lineage, and monitoring with Azure Machine Learning

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

In this article, learn about how do Machine Learning Operations (MLOps) in Azure Machine Learning to manage the lifecycle of your models. MLOps improves the quality and consistency of your machine learning solutions.

What is MLOps?

MLOps is based on [DevOps](#) principles and practices that increase the efficiency of workflows. Examples include continuous integration, delivery, and deployment. MLOps applies these principles to the machine learning process, with the goal of:

- Faster experimentation and development of models.
- Faster deployment of models into production.
- Quality assurance and end-to-end lineage tracking.

MLOps in Machine Learning

Machine Learning provides the following MLOps capabilities:

- **Create reproducible machine learning pipelines.** Use machine learning pipelines to define repeatable and reusable steps for your data preparation, training, and scoring processes.
- **Create reusable software environments.** Use these environments for training and deploying models.
- **Register, package, and deploy models from anywhere.** You can also track associated metadata required to use the model.
- **Capture the governance data for the end-to-end machine learning lifecycle.** The logged lineage information can include who is publishing models and why changes were made. It can also include when models were deployed or used in production.
- **Notify and alert on events in the machine learning lifecycle.** Event examples include experiment completion, model registration, model deployment, and data drift detection.
- **Monitor machine learning applications for operational and machine learning-related issues.** Compare model inputs between training and inference. Explore model-specific metrics. Provide monitoring and alerts on your machine learning infrastructure.
- **Automate the end-to-end machine learning lifecycle with Machine Learning and Azure Pipelines.** By using pipelines, you can frequently update models. You can also test new models. You can continually roll out new machine learning models alongside your other applications and services.

For more information on MLOps, see [Machine learning DevOps](#).

Create reproducible machine learning pipelines

Use machine learning pipelines from Machine Learning to stitch together all the steps in your model training process.

A machine learning pipeline can contain steps from data preparation to feature extraction to hyperparameter tuning to model evaluation. For more information, see [Machine learning pipelines](#).

If you use the [designer](#) to create your machine learning pipelines, you can at any time select the ... icon in the upper-right corner of the designer page. Then select **Clone**. When you clone your pipeline, you iterate your pipeline design without losing your old versions.

Create reusable software environments

By using Machine Learning environments, you can track and reproduce your projects' software dependencies as they evolve. You can use environments to ensure that builds are reproducible without manual software configurations.

Environments describe the pip and conda dependencies for your projects. You can use them for training and deployment of models. For more information, see [What are Machine Learning environments?](#).

Register, package, and deploy models from anywhere

The following sections discuss how to register, package, and deploy models.

Register and track machine learning models

With model registration, you can store and version your models in the Azure cloud, in your workspace. The model registry makes it easy to organize and keep track of your trained models.

TIP

A registered model is a logical container for one or more files that make up your model. For example, if you have a model that's stored in multiple files, you can register them as a single model in your Machine Learning workspace. After registration, you can then download or deploy the registered model and receive all the files that were registered.

Registered models are identified by name and version. Each time you register a model with the same name as an existing one, the registry increments the version. More metadata tags can be provided during registration. These tags are then used when you search for a model. Machine Learning supports any model that can be loaded by using Python 3.5.2 or higher.

TIP

You can also register models trained outside Machine Learning.

IMPORTANT

- When you use the **Filter by** `Tags` option on the **Models** page of Azure Machine Learning Studio, instead of using `TagName : TagValue`, use `TagName=TagValue` without spaces.
- You can't delete a registered model that's being used in an active deployment.

For more information, [Work with models in Azure Machine Learning](#).

Package and debug models

Before you deploy a model into production, it's packaged into a Docker image. In most cases, image creation happens automatically in the background during deployment. You can manually specify the image.

If you run into problems with the deployment, you can deploy on your local development environment for troubleshooting and debugging.

For more information, see [How to troubleshoot online endpoints](#).

Convert and optimize models

Converting your model to [Open Neural Network Exchange](#) (ONNX) might improve performance. On average, converting to ONNX can double performance.

For more information on ONNX with Machine Learning, see [Create and accelerate machine learning models](#).

Use models

Trained machine learning models are deployed as [endpoints](#) in the cloud or locally. Deployments use CPU, GPU for inferencing.

When deploying a model as an endpoint, you provide the following items:

- The models that are used to score data submitted to the service or device.
- An entry script. This script accepts requests, uses the models to score the data, and returns a response.
- A Machine Learning environment that describes the pip and conda dependencies required by the models and entry script.
- Any other assets such as text and data that are required by the models and entry script.

You also provide the configuration of the target deployment platform. For example, the VM family type, available memory, and number of cores. When the image is created, components required by Azure Machine Learning are also added. For example, assets needed to run the web service.

Batch scoring

Batch scoring is supported through batch endpoints. For more information, see [endpoints](#).

Online endpoints

You can use your models with an online endpoint. Online endpoints can use the following compute targets:

- Managed online endpoints
- Azure Kubernetes Service
- Local development environment

To deploy the model to an endpoint, you must provide the following items:

- The model or ensemble of models.
- Dependencies required to use the model. Examples are a script that accepts requests and invokes the model and conda dependencies.
- Deployment configuration that describes how and where to deploy the model.

For more information, see [Deploy online endpoints](#).

Controlled rollout

When deploying to an online endpoint, you can use controlled rollout to enable the following scenarios:

- Create multiple versions of an endpoint for a deployment
- Perform A/B testing by routing traffic to different deployments within the endpoint.
- Switch between endpoint deployments by updating the traffic percentage in endpoint configuration.

For more information, see [Controlled rollout of machine learning models](#).

Analytics

Microsoft Power BI supports using machine learning models for data analytics. For more information, see [Machine Learning integration in Power BI \(preview\)](#).

Capture the governance data required for MLOps

Machine Learning gives you the capability to track the end-to-end audit trail of all your machine learning assets by using metadata. For example:

- Machine Learning [integrates with Git](#) to track information on which repository, branch, and commit your code came from.
- [Machine Learning datasets](#) help you track, profile, and version data.
- [Interpretability](#) allows you to explain your models, meet regulatory compliance, and understand how models arrive at a result for specific input.
- Machine Learning Job history stores a snapshot of the code, data, and computes used to train a model.
- The Machine Learning Model Registry captures all the metadata associated with your model. For example, metadata includes which experiment trained it, where it's being deployed, and if its deployments are healthy.
- [Integration with Azure](#) allows you to act on events in the machine learning lifecycle. Examples are model registration, deployment, data drift, and training (job) events.

TIP

While some information on models and datasets is automatically captured, you can add more information by using *tags*. When you look for registered models and datasets in your workspace, you can use tags as a filter.

Associating a dataset with a registered model is an optional step. For information on how to reference a dataset when you register a model, see the [Model class reference](#).

Notify, automate, and alert on events in the machine learning lifecycle

Machine Learning publishes key events to Azure Event Grid, which can be used to notify and automate on events in the machine learning lifecycle. For more information, see [Use Event Grid](#).

Monitor for operational and machine learning issues

Monitoring enables you to understand what data is being sent to your model, and the predictions that it returns.

This information helps you understand how your model is being used. The collected input data might also be useful in training future versions of the model.

For more information, see [Enable model data collection](#).

Retrain your model on new data

Often, you'll want to validate your model, update it, or even retrain it from scratch, as you receive new information. Sometimes, receiving new data is an expected part of the domain. Other times, as discussed in [Detect data drift \(preview\) on datasets](#), model performance can degrade because of:

- Changes to a particular sensor.
- Natural data changes such as seasonal effects.
- Features shifting in their relation to other features.

There's no universal answer to "How do I know if I should retrain?" The Machine Learning event and monitoring tools previously discussed are good starting points for automation. After you've decided to retrain, you should:

- Preprocess your data by using a repeatable, automated process.
- Train your new model.
- Compare the outputs of your new model to the outputs of your old model.
- Use predefined criteria to choose whether to replace your old model.

A theme of the preceding steps is that your retraining should be automated, not improvised. [Machine Learning pipelines](#) are a good answer for creating workflows that relate to data preparation, training, validation, and

deployment. Read [Retrain models with Machine Learning designer](#) to see how pipelines and the Machine Learning designer fit into a retraining scenario.

Automate the machine learning lifecycle

You can use GitHub and Azure Pipelines to create a continuous integration process that trains a model. In a typical scenario, when a data scientist checks a change into the Git repo for a project, Azure Pipelines starts a training job. The results of the job can then be inspected to see the performance characteristics of the trained model. You can also create a pipeline that deploys the model as a web service.

The [Machine Learning extension](#) makes it easier to work with Azure Pipelines. It provides the following enhancements to Azure Pipelines:

- Enables workspace selection when you define a service connection.
- Enables release pipelines to be triggered by trained models created in a training pipeline.

For more information on using Azure Pipelines with Machine Learning, see:

- [Continuous integration and deployment of machine learning models with Azure Pipelines](#)
- [Machine Learning MLOps repository](#)
- [Machine Learning MLOpsPython repository](#)

You can also use Azure Data Factory to create a data ingestion pipeline that prepares data for use with training. For more information, see [Data ingestion pipeline](#).

Next steps

Learn more by reading and exploring the following resources:

- [Learning path: End-to-end MLOps with Azure Machine Learning](#)
- [How to deploy a model to an online endpoint with Machine Learning](#)
- [Tutorial: Train and deploy a model](#)
- [End-to-end MLOps examples repo](#)
- [CI/CD of machine learning models with Azure Pipelines](#)
- [Machine learning at scale](#)
- [Azure AI reference architectures and best practices repo](#)

What are Azure Machine Learning pipelines?

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v1 Python SDK azureml v1

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

An Azure Machine Learning pipeline is an independently executable workflow of a complete machine learning task. An Azure Machine Learning pipeline helps to standardize the best practices of producing a machine learning model, enables the team to execute at scale, and improves the model building efficiency.

Why are Azure Machine Learning pipelines needed?

The core of a machine learning pipeline is to split a complete machine learning task into a multistep workflow. Each step is a manageable component that can be developed, optimized, configured, and automated individually. Steps are connected through well-defined interfaces. The Azure Machine Learning pipeline service automatically orchestrates all the dependencies between pipeline steps. This modular approach brings two key benefits:

- Standardize the Machine learning operation (MLOPs) practice and support scalable team collaboration
- Training efficiency and cost reduction

Standardize the MLOps practice and support scalable team collaboration

Machine learning operation (MLOPs) automates the process of building machine learning models and taking the model to production. This is a complex process. It usually requires collaboration from different teams with different skills. A well-defined machine learning pipeline can abstract this complex process into a multiple steps workflow, mapping each step to a specific task such that each team can work independently.

For example, a typical machine learning project includes the steps of data collection, data preparation, model training, model evaluation, and model deployment. Usually, the data engineers concentrate on data steps, data scientists spend most time on model training and evaluation, the machine learning engineers focus on model deployment and automation of the entire workflow. By leveraging machine learning pipeline, each team only needs to work on building their own steps. The best way of building steps is using [Azure Machine Learning component](#), a self-contained piece of code that does one step in a machine learning pipeline. All these steps built by different users are finally integrated into one workflow through the pipeline definition. The pipeline is a collaboration tool for everyone in the project. The process of defining a pipeline and all its steps can be standardized by each company's preferred DevOps practice. The pipeline can be further versioned and automated. If the ML projects are described as a pipeline, then the best MLOps practice is already applied.

Training efficiency and cost reduction

Besides being the tool to put MLOps into practice, the machine learning pipeline also improves large model training's efficiency and reduces cost. Taking modern natural language model training as an example. It requires pre-processing large amounts of data and GPU intensive transformer model training. It takes hours to days to train a model each time. When the model is being built, the data scientist wants to test different training code or hyperparameters and run the training many times to get the best model performance. For most of these trainings, there's usually small changes from one training to another one. It will be a significant waste if every time the full training from data processing to model training takes place. By using machine learning pipeline, it can automatically calculate which steps result is unchanged and reuse outputs from previous training. Additionally, the machine learning pipeline supports running each step on different computation resources. Such that, the memory heavy data processing work and run-on high memory CPU machines, and the computation intensive training can run on expensive GPU machines. By properly choosing which step to run on

which type of machines, the training cost can be significantly reduced.

Getting started best practices

Depending on what a machine learning project already has, the starting point of building a machine learning pipeline may vary. There are a few typical approaches to building a pipeline.

The first approach usually applies to the team that hasn't used pipeline before and wants to take some advantage of pipeline like MLOps. In this situation, data scientists typically have developed some machine learning models on their local environment using their favorite tools. Machine learning engineers need to take data scientists' output into production. The work involves cleaning up some unnecessary code from original notebook or python code, changes the training input from local data to parameterized values, split the training code into multiple steps as needed, perform unit test of each step, and finally wraps all steps into a pipeline.

Once the teams get familiar with pipelines and want to do more machine learning projects using pipelines, they'll find the first approach is hard to scale. The second approach is set up a few pipeline templates, each try to solve one specific machine learning problem. The template predefines the pipeline structure including how many steps, each step's inputs and outputs, and their connectivity. To start a new machine learning project, the team first forks one template repo. The team leader then assigns members which step they need to work on. The data scientists and data engineers do their regular work. When they're happy with their result, they structure their code to fit in the pre-defined steps. Once the structured codes are checked-in, the pipeline can be executed or automated. If there's any change, each member only needs to work on their piece of code without touching the rest of the pipeline code.

Once a team has built a collection of machine learnings pipelines and reusable components, they could start to build the machine learning pipeline from cloning previous pipeline or tie existing reusable component together. At this stage, the team's overall productivity will be improved significantly.

Azure Machine Learning offers different methods to build a pipeline. For users who are familiar with DevOps practices, we recommend using [CLI](#). For data scientists who are familiar with python, we recommend writing pipeline using the [Azure ML SDK v1](#). For users who prefer to use UI, they could use the [designer to build pipeline by using registered components](#).

Which Azure pipeline technology should I use?

The Azure cloud provides several types of pipeline, each with a different purpose. The following table lists the different pipelines and what they're used for:

SCENARIO	PRIMARY PERSONA	AZURE OFFERING	OSS OFFERING	CANONICAL PIPE	STRENGTHS
Model orchestration (Machine learning)	Data scientist	Azure Machine Learning Pipelines	Kubeflow Pipelines	Data -> Model	Distribution, caching, code-first, reuse
Data orchestration (Data prep)	Data engineer	Azure Data Factory pipelines	Apache Airflow	Data -> Data	Strongly typed movement, data-centric activities
Code & app orchestration (CI/CD)	App Developer / Ops	Azure Pipelines	Jenkins	Code + Model -> App/Service	Most open and flexible activity support, approval queues, phases with gating

Next steps

Azure Machine Learning pipelines are a powerful facility that begins delivering value in the early development stages.

- [Define pipelines with the Azure ML CLI v2](#)
- [Define pipelines with the Azure ML SDK v2](#)
- [Define pipelines with Designer](#)
- Try out [CLI v2 pipeline example](#)
- Try out [Python SDK v2 pipeline example](#)

What is an Azure Machine Learning component?

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

An Azure Machine Learning component is a self-contained piece of code that does one step in a machine learning pipeline. A component is analogous to a function - it has a name, inputs, outputs, and a body. Components are the building blocks of the [Azure Machine Learning pipelines](#).

A component consists of three parts:

- Metadata: name, display_name, version, type, etc.
 - Interface: input/output specifications (name, type, description, default value, etc.).
 - Command, Code & Environment: command, code and environment required to run the component.

Components

- **Metadata**
name, display_name, version, type, etc.
 - **Interface**
input/output specifications (name, type, description, default value, etc)
 - **Command, Code & Environment**
command, code and environment required to run the component

```
    dataset {
        type: path
        description: "text Dataset Description used for finetuning tasks"
        name: "train"
        type: integer
        description: "TrainVal split (larger value of split, 800-split used as validation set, smaller as training set)"
        value: 800
        subtasks:
            training: true
            validation: false
            test: false
            eval: false
            validation_output: false
            eval_output: false
            code: generate_tf
            environment: null
            python: null
            python_process: null
            - source: ${inputs.source}
            - split: ${inputs.split}
            - validation: ${inputs.validation}
            - training: ${inputs.training}
            - validation_output: ${inputs.validation_output}
            - eval_output: ${inputs.eval_output}
```

CLL

SDK

U

Why should I use a component?

It's a good engineering practice to build a machine learning pipeline to split a complete machine learning task into a multi-step workflow. Such that, everyone can work on the specific step independently. In Azure Machine Learning, a component represents one reusable step in a pipeline. Components are designed to help improve the productivity of pipeline building. Specifically, components offer:

- **Well-defined interface:** Components require a well-defined interface (input and output). The interface allows the user to build steps and connect steps easily. The interface also hides the complex logic of a step and removes the burden of understanding how the step is implemented.
 - **Share and reuse:** As the building blocks of a pipeline, components can be easily shared and reused across pipelines, workspaces, and subscriptions. Components built by one team can be discovered and used by another team.
 - **Version control:** Components are versioned. The component producers can keep improving components and publish new versions. Consumers can use specific component versions in their pipelines. This gives them compatibility and reproducibility.

Unit testable: A component is a self-contained piece of code. It's easy to write unit test for a component.

Component and Pipeline

A machine learning pipeline is the workflow for a full machine learning task. Components are the building blocks of a machine learning pipeline. When you're thinking of a component, it must be under the context of pipeline.

To build components, the first thing is to define the machine learning pipeline. This requires breaking down the full machine learning task into a multi-step workflow. Each step is a component. For example, considering a simple machine learning task of using historical data to train a sales forecasting model, you may want to build a sequential workflow with data processing, model training, and model evaluation steps. For complex tasks, you may want to further break down. For example, split one single data processing step into data ingestion, data cleaning, data pre-processing, and feature engineering steps.

Once the steps in the workflow are defined, the next thing is to specify how each step is connected in the pipeline. For example, to connect your data processing step and model training step, you may want to define a data processing component to output a folder that contains the processed data. A training component takes a folder as input and outputs a folder that contains the trained model. These inputs and outputs definition will become part of your component interface definition.

Now, it's time to develop the code of executing a step. You can use your preferred languages (python, R, etc.). The code must be able to be executed by a shell command. During the development, you may want to add a few inputs to control how this step is going to be executed. For example, for a training step, you may like to add learning rate, number of epochs as the inputs to control the training. These additional inputs plus the inputs and outputs required to connect with other steps are the interface of the component. The argument of a shell command is used to pass inputs and outputs to the code. The environment to execute the command and the code needs to be specified. The environment could be a curated AzureML environment, a docker image or a conda environment.

Finally, you can package everything including code, cmd, environment, input, outputs, metadata together into a component. Then connects these components together to build pipelines for your machine learning workflow. One component can be used in multiple pipelines.

To learn more about how to build a component, see:

- How to [build a component using Azure MLCLI v2](#).
- How to [build a component using Azure ML SDK v2](#).

Next steps

- [Define component with the Azure ML CLI v2](#).
- [Define component with the Azure ML SDK v2](#).
- [Define component with Designer](#).
- [Component CLI v2 YAML reference](#).
- [What is Azure Machine Learning Pipeline?](#).
- Try out [CLI v2 component example](#).
- Try out [Python SDK v2 component example](#).

MLflow and Azure Machine Learning

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#) [Python SDK azure-ai-ml v2 \(preview\)](#)

[MLflow](#) is an open-source framework that's designed to manage the complete machine learning lifecycle. Its ability to train and serve models on different platforms allows you to use a consistent set of tools regardless of where your experiments are running: locally on your computer, on a remote compute target, on a virtual machine, or on an Azure Machine Learning compute instance.

TIP

Azure Machine Learning workspaces are MLflow-compatible, which means you can use Azure Machine Learning workspaces in the same way that you use an MLflow tracking server. Such compatibility has the following advantages:

- You can use Azure Machine Learning workspaces as your tracking server for any experiment you're running with MLflow, whether it runs on Azure Machine Learning or not. You only need to configure MLflow to point to the workspace where the tracking should happen.
- You can run any training routine that uses MLflow in Azure Machine Learning without changes. MLflow also supports model management and model deployment capabilities.

MLflow can manage the complete machine learning lifecycle by using four core capabilities:

- [Tracking](#) is a component of MLflow that logs and tracks your training job metrics, parameters, and model artifacts. It doesn't matter where your experiment's environment is--locally on your computer, on a remote compute target, on a virtual machine, or on an Azure Machine Learning compute instance.
- [Model Registry](#) is a component of MLflow that manages a model's versions in a centralized repository.
- [Model deployment](#) is a capability of MLflow that deploys models registered through the MLflow format to compute targets. Because of how MLflow models are stored, there's no need to provide scoring scripts for models in such a format.
- [Project](#) is a format for packaging data science code in a reusable and reproducible way, based primarily on conventions. It's supported in preview on Azure Machine Learning.

Tracking with MLflow

Azure Machine Learning uses MLflow Tracking for metric logging and artifact storage for your experiments, whether you created the experiments via the Azure Machine Learning Python SDK, the Azure Machine Learning CLI, or Azure Machine Learning studio. We recommend using MLflow for tracking experiments. To get started, see [Log metrics, parameters, and files with MLflow](#).

NOTE

Unlike the Azure Machine Learning SDK v1, there's no logging functionality in the SDK v2 (preview). We recommend that you use MLflow for logging.

With MLflow Tracking, you can connect Azure Machine Learning as the back end of your MLflow experiments. The workspace provides a centralized, secure, and scalable location to store training metrics and models. Capabilities include:

- [Track machine learning experiments and models running locally or in the cloud](#) with MLflow in Azure

Machine Learning.

- [Track Azure Databricks machine learning experiments with MLflow in Azure Machine Learning](#).
- [Track Azure Synapse Analytics machine learning experiments with MLflow in Azure Machine Learning](#).

IMPORTANT

- MLflow in R support is limited to tracking an experiment's metrics, parameters, and models on Azure Machine Learning jobs. RStudio or Jupyter Notebooks with R kernels are not supported. Model registries are not supported if you're using the MLflow R SDK. As an alternative, use the Azure Machine Learning CLI or Azure Machine Learning studio for model registration and management. View an [R example about using the MLflow tracking client with Azure Machine Learning](#).
- MLflow in Java support is limited to tracking an experiment's metrics and parameters on Azure Machine Learning jobs. Artifacts and models can't be tracked via the MLflow Java SDK. View a [Java example about using the MLflow tracking client with Azure Machine Learning](#).

To learn how to use MLflow to query experiments and runs in Azure Machine Learning, see [Manage experiments and runs with MLflow](#).

Model registries with MLflow

Azure Machine Learning supports MLflow for model management. This support represents a convenient way to support the entire model lifecycle for users who are familiar with the MLflow client.

To learn more about how to manage models by using the MLflow API in Azure Machine Learning, view [Manage model registries in Azure Machine Learning with MLflow](#).

Model deployments of MLflow models

You can [deploy MLflow models to Azure Machine Learning](#) so that you can apply the model management capabilities and no-code deployment offering in Azure Machine Learning. Azure Machine Learning supports deploying models to both real-time and batch endpoints. You can use the `azureml-mlflow` MLflow plug-in, the Azure Machine Learning CLI v2, and the user interface in Azure Machine Learning studio.

Learn more at [Deploy MLflow models to Azure Machine Learning](#).

Training MLflow projects (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

You can submit training jobs to Azure Machine Learning by using [MLflow projects](#) (preview). You can submit jobs locally with Azure Machine Learning tracking or migrate your jobs to the cloud via [Azure Machine Learning compute](#).

Learn more at [Train machine learning models with MLflow projects and Azure Machine Learning \(preview\)](#).

MLflow SDK, Azure Machine Learning v2, and Azure Machine Learning studio capabilities

The following table shows which operations are supported by each of the tools available in the machine

learning lifecycle.

FEATURE	MLFLOW SDK	AZURE MACHINE LEARNING V2 (CLI/SDK)	AZURE MACHINE LEARNING STUDIO
Track and log metrics, parameters, and models	✓		
Retrieve metrics, parameters, and models	✓ ¹	2	✓
Submit training jobs with MLflow projects	✓		
Submit training jobs with inputs and outputs		✓	✓
Submit training jobs by using machine learning pipelines		✓	
Manage experiments and runs	✓ ¹	✓	✓
Manage MLflow models	✓ ³	✓	✓
Manage non-MLflow models		✓	✓
Deploy MLflow models to Azure Machine Learning	✓ ⁴	✓	✓
Deploy non-MLflow models to Azure Machine Learning		✓	✓

NOTE

- ¹ View [Manage experiments and runs with MLflow](#) for details.
- ² Only artifacts and models can be downloaded.
- ³ View [Manage model registries in Azure Machine Learning with MLflow](#) for details.
- ⁴ View [Deploy MLflow models to Azure Machine Learning](#) for details. Deployment of MLflow models to batch inference by using the MLflow SDK is not possible at the moment.

Example notebooks

If you're getting started with MLflow in Azure Machine Learning, we recommend that you explore the [notebook examples about how to use MLflow](#):

- [Training and tracking an XGBoost classifier with MLflow](#): Demonstrates how to track experiments by using MLflow, log models, and combine multiple flavors into pipelines.
- [Training and tracking an XGBoost classifier with MLflow using service principal authentication](#): Demonstrates how to track experiments by using MLflow from compute that's running outside Azure Machine Learning. It shows how to authenticate against Azure Machine Learning services by using a service principal.
- [Hyper-parameter optimization using Hyperopt and nested runs in MLflow](#): Demonstrates how to use child

runs in MLflow to do hyper-parameter optimization for models by using the popular library Hyperopt. It shows how to transfer metrics, parameters, and artifacts from child runs to parent runs.

- [Logging models with MLflow](#): Demonstrates how to use the concept of models instead of artifacts with MLflow, including how to construct custom models.
- [Manage runs and experiments with MLflow](#): Demonstrates how to query experiments, runs, metrics, parameters, and artifacts from Azure Machine Learning by using MLflow.
- [Manage model registries with MLflow](#): Demonstrates how to manage models in registries by using MLflow.
- [Deploying models with MLflow](#): Demonstrates how to deploy no-code models in MLflow format to a deployment target in Azure Machine Learning.
- [Training models in Azure Databricks and deploying them on Azure Machine Learning](#): Demonstrates how to train models in Azure Databricks and deploy them in Azure Machine Learning. It also includes how to handle cases where you also want to track the experiments with the MLflow instance in Azure Databricks.
- [Migrating models with a scoring script to MLflow](#): Demonstrates how to migrate models with scoring scripts to no-code deployment with MLflow.
- [Using MLflow REST with Azure Machine Learning](#): Demonstrates how to work with the MLflow REST API when you're connected to Azure Machine Learning.

What is Responsible AI (preview)?

9/22/2022 • 7 minutes to read • [Edit Online](#)

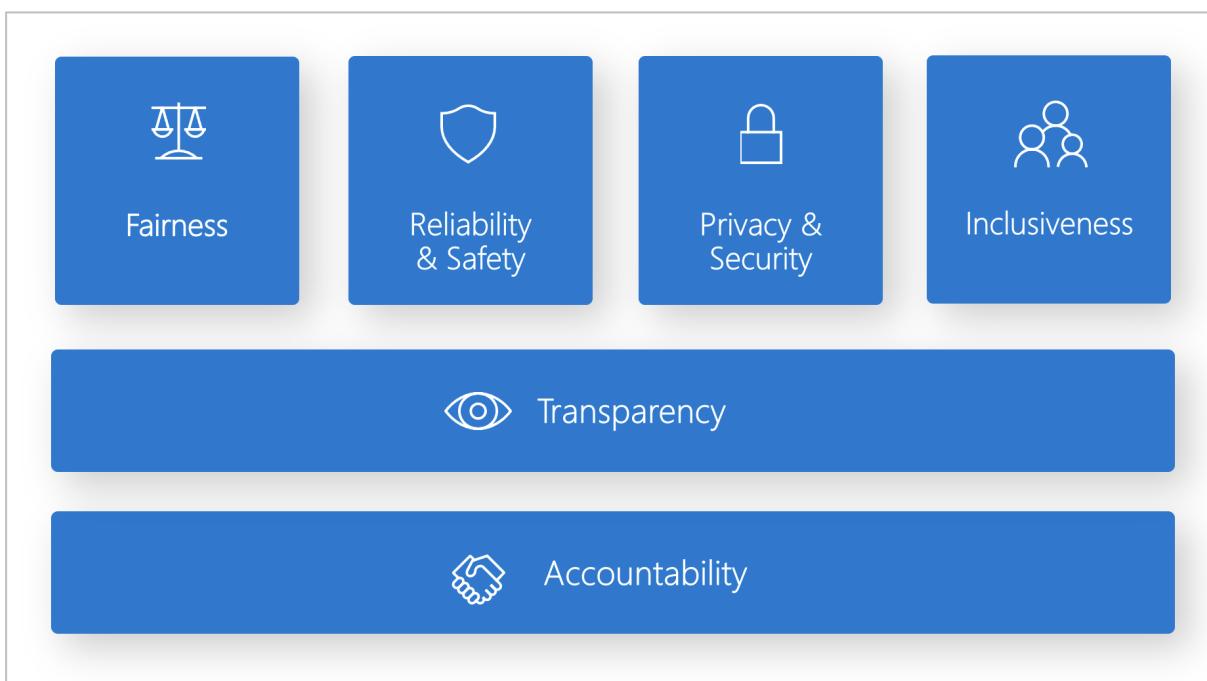
APPLIES TO:  Azure CLI ml extension v1  Python SDK azureml v1

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

Responsible Artificial Intelligence (Responsible AI) is an approach to developing, assessing, and deploying AI systems in a safe, trustworthy, and ethical way. AI systems are the product of many decisions made by those who develop and deploy them. From system purpose to how people interact with AI systems, Responsible AI can help proactively guide these decisions toward more beneficial and equitable outcomes. That means keeping people and their goals at the center of system design decisions and respecting enduring values like fairness, reliability, and transparency.

Microsoft has developed a [Responsible AI Standard](#). It's a framework for building AI systems according to six principles: fairness, reliability and safety, privacy and security, inclusiveness, transparency, and accountability. For Microsoft, these principles are the cornerstone of a responsible and trustworthy approach to AI, especially as intelligent technology becomes more prevalent in products and services that people use every day.

This article demonstrates how Azure Machine Learning supports tools for enabling developers and data scientists to implement and operationalize the six principles.



Fairness and inclusiveness

AI systems should treat everyone fairly and avoid affecting similarly situated groups of people in different ways. For example, when AI systems provide guidance on medical treatment, loan applications, or employment, they should make the same recommendations to everyone who has similar symptoms, financial circumstances, or professional qualifications.

Fairness and inclusiveness in Azure Machine Learning: The [fairness assessment](#) component of the [Responsible AI dashboard](#) enables data scientists and developers to assess model fairness across sensitive groups defined in terms of gender, ethnicity, age, and other characteristics.

Reliability and safety

To build trust, it's critical that AI systems operate reliably, safely, and consistently. These systems should be able to operate as they were originally designed, respond safely to unanticipated conditions, and resist harmful manipulation. How they behave and the variety of conditions they can handle reflect the range of situations and circumstances that developers anticipated during design and testing.

Reliability and safety in Azure Machine Learning: The [error analysis](#) component of the [Responsible AI dashboard](#) enables data scientists and developers to:

- Get a deep understanding of how failure is distributed for a model.
- Identify cohorts (subsets) of data with a higher error rate than the overall benchmark.

These discrepancies might occur when the system or model underperforms for specific demographic groups or for infrequently observed input conditions in the training data.

Transparency

When AI systems help inform decisions that have tremendous impacts on people's lives, it's critical that people understand how those decisions were made. For example, a bank might use an AI system to decide whether a person is creditworthy. A company might use an AI system to determine the most qualified candidates to hire.

A crucial part of transparency is *interpretability*: the useful explanation of the behavior of AI systems and their components. Improving interpretability requires stakeholders to comprehend how and why AI systems function the way they do. The stakeholders can then identify potential performance issues, fairness issues, exclusionary practices, or unintended outcomes.

Transparency in Azure Machine Learning: The [model interpretability](#) and [counterfactual what-if](#) components of the [Responsible AI dashboard](#) enable data scientists and developers to generate human-understandable descriptions of the predictions of a model.

The model interpretability component provides multiple views into a model's behavior:

- *Global explanations.* For example, what features affect the overall behavior of a loan allocation model?
- *Local explanations.* For example, why was a customer's loan application approved or rejected?
- *Model explanations for a selected cohort of data points.* For example, what features affect the overall behavior of a loan allocation model for low-income applicants?

The counterfactual what-if component enables understanding and debugging a machine learning model in terms of how it reacts to feature changes and perturbations.

Azure Machine Learning also supports a [Responsible AI scorecard](#). The scorecard is a customizable PDF report that developers can easily configure, generate, download, and share with their technical and non-technical stakeholders to educate them about their datasets and models health, achieve compliance, and build trust. This scorecard can also be used in audit reviews to uncover the characteristics of machine learning models.

Privacy and security

As AI becomes more prevalent, protecting privacy and securing personal and business information are becoming more important and complex. With AI, privacy and data security require close attention because access to data is essential for AI systems to make accurate and informed predictions and decisions about people. AI systems must comply with privacy laws that:

- Require transparency about the collection, use, and storage of data.
- Mandate that consumers have appropriate controls to choose how their data is used.

Privacy and security in Azure Machine Learning: Azure Machine Learning enables administrators and

developers to [create a secure configuration that complies](#) with their companies' policies. With Azure Machine Learning and the Azure platform, users can:

- Restrict access to resources and operations by user account or group.
- Restrict incoming and outgoing network communications.
- Encrypt data in transit and at rest.
- Scan for vulnerabilities.
- Apply and audit configuration policies.

Microsoft has also created two open-source packages that can enable further implementation of privacy and security principles:

- **SmartNoise**: Differential privacy is a set of systems and practices that help keep the data of individuals safe and private. In machine learning solutions, differential privacy might be required for regulatory compliance. SmartNoise is an open-source project (co-developed by Microsoft) that contains components for building differentially private systems that are global.
- **Counterfit**: Counterfit is an open-source project that comprises a command-line tool and generic automation layer to allow developers to simulate cyberattacks against AI systems. Anyone can download the tool and deploy it through Azure Cloud Shell to run in a browser, or deploy it locally in an Anaconda Python environment. It can assess AI models hosted in various cloud environments, on-premises, or in the edge. The tool is agnostic to AI models and supports various data types, including text, images, or generic input.

Accountability

The people who design and deploy AI systems must be accountable for how their systems operate.

Organizations should draw upon industry standards to develop accountability norms. These norms can ensure that AI systems aren't the final authority on any decision that affects people's lives. They can also ensure that humans maintain meaningful control over otherwise highly autonomous AI systems.

Accountability in Azure Machine Learning: [Machine learning operations \(MLOps\)](#) is based on DevOps principles and practices that increase the efficiency of AI workflows. Azure Machine Learning provides the following MLOps capabilities for better accountability of your AI systems:

- Register, package, and deploy models from anywhere. You can also track the associated metadata that's required to use the model.
- Capture the governance data for the end-to-end machine learning lifecycle. The logged lineage information can include who is publishing models, why changes were made, and when models were deployed or used in production.
- Notify and alert on events in the machine learning lifecycle. Examples include experiment completion, model registration, model deployment, and data drift detection.
- Monitor applications for operational issues and issues related to machine learning. Compare model inputs between training and inference, explore model-specific metrics, and provide monitoring and alerts on your machine learning infrastructure.

Besides the MLOps capabilities, the [Responsible AI scorecard](#) in Azure Machine Learning creates accountability by enabling cross-stakeholder communications. The scorecard also creates accountability by empowering developers to configure, download, and share their model health insights with their technical and non-technical stakeholders about AI data and model health. Sharing these insights can help build trust.

The machine learning platform also enables decision-making by informing business decisions through:

- Data-driven insights, to help stakeholders understand causal treatment effects on an outcome, by using historical data only. For example, "How would a medicine affect a patient's blood pressure?" These insights

are provided through the [causal inference](#) component of the [Responsible AI dashboard](#).

- Model-driven insights, to answer users' questions (such as "What can I do to get a different outcome from your AI next time?") so they can take action. Such insights are provided to data scientists through the [counterfactual what-if](#) component of the [Responsible AI dashboard](#).

Next steps

- For more information on how to implement Responsible AI in Azure Machine Learning, see [Responsible AI dashboard](#).
- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in your Responsible AI dashboard.
- Learn about the [Responsible AI Standard](#) for building AI systems according to six key principles.

Assess AI systems by using the Responsible AI dashboard (preview)

9/22/2022 • 11 minutes to read • [Edit Online](#)

Implementing Responsible AI in practice requires rigorous engineering. But rigorous engineering can be tedious, manual, and time-consuming without the right tooling and infrastructure.

The Responsible AI dashboard provides a single interface to help you implement Responsible AI in practice effectively and efficiently. It brings together several mature Responsible AI tools in the areas of:

- [Model performance and fairness assessment](#)
- Data exploration
- [Machine learning interpretability](#)
- [Error analysis](#)
- [Counterfactual analysis and perturbations](#)
- [Causal inference](#)

The dashboard offers a holistic assessment and debugging of models so you can make informed data-driven decisions. Having access to all of these tools in one interface empowers you to:

- Evaluate and debug your machine learning models by identifying model errors and fairness issues, diagnosing why those errors are happening, and informing your mitigation steps.
- Boost your data-driven decision-making abilities by addressing questions such as:

"What is the minimum change that users can apply to their features to get a different outcome from the model?"

"What is the causal effect of reducing or increasing a feature (for example, red meat consumption) on a real-world outcome (for example, diabetes progression)?"

You can customize the dashboard to include only the subset of tools that are relevant to your use case.

The Responsible AI dashboard is accompanied by a [PDF scorecard](#). The scorecard enables you to export Responsible AI metadata and insights into your data and models. You can then share them offline with the product and compliance stakeholders.

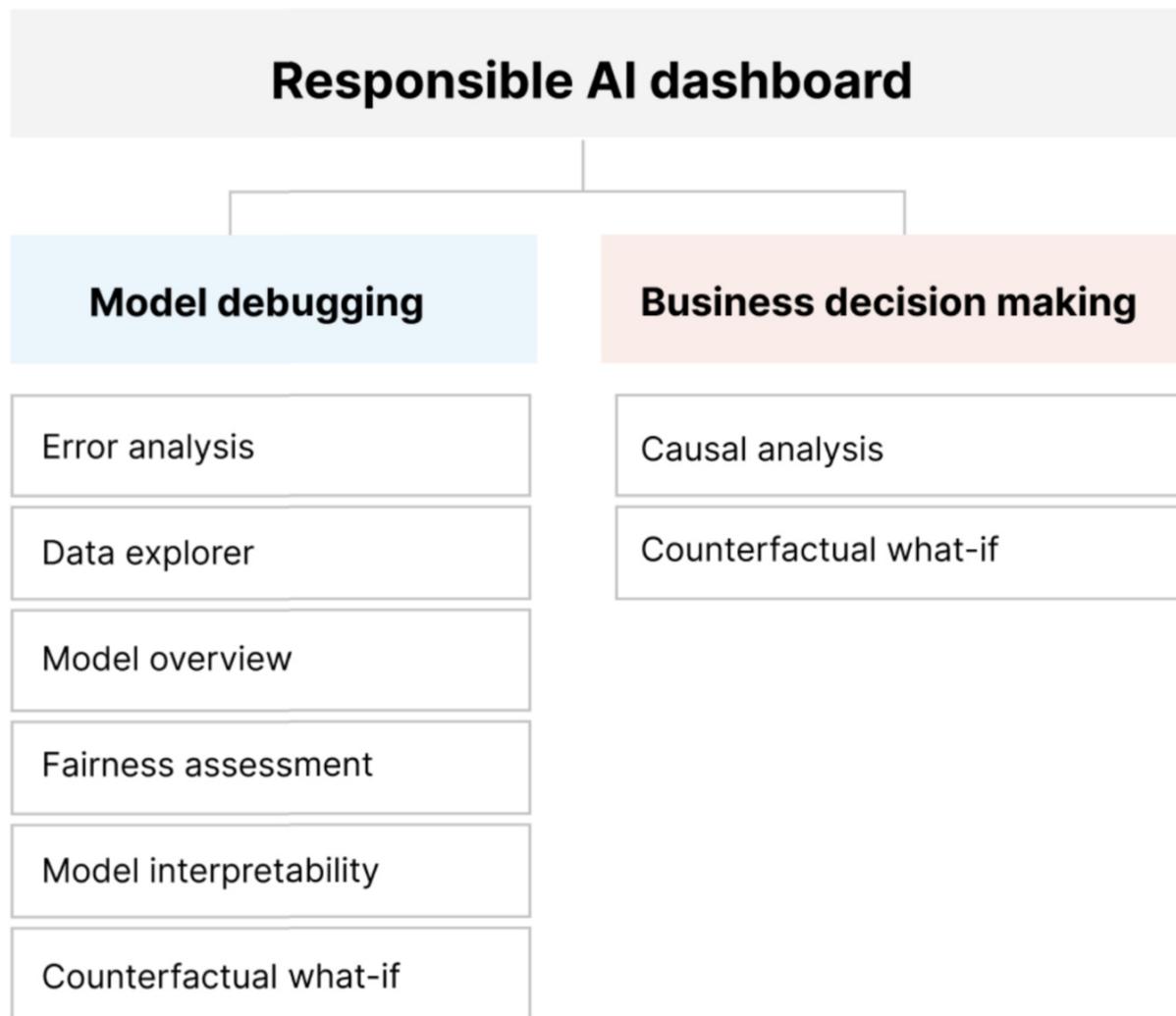
Responsible AI dashboard components

The Responsible AI dashboard brings together, in a comprehensive view, various new and pre-existing tools. The dashboard integrates these tools with [Azure Machine Learning CLI v2](#), [Azure Machine Learning Python SDK v2](#), and [Azure Machine Learning studio](#). The tools include:

- [Data explorer](#), to understand and explore your dataset distributions and statistics.
- [Model overview and fairness assessment](#), to evaluate the performance of your model and evaluate your model's group fairness issues (how your model's predictions affect diverse groups of people).
- [Error analysis](#), to view and understand how errors are distributed in your dataset.
- [Model interpretability](#) (importance values for aggregate and individual features), to understand your model's predictions and how those overall and individual predictions are made.
- [Counterfactual what-if](#), to observe how feature perturbations would affect your model predictions while providing the closest data points with opposing or different model predictions.

- **Causal analysis**, to use historical data to view the causal effects of treatment features on real-world outcomes.

Together, these tools will help you debug machine learning models, while informing your data-driven and model-driven business decisions. The following diagram shows how you can incorporate them into your AI lifecycle to improve your models and get solid data insights.



Model debugging

Assessing and debugging machine learning models is critical for model reliability, interpretability, fairness, and compliance. It helps determine how and why AI systems behave the way they do. You can then use this knowledge to improve model performance. Conceptually, model debugging consists of three stages:

1. **Identify**, to understand and recognize model errors and/or fairness issues by addressing the following questions:

"What kinds of errors does my model have?"

"In what areas are errors most prevalent?"

2. **Diagnose**, to explore the reasons behind the identified errors by addressing:

"What are the causes of these errors?"

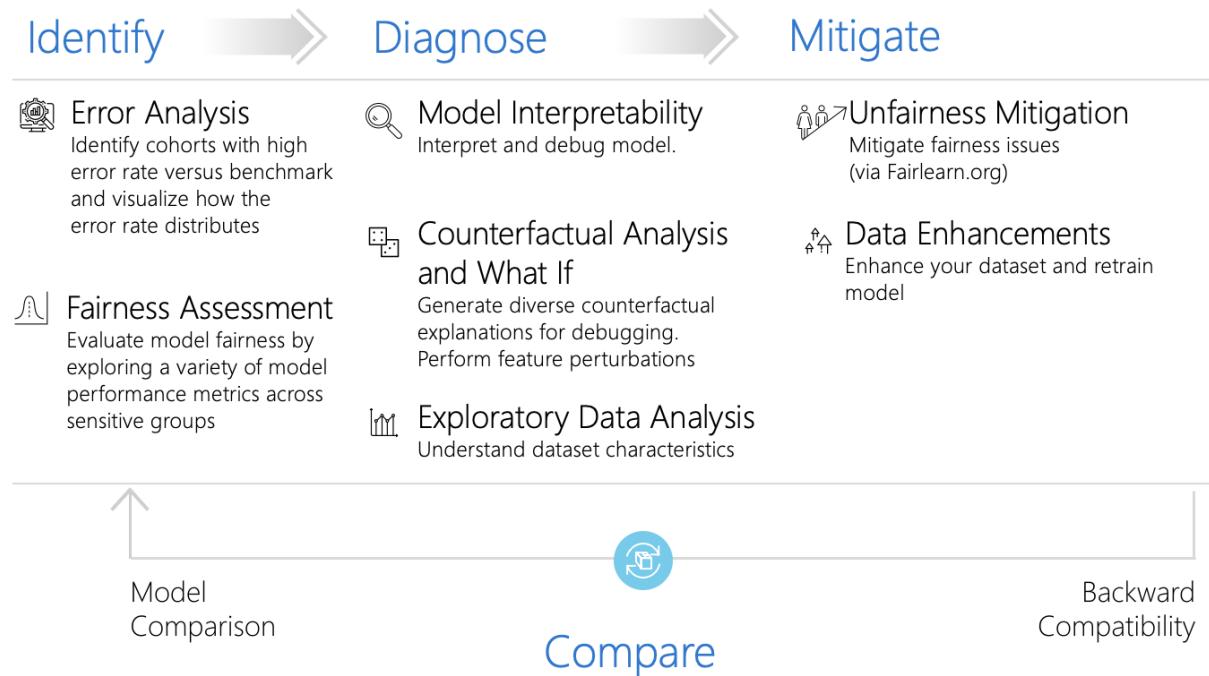
"Where should I focus my resources to improve my model?"

3. **Mitigate**, to use the identification and diagnosis insights from previous stages to take targeted mitigation steps and address questions such as:

"How can I improve my model?"

"What social or technical solutions exist for these issues?"

Model Debugging via Responsible AI dashboard



The following table describes when to use Responsible AI dashboard components to support model debugging:

STAGE	COMPONENT	DESCRIPTION
Identify	Error analysis	<p>The error analysis component helps you get a deeper understanding of model failure distribution and quickly identify erroneous cohorts (subgroups) of data.</p> <p>The capabilities of this component in the dashboard come from the Error Analysis package.</p>
Identify	Fairness analysis	<p>The fairness component defines groups in terms of sensitive attributes such as sex, race, and age. It then assesses how your model predictions affect these groups and how you can mitigate disparities. It evaluates the performance of your model by exploring the distribution of your prediction values and the values of your model performance metrics across the groups.</p> <p>The capabilities of this component in the dashboard come from the Fairlearn package.</p>

Stage	Component	Description
Identify	Model overview	The model overview component aggregates model assessment metrics in a high-level view of model prediction distribution for better investigation of its performance. This component also enables group fairness assessment by highlighting the breakdown of model performance across sensitive groups.
Diagnose	Data explorer	The data explorer visualizes datasets based on predicted and actual outcomes, error groups, and specific features. You can then identify issues of overrepresentation and underrepresentation, along with seeing how data is clustered in the dataset.
Diagnose	Model interpretability	<p>The interpretability component generates human-understandable explanations of the predictions of a machine learning model. It provides multiple views into a model's behavior:</p> <ul style="list-style-type: none"> - Global explanations (for example, which features affect the overall behavior of a loan allocation model) - Local explanations (for example, why an applicant's loan application was approved or rejected) <p>The capabilities of this component in the dashboard come from the InterpretML package.</p>
Diagnose	Counterfactual analysis and what-if	<p>This component consists of two functionalities for better error diagnosis:</p> <ul style="list-style-type: none"> - Generating a set of examples in which minimal changes to a particular point alter the model's prediction. That is, the examples show the closest data points with opposite model predictions. - Enabling interactive and custom what-if perturbations for individual data points to understand how the model reacts to feature changes. <p>The capabilities of this component in the dashboard come from the DICE package.</p>

Mitigation steps are available via standalone tools such as [Fairlearn](#). For more information, see the [unfairness mitigation algorithms](#).

Responsible decision-making

Decision-making is one of the biggest promises of machine learning. The Responsible AI dashboard can help you make informed business decisions through:

- Data-driven insights, to further understand causal treatment effects on an outcome by using historical data only. For example:

"How would a medicine affect a patient's blood pressure?"

"How would providing promotional values to certain customers affect revenue?"

These insights are provided through the [causal inference](#) component of the dashboard.

- Model-driven insights, to answer users' questions (such as "What can I do to get a different outcome from your AI next time?") so they can take action. These insights are provided to data scientists through the [counterfactual what-if](#) component.

Decision Making via Responsible AI dashboard

Understand data



Inform Actions

Exploratory-Data-Analysis Understand dataset characteristics

Causal Inference Understand the causal impact of your features on real-world outcomes

Counterfactual Analysis Generate diverse counterfactual explanations for informing end users

Exploratory data analysis, causal inference, and counterfactual analysis capabilities can help you make informed model-driven and data-driven decisions responsibly.

These components of the Responsible AI dashboard support responsible decision-making:

- **Data explorer:** You can reuse the data explorer component here to understand data distributions and to identify overrepresentation and underrepresentation. Data exploration is a critical part of decision making, because it isn't feasible to make informed decisions about a cohort that's underrepresented in the data.
- **Causal inference:** The causal inference component estimates how a real-world outcome changes in the presence of an intervention. It also helps construct promising interventions by simulating feature responses to various interventions and creating rules to determine which population cohorts would benefit from a particular intervention. Collectively, these functionalities allow you to apply new policies and effect real-world change.

The capabilities of this component come from the [EconML](#) package, which estimates heterogeneous treatment effects from observational data via machine learning.

- **Counterfactual analysis:** You can reuse the counterfactual analysis component here to generate minimum changes applied to a data point's features that lead to opposite model predictions. For example: Taylor would have obtained the loan approval from the AI if they earned \$10,000 more in annual income and had two fewer credit cards open.

Providing this information to users informs their perspective. It educates them on how they can take action to get the desired outcome from the AI in the future.

The capabilities of this component come from the [DiCE](#) package.

Reasons for using the Responsible AI dashboard

Although progress has been made on individual tools for specific areas of Responsible AI, data scientists often need to use various tools to holistically evaluate their models and data. For example: they might have to use model interpretability and fairness assessment together.

If data scientists discover a fairness issue with one tool, they then need to jump to a different tool to understand what data or model factors lie at the root of the issue before taking any steps on mitigation. The following factors further complicate this challenging process:

- There's no central location to discover and learn about the tools, extending the time it takes to research and learn new techniques.
- The different tools don't communicate with each other. Data scientists must wrangle the datasets, models, and other metadata as they pass them between the tools.
- The metrics and visualizations aren't easily comparable, and the results are hard to share.

The Responsible AI dashboard challenges this status quo. It's a comprehensive yet customizable tool that brings together fragmented experiences in one place. It enables you to seamlessly onboard to a single customizable framework for model debugging and data-driven decision-making.

By using the Responsible AI dashboard, you can create dataset cohorts, pass those cohorts to all of the supported components, and observe your model health for your identified cohorts. You can further compare insights from all supported components across a variety of prebuilt cohorts to perform disaggregated analysis and find the blind spots of your model.

When you're ready to share those insights with other stakeholders, you can extract them easily by using the [Responsible AI PDF scorecard](#). Attach the PDF report to your compliance reports, or share it with colleagues to build trust and get their approval.

Ways to customize the Responsible AI dashboard

The Responsible AI dashboard's strength lies in its customizability. It empowers users to design tailored, end-to-end model debugging and decision-making workflows that address their particular needs.

Need some inspiration? Here are some examples of how the dashboard's components can be put together to analyze scenarios in diverse ways:

RESPONSIBLE AI DASHBOARD FLOW	USE CASE
Model overview > error analysis > data explorer	To identify model errors and diagnose them by understanding the underlying data distribution
Model overview > fairness assessment > data explorer	To identify model fairness issues and diagnose them by understanding the underlying data distribution
Model overview > error analysis > counterfactuals analysis and what-if	To diagnose errors in individual instances with counterfactual analysis (minimum change to lead to a different model prediction)
Model overview > data explorer	To understand the root cause of errors and fairness issues introduced via data imbalances or lack of representation of a particular data cohort

RESPONSIBLE AI DASHBOARD FLOW	USE CASE
Model overview > interpretability	To diagnose model errors through understanding how the model has made its predictions
Data explorer > causal inference	To distinguish between correlations and causations in the data or decide the best treatments to apply to get a positive outcome
Interpretability > causal inference	To learn whether the factors that the model has used for prediction-making have any causal effect on the real-world outcome
Data explorer > counterfactuals analysis and what-if	To address customers' questions about what they can do next time to get a different outcome from an AI system

People who should use the Responsible AI dashboard

The following people can use the Responsible AI dashboard, and its corresponding [Responsible AI scorecard](#), to build trust with AI systems:

- Machine learning professionals and data scientists who are interested in debugging and improving their machine learning models before deployment
- Machine learning professionals and data scientists who are interested in sharing their model health records with product managers and business stakeholders to build trust and receive deployment permissions
- Product managers and business stakeholders who are reviewing machine learning models before deployment
- Risk officers who are reviewing machine learning models to understand fairness and reliability issues
- Providers of AI solutions who want to explain model decisions to users or help them improve the outcome
- Professionals in heavily regulated spaces who need to review machine learning models with regulators and auditors

Supported scenarios and limitations

- The Responsible AI dashboard currently supports regression and classification (binary and multi-class) models trained on tabular structured data.
- The Responsible AI dashboard currently supports MLflow models that are registered in Azure Machine Learning with a sklearn (scikit-learn) flavor only. The scikit-learn models should implement `predict()`/`predict_proba()` methods, or the model should be wrapped within a class that implements `predict()`/`predict_proba()` methods. The models must be loadable in the component environment and must be pickleable.
- The Responsible AI dashboard currently visualizes up to 5K of your data points on the dashboard UI. You should downsample your dataset to 5K or less before passing it to the dashboard.
- The dataset inputs to the Responsible AI dashboard must be pandas DataFrames in Parquet format. NumPy and SciPy sparse data is currently not supported.
- The Responsible AI dashboard currently supports numeric or categorical features. For categorical features, the user has to explicitly specify the feature names.
- The Responsible AI dashboard currently doesn't support datasets with more than 10K columns.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).

- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed on the Responsible AI dashboard.

Model interpretability (preview)

9/22/2022 • 7 minutes to read • [Edit Online](#)

This article describes methods you can use for model interpretability in Azure Machine Learning.

IMPORTANT

With the release of the Responsible AI dashboard, which includes model interpretability, we recommend that you migrate to the new experience, because the older SDK v1 preview model interpretability dashboard will no longer be actively maintained.

Why model interpretability is important to model debugging

When you're using machine learning models in ways that affect people's lives, it's critically important to understand what influences the behavior of models. Interpretability helps answer questions in scenarios such as:

- Model debugging: Why did my model make this mistake? How can I improve my model?
- Human-AI collaboration: How can I understand and trust the model's decisions?
- Regulatory compliance: Does my model satisfy legal requirements?

The interpretability component of the [Responsible AI dashboard](#) contributes to the "diagnose" stage of the model lifecycle workflow by generating human-understandable descriptions of the predictions of a machine learning model. It provides multiple views into a model's behavior:

- Global explanations: For example, what features affect the overall behavior of a loan allocation model?
- Local explanations: For example, why was a customer's loan application approved or rejected?

You can also observe model explanations for a selected cohort as a subgroup of data points. This approach is valuable when, for example, you're assessing fairness in model predictions for individuals in a particular demographic group. The **Local explanation** tab of this component also represents a full data visualization, which is great for general eyeballing of the data and looking at differences between correct and incorrect predictions of each cohort.

The capabilities of this component are founded by the [InterpretML](#) package, which generates model explanations.

Use interpretability when you need to:

- Determine how trustworthy your AI system's predictions are by understanding what features are most important for the predictions.
- Approach the debugging of your model by understanding it first and identifying whether the model is using healthy features or merely false correlations.
- Uncover potential sources of unfairness by understanding whether the model is basing predictions on sensitive features or on features that are highly correlated with them.
- Build user trust in your model's decisions by generating local explanations to illustrate their outcomes.
- Complete a regulatory audit of an AI system to validate models and monitor the impact of model decisions on humans.

How to interpret your model

In machine learning, *features* are the data fields you use to predict a target data point. For example, to predict

credit risk, you might use data fields for age, account size, and account age. Here, age, account size, and account age are features. Feature importance tells you how each data field affects the model's predictions. For example, although you might use age heavily in the prediction, account size and account age might not affect the prediction values significantly. Through this process, data scientists can explain resulting predictions in ways that give stakeholders visibility into the model's most important features.

By using the classes and methods in the Responsible AI dashboard and by using SDK v2 and CLI v2, you can:

- Explain model prediction by generating feature-importance values for the entire model (global explanation) or individual data points (local explanation).
- Achieve model interpretability on real-world datasets at scale.
- Use an interactive visualization dashboard to discover patterns in your data and its explanations at training time.

By using the classes and methods in the SDK v1, you can:

- Explain model prediction by generating feature-importance values for the entire model or individual data points.
- Achieve model interpretability on real-world datasets at scale during training and inference.
- Use an interactive visualization dashboard to discover patterns in your data and its explanations at training time.

Model interpretability classes are made available through the SDK v1 package. For more information, see [Install SDK packages for Azure Machine Learning](#) and `azureml.interpret`.

Supported model interpretability techniques

The Responsible AI dashboard and `azureml-interpret` use the interpretability techniques that were developed in [Interpret-Community](#), an open-source Python package for training interpretable models and helping to explain opaque-box AI systems. Opaque-box models are those for which we have no information about their internal workings.

Interpret-Community serves as the host for the following supported explainers, and currently supports the interpretability techniques presented in the next sections.

Supported in Responsible AI dashboard in Python SDK v2 and CLI v2

INTERPRETABILITY TECHNIQUE	DESCRIPTION	TYPE
----------------------------	-------------	------

INTERPRETABILITY TECHNIQUE	DESCRIPTION	TYPE
Mimic Explainer (Global Surrogate) + SHAP tree	<p>Mimic Explainer is based on the idea of training global surrogate models to mimic opaque-box models. A global surrogate model is an intrinsically interpretable model that's trained to approximate the predictions of any opaque-box model as accurately as possible.</p> <p>Data scientists can interpret the surrogate model to draw conclusions about the opaque-box model. The Responsible AI dashboard uses LightGBM (<code>LGBMExplainableModel</code>), paired with the SHAP (<code>SHapley Additive exPlanations</code>) Tree Explainer, which is a specific explainer to trees and ensembles of trees. The combination of LightGBM and SHAP tree provides model-agnostic global and local explanations of your machine learning models.</p>	Model-agnostic

Supported in Python SDK v1

INTERPRETABILITY TECHNIQUE	DESCRIPTION	TYPE
SHAP Tree Explainer	The SHAP Tree Explainer, which focuses on a polynomial, time-fast, SHAP value-estimation algorithm that's specific to <i>trees and ensembles of trees</i> .	Model-specific
SHAP Deep Explainer	Based on the explanation from SHAP, Deep Explainer is a "high-speed approximation algorithm for SHAP values in deep learning models that builds on a connection with DeepLIFT described in the SHAP NIPS paper . <i>TensorFlow</i> models and <i>Keras</i> models using the TensorFlow back end are supported (there's also preliminary support for PyTorch)."	Model-specific
SHAP Linear Explainer	The SHAP Linear Explainer computes SHAP values for a <i>linear model</i> , optionally accounting for inter-feature correlations.	Model-specific
SHAP Kernel Explainer	The SHAP Kernel Explainer uses a specially weighted local linear regression to estimate SHAP values for <i>any model</i> .	Model-agnostic

INTERPRETABILITY TECHNIQUE	DESCRIPTION	TYPE
Mimic Explainer (Global Surrogate)	Mimic Explainer is based on the idea of training global surrogate models to mimic opaque-box models. A global surrogate model is an intrinsically interpretable model that's trained to approximate the predictions of <i>any opaque-box model</i> as accurately as possible. Data scientists can interpret the surrogate model to draw conclusions about the opaque-box model. You can use one of the following interpretable models as your surrogate model: LightGBM (<code>LGBMExplainableModel</code>), Linear Regression (<code>LinearExplainableModel</code>), Stochastic Gradient Descent explainable model (<code>SGDExplainableModel</code>), or Decision Tree (<code>DecisionTreeExplainableModel</code>).	Model-agnostic
Permutation Feature Importance Explainer	Permutation Feature Importance (PFI) is a technique used to explain classification and regression models that's inspired by Breiman's Random Forests paper (see section 10). At a high level, the way it works is by randomly shuffling data one feature at a time for the entire dataset and calculating how much the performance metric of interest changes. The larger the change, the more important that feature is. PFI can explain the overall behavior of <i>any underlying model</i> but doesn't explain individual predictions.	Model-agnostic

Besides the interpretability techniques described above, we support another SHAP-based explainer, called Tabular Explainer. Depending on the model, Tabular Explainer uses one of the supported SHAP explainers:

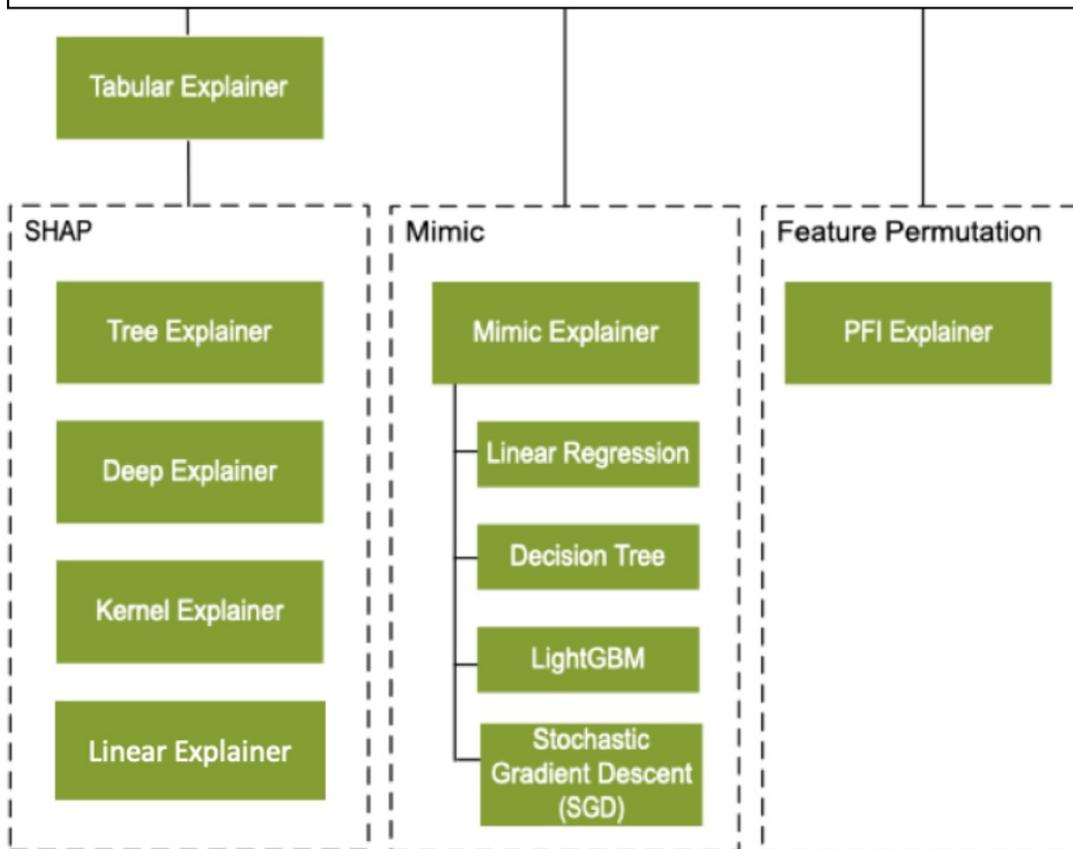
- Tree Explainer for all tree-based models
- Deep Explainer for deep neural network (DNN) models
- Linear Explainer for linear models
- Kernel Explainer for all other models

Tabular Explainer has also made significant feature and performance enhancements over the direct SHAP explainers:

- **Summarization of the initialization dataset:** When speed of explanation is most important, we summarize the initialization dataset and generate a small set of representative samples. This approach speeds up the generation of overall and individual feature importance values.
- **Sampling the evaluation data set:** If you pass in a large set of evaluation samples but don't actually need all of them to be evaluated, you can set the sampling parameter to `true` to speed up the calculation of overall model explanations.

The following diagram shows the current structure of supported explainers:

Tabular Data Interpretability Techniques



Supported machine learning models

The `azureml.interpret` package of the SDK supports models that are trained with the following dataset formats:

- `numpy.array`
- `pandas.DataFrame`
- `iml.datatypes.DenseData`
- `scipy.sparse.csr_matrix`

The explanation functions accept both models and pipelines as input. If a model is provided, it must implement the prediction function `predict` or `predict_proba` that conforms to the Scikit convention. If your model doesn't support this, you can wrap it in a function that generates the same outcome as `predict` or `predict_proba` in Scikit and use that wrapper function with the selected explainer.

If you provide a pipeline, the explanation function assumes that the running pipeline script returns a prediction. When you use this wrapping technique, `azureml.interpret` can support models that are trained via PyTorch, TensorFlow, and Keras deep learning frameworks as well as classic machine learning models.

Local and remote compute target

The `azureml.interpret` package is designed to work with both local and remote compute targets. If you run the package locally, the SDK functions won't contact any Azure services.

You can run the explanation remotely on Azure Machine Learning Compute and log the explanation info into the Azure Machine Learning Run History Service. After this information is logged, reports and visualizations from the explanation are readily available on Azure Machine Learning studio for analysis.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI v2 and SDK v2](#) or the [Azure Machine Learning studio UI](#).
- Explore the [supported interpretability visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.
- Learn how to enable [interpretability for automated machine learning models](#).

Model performance and fairness (preview)

9/22/2022 • 5 minutes to read • [Edit Online](#)

This article describes methods that you can use to understand your model performance and fairness in Azure Machine Learning.

What is machine learning fairness?

Artificial intelligence and machine learning systems can display unfair behavior. One way to define unfair behavior is by its harm, or its impact on people. AI systems can give rise to many types of harm. To learn more, see the [NeurIPS 2017 keynote by Kate Crawford](#).

Two common types of AI-caused harms are:

- **Harm of allocation:** An AI system extends or withholds opportunities, resources, or information for certain groups. Examples include hiring, school admissions, and lending, where a model might be better at picking good candidates among a specific group of people than among other groups.
- **Harm of quality-of-service:** An AI system doesn't work as well for one group of people as it does for another. For example, a voice recognition system might fail to work as well for women as it does for men.

To reduce unfair behavior in AI systems, you have to assess and mitigate these harms. The *model overview* component of the [Responsible AI dashboard](#) contributes to the identification stage of the model lifecycle by generating model performance metrics for your entire dataset and your identified cohorts of data. It generates these metrics across subgroups identified in terms of sensitive features or sensitive attributes.

NOTE

Fairness is a socio-technical challenge. Quantitative fairness metrics don't capture many aspects of fairness, such as justice and due process. Also, many quantitative fairness metrics can't all be satisfied simultaneously.

The goal of the Fairlearn open-source package is to enable humans to assess the impact and mitigation strategies. Ultimately, it's up to the humans who build AI and machine learning models to make trade-offs that are appropriate for their scenarios.

In this component of the Responsible AI dashboard, fairness is conceptualized through an approach known as *group fairness*. This approach asks: "Which groups of individuals are at risk for experiencing harm?" The term *sensitive features* suggests that the system designer should be sensitive to these features when assessing group fairness.

During the assessment phase, fairness is quantified through *disparity metrics*. These metrics can evaluate and compare model behavior across groups either as ratios or as differences. The Responsible AI dashboard supports two classes of disparity metrics:

- **Disparity in model performance:** These sets of metrics calculate the disparity (difference) in the values of the selected performance metric across subgroups of data. Here are a few examples:
 - Disparity in accuracy rate
 - Disparity in error rate
 - Disparity in precision
 - Disparity in recall
 - Disparity in mean absolute error (MAE)

- **Disparity in selection rate:** This metric contains the difference in selection rate (favorable prediction) among subgroups. An example of this is disparity in loan approval rate. Selection rate means the fraction of data points in each class classified as 1 (in binary classification) or distribution of prediction values (in regression).

The fairness assessment capabilities of this component come from the [Fairlearn](#) package. Fairlearn provides a collection of model fairness assessment metrics and unfairness mitigation algorithms.

NOTE

A fairness assessment is not a purely technical exercise. The Fairlearn open-source package can identify quantitative metrics to help you assess the fairness of a model, but it won't perform the assessment for you. You must perform a qualitative analysis to evaluate the fairness of your own models. The sensitive features noted earlier are an example of this kind of qualitative analysis.

Parity constraints for mitigating unfairness

After you understand your model's fairness issues, you can use the mitigation algorithms in the [Fairlearn](#) open-source package to mitigate those issues. These algorithms support a set of constraints on the predictor's behavior called *parity constraints* or criteria.

Parity constraints require some aspects of the predictor's behavior to be comparable across the groups that sensitive features define (for example, different races). The mitigation algorithms in the Fairlearn open-source package use such parity constraints to mitigate the observed fairness issues.

NOTE

The unfairness mitigation algorithms in the Fairlearn open-source package can provide suggested mitigation strategies to reduce unfairness in a machine learning model, but those strategies don't eliminate unfairness. Developers might need to consider other parity constraints or criteria for their machine learning models. Developers who use Azure Machine Learning must determine for themselves if the mitigation sufficiently reduces unfairness in their intended use and deployment of machine learning models.

The Fairlearn package supports the following types of parity constraints:

PARTY CONSTRAINT	PURPOSE	MACHINE LEARNING TASK
Demographic parity	Mitigate allocation harms	Binary classification, regression
Equalized odds	Diagnose allocation and quality-of-service harms	Binary classification
Equal opportunity	Diagnose allocation and quality-of-service harms	Binary classification
Bounded group loss	Mitigate quality-of-service harms	Regression

Mitigation algorithms

The Fairlearn open-source package provides two types of unfairness mitigation algorithms:

- **Reduction:** These algorithms take a standard black-box machine learning estimator (for example, a LightGBM model) and generate a set of retrained models by using a sequence of reweighted training datasets.

For example, applicants of a certain gender might be upweighted or downweighted to retrain models and reduce disparities across gender groups. Users can then pick a model that provides the best trade-off between accuracy (or another performance metric) and disparity, based on their business rules and cost calculations.

- **Post-processing:** These algorithms take an existing classifier and a sensitive feature as input. They then derive a transformation of the classifier's prediction to enforce the specified fairness constraints. The biggest advantage of one post-processing algorithm, threshold optimization, is its simplicity and flexibility because it doesn't need to retrain the model.

ALGORITHM	DESCRIPTION	MACHINE LEARNING TASK	SENSITIVE FEATURES	SUPPORTED PARITY CONSTRAINTS	ALGORITHM TYPE
<code>ExponentiatedGradient</code>	Black-box approach to fair classification described in A Reductions Approach to Fair Classification .	Binary classification	Categorical	Demographic parity, equalized odds	Reduction
<code>GridSearch</code>	Black-box approach described in A Reductions Approach to Fair Classification .	Binary classification	Binary	Demographic parity, equalized odds	Reduction
<code>GridSearch</code>	Black-box approach that implements a grid-search variant of fair regression with the algorithm for bounded group loss described in Fair Regression: Quantitative Definitions and Reduction-based Algorithms .	Regression	Binary	Bounded group loss	Reduction

ALGORITHM	DESCRIPTION	MACHINE LEARNING TASK	SENSITIVE FEATURES	SUPPORTED PARITY CONSTRAINTS	ALGORITHM TYPE
ThresholdOptimizer	Postprocessing algorithm based on the paper Equality of Opportunity in Supervised Learning . This technique takes as input an existing classifier and a sensitive feature. Then, it derives a monotone transformation of the classifier's prediction to enforce the specified parity constraints.	Binary classification	Categorical	Demographic parity, equalized odds	Post-processing

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Explore the [supported model overview and fairness assessment visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.
- Learn how to use the components by checking out Fairlearn's [GitHub repository](#), [user guide](#), [examples](#), and [sample notebooks](#).

Make data-driven policies and influence decision-making (preview)

9/22/2022 • 3 minutes to read • [Edit Online](#)

Machine learning models are powerful in identifying patterns in data and making predictions. But they offer little support for estimating how the real-world outcome changes in the presence of an intervention.

Practitioners have become increasingly focused on using historical data to inform their future decisions and business interventions. For example, how would the revenue be affected if a corporation pursued a new pricing strategy? Would a new medication improve a patient's condition, all else equal?

The *causal inference* component of the [Responsible AI dashboard](#) addresses these questions by estimating the effect of a feature on an outcome of interest on average, across a population or a cohort, and on an individual level. It also helps construct promising interventions by simulating feature responses to various interventions and creating rules to determine which population cohorts would benefit from an intervention. Collectively, these functionalities allow decision-makers to apply new policies and effect real-world change.

The capabilities of this component come from the [EconML](#) package. It estimates heterogeneous treatment effects from observational data via the [double machine learning](#) technique.

Use causal inference when you need to:

- Identify the features that have the most direct effect on your outcome of interest.
- Decide what overall treatment policy to take to maximize real-world impact on an outcome of interest.
- Understand how individuals with certain feature values would respond to a particular treatment policy.

How are causal inference insights generated?

NOTE

Only historical data is required to generate causal insights. The causal effects computed based on the treatment features are purely a data property. So, a trained model is optional when you're computing the causal effects.

Double machine learning is a method for estimating heterogeneous treatment effects when all potential confounders/controls (factors that simultaneously had a direct effect on the treatment decision in the collected data and the observed outcome) are observed but either of the following problems exists:

- There are too many for classical statistical approaches to be applicable. That is, they're *high-dimensional*.
- Their effect on the treatment and outcome can't be satisfactorily modeled by parametric functions. That is, they're *non-parametric*.

You can use machine learning techniques to address both problems. For an example, see [Chernozhukov2016](#).

Double machine learning reduces the problem by first estimating two predictive tasks:

- Predicting the outcome from the controls
- Predicting the treatment from the controls

Then the method combines these two predictive models in a final-stage estimation to create a model of the heterogeneous treatment effect. This approach allows for arbitrary machine learning algorithms to be used for the two predictive tasks while maintaining many favorable statistical properties related to the final model. These

properties include small mean squared error, asymptotic normality, and construction of confidence intervals.

What other tools does Microsoft provide for causal inference?

- [Project Azua](#) provides a novel framework that focuses on end-to-end causal inference.

Azua's DECI (deep end-to-end causal inference) technology is a single model that can simultaneously do causal discovery and causal inference. The user provides data, and the model can output the causal relationships among all variables.

By itself, this approach can provide insights into the data. It enables the calculation of metrics such as individual treatment effect (ITE), average treatment effect (ATE), and conditional average treatment effect (CATE). You can then use these calculations to make optimal decisions.

The framework is scalable for large data, in terms of both the number of variables and the number of data points. It can also handle missing data entries with mixed statistical types.

- [EconML](#) powers the back end of the Responsible AI dashboard's causal inference component. It's a Python package that applies machine learning techniques to estimate individualized causal responses from observational or experimental data.

The suite of estimation methods in EconML represents the latest advances in causal machine learning. By incorporating individual machine learning steps into interpretable causal models, these methods improve the reliability of what-if predictions and make causal analysis quicker and easier for a broad set of users.

- [DoWhy](#) is a Python library that aims to spark causal thinking and analysis. DoWhy provides a principled four-step interface for causal inference that focuses on explicitly modeling causal assumptions and validating them as much as possible.

The key feature of DoWhy is its state-of-the-art refutation API that can automatically test causal assumptions for any estimation method. It makes inference more robust and accessible to non-experts.

DoWhy supports estimation of the average causal effect for back-door, front-door, instrumental variable, and other identification methods. It also supports estimation of the CATE through an integration with the EconML library.

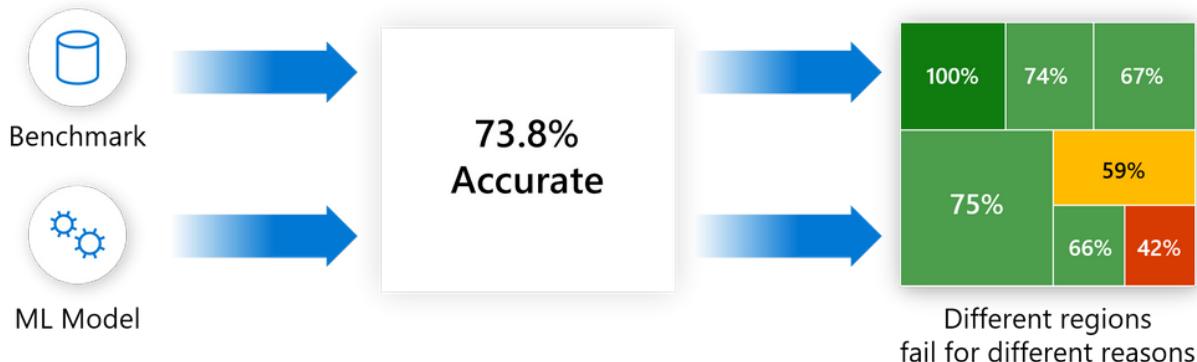
Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Explore the [supported causal inference visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

Assess errors in machine learning models (preview)

9/22/2022 • 2 minutes to read • [Edit Online](#)

One of the biggest challenges with current model-debugging practices is using aggregate metrics to score models on a benchmark dataset. Model accuracy might not be uniform across subgroups of data, and there might be input cohorts for which the model fails more often. The direct consequences of these failures are a lack of reliability and safety, the appearance of fairness issues, and a loss of trust in machine learning altogether.



Error analysis moves away from aggregate accuracy metrics. It exposes the distribution of errors to developers in a transparent way, and it enables them to identify and diagnose errors efficiently.

The error analysis component of the [Responsible AI dashboard](#) provides machine learning practitioners with a deeper understanding of model failure distribution and helps them quickly identify erroneous cohorts of data. This component identifies the cohorts of data with a higher error rate versus the overall benchmark error rate. It contributes to the identification stage of the model lifecycle workflow through:

- A decision tree that reveals cohorts with high error rates.
- A heatmap that visualizes how input features affect the error rate across cohorts.

Discrepancies in errors might occur when the system underperforms for specific demographic groups or infrequently observed input cohorts in the training data.

The capabilities of this component come from the [Error Analysis](#) package, which generates model error profiles.

Use error analysis when you need to:

- Gain a deep understanding of how model failures are distributed across a dataset and across several input and feature dimensions.
- Break down the aggregate performance metrics to automatically discover erroneous cohorts in order to inform your targeted mitigation steps.

Error tree

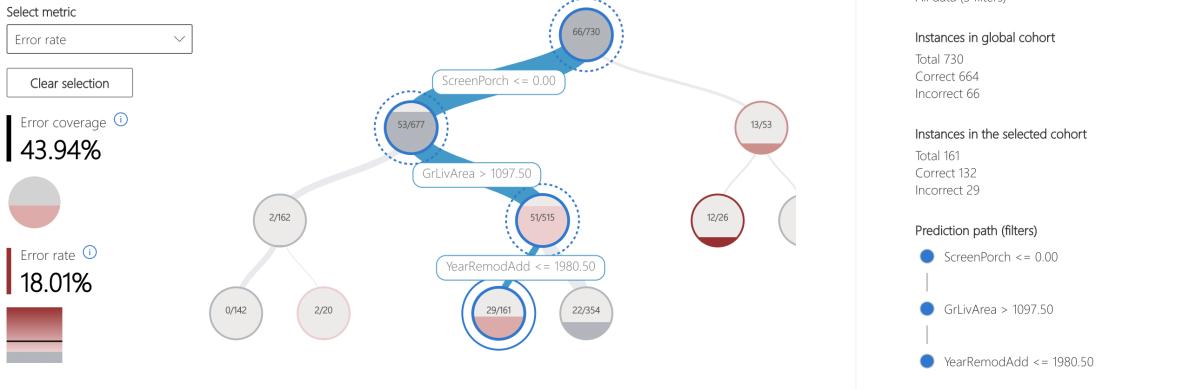
Often, error patterns are complex and involve more than one or two features. Developers might have difficulty exploring all possible combinations of features to discover hidden data pockets with critical failures.

To alleviate the burden, the binary tree visualization automatically partitions the benchmark data into interpretable subgroups that have unexpectedly high or low error rates. In other words, the tree uses the input features to maximally separate model error from success. For each node that defines a data subgroup, users can investigate the following information:

- **Error rate:** A portion of instances in the node for which the model is incorrect. It's shown through the intensity of the red color.
- **Error coverage:** A portion of all errors that fall into the node. It's shown through the fill rate of the node.
- **Data representation:** The number of instances in each node of the error tree. It's shown through the thickness of the incoming edge to the node, along with the total number of instances in the node.

Tree map Heat map Feature list

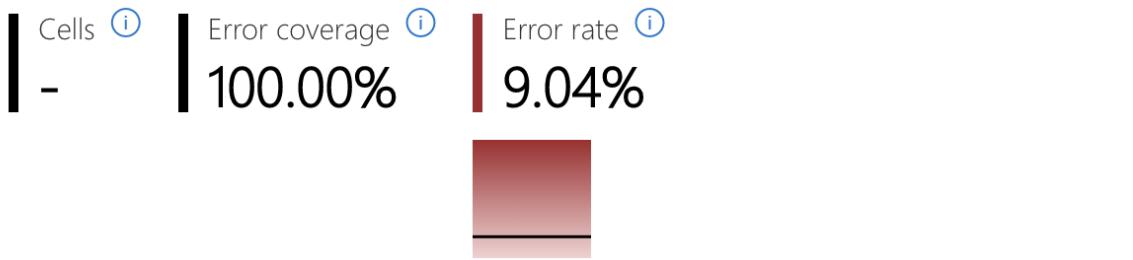
The tree visualization uses the mutual information between each feature and the error to best separate error instances from success instances hierarchically in the data. This simplifies the process of discovering and highlighting common failure patterns. To find important failure patterns, look for nodes with a stronger red color (i.e., high error rate) and a higher fill line (i.e., high error coverage). To edit the list of features being used in the tree, click on "Feature list." Use the "select metric" dropdown menu to learn more about your error and success nodes' performance. Please note that this metric selection will not impact the way your error tree is generated.



Error heatmap

The view slices the data based on a one-dimensional or two-dimensional grid of input features. Users can choose the input features of interest for analysis.

The heatmap visualizes cells with high error by using a darker red color to bring the user's attention to those regions. This feature is especially beneficial when the error themes are different across partitions, which happens often in practice. In this error identification view, the analysis is highly guided by the users and their knowledge or hypotheses of what features might be most important for understanding failures.



Select metric

Error rate v

Rows: Feature 1

Fireplaces v

Columns: Feature 2

YearBuilt v

Clear all Select all Quantile binning i Off Binning threshold i 5



37e+3,1.90e- 30e+3,1.93e- 33e+3,1.95e- 35e+3,1.98e- 38e+3,2.01e-

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Explore the [supported error analysis visualizations](#).
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

Understand your datasets (preview)

9/22/2022 • 2 minutes to read • [Edit Online](#)

Machine learning models "learn" from historical decisions and actions captured in training data. As a result, their performance in real-world scenarios is heavily influenced by the data they're trained on. When feature distribution in a dataset is skewed, it can cause a model to incorrectly predict data points that belong to an underrepresented group or to be optimized along an inappropriate metric.

For example, while a model was training an AI system for predicting house prices, the training set was representing 75 percent of newer houses that had less than median prices. As a result, it was much less accurate in successfully identifying more expensive historic houses. The fix was to add older and expensive houses to the training data and augment the features to include insights about historical value. That data augmentation improved results.

The data explorer component of the [Responsible AI dashboard](#) helps visualize datasets based on predicted and actual outcomes, error groups, and specific features. It helps you identify issues of overrepresentation and underrepresentation and to see how data is clustered in the dataset. Data visualizations consist of aggregate plots or individual data points.

When to use the data explorer

Use the data explorer when you need to:

- Explore your dataset statistics by selecting different filters to slice your data into different dimensions (also known as cohorts).
- Understand the distribution of your dataset across different cohorts and feature groups.
- Determine whether your findings related to fairness, error analysis, and causality (derived from other dashboard components) are a result of your dataset's distribution.
- Decide in which areas to collect more data to mitigate errors that come from representation issues, label noise, feature noise, label bias, and similar factors.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLI and SDK](#) or [Azure Machine Learning studio UI](#).
- Explore the [supported data explorer visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

Counterfactuals analysis and what-if (preview)

9/22/2022 • 2 minutes to read • [Edit Online](#)

What-if counterfactuals address the question of what the model would predict if you changed the action input. They enable understanding and debugging of a machine learning model in terms of how it reacts to input (feature) changes.

Standard interpretability techniques approximate a machine learning model or rank features by their predictive importance. By contrast, counterfactual analysis "interrogates" a model to determine what changes to a particular data point would flip the model decision.

Such an analysis helps in disentangling the impact of correlated features in isolation. It also helps you get a more nuanced understanding of how much of a feature change is needed to see a model decision flip for classification models and a decision change for regression models.

The *counterfactual analysis and what-if* component of the [Responsible AI dashboard](#) has two functions:

- Generate a set of examples with minimal changes to a particular point such that they change the model's prediction (showing the closest data points with opposite model predictions).
- Enable users to generate their own what-if perturbations to understand how the model reacts to feature changes.

One of the top differentiators of the Responsible AI dashboard's counterfactual analysis component is the fact that you can identify which features to vary and their permissible ranges for valid and logical counterfactual examples.

The capabilities of this component come from the [DiCE](#) package.

Use what-if counterfactuals when you need to:

- Examine fairness and reliability criteria as a decision evaluator by perturbing sensitive attributes such as gender and ethnicity, and then observing whether model predictions change.
- Debug specific input instances in depth.
- Provide solutions to users and determine what they can do to get a desirable outcome from the model.

How are counterfactual examples generated?

To generate counterfactuals, DiCE implements a few model-agnostic techniques. These methods apply to any opaque-box classifier or regressor. They're based on sampling nearby points to an input point, while optimizing a loss function based on proximity (and optionally, sparsity, diversity, and feasibility). Currently supported methods are:

- **Randomized search:** This method samples points randomly near a query point and returns counterfactuals as points whose predicted label is the desired class.
- **Genetic search:** This method samples points by using a genetic algorithm, given the combined objective of optimizing proximity to the query point, changing as few features as possible, and seeking diversity among the generated counterfactuals.
- **KD tree search:** This algorithm returns counterfactuals from the training dataset. It constructs a KD tree over the training data points based on a distance function and then returns the closest points to a particular query point that yields the desired predicted label.

Next steps

- Learn how to generate the Responsible AI dashboard via [CLIV2 and SDKv2 or studio UI](#).
- Explore the [supported counterfactual analysis and what-if perturbation visualizations](#) of the Responsible AI dashboard.
- Learn how to generate a [Responsible AI scorecard](#) based on the insights observed in the Responsible AI dashboard.

What is an Azure Machine Learning workspace?

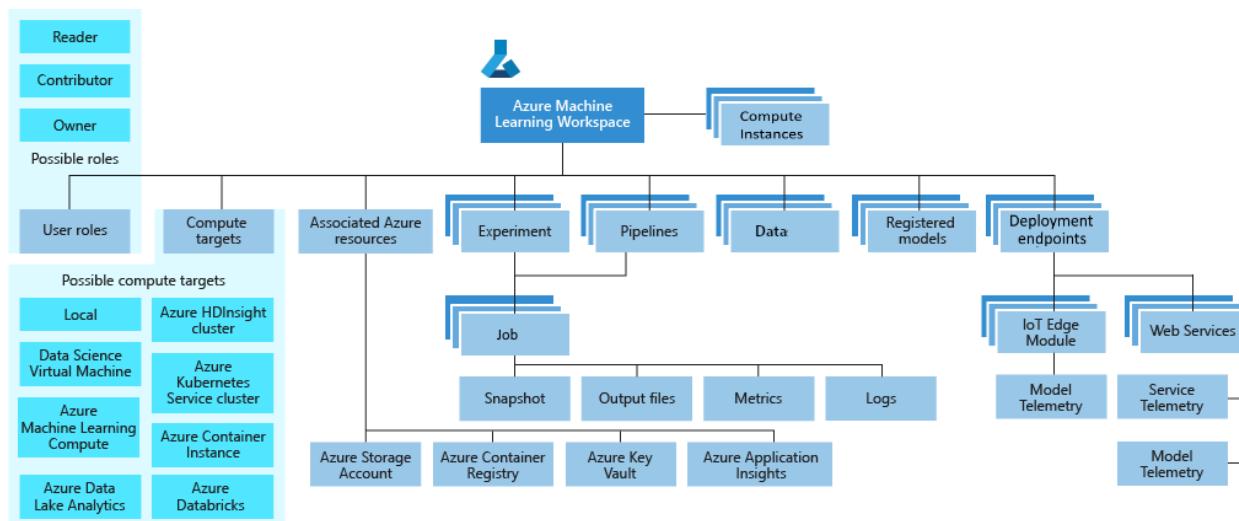
9/22/2022 • 5 minutes to read • [Edit Online](#)

The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. The workspace keeps a history of all training runs, including logs, metrics, output, and a snapshot of your scripts. You use this information to determine which training run produces the best model.

Once you have a model you like, you register it with the workspace. You then use the registered model and scoring scripts to deploy to an [online endpoint](#) as a REST-based HTTP endpoint.

Taxonomy

A taxonomy of the workspace is illustrated in the following diagram:



The diagram shows the following components of a workspace:

- A workspace can contain [Azure Machine Learning compute instances](#), cloud resources configured with the Python environment necessary to run Azure Machine Learning.
- [User roles](#) enable you to share your workspace with other users, teams, or projects.
- [Compute targets](#) are used to run your experiments.
- When you create the workspace, [associated resources](#) are also created for you.
- Jobs are training runs you use to build your models. You can organize your jobs into Experiments.
- [Pipelines](#) are reusable workflows for training and retraining your model.
- [Data assets](#) aid in management of the data you use for model training and pipeline creation.
- Once you have a model you want to deploy, you create a registered model.
- Use the registered model and a scoring script to create an [online endpoint](#).

Tools for workspace interaction

You can interact with your workspace in the following ways:

- On the web:
 - [Azure Machine Learning studio](#)
 - [Azure Machine Learning designer](#)
- In any Python environment with the [Azure Machine Learning SDK for Python](#).
- On the command line using the Azure Machine Learning [CLI extension](#)
- [Azure Machine Learning VS Code Extension](#)

Machine learning with a workspace

Machine learning tasks read and/or write artifacts to your workspace.

- Run an experiment to train a model - writes job run results to the workspace.
- Use automated ML to train a model - writes training results to the workspace.
- Register a model in the workspace.
- Deploy a model - uses the registered model to create a deployment.
- Create and run reusable workflows.
- View machine learning artifacts such as jobs, pipelines, models, deployments.
- Track and monitor models.

Workspace management

You can also perform the following workspace management tasks:

WORKSPACE MANAGEMENT TASK	PORTAL	STUDIO	PYTHON SDK	AZURE CLI	VS CODE
Create a workspace	✓	✓	✓	✓	✓
Manage workspace access	✓			✓	
Create and manage compute resources	✓	✓	✓	✓	✓
Create a compute instance		✓	✓	✓	✓

WARNING

Moving your Azure Machine Learning workspace to a different subscription, or moving the owning subscription to a new tenant, is not supported. Doing so may cause errors.

Create a workspace

There are multiple ways to create a workspace:

- Use [Azure Machine Learning studio](#) to quickly create a workspace with default settings.
- Use the [Azure portal](#) for a point-and-click interface with more options.

- Use the [Azure Machine Learning SDK for Python](#) to create a workspace on the fly from Python scripts or Jupyter notebooks.
- Use an [Azure Resource Manager template](#) or the [Azure Machine Learning CLI](#) when you need to automate or customize the creation with corporate security standards.
- If you work in Visual Studio Code, use the [VS Code extension](#).

NOTE

The workspace name is case-insensitive.

Sub resources

These sub resources are the main resources that are made in the AzureML workspace.

- VMs: provide computing power for your AzureML workspace and are an integral part in deploying and training models.
- Load Balancer: a network load balancer is created for each compute instance and compute cluster to manage traffic even while the compute instance/cluster is stopped.
- Virtual Network: these help Azure resources communicate with one another, the internet, and other on-premises networks.
- Bandwidth: encapsulates all outbound data transfers across regions.

Associated resources

When you create a new workspace, it automatically creates several Azure resources that are used by the workspace:

- [Azure Storage account](#): Is used as the default datastore for the workspace. Jupyter notebooks that are used with your Azure Machine Learning compute instances are stored here as well.

IMPORTANT

By default, the storage account is a general-purpose v1 account. You can [upgrade this to general-purpose v2](#) after the workspace has been created. Do not enable hierarchical namespace on the storage account after upgrading to general-purpose v2.

To use an existing Azure Storage account, it cannot be of type BlobStorage or a premium account (Premium_LRS and Premium_GRS). It also cannot have a hierarchical namespace (used with Azure Data Lake Storage Gen2). Neither premium storage nor hierarchical namespaces are supported with the *default* storage account of the workspace. You can use premium storage or hierarchical namespace with *non-default* storage accounts.

- [Azure Container Registry](#): Registers docker containers that are used for the following components:
 - [Azure Machine Learning environments](#) when training and deploying models
 - [AutoML](#) when deploying
 - [Data profiling](#)

To minimize costs, ACR is **lazy-loaded** until images are needed.

NOTE

If your subscription setting requires adding tags to resources under it, Azure Container Registry (ACR) created by Azure Machine Learning will fail, since we cannot set tags to ACR.

- [Azure Application Insights](#): Stores monitoring and diagnostics information. For more information, see [Monitor online endpoints](#).

NOTE

You can delete the Application Insights instance after cluster creation if you want. Deleting it limits the information gathered from the workspace, and may make it more difficult to troubleshoot problems. **If you delete the Application Insights instance created by the workspace, you cannot re-create it without deleting and recreating the workspace.**

- [Azure Key Vault](#): Stores secrets that are used by compute targets and other sensitive information that's needed by the workspace.

NOTE

You can instead use existing Azure resource instances when you create the workspace with the [Python SDK](#) or the [Azure Machine Learning CLI](#) [using an ARM template](#).

Next steps

To learn more about planning a workspace for your organization's requirements, see [Organize and set up Azure Machine Learning](#).

To get started with Azure Machine Learning, see:

- [What is Azure Machine Learning?](#)
- [Create and manage a workspace](#)
- [Tutorial: Get started with Azure Machine Learning](#)
- [Tutorial: Create your first classification model with automated machine learning](#)
- [Tutorial: Predict automobile price with the designer](#)

What are Azure Machine Learning environments?

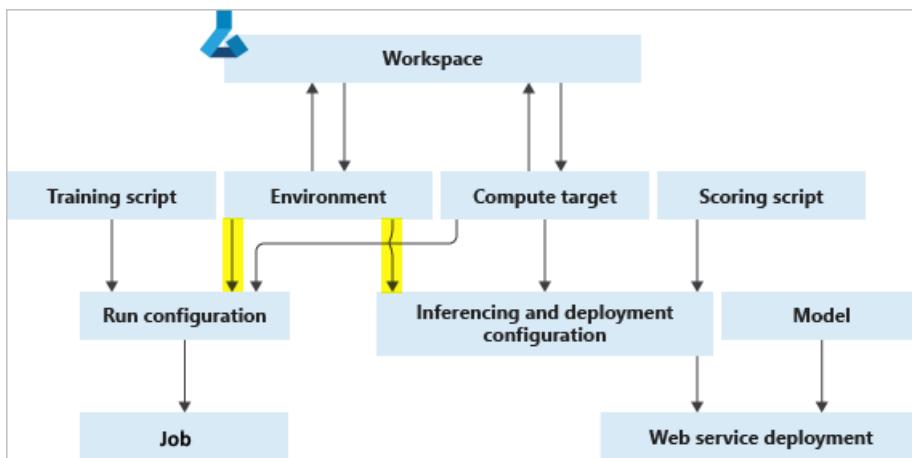
9/22/2022 • 6 minutes to read • [Edit Online](#)

Azure Machine Learning environments are an encapsulation of the environment where your machine learning training happens. They specify the Python packages, environment variables, and software settings around your training and scoring scripts. They also specify runtimes (Python, Spark, or Docker). The environments are managed and versioned entities within your Machine Learning workspace that enable reproducible, auditable, and portable machine learning workflows across a variety of compute targets.

You can use an `Environment` object on your local compute to:

- Develop your training script.
- Reuse the same environment on Azure Machine Learning Compute for model training at scale.
- Deploy your model with that same environment.
- Revisit the environment in which an existing model was trained.

The following diagram illustrates how you can use a single `Environment` object in both your job configuration (for training) and your inference and deployment configuration (for web service deployments).



The environment, compute target and training script together form the job configuration: the full specification of a training job.

Types of environments

Environments can broadly be divided into three categories: *curated*, *user-managed*, and *system-managed*.

Curated environments are provided by Azure Machine Learning and are available in your workspace by default. Intended to be used as is, they contain collections of Python packages and settings to help you get started with various machine learning frameworks. These pre-created environments also allow for faster deployment time. For a full list, see the [curated environments article](#).

In user-managed environments, you're responsible for setting up your environment and installing every package that your training script needs on the compute target. Also be sure to include any dependencies needed for model deployment.

You use system-managed environments when you want `conda` to manage the Python environment for you. A new `conda` environment is materialized from your `conda` specification on top of a base docker image.

Create and manage environments

You can create environments from clients like the AzureML Python SDK, Azure Machine Learning CLI, Environments page in Azure Machine Learning studio, and [VS Code extension](#). Every client allows you to customize the base image, Dockerfile, and Python layer if needed.

For specific code samples, see the "Create an environment" section of [How to use environments](#).

Environments are also easily managed through your workspace, which allows you to:

- Register environments.
- Fetch environments from your workspace to use for training or deployment.
- Create a new instance of an environment by editing an existing one.
- View changes to your environments over time, which ensures reproducibility.
- Build Docker images automatically from your environments.

"Anonymous" environments are automatically registered in your workspace when you submit an experiment. They will not be listed but may be retrieved by version.

For code samples, see the "Manage environments" section of [How to use environments](#).

Environment building, caching, and reuse

Azure Machine Learning builds environment definitions into Docker images and conda environments. It also caches the environments so they can be reused in subsequent training jobs and service endpoint deployments. Running a training script remotely requires the creation of a Docker image, but a local job can use a conda environment directly.

Submitting a job using an environment

When you first submit a remote job using an environment, the Azure Machine Learning service invokes an [ACR Build Task](#) on the Azure Container Registry (ACR) associated with the Workspace. The built Docker image is then cached on the Workspace ACR. Curated environments are backed by Docker images that are cached in Global ACR. At the start of the job execution, the image is retrieved by the compute target from the relevant ACR.

For local jobs, a Docker or conda environment is created based on the environment definition. The scripts are then executed on the target compute - a local runtime environment or local Docker engine.

Building environments as Docker images

If the image for a particular environment definition doesn't already exist in the workspace ACR, a new image will be built. The image build consists of two steps:

1. Downloading a base image, and executing any Docker steps
2. Building a conda environment according to conda dependencies specified in the environment definition.

The second step is omitted if you specify [user-managed dependencies](#). In this case you're responsible for installing any Python packages, by including them in your base image, or specifying custom Docker steps within the first step. You're also responsible for specifying the correct location for the Python executable. It is also possible to use a [custom Docker base image](#).

Image caching and reuse

If you use the same environment definition for another job, Azure Machine Learning reuses the cached image from the Workspace ACR to save time.

To view the details of a cached image, check the Environments page in Azure Machine Learning studio or use the [Environment.get_image_details](#) method.

To determine whether to reuse a cached image or build a new one, AzureML computes a [hash value](#) from the environment definition and compares it to the hashes of existing environments. The hash is based on the

environment definition's:

- Base image
- Custom docker steps
- Python packages
- Spark packages

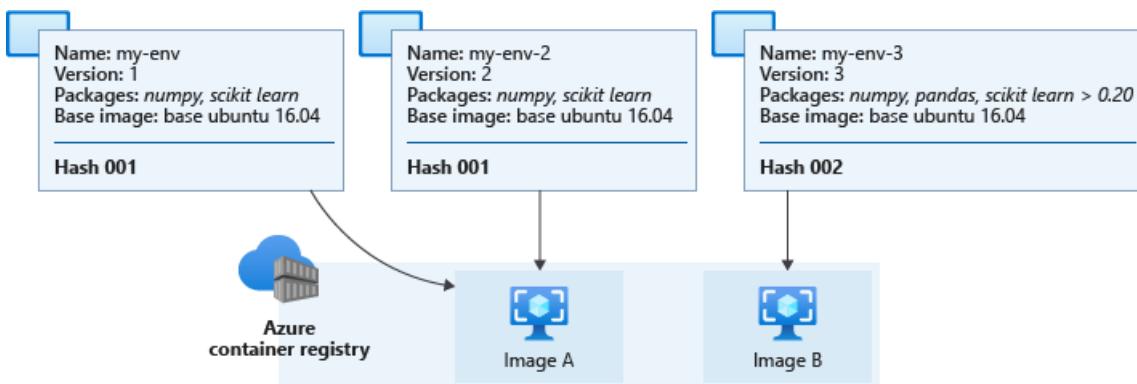
The hash isn't affected by the environment name or version. If you rename your environment or create a new one with the same settings and packages as another environment, then the hash value will remain the same. However, environment definition changes like adding or removing a Python package or changing a package version will result cause the resulting hash value to change. Changing the order of dependencies or channels in an environment will also change the hash and require a new image build. Similarly, any change to a curated environment will result in the creation of a new "non-curated" environment.

NOTE

You will not be able to submit any local changes to a curated environment without changing the name of the environment. The prefixes "AzureML-" and "Microsoft" are reserved exclusively for curated environments, and your job submission will fail if the name starts with either of them.

The environment's computed hash value is compared with those in the Workspace and global ACR, or on the compute target (local jobs only). If there is a match then the cached image is pulled and used, otherwise an image build is triggered.

The following diagram shows three environment definitions. Two of them have different names and versions but identical base images and Python packages, which results in the same hash and corresponding cached image. The third environment has different Python packages and versions, leading to a different hash and cached image.



Actual cached images in your workspace ACR will have names like

`azureml/azureml_e9607b2514b066c851012848913ba19f` with the hash appearing at the end.

IMPORTANT

- If you create an environment with an unpinned package dependency (for example, `numpy`), the environment uses the package version that was *available when the environment was created*. Any future environment that uses a matching definition will use the original version.

To update the package, specify a version number to force an image rebuild. An example of this would be changing `numpy` to `numpy==1.18.1`. New dependencies--including nested ones--will be installed, and they might break a previously working scenario.

- Using an unpinned base image like `mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04` in your environment definition results in rebuilding the image every time the `latest` tag is updated. This helps the image receive the latest patches and system updates.

WARNING

The `Environment.build` method will rebuild the cached image, with the possible side-effect of updating unpinned packages and breaking reproducibility for all environment definitions corresponding to that cached image.

Image patching

Microsoft is responsible for patching the base images for known security vulnerabilities. Updates for supported images are released every two weeks, with a commitment of no unpatched vulnerabilities older than 30 days in the latest version of the image. Patched images are released with a new immutable tag and the `:latest` tag is updated to the latest version of the patched image.

If you provide your own images, you are responsible for updating them.

For more information on the base images, see the following links:

- [Azure Machine Learning base images](#) GitHub repository.
- [Train a model using a custom image](#).
- [Deploy a TensorFlow model using a custom container](#)

Next steps

- Learn how to [create and use environments](#) in Azure Machine Learning.
- See the Python SDK reference documentation for the [environment class](#).

What is an Azure Machine Learning compute instance?

9/22/2022 • 7 minutes to read • [Edit Online](#)

An Azure Machine Learning compute instance is a managed cloud-based workstation for data scientists. Each compute instance has only one owner, although you can share files between multiple compute instances.

Compute instances make it easy to get started with Azure Machine Learning development as well as provide management and enterprise readiness capabilities for IT administrators.

Use a compute instance as your fully configured and managed development environment in the cloud for machine learning. They can also be used as a compute target for training and inferencing for development and testing purposes.

For compute instance Jupyter functionality to work, ensure that web socket communication is not disabled.

Please ensure your network allows websocket connections to *.instances.azureml.net and *.instances.azureml.ms.

IMPORTANT

Items marked (preview) in this article are currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Why use a compute instance?

A compute instance is a fully managed cloud-based workstation optimized for your machine learning development environment. It provides the following benefits:

KEY BENEFITS	DESCRIPTION
Productivity	You can build and deploy models using integrated notebooks and the following tools in Azure Machine Learning studio: - Jupyter - JupyterLab - VS Code (preview) Compute instance is fully integrated with Azure Machine Learning workspace and studio. You can share notebooks and data with other data scientists in the workspace.

KEY BENEFITS	DESCRIPTION
Managed & secure	<p>Reduce your security footprint and add compliance with enterprise security requirements. Compute instances provide robust management policies and secure networking configurations such as:</p> <ul style="list-style-type: none"> - Autoprovisioning from Resource Manager templates or Azure Machine Learning SDK - Azure role-based access control (Azure RBAC) - Virtual network support - Azure policy to disable SSH access - Azure policy to enforce creation in a virtual network - Auto-shutdown/auto-start based on schedule - TLS 1.2 enabled
Preconfigured for ML	Save time on setup tasks with pre-configured and up-to-date ML packages, deep learning frameworks, GPU drivers.
Fully customizable	Broad support for Azure VM types including GPUs and persisted low-level customization such as installing packages and drivers makes advanced scenarios a breeze. You can also use setup scripts to automate customization

- Secure your compute instance with [No public IP \(preview\)](#)
- The compute instance is also a secure training compute target similar to [compute clusters](#), but it is single node.
- You can [create a compute instance](#) yourself, or an administrator can [create a compute instance on your behalf](#).
- You can also [use a setup script \(preview\)](#) for an automated way to customize and configure the compute instance as per your needs.
- To save on costs, [create a schedule \(preview\)](#) to automatically start and stop the compute instance.

Tools and environments

Azure Machine Learning compute instance enables you to author, train, and deploy models in a fully integrated notebook experience in your workspace.

You can run Jupyter notebooks in [VS Code](#) using compute instance as the remote server with no SSH needed. You can also enable VS Code integration through [remote SSH extension](#).

You can [install packages](#) and [add kernels](#) to your compute instance.

Following tools and environments are already installed on the compute instance:

GENERAL TOOLS & ENVIRONMENTS	DETAILS
Drivers	<div style="display: flex; justify-content: space-around; align-items: center;"> CUDA cuDNN NVIDIA Blob FUSE </div>
Intel MPI library	
Azure CLI	

GENERAL TOOLS & ENVIRONMENTS	DETAILS
Azure Machine Learning samples	
Docker	
Nginx	
NCCL 2.0	
Protobuf	
R TOOLS & ENVIRONMENTS	DETAILS
R kernel	

You can [Add RStudio](#) when you create the instance.

PYTHON TOOLS & ENVIRONMENTS	DETAILS
Anaconda Python	
Jupyter and extensions	
Jupyterlab and extensions	
Azure Machine Learning SDK for Python from PyPI	<p>Includes most of the azureml extra packages. To see the full list, open a terminal window on your compute instance and run</p> <pre>conda list -n azureml_py36 azureml*</pre>
Other PyPI packages	<pre>jupyter tensorboard nbconvert notebook Pillow</pre>
Conda packages	<pre>cython numpy ipykernel scikit-learn matplotlib tqdm joblib nodejs nb_conda_kernels</pre>
Deep learning packages	<pre>PyTorch TensorFlow Keras Horovod MLFlow pandas-ml scrapbook</pre>

PYTHON TOOLS & ENVIRONMENTS	DETAILS
ONNX packages	<pre>keras2onnx onnx onnxconverter-common sk12onnx onnxmltools</pre>
Azure Machine Learning Python samples	

Python packages are all installed in the **Python 3.8 - AzureML** environment. Compute instance has Ubuntu 18.04 as the base OS.

Accessing files

Notebooks and R scripts are stored in the default storage account of your workspace in Azure file share. These files are located under your “User files” directory. This storage makes it easy to share notebooks between compute instances. The storage account also keeps your notebooks safely preserved when you stop or delete a compute instance.

The Azure file share account of your workspace is mounted as a drive on the compute instance. This drive is the default working directory for Jupyter, Jupyter Labs, and RStudio. This means that the notebooks and other files you create in Jupyter, JupyterLab, or RStudio are automatically stored on the file share and available to use in other compute instances as well.

The files in the file share are accessible from all compute instances in the same workspace. Any changes to these files on the compute instance will be reliably persisted back to the file share.

You can also clone the latest Azure Machine Learning samples to your folder under the user files directory in the workspace file share.

Writing small files can be slower on network drives than writing to the compute instance local disk itself. If you are writing many small files, try using a directory directly on the compute instance, such as a `/tmp` directory. Note these files will not be accessible from other compute instances.

Do not store training data on the notebooks file share. You can use the `/tmp` directory on the compute instance for your temporary data. However, do not write very large files of data on the OS disk of the compute instance. OS disk on compute instance has 128 GB capacity. You can also store temporary training data on temporary disk mounted on `/mnt`. Temporary disk size is configurable based on the VM size chosen and can store larger amounts of data if a higher size VM is chosen. You can also mount [datastores and datasets](#). Any software packages you install are saved on the OS disk of compute instance. Please note customer managed key encryption is currently not supported for OS disk. The OS disk for compute instance is encrypted with Microsoft-managed keys.

Create

As an administrator, you can [create a compute instance for others in the workspace \(preview\)](#).

You can also [use a setup script \(preview\)](#) for an automated way to customize and configure the compute instance.

To create a compute instance for yourself, use your workspace in Azure Machine Learning studio, [create a new compute instance](#) from either the **Compute** section or in the **Notebooks** section when you are ready to run one of your notebooks.

You can also create an instance

- Directly from the [integrated notebooks experience](#)
- In Azure portal
- From Azure Resource Manager template. For an example template, see the [create an Azure Machine Learning compute instance template](#).
- With [Azure Machine Learning SDK](#)
- From the [CLI extension for Azure Machine Learning](#)

The dedicated cores per region per VM family quota and total regional quota, which applies to compute instance creation, is unified and shared with Azure Machine Learning training compute cluster quota. Stopping the compute instance does not release quota to ensure you will be able to restart the compute instance. Please do not stop the compute instance through the OS terminal by doing a sudo shutdown.

Compute instance comes with P10 OS disk. Temp disk type depends on the VM size chosen. Currently, it is not possible to change the OS disk type.

Compute target

Compute instances can be used as a [training compute target](#) similar to Azure Machine Learning [compute training clusters](#). But a compute instance has only a single node, while a compute cluster can have more nodes.

A compute instance:

- Has a job queue.
- Runs jobs securely in a virtual network environment, without requiring enterprises to open up SSH port. The job executes in a containerized environment and packages your model dependencies in a Docker container.
- Can run multiple small jobs in parallel (preview). One job per core can run in parallel while the rest of the jobs are queued.
- Supports single-node multi-GPU [distributed training](#) jobs

You can use compute instance as a local inferencing deployment target for test/debug scenarios.

TIP

The compute instance has 120GB OS disk. If you run out of disk space and get into an unusable state, please clear at least 5 GB disk space on OS disk (mounted on `/`) through the compute instance terminal by removing files/folders and then do `sudo reboot`. To access the terminal go to compute list page or compute instance details page and click on **Terminal** link. You can check available disk space by running `df -h` on the terminal. Clear at least 5 GB space before doing `sudo reboot`. Please do not stop or restart the compute instance through the Studio until 5 GB disk space has been cleared.

Next steps

- [Create and manage a compute instance](#)
- [Tutorial: Train your first ML model](#) shows how to use a compute instance with an integrated notebook.

What are compute targets in Azure Machine Learning?

9/22/2022 • 8 minutes to read • [Edit Online](#)

A *compute target* is a designated compute resource or environment where you run your training script or host your service deployment. This location might be your local machine or a cloud-based compute resource. Using compute targets makes it easy for you to later change your compute environment without having to change your code.

In a typical model development lifecycle, you might:

1. Start by developing and experimenting on a small amount of data. At this stage, use your local environment, such as a local computer or cloud-based virtual machine (VM), as your compute target.
2. Scale up to larger data, or do [distributed training](#) by using one of these [training compute targets](#).
3. After your model is ready, deploy it to a web hosting environment with one of these [deployment compute targets](#).

The compute resources you use for your compute targets are attached to a [workspace](#). Compute resources other than the local machine are shared by users of the workspace.

Training compute targets

Azure Machine Learning has varying support across different compute targets. A typical model development lifecycle starts with development or experimentation on a small amount of data. At this stage, use a local environment like your local computer or a cloud-based VM. As you scale up your training on larger datasets or perform [distributed training](#), use Azure Machine Learning compute to create a single- or multi-node cluster that autoscales each time you submit a job. You can also attach your own compute resource, although support for different scenarios might vary.

Compute targets can be reused from one training job to the next. For example, after you attach a remote VM to your workspace, you can reuse it for multiple jobs. For machine learning pipelines, use the appropriate [pipeline step](#) for each compute target.

You can use any of the following resources for a training compute target for most jobs. Not all resources can be used for automated machine learning, machine learning pipelines, or designer. Azure Databricks can be used as a training resource for local runs and machine learning pipelines, but not as a remote target for other training.

TRAINING TARGETS	AUTOMATED MACHINE LEARNING	MACHINE LEARNING PIPELINES	AZURE MACHINE LEARNING DESIGNER
Local computer	Yes		
Azure Machine Learning compute cluster	Yes	Yes	Yes
Azure Machine Learning compute instance	Yes (through SDK)	Yes	Yes
Azure Machine Learning Kubernetes	Yes	Yes	Yes

TRAINING TARGETS	AUTOMATED MACHINE LEARNING	MACHINE LEARNING PIPELINES	AZURE MACHINE LEARNING DESIGNER
Remote VM	Yes	Yes	
Apache Spark pools (preview)	Yes (SDK local mode only)	Yes	
Azure Databricks	Yes (SDK local mode only)	Yes	
Azure Data Lake Analytics		Yes	
Azure HDInsight		Yes	
Azure Batch		Yes	

TIP

The compute instance has 120GB OS disk. If you run out of disk space, [use the terminal](#) to clear at least 1-2 GB before you [stop or restart](#) the compute instance.

Compute targets for inference

When performing inference, Azure Machine Learning creates a Docker container that hosts the model and associated resources needed to use it. This container is then used in a compute target.

The compute target you use to host your model will affect the cost and availability of your deployed endpoint. Use this table to choose an appropriate compute target.

COMPUTE TARGET	USED FOR	GPU SUPPORT	DESCRIPTION
Local web service	Testing/debugging		Use for limited testing and troubleshooting. Hardware acceleration depends on use of libraries in the local system.
Azure Machine Learning Kubernetes	Real-time inference Batch inference	Yes	Run inferencing workloads on on-premises, cloud, and edge Kubernetes clusters.
Azure Container Instances	Real-time inference Recommended for dev/test purposes only.		Use for low-scale CPU-based workloads that require less than 48 GB of RAM. Doesn't require you to manage a cluster. Supported in the designer.
Azure Machine Learning compute clusters	Batch inference	Yes (machine learning pipeline)	Run batch scoring on serverless compute. Supports normal and low-priority VMs. No support for real-time inference.

NOTE

Although compute targets like local, and Azure Machine Learning compute clusters support GPU for training and experimentation, using GPU for inference *when deployed as a web service* is supported only on Azure Machine Learning Kubernetes.

Using a GPU for inference *when scoring with a machine learning pipeline* is supported only on Azure Machine Learning compute.

When choosing a cluster SKU, first scale up and then scale out. Start with a machine that has 150% of the RAM your model requires, profile the result and find a machine that has the performance you need. Once you've learned that, increase the number of machines to fit your need for concurrent inference.

NOTE

- Container instances are suitable only for small models less than 1 GB in size.

Learn where and how to deploy your model to a compute target.

Azure Machine Learning compute (managed)

A managed compute resource is created and managed by Azure Machine Learning. This compute is optimized for machine learning workloads. Azure Machine Learning compute clusters and [compute instances](#) are the only managed computes.

You can create Azure Machine Learning compute instances or compute clusters from:

- [Azure Machine Learning studio](#).
- The Python SDK and the Azure CLI:
 - [Compute instance](#).
 - [Compute cluster](#).
- An Azure Resource Manager template. For an example template, see [Create an Azure Machine Learning compute cluster](#).

When created, these compute resources are automatically part of your workspace, unlike other kinds of compute targets.

CAPABILITY	COMPUTE CLUSTER	COMPUTE INSTANCE
Single- or multi-node cluster	✓	Single node cluster
Autoscales each time you submit a job	✓	
Automatic cluster management and job scheduling	✓	✓
Support for both CPU and GPU resources	✓	✓

NOTE

When a compute *cluster* is idle, it autoscales to 0 nodes, so you don't pay when it's not in use. A compute *instance* is always on and doesn't autoscale. You should [stop the compute instance](#) when you aren't using it to avoid extra cost.

Supported VM series and sizes

NOTE

H-series virtual machine series will be retired on August 31, 2022. Create compute instance and compute clusters with alternate VM sizes. Existing compute instances and clusters with H-series virtual machines will not work after August 31, 2022.

When you select a node size for a managed compute resource in Azure Machine Learning, you can choose from among select VM sizes available in Azure. Azure offers a range of sizes for Linux and Windows for different workloads. To learn more, see [VM types and sizes](#).

There are a few exceptions and limitations to choosing a VM size:

- Some VM series aren't supported in Azure Machine Learning.
- There are some VM series, such as GPUs and other special SKUs, which may not initially appear in your list of available VMs. But you can still use them, once you request a quota change. For more information about requesting quotas, see [Request quota increases](#). See the following table to learn more about supported series.

SUPPORTED VM SERIES	CATEGORY	SUPPORTED BY
DDSv4	General purpose	Compute clusters and instance
Dv2	General purpose	Compute clusters and instance
Dv3	General purpose	Compute clusters and instance
DSv2	General purpose	Compute clusters and instance
DSv3	General purpose	Compute clusters and instance
EAv4	Memory optimized	Compute clusters and instance
Ev3	Memory optimized	Compute clusters and instance
ESv3	Memory optimized	Compute clusters and instance
FSv2	Compute optimized	Compute clusters and instance
FX	Compute optimized	Compute clusters
H	High performance compute	Compute clusters and instance
HB	High performance compute	Compute clusters and instance
HBv2	High performance compute	Compute clusters and instance

SUPPORTED VM SERIES	CATEGORY	SUPPORTED BY
HBv3	High performance compute	Compute clusters and instance
HC	High performance compute	Compute clusters and instance
LSv2	Storage optimized	Compute clusters and instance
M	Memory optimized	Compute clusters and instance
NC	GPU	Compute clusters and instance
NC Promo	GPU	Compute clusters and instance
NCv2	GPU	Compute clusters and instance
NCv3	GPU	Compute clusters and instance
ND	GPU	Compute clusters and instance
NDv2	GPU	Compute clusters and instance
NV	GPU	Compute clusters and instance
NVv3	GPU	Compute clusters and instance
NCasT4_v3	GPU	Compute clusters and instance
NDasrA100_v4	GPU	Compute clusters and instance

While Azure Machine Learning supports these VM series, they might not be available in all Azure regions. To check whether VM series are available, see [Products available by region](#).

NOTE

Azure Machine Learning doesn't support all VM sizes that Azure Compute supports. To list the available VM sizes, use one of the following methods:

- [REST API](#)
- [Python SDK](#)

If using the GPU-enabled compute targets, it is important to ensure that the correct CUDA drivers are installed in the training environment. Use the following table to determine the correct CUDA version to use:

GPU ARCHITECTURE	AZURE VM SERIES	SUPPORTED CUDA VERSIONS
Ampere	NDA100_v4	11.0+
Turing	NCT4_v3	10.0+
Volta	NCv3, NDv2	9.0+

GPU ARCHITECTURE	AZURE VM SERIES	SUPPORTED CUDA VERSIONS
Pascal	NCv2, ND	9.0+
Maxwell	NV, NVv3	9.0+
Kepler	NC, NC Promo	9.0+

In addition to ensuring the CUDA version and hardware are compatible, also ensure that the CUDA version is compatible with the version of the machine learning framework you are using:

- For PyTorch, you can check the compatibility by visiting [Pytorch's previous versions page](#).
- For Tensorflow, you can check the compatibility by visiting [Tensorflow's build from source page](#).

Compute isolation

Azure Machine Learning compute offers VM sizes that are isolated to a specific hardware type and dedicated to a single customer. Isolated VM sizes are best suited for workloads that require a high degree of isolation from other customers' workloads for reasons that include meeting compliance and regulatory requirements. Utilizing an isolated size guarantees that your VM will be the only one running on that specific server instance.

The current isolated VM offerings include:

- Standard_M128ms
- Standard_F72s_v2
- Standard_NC24s_v3
- Standard_NC24rs_v3*

*RDMA capable

To learn more about isolation, see [Isolation in the Azure public cloud](#).

Unmanaged compute

An unmanaged compute target is *not* managed by Azure Machine Learning. You create this type of compute target outside Azure Machine Learning and then attach it to your workspace. Unmanaged compute resources can require additional steps for you to maintain or to improve performance for machine learning workloads.

Azure Machine Learning supports the following unmanaged compute types:

- Your local computer
- Remote virtual machines
- Azure HDInsight
- Azure Batch
- Azure Databricks
- Azure Data Lake Analytics
- Azure Container Instance
- Kubernetes

For more information, see [set up compute targets for model training and deployment](#)

Next steps

Learn how to:

- [Deploy your model to a compute target](#)

Data in Azure Machine Learning

9/22/2022 • 8 minutes to read • [Edit Online](#)

Azure Machine Learning lets you bring data from a local machine or an existing cloud-based storage. In this article you will learn the main data concepts in Azure Machine Learning, including:

- **URIs** - A Uniform Resource Identifier that is a reference to a storage location on your local computer or in the cloud that makes it very easy to access data in your jobs.
- **Data asset** - Create data assets in your workspace to share with team members, version, and track data lineage.
- **Datastore** - Azure Machine Learning Datastores securely keep the connection information to your data storage on Azure, so you don't have to code it in your scripts.
- **MLTable** - a method to abstract the schema definition for tabular data so that it is easier for consumers of the data to materialize the table into a Pandas/Dask/Spark dataframe.

URIs

A URI (uniform resource identifier) represents a storage location on your local computer, an attached Datastore, blob/ADLS storage, or a publicly available http(s) location. In addition to local paths (for example:

`./path_to_my_data/`), several different protocols are supported for cloud storage locations:

- `http(s)` - Private/Public Azure Blob Storage Locations, or publicly available http(s) location
- `abfs(s)` - Azure Data Lake Storage Gen2 storage location
- `azureml` - An Azure Machine Learning **Datastore** location

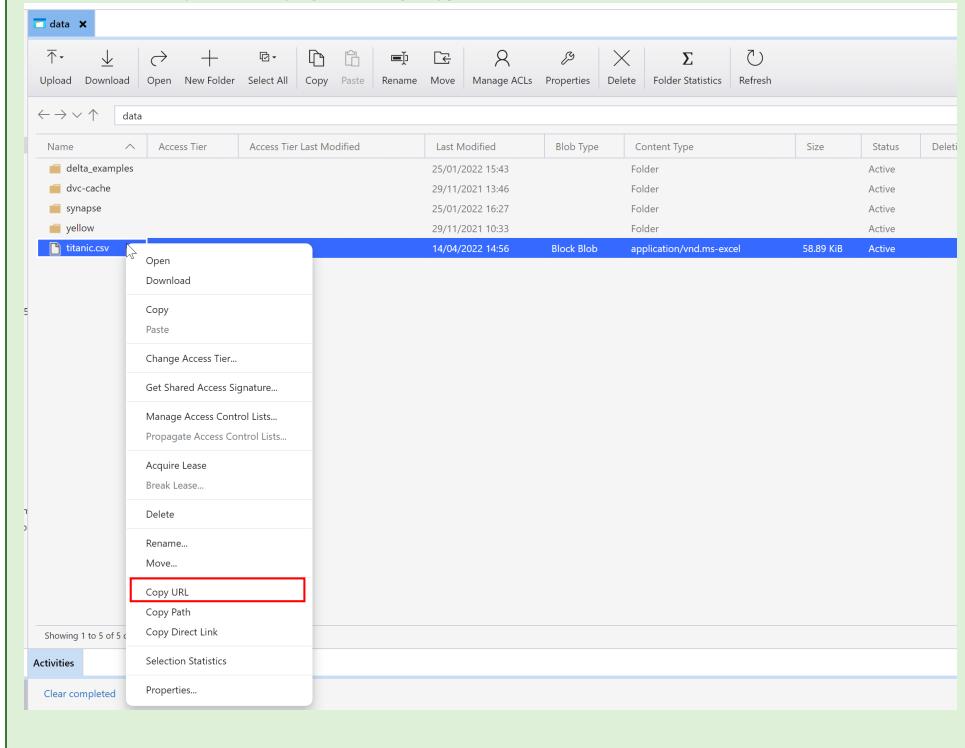
Azure Machine Learning distinguishes two types of URIs:

DATA TYPE	DESCRIPTION	EXAMPLES
<code>uri_file</code>	Refers to a specific file location	<code>https://<account_name>.blob.core.windows.net/<container_name></code> <code>azureml://datastores/<datastore_name>/paths/<folder>/<file></code> <code>abfss://<file_system>@<account_name>.dfs.core.windows.net/<f</code>
<code>uri_folder</code>	Refers to a specific folder location	<code>https://<account_name>.blob.core.windows.net/<container_name></code> <code>azureml://datastores/<datastore_name>/paths/<folder></code> <code>abfss://<file_system>@<account_name>.dfs.core.windows.net/<f</code>

URIs are mapped to the filesystem on the compute target, hence using URIs is like using files or folders in the command that consumes/produces them. URIs leverage **identity-based authentication** to connect to storage services with either your Azure Active Directory ID (default) or Managed Identity.

TIP

For data located in an Azure storage account we recommend using the [Azure Storage Explorer](#). You can browse data and obtain the URI for any file/folder by right-selecting **Copy URL**:



Examples

- [uri_file](#)
- [uri_folder](#)

Below is an example of a job specification that shows how to access a file from a public blob store. In this example, the job executes the Linux `ls` command.

```
# hello-data-uri-file.yml
$schema: https://azureschemas.azureedge.net/latest/commandJob.schema.json
command: |
    ls ${{inputs.my_csv_file}}

inputs:
    my_csv_file:
        type: uri_file
        path: https://azurermexamples.blob.core.windows.net/datasets/titanic.csv
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster
```

Create the job using the CLI:

```
az ml job create --file hello-data-uri-file.yml
```

When the job has completed the user logs will show the standard output of the Linux command

```
ls ${{inputs.my_csv_file}} :
```

A screenshot of the Azure ML Compute job log interface. The top bar includes Refresh, Connect to compute, Edit and submit, Create model, Cancel, Delete, Download all, Enable log streaming, and Word wrap. The main area shows the 'Outputs + logs' tab selected. Under 'Logs', there is a tree view with system_logs and user_logs. A file named std_log.txt is selected. The log content shows three lines of text: 1 /stderr/bin/bash /azurerm-envs/sklearn-1.0/lib/libtinfo.so.6: no version information available (required by /bin/bash) 2 /mnt/azurerm/cr/1ad34c235e5c43f6b4d24d37bbdd6670/cap/data-capability/wd/my_csv_file/titanic.csv 3

Notice that the file has been mapped to the filesystem on the compute target and `${{inputs.my_csv_file}}` resolves to that location.

Data asset

Azure Machine Learning allows you to create and version data assets in a workspace so that other members of your team can easily consume the data asset by using a name/version.

Example usage

- [Create data asset](#)
- [Consume data asset](#)

To create a data asset, firstly define a data specification in a YAML file that provides a name, type and path for the data:

```
# data-example.yml
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: <name>
description: <description>
type: <type> # uri_file, uri_folder, mltable
path: https://<storage_name>.blob.core.windows.net/<container_name>/path
```

Then in the CLI, create the data asset:

```
az ml data create --file data-example.yml --version 1
```

Datastore

An Azure Machine Learning datastore is a *reference* to an *existing* storage account on Azure. The benefits of creating and using a datastore are:

1. A common and easy-to-use API to interact with different storage types (Blob/Files/ADLS).
2. Easier to discover useful datastores when working as a team.
3. When using credential-based access (service principal/SAS/key), the connection information is secured so you don't have to code it in your scripts.

When you create a datastore with an existing storage account on Azure, you have the choice between two different authentication methods:

- **Credential-based** - authenticate access to the data using a service principal, shared access signature (SAS) token or account key. These credentials can be accessed by users who have *Reader* access to the workspace.
- **Identity-based** - authenticate access to the data using your Azure Active Directory identity or managed identity.

The table below summarizes which cloud-based storage services in Azure can be created as an Azure Machine Learning datastore and what authentication type can be used to access them.

SUPPORTED STORAGE SERVICE	CREDENTIAL-BASED AUTHENTICATION	IDENTITY-BASED AUTHENTICATION
Azure Blob Container	✓	✓
Azure File Share	✓	
Azure Data Lake Gen1	✓	✓
Azure Data Lake Gen2	✓	✓

NOTE

The URI format to refer to a file/folder/mltable on a datastore is: `azurerm://datastores/<name>/paths/<path>`

MLTable

`mltable` is a way to abstract the schema definition for tabular data so that it is easier for consumers of the data to materialize the table into a Pandas/Dask/Spark dataframe.

TIP

The ideal scenarios to use `mltable` are:

- The schema of your data is complex and/or changes frequently.
- You only need a subset of data (for example: a sample of rows or files, specific columns, etc).
- AutoML jobs requiring tabular data.

If your scenario does not fit the above then it is likely that URLs are a more suitable type.

A motivating example

Imagine a scenario where you have many text files in a folder:

```
|   └── my_data
|       ├── file1.txt
|       ├── file1_use_this.txt
|       ├── file2.txt
|       ├── file2_use_this.txt
|       .
|       .
|       └── file1000.txt
|           └── file1000_use_this.txt
```

Each text file has the following structure:

```
store_location date zip_code amount x y noise_col1 noise_col2
Seattle 20/04/2022 12324 123.4 true false true blah blah
.
.
.
London 20/04/2022 XX358YY 156 true true true blah blah
```

Some important features of this data are:

- The data of interest is only in files that have the following suffix: `_use_this.txt` and other file names that don't match should be ignored.
- The date should be represented as a date and not a string.
- The x, y, z columns are booleans, not strings.
- The store location is an index that is useful for generating subsets of data.
- The file is encoded in `ascii` format.
- Every file in the folder contains the same header.
- The first million records for zip_code are numeric but later on you can see they're alphanumeric.
- There are some dummy (noisy) columns in the data that aren't useful for machine learning.

You could materialize the above text files into a dataframe using Pandas and a URI:

```

import glob
import datetime
import os
import argparse
import pandas as pd

parser = argparse.ArgumentParser()
parser.add_argument("--input_folder", type=str)
args = parser.parse_args()

path = os.path.join(args.input_folder, "*_use_this.txt")
files = glob.glob(path)

# create empty list
df1 = []

# dict of column types
col_types = {
    "zip": str,
    "date": datetime.date,
    "x": bool,
    "y": bool,
    "z": bool
}

# enumerate files into a list of dfs
for f in files:
    csv = pd.read_table(
        path=f,
        delimiter=" ",
        header=0,
        usecols=["store_location", "zip_code", "date", "amount", "x", "y", "z"],
        dtype=col_types,
        encoding='ascii'
    )
    df1.append(csv)

# concatenate the list of dataframes
df = pd.concat(df1)
# set the index column
df.index_columns("store_location")

```

However, it will be the responsibility of the *consumer* of the data asset to parse the schema into a dataframe. In the scenario defined above, that means the consumers will need to independently ascertain the Python code to materialize the data into a dataframe.

Passing responsibility to the consumer of the data asset will cause problems when:

- **The schema changes (for example, a column name changes):** All consumers of the data must update their Python code independently. Other examples can be type changes, columns being added/removed, encoding change, etc.
- **The data size increases -** If the data gets too large for Pandas to process, then all the consumers of the data need to switch to a more scalable library (PySpark/Dask).

Under the above two conditions, `mltable` can help because it enables the creator of the data asset to define the schema in a single file and the consumers can materialize the data into a dataframe easily without needing to write Python code to parse the schema. For the above example, the creator of the data asset defines an MLTable file in the same directory as the data:

```

my_data
├── MLTable
├── file1.txt
└── file1_use_this.txt
.
.
.

```

The MLTable file has the following definition that specifies how the data should be processed into a dataframe:

```
type: mltable

paths:
  - pattern: ./*_use_this.txt

traits:
  - index_columns: store_location

transformations:
  - read_delimited:
    encoding: ascii
    header: all_files_same_headers
    delimiter: " "
  - keep_columns: ["store_location", "zip_code", "date", "amount", "x", "y", "z"]
  - convert_column_types:
    - columns: ["x", "y", "z"]
      to_type: boolean
    - columns: "date"
      to_type: datetime
```

The consumers can read the data into dataframe using three lines of Python code:

```
import mltable

tbl = mltable.load("./my_data")
df = tbl.to_pandas_dataframe()
```

If the schema of the data changes, then it can be updated in a single place (the MLTable file) rather than having to make code changes in multiple places.

Just like `uri_file` and `uri_folder`, you can create a data asset with `mltable` types.

Next steps

- [Install and set up the CLI \(v2\)](#)
- [Create datastores](#)
- [Create data assets](#)
- [Read and write data in a job](#)
- [Data administration](#)

What is "human data" and why is it important to source responsibly?

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

APPLIES TO:  Azure CLI ml extension v2 (current)

Human data is data collected directly from, or about, people. Human data may include personal data such as names, age, images, or voice clips and sensitive data such as genetic data, biometric data, gender identity, religious beliefs, or political affiliations.

Collecting this data can be important to building AI systems that work for all users. But certain practices should be avoided, especially ones that can cause physical and psychological harm to data contributors.

The best practices in this article will help you conduct manual data collection projects from volunteers where everyone involved is treated with respect, and potential harms—especially those faced by vulnerable groups—are anticipated and mitigated. This means that:

- People contributing data are not coerced or exploited in any way, and they have control over what personal data is collected.
- People collecting and labeling data have adequate training.

These practices can also help ensure more-balanced and higher-quality datasets and better stewardship of human data.

These are emerging practices, and we are continually learning. The best practices below are a starting point as you begin your own responsible human data collections. These best practices are provided for informational purposes only and should not be treated as legal advice. All human data collections should undergo specific privacy and legal reviews.

General best practices

We suggest the following best practices for manually collecting human data directly from people.

Best Practice

Why?

Obtain voluntary informed consent.

- Participants should understand and consent to data collection and how their data will be used.
- Data should only be stored, processed, and used for purposes that are part of the original documented informed consent.
- Consent documentation should be properly stored and associated with the collected data.

Compensate data contributors appropriately.

- Data contributors should not be pressured or coerced into data collections and should be fairly compensated for their time and data.
- Inappropriate compensation can be exploitative or coercive.

Let contributors self-identify demographic information.

- Demographic information that is not self-reported by data contributors but assigned by data collectors may 1) result in inaccurate metadata and 2) be disrespectful to data contributors.

Anticipate harms when recruiting vulnerable groups.

- Collecting data from vulnerable population groups introduces risk to data contributors and your organization.

Treat data contributors with respect.

- Improper interactions with data contributors at any phase of the data collection can negatively impact data quality, as well as the overall data collection experience for data contributors and data collectors.

Qualify external suppliers carefully.

- Data collections with unqualified suppliers may result in low quality data, poor data management, unprofessional practices, and potentially harmful outcomes for data contributors and data collectors (including violations of human rights).
- Annotation or labeling work (e.g., audio transcription, image tagging) with unqualified suppliers may result in low quality or biased datasets, insecure data management, unprofessional practices, and potentially harmful outcomes for data contributors (including violations of human rights).

Communicate expectations clearly in the Statement of Work (SOW) (contracts or agreements) with suppliers.

- A contract which lacks requirements for responsible data collection work may result in low-quality or poorly collected data.

Qualify geographies carefully.

- When applicable, collecting data in areas of high geopolitical risk and/or unfamiliar geographies may result in unusable or low-quality data and may impact the safety of involved parties.

Be a good steward of your datasets.

- Improper data management and poor documentation can result in data misuse.

NOTE

This article focuses on recommendations for human data, including personal data and sensitive data such as biometric data, health data, racial or ethnic data, data collected manually from the general public or company employees, as well as metadata relating to human characteristics, such as age, ancestry, and gender identity, that may be created via annotation or labeling.

[Download the full recommendations here](#)

Best practices for collecting age, ancestry, and gender identity

In order for AI systems to work well for everyone, the datasets used for training and evaluation should reflect the diversity of people who will use or be affected by those systems. In many cases, age, ancestry, and gender identity can help approximate the range of factors that might affect how well a product performs for a variety of people; however, collecting this information requires special consideration.

If you do collect this data, always let data contributors self-identify (choose their own responses) instead of having data collectors make assumptions, which might be incorrect. Also include a “prefer not to answer” option for each question. These practices will show respect for the data contributors and yield more balanced and higher-quality data.

These best practices have been developed based on three years of research with intended stakeholders and collaboration with many teams at Microsoft: [fairness and inclusiveness working groups](#), [Global Diversity & Inclusion](#), [Global Readiness](#), [Office of Responsible AI](#), and others.

To enable people to self-identify, consider using the following survey questions.

Age

How old are you?

Select your age range

[Include appropriate age ranges as defined by project purpose, geographical region, and guidance from domain experts]

- # to #
- # to #
- # to #
- Prefer not to answer

Ancestry

Please select the categories that best describe your ancestry

May select multiple

[Include appropriate categories as defined by project purpose, geographical region, and guidance from domain experts]

- Ancestry group
- Ancestry group
- Ancestry group
- Multiple (multiracial, mixed Ancestry)
- Not listed, I describe myself as: _____
- Prefer not to answer

Gender identity

How do you identify?

May select multiple

[Include appropriate gender identities as defined by project purpose, geographical region, and guidance from domain experts]

- Gender identity
- Gender identity
- Gender identity
- Prefer to self-describe: _____
- Prefer not to answer

Caution

In some parts of the world, there are laws that criminalize specific gender categories, so it may be dangerous for data contributors to answer this question honestly. Always give people a way to opt out. And work with regional experts and attorneys to conduct a careful review of the laws and cultural norms of each place where you plan

to collect data, and if needed, avoid asking this question entirely.

[Download the full guidance here.](#)

Next steps

For more information on how to work with your data:

- [Secure data access in Azure Machine Learning](#)
- [Data ingestion options for Azure Machine Learning workflows](#)
- [Optimize data processing with Azure Machine Learning](#)
- [Use differential privacy with Azure Machine Learning SDK](#)

Follow these how-to guides to work with your data after you've collected it:

- [Set up image labeling](#)
- [Label images and text](#)

Enterprise security and governance for Azure Machine Learning

9/22/2022 • 6 minutes to read • [Edit Online](#)

In this article, you'll learn about security and governance features available for Azure Machine Learning. These features are useful for administrators, DevOps, and MLOps who want to create a secure configuration that is compliant with your company's policies. With Azure Machine Learning and the Azure platform, you can:

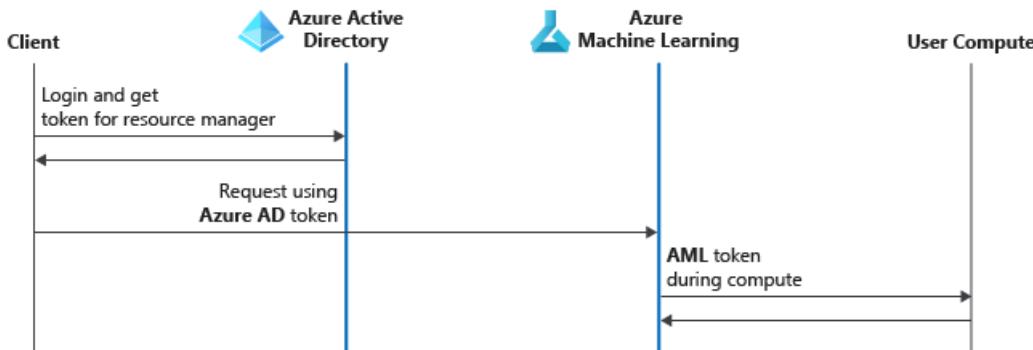
- Restrict access to resources and operations by user account or groups
- Restrict incoming and outgoing network communications
- Encrypt data in transit and at rest
- Scan for vulnerabilities
- Apply and audit configuration policies

Restrict access to resources and operations

[Azure Active Directory \(Azure AD\)](#) is the identity service provider for Azure Machine Learning. It allows you to create and manage the security objects (user, group, service principal, and managed identity) that are used to *authenticate* to Azure resources. Multi-factor authentication is supported if Azure AD is configured to use it.

Here's the authentication process for Azure Machine Learning using multi-factor authentication in Azure AD:

1. The client signs in to Azure AD and gets an Azure Resource Manager token.
2. The client presents the token to Azure Resource Manager and to all Azure Machine Learning.
3. Azure Machine Learning provides a Machine Learning service token to the user compute target (for example, Azure Machine Learning compute cluster). This token is used by the user compute target to call back into the Machine Learning service after the job is complete. The scope is limited to the workspace.



Each workspace has an associated system-assigned [managed identity](#) that has the same name as the workspace. This managed identity is used to securely access resources used by the workspace. It has the following Azure RBAC permissions on associated resources:

RESOURCE	PERMISSIONS
Workspace	Contributor
Storage account	Storage Blob Data Contributor
Key vault	Access to all keys, secrets, certificates

RESOURCE	PERMISSIONS
Azure Container Registry	Contributor
Resource group that contains the workspace	Contributor

The system-assigned managed identity is used for internal service-to-service authentication between Azure Machine Learning and other Azure resources. The identity token is not accessible to users and cannot be used by them to gain access to these resources. Users can only access the resources through [Azure Machine Learning control and data plane APIs](#), if they have sufficient RBAC permissions.

The managed identity needs Contributor permissions on the resource group containing the workspace in order to provision the associated resources, and to [deploy Azure Container Instances for web service endpoints](#).

We don't recommend that admins revoke the access of the managed identity to the resources mentioned in the preceding table. You can restore access by using the [resync keys operation](#).

NOTE

If your Azure Machine Learning workspaces has compute targets (compute cluster, compute instance, Azure Kubernetes Service, etc.) that were created **before May 14th, 2021**, you may also have an additional Azure Active Directory account. The account name starts with `Microsoft-AzureML-Support-App-` and has contributor-level access to your subscription for every workspace region.

If your workspace does not have an Azure Kubernetes Service (AKS) attached, you can safely delete this Azure AD account.

If your workspace has attached AKS clusters, *and they were created before May 14th, 2021*, **do not delete this Azure AD account**. In this scenario, you must first delete and recreate the AKS cluster before you can delete the Azure AD account.

You can provision the workspace to use user-assigned managed identity, and grant the managed identity additional roles, for example to access your own Azure Container Registry for base Docker images. For more information, see [Use managed identities for access control](#).

You can also configure managed identities for use with Azure Machine Learning compute cluster. This managed identity is independent of workspace managed identity. With a compute cluster, the managed identity is used to access resources such as secured datastores that the user running the training job may not have access to. For more information, see [Identity-based data access to storage services on Azure](#).

TIP

There are some exceptions to the use of Azure AD and Azure RBAC within Azure Machine Learning:

- You can optionally enable SSH access to compute resources such as Azure Machine Learning compute instance and compute cluster. SSH access is based on public/private key pairs, not Azure AD. SSH access is not governed by Azure RBAC.
- You can authenticate to models deployed as online endpoints using **key** or **token**-based authentication. Keys are static strings, while tokens are retrieved using an Azure AD security object. For more information, see [How to authenticate online endpoints](#).

For more information, see the following articles:

- [Authentication for Azure Machine Learning workspace](#)
- [Manage access to Azure Machine Learning](#)

- [Connect to storage services](#)
- [Use Azure Key Vault for secrets when training](#)
- [Use Azure AD managed identity with Azure Machine Learning](#)

Network security and isolation

To restrict network access to Azure Machine Learning resources, you can use [Azure Virtual Network \(VNet\)](#).

VNets allow you to create network environments that are partially, or fully, isolated from the public internet. This reduces the attack surface for your solution, as well as the chances of data exfiltration.

You might use a virtual private network (VPN) gateway to connect individual clients, or your own network, to the VNet

The Azure Machine Learning workspace can use [Azure Private Link](#) to create a private endpoint behind the VNet. This provides a set of private IP addresses that can be used to access the workspace from within the VNet. Some of the services that Azure Machine Learning relies on can also use Azure Private Link, but some rely on network security groups or user-defined routing.

For more information, see the following documents:

- [Virtual network isolation and privacy overview](#)
- [Secure workspace resources](#)
- [Secure training environment](#)
- For securing inference, see the following documents:
 - If using CLI v1 or SDK v1 - [Secure inference environment](#)
 - If using CLI v2 or SDK v2 - [Network isolation for managed online endpoints](#)
- [Use studio in a secured virtual network](#)
- [Use custom DNS](#)
- [Configure firewall](#)

Data encryption

Azure Machine Learning uses a variety of compute resources and data stores on the Azure platform. To learn more about how each of these supports data encryption at rest and in transit, see [Data encryption with Azure Machine Learning](#).

When deploying models as web services, you can enable transport-layer security (TLS) to encrypt data in transit. For more information, see [Configure a secure web service](#).

Data exfiltration prevention (preview)

Azure Machine Learning has several inbound and outbound network dependencies. Some of these dependencies can expose a data exfiltration risk by malicious agents within your organization. These risks are associated with the outbound requirements to Azure Storage, Azure Front Door, and Azure Monitor. For recommendations on mitigating this risk, see the [Azure Machine Learning data exfiltration prevention](#) article.

Vulnerability scanning

[Microsoft Defender for Cloud](#) provides unified security management and advanced threat protection across hybrid cloud workloads. For Azure machine learning, you should enable scanning of your [Azure Container Registry](#) resource and Azure Kubernetes Service resources. For more information, see [Azure Container Registry image scanning by Defender for Cloud](#) and [Azure Kubernetes Services integration with Defender for Cloud](#).

Audit and manage compliance

[Azure Policy](#) is a governance tool that allows you to ensure that Azure resources are compliant with your policies. You can set policies to allow or enforce specific configurations, such as whether your Azure Machine Learning workspace uses a private endpoint. For more information on Azure Policy, see the [Azure Policy documentation](#). For more information on the policies specific to Azure Machine Learning, see [Audit and manage compliance with Azure Policy](#).

Next steps

- [Azure Machine Learning best practices for enterprise security](#)
- [Secure Azure Machine Learning web services with TLS](#)
- [Use Azure Machine Learning with Azure Firewall](#)
- [Use Azure Machine Learning with Azure Virtual Network](#)
- [Data encryption at rest and in transit](#)
- [Build a real-time recommendation API on Azure](#)

Network traffic flow when using a secured workspace

9/22/2022 • 10 minutes to read • [Edit Online](#)

When your Azure Machine Learning workspace and associated resources are secured in an Azure Virtual Network, it changes the network traffic between resources. Without a virtual network, network traffic flows over the public internet or within an Azure data center. Once a virtual network (VNet) is introduced, you may also want to harden network security. For example, blocking inbound and outbound communications between the VNet and public internet. However, Azure Machine Learning requires access to some resources on the public internet. For example, Azure Resource Management is used for deployments and management operations.

This article lists the required traffic to/from the public internet. It also explains how network traffic flows between your client development environment and a secured Azure Machine Learning workspace in the following scenarios:

- Using Azure Machine Learning **studio** to work with:
 - Your workspace
 - AutoML
 - Designer
 - Datasets and datastores

TIP

Azure Machine Learning studio is a web-based UI that runs partially in your web browser, and makes calls to Azure services to perform tasks such as training a model, using designer, or viewing datasets. Some of these calls use a different communication flow than if you are using the SDK, CLI, REST API, or VS Code.

- Using Azure Machine Learning **studio**, **SDK**, **CLI**, or **REST API** to work with:
 - Compute instances and clusters
 - Azure Kubernetes Service
 - Docker images managed by Azure Machine Learning

TIP

If a scenario or task is not listed here, it should work the same with or without a secured workspace.

Assumptions

This article assumes the following configuration:

- Azure Machine Learning workspace using a private endpoint to communicate with the VNet.
- The Azure Storage Account, Key Vault, and Container Registry used by the workspace also use a private endpoint to communicate with the VNet.
- A VPN gateway or Express Route is used by the client workstations to access the VNet.

Inbound and outbound requirements

SCENARIO	REQUIRED INBOUND	REQUIRED OUTBOUND	ADDITIONAL CONFIGURATION
Access workspace from studio	NA	<ul style="list-style-type: none"> Azure Active Directory Azure Front Door Azure Machine Learning service 	You may need to use a custom DNS server. For more information, see Use your workspace with a custom DNS .
Use AutoML, designer, dataset, and datastore from studio	NA	NA	<ul style="list-style-type: none"> Workspace service principal configuration Allow access from trusted Azure services <p>For more information, see How to secure a workspace in a virtual network.</p>
Use compute instance and compute cluster	<ul style="list-style-type: none"> Azure Machine Learning service on port 44224 Azure Batch Management service on ports 29876-29877 	<ul style="list-style-type: none"> Azure Active Directory Azure Resource Manager Azure Machine Learning service Azure Storage Account Azure Key Vault 	If you use a firewall, create user-defined routes. For more information, see Configure inbound and outbound traffic .
Use Azure Kubernetes Service	NA	For information on the outbound configuration for AKS, see How to deploy to Azure Kubernetes Service .	Configure the Internal Load Balancer. For more information, see How to deploy to Azure Kubernetes Service .
Use Docker images managed by Azure Machine Learning	NA	<ul style="list-style-type: none"> Microsoft Container Registry <code>viennaglobal.azurecr.io</code> global container registry 	If the Azure Container Registry for your workspace is behind the VNet, configure the workspace to use a compute cluster to build images. For more information, see How to secure a workspace in a virtual network .

IMPORTANT

Azure Machine Learning uses multiple storage accounts. Each stores different data, and has a different purpose:

- **Your storage:** The Azure Storage Account(s) in your Azure subscription are used to store your data and artifacts such as models, training data, training logs, and Python scripts. For example, the *default* storage account for your workspace is in your subscription. The Azure Machine Learning compute instance and compute clusters access **file** and **blob** data in this storage over ports 445 (SMB) and 443 (HTTPS).

When using a **compute instance** or **compute cluster**, your storage account is mounted as a **file share** using the SMB protocol. The compute instance and cluster use this file share to store the data, models, Jupyter notebooks, datasets, etc. The compute instance and cluster use the private endpoint when accessing the storage account.

- **Microsoft storage:** The Azure Machine Learning compute instance and compute clusters rely on Azure Batch, and access storage located in a Microsoft subscription. This storage is used only for the management of the compute instance/cluster. None of your data is stored here. The compute instance and compute cluster access the **blob**, **table**, and **queue** data in this storage, using port 443 (HTTPS).

Machine Learning also stores metadata in an Azure Cosmos DB instance. By default, this instance is hosted in a Microsoft subscription and managed by Microsoft. You can optionally use an Azure Cosmos DB instance in your Azure subscription. For more information, see [Data encryption with Azure Machine Learning](#).

Scenario: Access workspace from studio

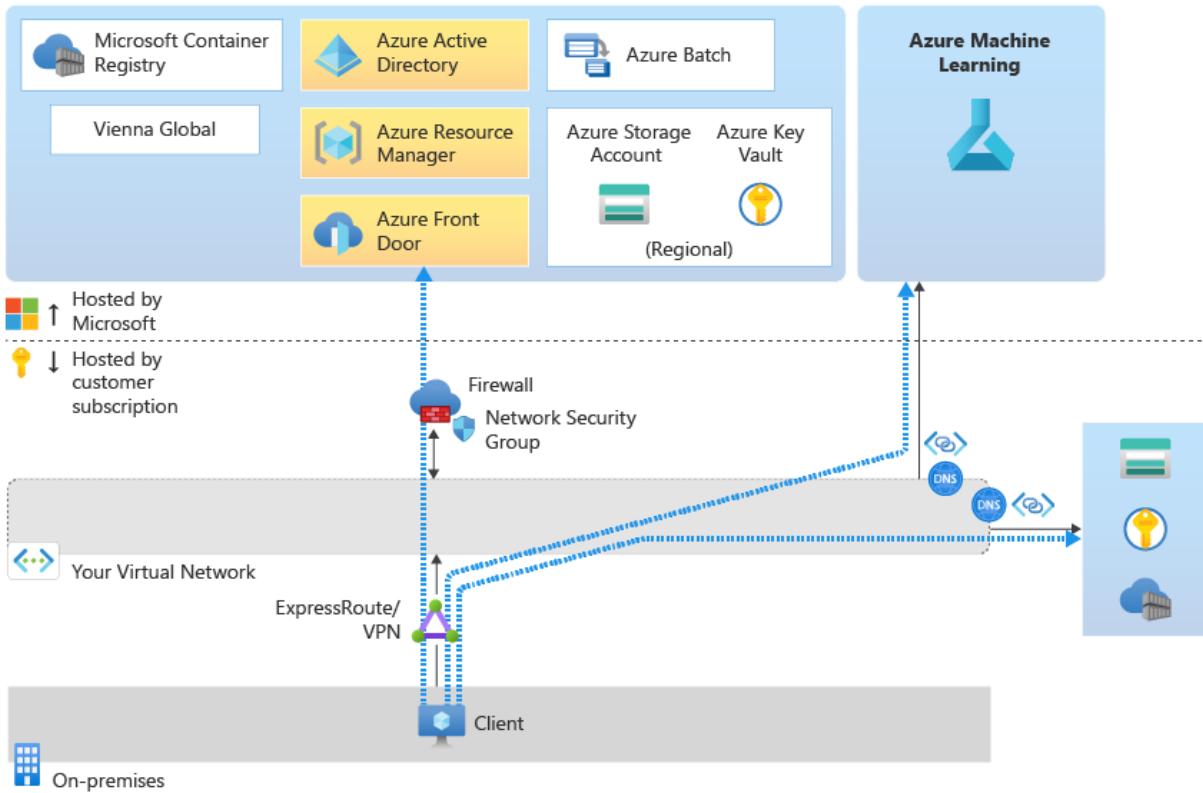
NOTE

The information in this section is specific to using the workspace from the Azure Machine Learning studio. If you use the Azure Machine Learning SDK, REST API, CLI, or Visual Studio Code, the information in this section does not apply to you.

When accessing your workspace from studio, the network traffic flows are as follows:

- To authenticate to resources, **Azure Active Directory** is used.
- For management and deployment operations, **Azure Resource Manager** is used.
- For Azure Machine Learning specific tasks, **Azure Machine Learning service** is used
- For access to Azure Machine Learning studio (<https://ml.azure.com>), **Azure FrontDoor** is used.
- For most storage operations, traffic flows through the private endpoint of the default storage for your workspace. Exceptions are discussed in the [Use AutoML, designer, dataset, and datastore](#) section.
- You also need to configure a DNS solution that allows you to resolve the names of the resources within the VNet. For more information, see [Use your workspace with a custom DNS](#).

Other Azure resources



Scenario: Use AutoML, designer, dataset, and datastore from studio

The following features of Azure Machine Learning studio use *data profiling*:

- Dataset: Explore the dataset from studio.
- Designer: Visualize module output data.
- AutoML: View a data preview/profile and choose a target column.
- Labeling

Data profiling depends on the Azure Machine Learning managed service being able to access the default Azure Storage Account for your workspace. The managed service *doesn't exist in your VNet*, so can't directly access the storage account in the VNet. Instead, the workspace uses a service principal to access storage.

TIP

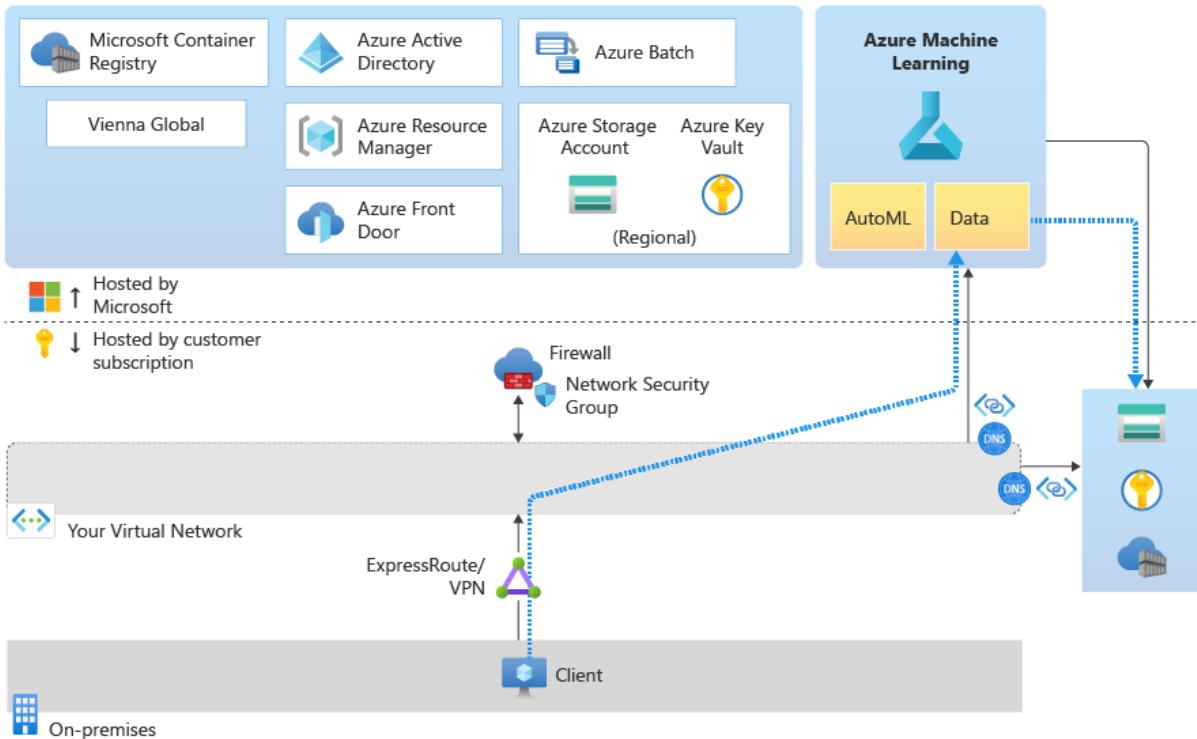
You can provide a service principal when creating the workspace. If you do not, one is created for you and will have the same name as your workspace.

To allow access to the storage account, configure the storage account to allow a **resource instance** for your workspace or select the **Allow Azure services on the trusted services list to access this storage account**. This setting allows the managed service to access storage through the Azure data center network.

Next, add the service principal for the workspace to the **Reader** role to the private endpoint of the storage account. This role is used to verify the workspace and storage subnet information. If they're the same, access is allowed. Finally, the service principal also requires **Blob data contributor** access to the storage account.

For more information, see the Azure Storage Account section of [How to secure a workspace in a virtual network](#).

Other Azure resources



Scenario: Use compute instance and compute cluster

Azure Machine Learning compute instance and compute cluster are managed services hosted by Microsoft. They're built on top of the Azure Batch service. While they exist in a Microsoft managed environment, they're also injected into your VNet.

When you create a compute instance or compute cluster, the following resources are also created in your VNet:

- A Network Security Group with required outbound rules. These rules allow **inbound** access from the Azure Machine Learning (TCP on port 44224) and Azure Batch service (TCP on ports 29876-29877).

IMPORTANT

If you use a firewall to block internet access into the VNet, you must configure the firewall to allow this traffic. For example, with Azure Firewall you can create user-defined routes. For more information, see [How to use Azure Machine Learning with a firewall](#).

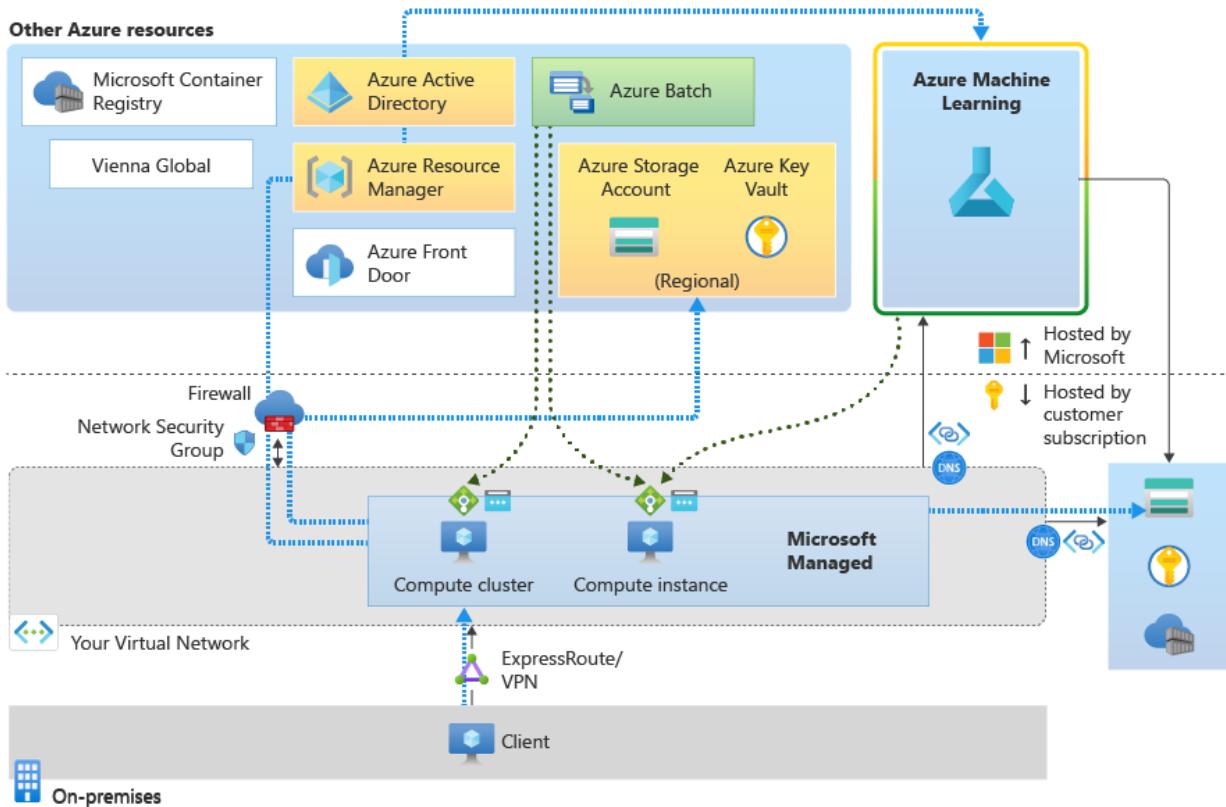
- A load balancer with a public IP.

Also allow **outbound** access to the following service tags. For each tag, replace `region` with the Azure region of your compute instance/cluster:

- `Storage.region` - This outbound access is used to connect to the Azure Storage Account inside the Azure Batch service-managed VNet.
- `Keyvault.region` - This outbound access is used to connect to the Azure Key Vault account inside the Azure Batch service-managed VNet.

Data access from your compute instance or cluster goes through the private endpoint of the Storage Account for your VNet.

If you use Visual Studio Code on a compute instance, you must allow other outbound traffic. For more information, see [How to use Azure Machine Learning with a firewall](#).



Scenario: Use online endpoints

Securing an online endpoint with a private endpoint is a preview feature.

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Inbound communication with the scoring URL of the online endpoint can be secured using the `public_network_access` flag on the endpoint. Setting the flag to `disabled` restricts the online endpoint to receiving traffic only from the virtual network. For secure inbound communications, the Azure Machine Learning workspace's private endpoint is used.

Outbound communication from a deployment can be secured on a per-deployment basis by using the `egress_public_network_access` flag. Outbound communication in this case is from the deployment to Azure Container Registry, storage blob, and workspace. Setting the flag to `true` will restrict communication with these resources to the virtual network.

NOTE

For secure outbound communication, a private endpoint is created for each deployment where `egress_public_network_access` is set to `disabled`.

Visibility of the endpoint is also governed by the `public_network_access` flag of the Azure Machine Learning workspace. If this flag is `disabled`, then the scoring endpoints can only be accessed from virtual networks that contain a private endpoint for the workspace. If it is `enabled`, then the scoring endpoint can be accessed from the virtual network and public networks.

Supported configurations

CONFIGURATION	INBOUND (ENDPOINT PROPERTY)	OUTBOUND (DEPLOYMENT PROPERTY)	SUPPORTED?
secure inbound with secure outbound	<code>public_network_access</code> is disabled	<code>egress_public_network_access</code> is disabled	Yes
secure inbound with public outbound	<code>public_network_access</code> is disabled	<code>egress_public_network_access</code> is enabled	Yes
public inbound with secure outbound	<code>public_network_access</code> is enabled	<code>egress_public_network_access</code> is disabled	Yes
public inbound with public outbound	<code>public_network_access</code> is enabled	<code>egress_public_network_access</code> is enabled	Yes

Scenario: Use Azure Kubernetes Service

For information on the outbound configuration required for Azure Kubernetes Service, see the connectivity requirements section of [How to deploy to Azure Kubernetes Service](#).

NOTE

The Azure Kubernetes Service load balancer is not the same as the load balancer created by Azure Machine Learning. If you want to host your model as a secured application, only available on the VNet, use the internal load balancer created by Azure Machine Learning. If you want to allow public access, use the public load balancer created by Azure Machine Learning.

If your model requires extra inbound or outbound connectivity, such as to an external data source, use a network security group or your firewall to allow the traffic.

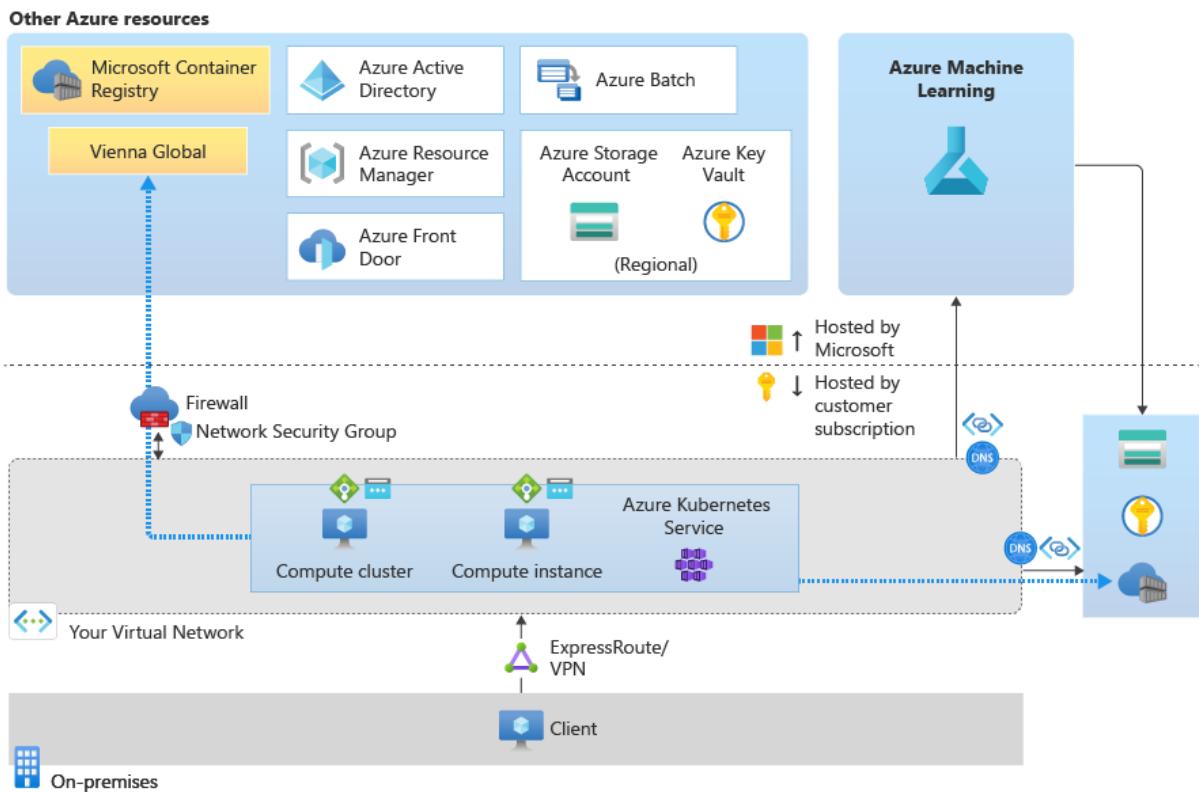
Scenario: Use Docker images managed by Azure ML

Azure Machine Learning provides Docker images that can be used to train models or perform inference. If you don't specify your own images, the ones provided by Azure Machine Learning are used. These images are hosted on the Microsoft Container Registry (MCR). They're also hosted on a geo-replicated Azure Container Registry named `viennaglobal.azurecr.io`.

If you provide your own docker images, such as on an Azure Container Registry that you provide, you don't need the outbound communication with MCR or `viennaglobal.azurecr.io`.

TIP

If your Azure Container Registry is secured in the VNet, it cannot be used by Azure Machine Learning to build Docker images. Instead, you must designate an Azure Machine Learning compute cluster to build images. For more information, see [How to secure a workspace in a virtual network](#).



Next steps

Now that you've learned how network traffic flows in a secured configuration, learn more about securing Azure ML in a virtual network by reading the [Virtual network isolation and privacy overview](#) article.

For information on best practices, see the [Azure Machine Learning best practices for enterprise security](#) article.

Azure Policy Regulatory Compliance controls for Azure Machine Learning

9/22/2022 • 4 minutes to read • [Edit Online](#)

Regulatory Compliance in Azure Policy provides Microsoft created and managed initiative definitions, known as **built-ins**, for the **compliance domains** and **security controls** related to different compliance standards. This page lists the **compliance domains** and **security controls** for Azure Machine Learning. You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the [Azure Policy GitHub repo](#).

IMPORTANT

Each control is associated with one or more [Azure Policy](#) definitions. These policies might help you [assess compliance](#) with the control. However, there often isn't a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves. This doesn't ensure that you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards can change over time.

Azure Security Benchmark

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Network Security	NS-2	Secure cloud services with network controls	Azure Machine Learning workspaces should use private link	1.1.0
Data Protection	DP-5	Use customer-managed key option in data at rest encryption when required	Azure Machine Learning workspaces should be encrypted with a customer-managed key	1.0.3

FedRAMP High

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP High](#). For more information about this compliance standard,

see [FedRAMP High](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Azure Machine Learning workspaces should use private link	1.1.0
Access Control	AC-17	Remote Access	Azure Machine Learning workspaces should use private link	1.1.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-7	Boundary Protection	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-7 (3)	Access Points	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Azure Machine Learning workspaces should be encrypted with a customer-managed key	1.0.3

FedRAMP Moderate

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP Moderate](#). For more information about this compliance standard, see [FedRAMP Moderate](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Azure Machine Learning workspaces should use private link	1.1.0
Access Control	AC-17	Remote Access	Azure Machine Learning workspaces should use private link	1.1.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Access Control	AC-17 (1)	Automated Monitoring / Control	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-7	Boundary Protection	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-7 (3)	Access Points	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Azure Machine Learning workspaces should be encrypted with a customer-managed key	1.0.3

New Zealand ISM Restricted

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - New Zealand ISM Restricted](#). For more information about this compliance standard, see [New Zealand ISM Restricted](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Infrastructure	INF-9	10.8.35 Security Architecture	Azure Machine Learning workspaces should use private link	1.1.0
Cryptography	CR-3	17.1.46 Reducing storage and physical transfer requirements	Azure Machine Learning workspaces should be encrypted with a customer-managed key	1.0.3

NIST SP 800-53 Rev. 5

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 5](#). For more information about this compliance standard, see [NIST SP 800-53 Rev. 5](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Azure Machine Learning workspaces should use private link	1.1.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Access Control	AC-17	Remote Access	Azure Machine Learning workspaces should use private link	1.1.0
Access Control	AC-17 (1)	Monitoring and Control	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-7	Boundary Protection	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-7 (3)	Access Points	Azure Machine Learning workspaces should use private link	1.1.0
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Azure Machine Learning workspaces should be encrypted with a customer-managed key	1.0.3

NZ ISM Restricted v3.5

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NZ ISM Restricted v3.5](#). For more information about this compliance standard, see [NZ ISM Restricted v3.5](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Cryptography	NZISM Security Benchmark CR-3	17.1.53 Reducing storage and physical transfer requirements	Azure Machine Learning workspaces should be encrypted with a customer-managed key	1.0.3
Infrastructure	NZISM Security Benchmark INF-9	10.8.35 Security Architecture	Azure Machine Learning workspaces should use private link	1.1.0

Next steps

- Learn more about [Azure Policy Regulatory Compliance](#).
- See the built-ins on the [Azure Policy GitHub repo](#).

Data encryption with Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

Azure Machine Learning uses a variety of Azure data storage services and compute resources when training models and performing inference. Each of these has their own story on how they provide encryption for data at rest and in transit. In this article, learn about each one and which is best for your scenario.

IMPORTANT

For production grade encryption during **training**, Microsoft recommends using Azure Machine Learning compute cluster.

For production grade encryption during **inference**, Microsoft recommends using Azure Kubernetes Service.

Azure Machine Learning compute instance is a dev/test environment. When using it, we recommend that you store your files, such as notebooks and scripts, in a file share. Your data should be stored in a datastore.

Encryption at rest

Azure Machine Learning relies on multiple Azure Services, each of which have their own encryption capabilities.

Azure Blob storage

Azure Machine Learning stores snapshots, output, and logs in the Azure Blob storage account (default storage account) that's tied to the Azure Machine Learning workspace and your subscription. All the data stored in Azure Blob storage is encrypted at rest with Microsoft-managed keys.

For information on how to use your own keys for data stored in Azure Blob storage, see [Azure Storage encryption with customer-managed keys in Azure Key Vault](#).

Training data is typically also stored in Azure Blob storage so that it's accessible to training compute targets. This storage isn't managed by Azure Machine Learning but mounted to compute targets as a remote file system.

If you need to **rotate or revoke** your key, you can do so at any time. When rotating a key, the storage account will start using the new key (latest version) to encrypt data at rest. When revoking (disabling) a key, the storage account takes care of failing requests. It usually takes an hour for the rotation or revocation to be effective.

For information on regenerating the access keys, see [Regenerate storage access keys](#).

Azure Cosmos DB

Azure Machine Learning stores metadata in an Azure Cosmos DB instance. This instance is associated with a Microsoft subscription managed by Azure Machine Learning. All the data stored in Azure Cosmos DB is encrypted at rest with Microsoft-managed keys.

When using your own (customer-managed) keys to encrypt the Azure Cosmos DB instance, a Microsoft managed Azure Cosmos DB instance is created in your subscription. This instance is created in a Microsoft-managed resource group, which is different than the resource group for your workspace. For more information, see [Customer-managed keys](#).

Azure Container Registry

All container images in your registry (Azure Container Registry) are encrypted at rest. Azure automatically encrypts an image before storing it and decrypts it when Azure Machine Learning pulls the image.

To use your own (customer-managed) keys to encrypt your Azure Container Registry, you need to create your own ACR and attach it while provisioning the workspace or encrypt the default instance that gets created at the

time of workspace provisioning.

IMPORTANT

Azure Machine Learning requires the admin account be enabled on your Azure Container Registry. By default, this setting is disabled when you create a container registry. For information on enabling the admin account, see [Admin account](#).

Once an Azure Container Registry has been created for a workspace, do not delete it. Doing so will break your Azure Machine Learning workspace.

For an example of creating a workspace using an existing Azure Container Registry, see the following articles:

- [Create a workspace for Azure Machine Learning with Azure CLI](#).
- [Create a workspace with Python SDK](#).
- [Use an Azure Resource Manager template to create a workspace for Azure Machine Learning](#)

Azure Container Instance

You may encrypt a deployed Azure Container Instance (ACI) resource using customer-managed keys. The customer-managed key used for ACI can be stored in the Azure Key Vault for your workspace. For information on generating a key, see [Encrypt data with a customer-managed key](#).

To use the key when deploying a model to Azure Container Instance, create a new deployment configuration using `AciWebservice.deploy_configuration()`. Provide the key information using the following parameters:

- `cmk_vault_base_url` : The URL of the key vault that contains the key.
- `cmk_key_name` : The name of the key.
- `cmk_key_version` : The version of the key.

For more information on creating and using a deployment configuration, see the following articles:

- [AciWebservice.deploy_configuration\(\) reference](#)
- [Where and how to deploy](#)

For more information on using a customer-managed key with ACI, see [Encrypt data with a customer-managed key](#).

Azure Kubernetes Service

You may encrypt a deployed Azure Kubernetes Service resource using customer-managed keys at any time. For more information, see [Bring your own keys with Azure Kubernetes Service](#).

This process allows you to encrypt both the Data and the OS Disk of the deployed virtual machines in the Kubernetes cluster.

IMPORTANT

This process only works with AKS K8s version 1.17 or higher. Azure Machine Learning added support for AKS 1.17 on Jan 13, 2020.

Machine Learning Compute

Compute cluster The OS disk for each compute node stored in Azure Storage is encrypted with Microsoft-managed keys in Azure Machine Learning storage accounts. This compute target is ephemeral, and clusters are typically scaled down when no jobs are queued. The underlying virtual machine is de-provisioned, and the OS disk is deleted. Azure Disk Encryption isn't supported for the OS disk.

Each virtual machine also has a local temporary disk for OS operations. If you want, you can use the disk to

stage training data. If the workspace was created with the `hbi_workspace` parameter set to `TRUE`, the temporary disk is encrypted. This environment is short-lived (only for the duration of your job,) and encryption support is limited to system-managed keys only.

Compute instance The OS disk for compute instance is encrypted with Microsoft-managed keys in Azure Machine Learning storage accounts. If the workspace was created with the `hbi_workspace` parameter set to `TRUE`, the local temporary disk on compute instance is encrypted with Microsoft managed keys. Customer managed key encryption is not supported for OS and temp disk.

For more information, see [Customer-managed keys](#).

Azure Databricks

Azure Databricks can be used in Azure Machine Learning pipelines. By default, the Databricks File System (DBFS) used by Azure Databricks is encrypted using a Microsoft-managed key. To configure Azure Databricks to use customer-managed keys, see [Configure customer-managed keys on default \(root\) DBFS](#).

Microsoft-generated data

When using services such as Automated Machine Learning, Microsoft may generate a transient, pre-processed data for training multiple models. This data is stored in a datastore in your workspace, which allows you to enforce access controls and encryption appropriately.

You may also want to encrypt [diagnostic information logged from your deployed endpoint](#) into your Azure Application Insights instance.

Encryption in transit

Azure Machine Learning uses TLS to secure internal communication between various Azure Machine Learning microservices. All Azure Storage access also occurs over a secure channel.

To secure external calls made to the scoring endpoint, Azure Machine Learning uses TLS. For more information, see [Use TLS to secure a web service through Azure Machine Learning](#).

Data collection and handling

Microsoft collected data

Microsoft may collect non-user identifying information like resource names (for example the dataset name, or the machine learning experiment name), or job environment variables for diagnostic purposes. All such data is stored using Microsoft-managed keys in storage hosted in Microsoft owned subscriptions and follows [Microsoft's standard Privacy policy and data handling standards](#). This data is kept within the same region as your workspace.

Microsoft also recommends not storing sensitive information (such as account key secrets) in environment variables. Environment variables are logged, encrypted, and stored by us. Similarly when naming your jobs, avoid including sensitive information such as user names or secret project names. This information may appear in telemetry logs accessible to Microsoft Support engineers.

You may opt out from diagnostic data being collected by setting the `hbi_workspace` parameter to `TRUE` while provisioning the workspace. This functionality is supported when using the AzureML Python SDK, the Azure CLI, REST APIs, or Azure Resource Manager templates.

Using Azure Key Vault

Azure Machine Learning uses the Azure Key Vault instance associated with the workspace to store credentials of various kinds:

- The associated storage account connection string

- Passwords to Azure Container Repository instances
- Connection strings to data stores

SSH passwords and keys to compute targets like Azure HDInsight and VMs are stored in a separate key vault that's associated with the Microsoft subscription. Azure Machine Learning doesn't store any passwords or keys provided by users. Instead, it generates, authorizes, and stores its own SSH keys to connect to VMs and HDInsight to run the experiments.

Each workspace has an associated system-assigned managed identity that has the same name as the workspace. This managed identity has access to all keys, secrets, and certificates in the key vault.

Next steps

- [Connect to Azure storage](#)
- [Get data from a datastore](#)
- [Connect to data](#)
- [Train with datasets](#)
- [Customer-managed keys.](#)

Customer-managed keys for Azure Machine Learning

9/22/2022 • 6 minutes to read • [Edit Online](#)

Azure Machine Learning is built on top of multiple Azure services. While the data is stored securely using encryption keys that Microsoft provides, you can enhance security by also providing your own (customer-managed) keys. The keys you provide are stored securely using Azure Key Vault.

Customer-managed keys are used with the following services that Azure Machine Learning relies on:

SERVICE	WHAT IT'S USED FOR
Azure Cosmos DB	Stores metadata for Azure Machine Learning
Azure Cognitive Search	Stores workspace metadata for Azure Machine Learning
Azure Storage Account	Stores workspace metadata for Azure Machine Learning
Azure Container Instance	Hosting trained models as inference endpoints
Azure Kubernetes Service	Hosting trained models as inference endpoints

TIP

- Azure Cosmos DB, Cognitive Search, and Storage Account are secured using the same key. You can use a different key for Azure Kubernetes Service and Container Instance.
- To use a customer-managed key with Azure Cosmos DB, Cognitive Search, and Storage Account, the key is provided when you create your workspace. The key(s) used with Azure Container Instance and Kubernetes Service are provided when configuring those resources.

In addition to customer-managed keys, Azure Machine Learning also provides a [hbi_workspace flag](#). Enabling this flag reduces the amount of data Microsoft collects for diagnostic purposes and enables [extra encryption in Microsoft-managed environments](#). This flag also enables the following behaviors:

- Starts encrypting the local scratch disk in your Azure Machine Learning compute cluster, provided you haven't created any previous clusters in that subscription. Else, you need to raise a support ticket to enable encryption of the scratch disk of your compute clusters.
- Cleans up your local scratch disk between jobs.
- Securely passes credentials for your storage account, container registry, and SSH account from the execution layer to your compute clusters using your key vault.

TIP

The `hbi_workspace` flag does not impact encryption in transit, only encryption at rest.

Prerequisites

- An Azure subscription.

- An Azure Key Vault instance. The key vault contains the key(s) used to encrypt your services.

- The key vault instance must enable soft delete and purge protection.
- The managed identity for the services secured by a customer-managed key must have the following permissions in key vault:
 - wrap key
 - unwrap key
 - get

For example, the managed identity for Azure Cosmos DB would need to have those permissions to the key vault.

Limitations

- The customer-managed key for resources the workspace depends on can't be updated after workspace creation.
- Resources managed by Microsoft in your subscription can't transfer ownership to you.
- You can't delete Microsoft-managed resources used for customer-managed keys without also deleting your workspace.

How workspace metadata is stored

The following resources store metadata for your workspace:

SERVICE	HOW IT'S USED
Azure Cosmos DB	Stores job history data.
Azure Cognitive Search	Stores indices that are used to help query your machine learning content.
Azure Storage Account	Stores other metadata such as Azure Machine Learning pipelines data.

Your Azure Machine Learning workspace reads and writes data using its managed identity. This identity is granted access to the resources using a role assignment (Azure role-based access control) on the data resources. The encryption key you provide is used to encrypt data that is stored on Microsoft-managed resources. It's also used to create indices for Azure Cognitive Search, which are created at runtime.

Customer-managed keys

When you **don't use a customer-managed key**, Microsoft creates and manages these resources in a Microsoft owned Azure subscription and uses a Microsoft-managed key to encrypt the data.

When you **use a customer-managed key**, these resources are *in your Azure subscription* and encrypted with your key. While they exist in your subscription, these resources are **managed by Microsoft**. They're automatically created and configured when you create your Azure Machine Learning workspace.

IMPORTANT

When using a customer-managed key, the costs for your subscription will be higher because these resources are in your subscription. To estimate the cost, use the [Azure pricing calculator](#).

These Microsoft-managed resources are located in a new Azure resource group created in your subscription. This group is in addition to the resource group for your workspace. This resource group will contain the Microsoft-managed resources that your key is used with. The resource group will be named using the formula of <Azure Machine Learning workspace resource group name><GUID>.

TIP

- The [Request Units](#) for the Azure Cosmos DB automatically scale as needed.
- If your Azure Machine Learning workspace uses a private endpoint, this resource group will also contain a Microsoft-managed Azure Virtual Network. This VNet is used to secure communications between the managed services and the workspace. You **cannot provide your own VNet for use with the Microsoft-managed resources**. You also **cannot modify the virtual network**. For example, you cannot change the IP address range that it uses.

IMPORTANT

If your subscription does not have enough quota for these services, a failure will occur.

WARNING

Don't delete the resource group that contains this Azure Cosmos DB instance, or any of the resources automatically created in this group. If you need to delete the resource group or Microsoft-managed services in it, you must delete the Azure Machine Learning workspace that uses it. The resource group resources are deleted when the associated workspace is deleted.

How compute data is stored

Azure Machine Learning uses compute resources to train and deploy machine learning models. The following table describes the compute options and how data is encrypted by each one:

COMPUTE	ENCRYPTION
Azure Container Instance	Data is encrypted by a Microsoft-managed key or a customer-managed key. For more information, see Encrypt data with a customer-managed key .
Azure Kubernetes Service	Data is encrypted by a Microsoft-managed key or a customer-managed key. For more information, see Bring your own keys with Azure disks in Azure Kubernetes Services .
Azure Machine Learning compute instance	Local scratch disk is encrypted if the <code>hbi_workspace</code> flag is enabled for the workspace.
Azure Machine Learning compute cluster	OS disk encrypted in Azure Storage with Microsoft-managed keys. Temporary disk is encrypted if the <code>hbi_workspace</code> flag is enabled for the workspace.

Compute cluster The OS disk for each compute node stored in Azure Storage is encrypted with Microsoft-managed keys in Azure Machine Learning storage accounts. This compute target is ephemeral, and clusters are typically scaled down when no jobs are queued. The underlying virtual machine is de-provisioned, and the OS disk is deleted. Azure Disk Encryption isn't supported for the OS disk.

Each virtual machine also has a local temporary disk for OS operations. If you want, you can use the disk to stage training data. If the workspace was created with the `hbi_workspace` parameter set to `TRUE`, the temporary disk is encrypted. This environment is short-lived (only during your job) and encryption support is limited to system-managed keys only.

Compute instance The OS disk for compute instance is encrypted with Microsoft-managed keys in Azure Machine Learning storage accounts. If the workspace was created with the `hbi_workspace` parameter set to `TRUE`, the local temporary disk on compute instance is encrypted with Microsoft managed keys. Customer managed key encryption isn't supported for OS and temp disk.

HBI_workspace flag

- The `hbi_workspace` flag can only be set when a workspace is created. It can't be changed for an existing workspace.
- When this flag is set to True, it may increase the difficulty of troubleshooting issues because less telemetry data is sent to Microsoft. There's less visibility into success rates or problem types. Microsoft may not be able to react as proactively when this flag is True.

To enable the `hbi_workspace` flag when creating an Azure Machine Learning workspace, follow the steps in one of the following articles:

- [How to create and manage a workspace.](#)
- [How to create and manage a workspace using the Azure CLI.](#)
- [How to create a workspace using Hashicorp Terraform.](#)
- [How to create a workspace using Azure Resource Manager templates.](#)

Next Steps

- [How to configure customer-managed keys with Azure Machine Learning.](#)

Vulnerability management for Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

Vulnerability management involves detecting, assessing, mitigating, and reporting on any security vulnerabilities that exist in an organization's systems and software. Vulnerability management is a shared responsibility between you and Microsoft.

In this article, we discuss these responsibilities and outline the vulnerability management controls provided by Azure Machine Learning. You'll learn how to keep your service instance and applications up to date with the latest security updates, and how to minimize the window of opportunity for attackers.

Microsoft-managed VM images

Azure Machine Learning manages host OS VM images for Azure ML compute instance, Azure ML compute clusters, and Data Science Virtual Machines. The update frequency is monthly and includes the following:

- For each new VM image version, the latest updates are sourced from the original publisher of the OS. Using the latest updates ensures that all OS-related patches that are applicable are picked. For Azure Machine Learning, the publisher is Canonical for all the Ubuntu 18 images. These images are used for Azure Machine Learning compute instances, compute clusters, and Data Science Virtual Machines.
- VM images are updated monthly.
- In addition to patches applied by the original publisher, Azure Machine Learning updates system packages when updates are available.
- Azure Machine Learning checks and validates any machine learning packages that may require an upgrade. In most circumstances, new VM images contain the latest package versions.
- All VM images are built on secure subscriptions that run vulnerability scanning regularly. Any unaddressed vulnerabilities are flagged and are to be fixed within the next release.
- The frequency is on a monthly interval for most images. For compute instance, the image release is aligned with the Azure ML SDK release cadence as it comes preinstalled in the environment.

Next to the regular release cadence, hot fixes are applied in the case vulnerabilities are discovered. Hot fixes get rolled out within 72 hours for Azure ML compute and within a week for Compute Instance.

NOTE

The host OS is not the OS version you might specify for an [environment](#) when training or deploying a model. Environments run inside Docker. Docker runs on the host OS.

Microsoft-managed container images

[Base docker images](#) maintained by Azure Machine Learning get security patches frequently to address newly discovered vulnerabilities.

Azure Machine Learning releases updates for supported images every two weeks to address vulnerabilities. As a commitment, we aim to have no vulnerabilities older than 30 days in the latest version of supported images.

Patched images are released under new immutable tag and also updated `:latest` tag. Using the `:latest` tag or pinning to a particular image version may be a trade-off of security and environment reproducibility for your

machine learning job.

Managing environments and container images

Reproducibility is a key aspect of software development and machine learning experimentation. [Azure Machine Learning Environment](#) component's primary focus is to guarantee reproducibility of the environment where user's code gets executed. To ensure reproducibility for any machine learning job, earlier built images will be pulled to the compute nodes without a need of rematerialization.

While Azure Machine Learning patches base images with each release, whether you use the latest image may be tradeoff between reproducibility and vulnerability management. So, it's your responsibility to choose the environment version used for your jobs or model deployments.

By default, dependencies are layered on top of base images provided by Azure ML when building environments. You can also use your own base images when using environments in Azure Machine Learning. Once you install more dependencies on top of the Microsoft-provided images, or bring your own base images, vulnerability management becomes your responsibility.

Associated to your Azure Machine Learning workspace is an Azure Container Registry instance that's used as a cache for container images. Any image materialized, is pushed to the container registry, and used if experimentation or deployment is triggered for the corresponding environment. Azure Machine Learning doesn't delete any image from your container registry, and it's your responsibility to evaluate the need of an image over time. To monitor and maintain environment hygiene, you can use [Microsoft Defender for Container Registry](#) to help scan your images for vulnerabilities. To automate your processes based on triggers from Microsoft Defender, see [Automate responses to Microsoft Defender for Cloud triggers](#).

Using a private package repository

Azure Machine Learning uses Conda for package installations. By default, packages are downloaded from public repositories. In case your organization requires packages to be sourced only from private repositories, you may override the conda configuration as part of your base image. Below example configuration shows how to remove the default channels, and add your own private conda feed.

```
RUN conda config --set offline false \
&& conda config --remove channels defaults || true \
&& conda config --add channels https://my.private.conda.feed/conda/feed
```

See [use your own dockerfile](#) to learn how to specify your own base images in Azure Machine Learning. For more details on configuring Conda environments, see [Conda - Creating an environment file manually](#).

Vulnerability management on compute hosts

Managed compute nodes in Azure Machine Learning make use of Microsoft-managed OS VM images and pull the latest updated VM image at the time that a node gets provisioned. This applies to compute instance, compute cluster, and managed inference compute SKUs. While OS VM images are regularly patched, compute nodes are not actively scanned for vulnerabilities while in use. For an extra layer of protection, consider network isolation of your compute.

It's a shared responsibility between you and Microsoft to ensure that your environment is up-to-date and compute nodes use the latest OS version. Nodes that are non-idle can't get updated to the latest VM image. Considerations are slightly different for each compute type, as listed in the following sections.

Compute instance

- Compute instances get latest VM images at time of provisioning.
- Microsoft doesn't provide active OS patching for compute instance. To obtain the latest VM image, delete

and recreate the compute instance.

- You could use set up scripts to install extra scanning software. Azure Defender agents are currently not supported.
- To query resource age, you could use the following log analytics query:

```
AmlComputeClusterEvent
| where ClusterType == "DSI" and EventType == "CreateOperationCompleted" and split(_ResourceId, "/")[-1]==<wsname>
| project ClusterName, TimeCreated=TimeGenerated
| summarize Last_Time_Created=arg_max(TimeGenerated, *) by ClusterName
| join kind = leftouter (AmlComputeClusterEvent
    | where ClusterType == "DSI" and EventType == "DeleteOperationCompleted"
    | project ClusterName, TimeGenerated
    | summarize Last_Time_Deleted=arg_max(TimeGenerated, *) by ClusterName)
on ClusterName
| where (Last_Time_Created > Last_Time_Deleted or isnull(Last_Time_Deleted)) and Last_Time_Created < ago(30days)
| project ClusterName, Last_Time_Created, Last_Time_Deleted
```

Compute clusters

Compute clusters automatically upgrade to the latest VM image. If the cluster is configured with min nodes = 0, it automatically upgrades nodes to the latest VM image version when all jobs are completed and the cluster reduces to zero nodes.

- There are conditions in which cluster nodes do not scale down, and as a result are unable to get the latest VM images.
 - Cluster minimum node count may be set to a value greater than 0.
 - Jobs may be scheduled continuously on your cluster.
- It is your responsibility to scale non-idle cluster nodes down to get the latest OS VM image updates. Azure Machine Learning does not abort any running workloads on compute nodes to issue VM updates.
 - Temporarily change the minimum nodes to zero and allow the cluster to reduce to zero nodes.

Managed online endpoints

- Managed Online Endpoints automatically receive OS host image updates that include vulnerability fixes. The update frequency of images is at least once a month.
- Compute nodes get automatically upgraded to the latest VM image version once released. There's no action required on you.

Customer managed Kubernetes clusters

[Kubernetes compute](#) lets you configure Kubernetes clusters to train, inference, and manage models in Azure Machine Learning.

- Because you manage the environment with Kubernetes, both OS VM vulnerabilities and container image vulnerability management is your responsibility.
- Azure Machine Learning frequently publishes new versions of AzureML extension container images into Microsoft Container Registry. It's Microsoft's responsibility to ensure new image versions are free from vulnerabilities. Vulnerabilities are fixed with [each release](#).
- When your clusters run jobs without interruption, running jobs may run outdated container image versions. Once you upgrade the amlarc extension to a running cluster, newly submitted jobs will start to use the latest image version. When upgrading the AMLArc extension to its latest version, clean up the old container image versions from the clusters as required.
- Observability on whether your Azure Arc cluster is running the latest version of AMLArc, you can find via the Azure portal. Under your Arc resource of the type 'Kubernetes - Azure Arc', see 'Extensions' to find the

version of the AMLArc extension.

Automated ML and Designer environments

For code-based training experiences, you control which Azure Machine Learning environment is used. With AutoML and Designer, the environment is encapsulated as part of the service. These types of jobs can run on computes configured by you, allowing for extra controls such as network isolation.

- Automated ML jobs run on environments that layer on top of Azure ML [base docker images](#).
- Designer jobs are compartmentalized into [Components](#). Each component has its own environment that layers on top of the Azure ML base docker images. For more information on components, see the [Component reference](#).

Next steps

- [Azure Machine Learning Base Images Repository](#)
- [Data Science Virtual Machine release notes](#)
- [AzureML Python SDK Release Notes](#)
- [Machine learning enterprise security](#)

How to create a secure workspace

9/22/2022 • 17 minutes to read • [Edit Online](#)

In this article, learn how to create and connect to a secure Azure Machine Learning workspace. A secure workspace uses Azure Virtual Network to create a security boundary around resources used by Azure Machine Learning.

In this tutorial, you accomplish the following tasks:

- Create an Azure Virtual Network (VNet) to **secure communications between services in the virtual network**.
- Create an Azure Storage Account (blob and file) behind the VNet. This service is used as **default storage for the workspace**.
- Create an Azure Key Vault behind the VNet. This service is used to **store secrets used by the workspace**. For example, the security information needed to access the storage account.
- Create an Azure Container Registry (ACR). This service is used as a repository for Docker images. **Docker images provide the compute environments needed when training a machine learning model or deploying a trained model as an endpoint**.
- Create an Azure Machine Learning workspace.
- Create a jump box. A jump box is an Azure Virtual Machine that is behind the VNet. Since the VNet restricts access from the public internet, **the jump box is used as a way to connect to resources behind the VNet**.
- Configure Azure Machine Learning studio to work behind a VNet. The studio provides a **web interface for Azure Machine Learning**.
- Create an Azure Machine Learning compute cluster. A compute cluster is used when **training machine learning models in the cloud**. In configurations where Azure Container Registry is behind the VNet, it is also used to build Docker images.
- Connect to the jump box and use the Azure Machine Learning studio.

TIP

If you're looking for a template (Microsoft Bicep or Hashicorp Terraform) that demonstrates how to create a secure workspace, see [Tutorial - Create a secure workspace using a template](#).

Prerequisites

- Familiarity with Azure Virtual Networks and IP networking. If you are not familiar, try the [Fundamentals of computer networking](#) module.
- While most of the steps in this article use the Azure portal or the Azure Machine Learning studio, some steps use the Azure CLI extension for Machine Learning v2.

Limitations

The steps in this article put Azure Container Registry behind the VNet. In this configuration, you can't deploy models to Azure Container Instances inside the VNet. For more information, see [Secure the inference environment](#).

TIP

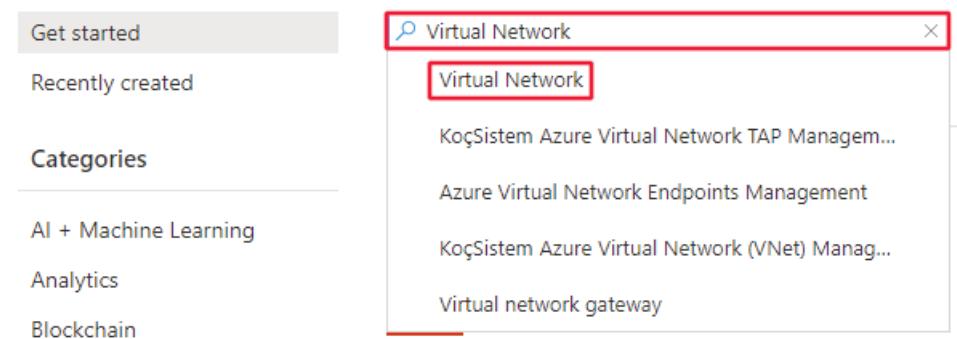
As an alternative to Azure Container Instances, try Azure Machine Learning managed online endpoints. For more information, see [Enable network isolation for managed online endpoints \(preview\)](#).

Create a virtual network

To create a virtual network, use the following steps:

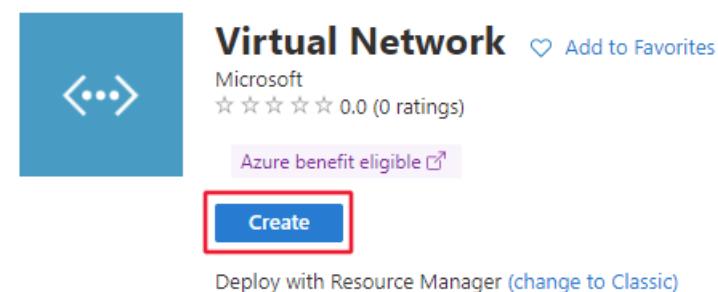
1. In the [Azure portal](#), select the portal menu in the upper left corner. From the menu, select **+ Create a resource** and then enter **Virtual Network** in the search field. Select the **Virtual Network** entry, and then select **Create**.

Create a resource



Virtual Network

Microsoft



2. From the **Basics** tab, select the **Azure subscription** to use for this resource and then select or create a new **resource group**. Under **Instance details**, enter a friendly **name** for your virtual network and select the **region** to create it in.

Create virtual network ...

Basics IP Addresses Security Tags Review + create

Azure Virtual Network (VNet) is the fundamental building block for your private network in Azure. VNet enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks. VNet is similar to a traditional network that you'd operate in your own data center, but brings with it additional benefits of Azure's infrastructure such as scale, availability, and isolation. [Learn more about virtual network](#)

Project details

Subscription *	documentationteam
Resource group *	(New) docs-ml-rg Create new

Instance details

Name *	docs-ml-vnet
Region *	(US) East US 2

[Review + create](#)

< Previous

Next : IP Addresses >

[Download a template for automation](#)

3. Select **IP Addresses** tab. The default settings should be similar to the following image:

Create virtual network ...

Basics IP Addresses Security Tags Review + create

The virtual network's address space, specified as one or more address prefixes in CIDR notation (e.g. 192.168.1.0/24).

IPv4 address space

172.16.0.0/16 172.16.0.0 - 172.16.255.255 (65536 addresses)



Add IPv6 address space

The subnet's address range in CIDR notation (e.g. 192.168.1.0/24). It must be contained by the address space of the virtual network.

[+ Add subnet](#) [Remove subnet](#)

<input type="checkbox"/> Subnet name	Subnet address range	NAT gateway
<input type="checkbox"/> default	172.16.0.0/24	-

i Use of a NAT gateway is recommended for outbound internet access from a subnet. You can deploy a NAT gateway and assign it to a subnet after you create the virtual network. [Learn more](#)

[Review + create](#)

< Previous

Next : Security >

[Download a template for automation](#)

Use the following steps to configure the IP address and configure a subnet for training and scoring resources:

TIP

While you can use a single subnet for all Azure ML resources, the steps in this article show how to create two subnets to separate the training & scoring resources.

The workspace and other dependency services will go into the training subnet. They can still be used by resources in other subnets, such as the scoring subnet.

- a. Look at the default **IPv4 address space** value. In the screenshot, the value is **172.16.0.0/16**. **The value may be different for you**. While you can use a different value, the rest of the steps in this tutorial are based on the **172.16.0.0/16** value.

IMPORTANT

We do not recommend using the 172.17.0.0/16 IP address range for your VNet. This is the default subnet range used by the Docker bridge network. Other ranges may also conflict depending on what you want to connect to the virtual network. For example, if you plan to connect your on-premises network to the VNet, and your on-premises network also uses the 172.16.0.0/16 range. Ultimately, it is up to **you** to plan your network infrastructure.

- b. Select the **Default** subnet and then select **Remove subnet**.

A screenshot of the Azure portal showing a list of subnets. At the top, there are buttons for '+ Add subnet' and 'Remove subnet'. Below is a table with two columns: 'Subnet name' and 'Subnet address range'. The first row shows 'default' with '172.16.0.0/24'. The 'Remove subnet' button is highlighted with a red box.

Subnet name	Subnet address range	NAT gateway
default	172.16.0.0/24	-

- c. To create a subnet to contain the workspace, dependency services, and resources used for training, select **+ Add subnet** and set the subnet name and address range. The following are the values used in this tutorial:

- **Subnet name:** Training
- **Subnet address range:** 172.16.0.0/24

A screenshot of the 'Create virtual network' blade in the Azure portal. The 'IP Addresses' tab is selected. A new subnet named 'Training' is being added with the address range '172.16.0.0/24'. The 'Add subnet' button is highlighted with a red box. The 'Review + create' button is at the bottom left, and the 'Add' button is highlighted with a red box at the bottom right.

TIP

If you plan on using a *service endpoint* to add your Azure Storage Account, Azure Key Vault, and Azure Container Registry to the VNet, select the following under **Services**:

- Microsoft.Storage
- Microsoft.KeyVault
- Microsoft.ContainerRegistry

If you plan on using a *private endpoint* to add these services to the VNet, you do not need to select these entries. The steps in this article use a private endpoint for these services, so you do not need to select them when following these steps.

- d. To create a subnet for compute resources used to score your models, select **+ Add subnet** again, and set the name and address range:

- **Subnet name:** Scoring
- **Subnet address range:** 172.16.1.0/24

The screenshot shows the 'Add subnet' dialog in the Azure portal. At the top, it says 'Add subnet'. Below that, there are fields for 'Subnet name *' (set to 'Scoring') and 'Subnet address range *' (set to '172.16.1.0/24'). A note below says '172.16.1.0 - 172.16.1.255 (251 + 5 Azure reserved addresses)'. Under 'NAT GATEWAY', it says 'Simplify connectivity to the internet using a network address translation gateway. Outbound connectivity is possible without a load balancer or public IP addresses attached to your virtual machines.' A note below says 'Use of a NAT gateway is recommended for outbound internet access from a subnet. You can deploy a NAT gateway and assign it to a subnet after you create the virtual network.' A 'NAT gateway' dropdown is set to 'None'. In the 'SERVICE ENDPOINTS' section, it says 'Create service endpoint policies to allow traffic to specific azure resources from your virtual network over service endpoints.' A 'Services' dropdown is set to '0 selected'. At the bottom, there are 'Review + create' and 'Add' buttons.

TIP

If you plan on using a *service endpoint* to add your Azure Storage Account, Azure Key Vault, and Azure Container Registry to the VNet, select the following under **Services**:

- Microsoft.Storage
- Microsoft.KeyVault
- Microsoft.ContainerRegistry

If you plan on using a *private endpoint* to add these services to the VNet, you do not need to select these entries. The steps in this article use a private endpoint for these services, so you do not need to select them when following these steps.

4. Select **Security**. For **BastionHost**, select **Enable**. [Azure Bastion](#) provides a secure way to access the VM jump box you will create inside the VNet in a later step. Use the following values for the remaining fields:

- **Bastion name:** A unique name for this Bastion instance

- **AzureBastionSubnetAddress space:** 172.16.2.0/27
- **Public IP address:** Create a new public IP address.

Leave the other fields at the default values.

Home > Create a resource > Virtual Network >

Create virtual network ...

Basics IP Addresses **Security** Tags Review + create

BastionHost i Disable Enable

Bastion name * docs-ml-bastion ✓

AzureBastionSubnet address space * 172.16.2.0/27 ✓
172.16.2.0 - 172.16.2.31 (32 addresses)

Public IP address * (New) docs-ml-public-ip ▼
[Create new](#)

DDoS Protection Standard i Disable Enable

Firewall i Disable Enable

Review + create < Previous Next : Tags > Download a template for automation

5. Select **Review + create**.

Create virtual network

Basics IP Addresses Security Tags Review + create

The virtual network's address space, specified as one or more address prefixes in CIDR notation (e.g. 192.168.1.0/24).

IPv4 address space

172.16.0.0/16 172.16.0.0 - 172.16.255.255 (65536 addresses)



Add IPv6 address space ⓘ

The subnet's address range in CIDR notation (e.g. 192.168.1.0/24). It must be contained by the address space of the virtual network.

+ Add subnet Remove subnet

<input type="checkbox"/> Subnet name	Subnet address range	NAT gateway
<input type="checkbox"/> Training	172.16.0.0/24	-
<input type="checkbox"/> Scoring	172.16.1.0/24	-

Use of a NAT gateway is recommended for outbound internet access from a subnet. You can deploy a NAT gateway and assign it to a subnet after you create the virtual network. [Learn more](#)

Review + create

< Previous

Next : Security >

Download a template for automation

6. Verify that the information is correct, and then select **Create**.

Create virtual network

Validation passed

Basics IP Addresses Security Tags **Review + create**

Basics

Subscription documentationteam
Resource group (new) docs-ml-rg
Name docs-ml-vnet
Region East US 2

IP addresses

Address space 172.16.0.0/16
Subnet Training (172.16.0.0/24), Scoring (172.16.1.0/24)

Tags

None

Security

BastionHost Enabled
DDoS protection plan Basic
Firewall Disabled

Create

< Previous

Next >

Download a template for automation

Create a storage account

1. In the [Azure portal](#), select the portal menu in the upper left corner. From the menu, select **+ Create a resource** and then enter **Storage account**. Select the **Storage Account** entry, and then select **Create**.
2. From the **Basics** tab, select the **subscription**, **resource group**, and **region** you previously used for the virtual network. Enter a unique **Storage account name**, and set **Redundancy** to **Locally-redundant storage (LRS)**.

Create a storage account

Basics Advanced Networking Data protection Tags Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more about Azure storage accounts](#)

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *	documentationteam
Resource group *	docs-ml-rg
	Create new

Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name ⓘ *	docsmlstore
Region ⓘ *	(US) East US 2
Performance ⓘ *	<input checked="" type="radio"/> Standard: Recommended for most scenarios (general-purpose v2 account) <input type="radio"/> Premium: Recommended for scenarios that require low latency.
Redundancy ⓘ *	Locally-redundant storage (LRS)

[Review + create](#)

< Previous

Next : Advanced >

3. From the **Networking** tab, select **Private endpoint** and then select **+ Add private endpoint**.

Create a storage account ...

Basics Advanced **Networking** Data protection Tags Review + create

Network connectivity
You can connect to your storage account either publicly, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method * Public endpoint (all networks) Public endpoint (selected networks) Private endpoint

Private endpoint
Create a private endpoint to allow a private connection to this resource. Additional private endpoint connections can be created within the storage account or private link center.

Add private endpoint

Name	Subscription	Resource g...	Region	Target sub...	Subnet	Private DN...	Private DN...
Click on add to create a private endpoint							

Network routing
Determine how to route your traffic as it travels from the source to its Azure endpoint. Microsoft network routing is recommended for most customers.

Routing preference ⓘ * Microsoft network routing Internet routing

[Review + create](#) [< Previous](#) [Next : Data protection >](#)

4. On the **Create private endpoint** form, use the following values:

- **Subscription:** The same Azure subscription that contains the previous resources you've created.
- **Resource group:** The same Azure resource group that contains the previous resources you've created.
- **Location:** The same Azure region that contains the previous resources you've created.
- **Name:** A unique name for this private endpoint.
- **Target sub-resource:** blob
- **Virtual network:** The virtual network you created earlier.
- **Subnet:** Training (172.16.0.0/24)
- **Private DNS integration:** Yes
- **Private DNS Zone:** privatelink.blob.core.windows.net

Select OK to create the private endpoint.

5. Select **Review + create**. Verify that the information is correct, and then select **Create**.

6. Once the Storage Account has been created, select **Go to resource**:

✓ Your deployment is complete

Deployment name: docsmlstore_1620308306892
Subscription: documentationteam
Resource group: docs-ml-rg

⌄ Deployment details [\(Download\)](#)

⌂ Next steps

[Go to resource](#)

- From the left navigation, select **Networking** the **Private endpoint connections** tab, and then select **+ Private endpoint**:

NOTE

While you created a private endpoint for Blob storage in the previous steps, you must also create one for File storage.

The screenshot shows the Azure Storage account 'docsmlstore' under the 'Networking' blade. The 'Private endpoint connections' tab is selected, indicated by a red box. On the left, the 'Networking' section is highlighted with a red box. The main area displays a table with one row:

Connection name	Connection state	Private endpoint	Description
docsmlstore.8fce271c-26b3-4...	Approved	docs-ml-store-blob-pe	Auto-Approved

- On the **Create a private endpoint** form, use the same **subscription**, **resource group**, and **Region** that you have used for previous resources. Enter a unique **Name**.

Create a private endpoint ...

1 Basics 2 Resource 3 Configuration 4 Tags 5 Review + create

Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to. [Learn more](#)

Project details

Subscription * ⓘ documentationteam

Resource group * ⓘ docs-ml-rg
[Create new](#)

Instance details

Name * docs-ml-store-file-pe

Region * (US) East US 2

< Previous **Next : Resource >**

9. Select **Next : Resource**, and then set **Target sub-resource** to **file**.

Create a private endpoint ...

✓ Basics 2 **Resource** 3 Configuration 4 Tags 5 Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Subscription documentationteam

Resource type Microsoft.Storage/storageAccounts

Resource docsmlstore

Target sub-resource * ⓘ file

< Previous **Next : Configuration >**

10. Select **Next : Configuration**, and then use the following values:

- **Virtual network:** The network you created previously
- **Subnet:** Training
- **Integrate with private DNS zone:** Yes
- **Private DNS zone:** privatelink.file.core.windows.net

Create a private endpoint ...

✓ Basics ✓ Resource **3 Configuration** Tags Review + create

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network * Subnet *

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone Yes No

Configuration name	Subscription	Private DNS zone
privatelink-file-core-w...	documentationteam	(New) privatelink.file.core.windows.net

Review + create < Previous Next : Tags >

11. Select **Review + Create**. Verify that the information is correct, and then select **Create**.

TIP

If you plan to use **ParallelRunStep** in your pipeline, it is also required to configure private endpoints target **queue** and **table** sub-resources. ParallelRunStep uses queue and table under the hood for task scheduling and dispatching.

Create a key vault

1. In the [Azure portal](#), select the portal menu in the upper left corner. From the menu, select **+ Create a resource** and then enter **Key Vault**. Select the **Key Vault** entry, and then select **Create**.
2. From the **Basics** tab, select the **subscription**, **resource group**, and **region** you previously used for the virtual network. Enter a unique **Key vault name**. Leave the other fields at the default value.

Create key vault ...

Basics Access policy Networking Tags Review + create

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	documentationteam
Resource group *	docs-ml-rg
	Create new

Instance details

Key vault name * ⓘ	docs-ml-kv
Region *	East US 2
Pricing tier * ⓘ	Standard

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or by Microsoft.

Soft-delete ⓘ	Enabled
Days to retain deleted vaults * ⓘ	90
Purge protection ⓘ	<input checked="" type="radio"/> Disable purge protection (allow key vault and objects to be purged during retention period) <input type="radio"/> Enable purge protection (enforce a mandatory retention period for deleted vaults and vault objects)

[Review + create](#)

< Previous

Next : Access policy >

- From the **Networking** tab, select **Private endpoint** and then select **+ Add**.

Create key vault ...

Basics Access policy Networking Tags Review + create

Network connectivity
You can connect to this key vault either publicly, via public IP addresses or service endpoints, or privately, using a private endpoint.

Connectivity method

Public endpoint (all networks)
 Public endpoint (selected networks)
 Private endpoint

Private endpoint
Create a private endpoint to allow a private connection to this resource. Additional private endpoint connections can be created within the key vault or private link center.

+ Add

Name	Subscription	Resource group	Region	Subnet	Private DNS Zone
Click on add button to add private endpoint					

Review + create < Previous Next : Tags >

4. On the **Create private endpoint** form, use the following values:

- **Subscription:** The same Azure subscription that contains the previous resources you've created.
- **Resource group:** The same Azure resource group that contains the previous resources you've created.
- **Location:** The same Azure region that contains the previous resources you've created.
- **Name:** A unique name for this private endpoint.
- **Target sub-resource:** Vault
- **Virtual network:** The virtual network you created earlier.
- **Subnet:** Training (172.16.0.0/24)
- **Private DNS integration:** Yes
- **Private DNS Zone:** privatelink.vaultcore.azure.net

Select OK to create the private endpoint.

Create private endpoint

X

Subscription * ⓘ documentationteam

Resource group * ⓘ docs-ml-rg
Create new

Location * (US) East US 2

Name * ⓘ docs-ml-kv-pe

Target sub-resource * Vault

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more about private endpoint networking](#)

Virtual network * ⓘ docs-ml-vnet

Subnet * ⓘ Training (172.16.0.0/24)

ⓘ If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more about private DNS integration](#)

Integrate with private DNS zone ⓘ Yes No

Private DNS Zone * ⓘ (New) privatelink.vaultcore.azure.net

OK Discard

5. Select **Review + create**. Verify that the information is correct, and then select **Create**.

Create a container registry

1. In the [Azure portal](#), select the portal menu in the upper left corner. From the menu, select **+ Create a resource** and then enter **Container Registry**. Select the **Container Registry** entry, and then select **Create**.
2. From the **Basics** tab, select the **subscription**, **resource group**, and **location** you previously used for the virtual network. Enter a unique **Registry name** and set the **SKU** to **Premium**.

Create container registry

Basics Networking Encryption Tags Review + create

Azure Container Registry allows you to build, store, and manage container images and artifacts in a private registry for all types of container deployments. Use Azure container registries with your existing container development and deployment pipelines. Use Azure Container Registry Tasks to build container images in Azure on-demand, or automate builds triggered by source code updates, updates to a container's base image, or timers. [Learn more](#)

Project details

Subscription *

documentationteam

Resource group *

docs-ml-rg

[Create new](#)

Instance details

Registry name *

docsmlacr

.azurecr.io

Location *

East US 2

Availability zones (i)

Enabled

⚠ During preview, availability zone enablement can not be changed.

SKU * (i)

Premium

[Review + create](#)

[< Previous](#)

[Next: Networking >](#)

3. From the **Networking** tab, select **Private endpoint** and then select **+ Add**.

Create container registry

Basics **Networking** Encryption Tags Review + create

Network connectivity

You can connect to this registry either publicly, via public IP addresses, or privately, using a private endpoint. [Learn more](#)

Connectivity method

Public endpoint (all networks)

Private endpoint

Private endpoint

Click on add to create a private endpoint

[+ Add](#)

[Review + create](#)

[< Previous](#)

[Next: Encryption >](#)

4. On the **Create private endpoint** form, use the following values:

- Subscription:** The same Azure subscription that contains the previous resources you've created.
- Resource group:** The same Azure resource group that contains the previous resources you've created.

- **Location:** The same Azure region that contains the previous resources you've created.
- **Name:** A unique name for this private endpoint.
- **Target sub-resource:** registry
- **Virtual network:** The virtual network you created earlier.
- **Subnet:** Training (172.16.0.0/24)
- **Private DNS integration:** Yes
- **Private DNS Zone:** privatelink.azurecr.io

Select **OK** to create the private endpoint.

Create private endpoint

Subscription * ⓘ documentationteam

Resource group * ⓘ docs-ml-rg [Create new](#)

Location * (US) East US 2

Name * ⓘ docs-ml-acr-pe

registry sub-resource * ⓘ registry

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more about private endpoint networking](#)

Virtual network * ⓘ docs-ml-vnet

Subnet * ⓘ Training (172.16.0.0/24)

If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more about private DNS integration](#)

Integrate with private DNS zone ⓘ Yes No

Private DNS Zone * ⓘ (New) privatelink.azurecr.io

OK **Discard**

5. Select **Review + create**. Verify that the information is correct, and then select **Create**.

6. After the container registry has been created, select **Go to resource**.

The screenshot shows the Microsoft Container Registry Overview page. At the top, there's a search bar and several action buttons: Delete, Cancel, Redeploy, and Refresh. Below the search bar is a navigation menu with links to Overview, Inputs, Outputs, and Template. A feedback message says "We'd love your feedback! →". The main content area displays a green checkmark icon and the message "Your deployment is complete". It provides deployment details: Deployment name: Microsoft.ContainerRegistry, Subscription: documentationteam, Resource group: docs-ml-rg. There are two expandable sections: "Deployment details (Download)" and "Next steps". The "Next steps" section contains a prominent blue button with white text that says "Go to resource", which is highlighted with a red border.

7. From the left of the page, select **Access keys**, and then enable **Admin user**. This setting is required when using Azure Container Registry inside a virtual network with Azure Machine Learning.

The screenshot shows the "Access keys" page for a container registry named "docsmlacr". On the left, there's a sidebar with links to Overview, Activity log, Access control (IAM), Tags, Quick start, Events, and Settings. The "Access keys" link is highlighted with a red border. The main form has fields for Registry name (docsmlacr), Login server (docsmlacr.azurecr.io), Admin user (with a toggle switch set to "Enabled" and a red border around it), Username (docsmlacr), Name, Password, and two password confirmation fields. The "Regenerate" button is visible next to the password fields. The "password" field contains the value "2B+gdwg+NpQjLhoU7eEcq56C1FDdbJnN/" and the "password2" field contains "Vk7Xb4I+y0lgUEEaSSel63t0ulBV7mWQ".

Create a workspace

1. In the [Azure portal](#), select the portal menu in the upper left corner. From the menu, select **+ Create a resource** and then enter **Machine Learning**. Select the **Machine Learning** entry, and then select **Create**.

The screenshot shows the Azure Marketplace search results for "Machine Learning". The top result is a card for "Machine Learning" by Microsoft, which has a 4.3 rating from 112 reviews. The card includes a blue 3D pyramid icon, a "Create" button at the bottom, and a note that it is "Azure benefit eligible". The "Create" button is highlighted with a red border.

2. From the **Basics** tab, select the **subscription**, **resource group**, and **Region** you previously used for the virtual network. Use the following values for the other fields:
 - **Workspace name:** A unique name for your workspace.
 - **Storage account:** Select the storage account you created previously.

- **Key vault:** Select the key vault you created previously.
- **Application insights:** Use the default value.
- **Container registry:** Use the container registry you created previously.

Machine learning ...

Create a machine learning workspace

Basics Networking Advanced Tags Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

documentationteam ▼

Resource group * ⓘ

docs-ml-rg ▼

[Create new](#)

Workspace details

Specify the name and region for the workspace.

Workspace name * ⓘ

docs-ml-ws ✓

Region * ⓘ

East US 2 ▼

Storage account * ⓘ

docsmlstore ▼

[Create new](#)

Key vault * ⓘ

docs-ml-kv ▼

[Create new](#)

Application insights * ⓘ

(new) docsmlws6199458463 ▼

[Create new](#)

Container registry * ⓘ

docsmlacr ▼

[Create new](#)

[Review + create](#)

[< Previous](#)

[Next : Networking](#)

3. From the **Networking** tab, select **Private endpoint** and then select **+ add**.

Machine learning ...

Create a machine learning workspace

Basics Networking Advanced Tags Review + create

Network connectivity

You can connect to your workspace either publicly or privately using a private endpoint.

Connectivity method *

Public endpoint (all networks)

Private endpoint

Private endpoint

Create a private endpoint to allow a private connection to this resource.

Name	Subscription	Resource group	Region	Subnet	Private DNS Zone
------	--------------	----------------	--------	--------	------------------

Click on add to create a private endpoint

[+ Add](#)

[Review + create](#)

[< Previous](#)

[Next : Advanced](#)

4. On the **Create private endpoint** form, use the following values:

- **Subscription:** The same Azure subscription that contains the previous resources you've created.
- **Resource group:** The same Azure resource group that contains the previous resources you've created.
- **Location:** The same Azure region that contains the previous resources you've created.
- **Name:** A unique name for this private endpoint.
- **Target sub-resource:** amlworkspace
- **Virtual network:** The virtual network you created earlier.
- **Subnet:** Training (172.16.0.0/24)
- **Private DNS integration:** Yes
- **Private DNS Zone:** Leave the two private DNS zones at the default values of **privatelink.api.azureml.ms** and **privatelink.notebooks.azure.net**.

Select **OK** to create the private endpoint.

Create private endpoint X

Subscription * ⓘ documentationteam

Resource group * ⓘ docs-ml-rg
[Create new](#)

Location * (US) East US 2

Name * ⓘ docs-ml-ws-pe ✓

Workspace sub-resource ⓘ amlworkspace

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more about private endpoint networking](#)

Virtual network * ⓘ docs-ml-vnet

Subnet * ⓘ Training (172.16.0.0/24)

ⓘ If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more about private DNS integration](#)

Integrate with private DNS zone ⓘ Yes No

Configuration name	Subscription	Private DNS zone
privatelink-api-azureml-ms	documentationteam	(New) privatelink.api.azureml.ms
privatelink-notebooks-az...	documentationteam	(New) privatelink.notebooks.azure.net

OK Discard

5. Select **Review + create**. Verify that the information is correct, and then select **Create**.

6. Once the workspace has been created, select **Go to resource**.

7. From the **Settings** section on the left, select **Private endpoint connections** and then select the link in the **Private endpoint** column:

docs-ml-ws | Private endpoint connections

Machine learning

Search (Ctrl+ /) Private endpoint Approve Reject Remove Refresh

Settings

- Private endpoint connections (highlighted)
- Properties
- Locks

Filter by name... All connection states

Connection name	Connection state	Private endpoint	Description
docs-ml-ws.6f27c06a-abdf-44e9-a...	Approved	docs-ml-ws-pe (highlighted)	Auto-Approved

- Once the private endpoint information appears, select **DNS configuration** from the left of the page. Save the IP address and fully qualified domain name (FQDN) information on this page, as it will be used later.

dns docs-ml-ws-pe | DNS configuration

Private endpoint

Search (Ctrl+ /) Add configuration Refresh

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint using a private DNS zone. You can also utilize your own DNS servers. [Learn more](#)

Customer Visible FQDNs

DNS records visible to the customer

Network Interface	IP addresses	FQDN
docs-ml-ws-pe.nic.30721b95-c716-4...	172.16.0.9	62ba24ca-e096-40dd-96ac-f25405d82b1b.workspace.eastus2.api.azureml.ms 62ba24ca-e096-40dd-96ac-f25405d82b1b.workspace.eastus2.cert.api.azureml.ms
	172.16.0.10	ml-docs-ml-ws-eastus2-62ba24ca-e096-40dd-96ac-f25405d82b1b.notebooks.azure.net

IMPORTANT

There are still some configuration steps needed before you can fully use the workspace. However, these require you to connect to the workspace.

Enable studio

Azure Machine Learning studio is a web-based application that lets you easily manage your workspace. However, it needs some extra configuration before it can be used with resources secured inside a VNet. Use the following steps to enable studio:

- When using an Azure Storage Account that has a private endpoint, add the service principal for the workspace as a **Reader** for the storage private endpoint(s). From the Azure portal, select your storage account and then select **Networking**. Next, select **Private endpoint connections**.

Home > Microsoft.PrivateEndpoint-20210611095057 > docs-ml-rg > docsmlstore

docsmlstore | Networking

Storage account

File shares Queues Tables Security + networking Networking (highlighted)

Firewalls and virtual networks Private endpoint connections Custom domain

Search (Ctrl+ /) Private endpoint Approve Reject Remove Refresh

Filter by name... All connection states

Connection name	Connection state	Private endpoint	Description
docsmlstore.f924e769-eb5...	Approved	docs-ml-store-blob-pe (highlighted)	Auto-Approved
docsmlstore.d47b6994-6c1...	Approved	docs-ml-store-file-pe	Auto-Approved

- For each private endpoint listed, use the following steps:

- Select the link in the **Private endpoint** column.

Home > Microsoft.PrivateEndpoint-20210611095057 > docs-ml-rg > docsmilstore

docsmlstore | Networking Storage account

Firewalls and virtual networks Private endpoint connections Custom domain

+ Private endpoint ✓ Approve ✗ Reject 🗑 Remove ⏪ Refresh

Filter by name... All connection states

Connection name	Connection state	Private endpoint	Description
docsmlstore.f924e769-eb5...	Approved	docs-ml-store-blob-pe	Auto-Approved
docsmlstore.d47b6994-6c1...	Approved	docs-ml-store-file-pe	Auto-Approved

- Select Access control (IAM) from the left side.
- Select + Add, and then Add role assignment (Preview).

Home >

Access control (IAM)

Search (Ctrl+ /) + Add Download role assignments Edit columns

Overview Activity log Access control (IAM) Tags Resource visualizer

Add role assignment

Add co-administrator Roles Deny assignments

Add custom role

View my level of access to this resource.

View my access

- On the Role tab, select the Reader.

Home >

Add role assignment

Role Members Review + assign

A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. [Learn more](#)

Search by role name or description Type : All Category : All

Name ↑	Description ↑	Type ↑	Category ↑	Details
Owner	Grants full access to manage all resources, including the ability to a...	BuiltinRole	General	View
Contributor	Grants full access to manage all resources, but does not allow you to ...	BuiltinRole	General	View
Reader	View all resources, but does not allow you to make any changes.	BuiltinRole	General	View
AcrDelete	acr delete	BuiltinRole	Containers	View
AcrImageSigner	acr image signer	BuiltinRole	Containers	View
AcrPull	acr pull	BuiltinRole	Containers	View
AcrPush	acr push	BuiltinRole	Containers	View
AcrQuarantineReader	acr quarantine data reader	BuiltinRole	Containers	View
AcrQuarantineWriter	acr quarantine data writer	BuiltinRole	Containers	View

Review + assign Previous Next

- On the Members tab, select User, group, or service principal in the Assign access to area and then select + Select members. In the Select members dialog, enter the name as your Azure Machine Learning workspace. Select the service principal for the workspace, and then use the Select button.
- On the Review + assign tab, select Review + assign to assign the role.

Connect to the workspace

There are several ways that you can connect to the secured workspace. The steps in this article use a **jump box**, which is a virtual machine in the VNet. You can connect to it using your web browser and Azure Bastion. The following table lists several other ways that you might connect to the secure workspace:

METHOD	DESCRIPTION
Azure VPN gateway	Connects on-premises networks to the VNet over a private connection. Connection is made over the public internet.
ExpressRoute	Connects on-premises networks into the cloud over a private connection. Connection is made using a connectivity provider.

IMPORTANT

When using a **VPN gateway** or **ExpressRoute**, you will need to plan how name resolution works between your on-premises resources and those in the VNet. For more information, see [Use a custom DNS server](#).

Create a jump box (VM)

Use the following steps to create a Data Science Virtual Machine for use as a jump box:

1. In the [Azure portal](#), select the portal menu in the upper left corner. From the menu, select **+ Create a resource** and then enter **Data science virtual machine**. Select the **Data science virtual machine - Windows** entry, and then select **Create**.
2. From the **Basics** tab, select the **subscription**, **resource group**, and **Region** you previously used for the virtual network. Provide values for the following fields:
 - **Virtual machine name**: A unique name for the VM.
 - **Username**: The username you will use to login to the VM.
 - **Password**: The password for the username.
 - **Security type**: Standard.
 - **Image**: Data Science Virtual Machine - Windows Server 2019 - Gen1.

IMPORTANT

Do not select a Gen2 image.

You can leave other fields at the default values.

Create a virtual machine

X

⚠️ Changing Basic options may reset selections you have made. Review all options prior to creating the virtual machine.

Basics Disks Networking Management Advanced Tags Review + create

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. [Learn more ↗](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

ML-docs

Resource group * ⓘ

docs-ml-rg

[Create new](#)

Instance details

Virtual machine name * ⓘ

docs-ml-jump

Region * ⓘ

(US) East US 2

Availability options ⓘ

Availability zone

Availability zone * ⓘ

1

Image * ⓘ

Data Science Virtual Machine - Windows Server 2019 - Gen1

[See all images](#)

Size * ⓘ

Standard_DS1_v2 - 1 vcpu, 3.5 GiB memory

[See all sizes](#)

Administrator account

Username * ⓘ

myusername

Password * ⓘ

.....

Confirm password * ⓘ

.....

Licensing

Save up to 49% with a license you already own using Azure Hybrid Benefit. [Learn more ↗](#)

Would you like to use an existing
Windows Server license? * ⓘ

[Review Azure hybrid benefit compliance](#)

[Review + create](#)

< Previous

Next : Disks >

3. Select **Networking**, and then select the **Virtual network** you created earlier. Use the following information to set the remaining fields:

- Select the **Training** subnet.
- Set the **Public IP** to **None**.
- Leave the other fields at the default value.

Create a virtual machine

X

[Basics](#) [Disks](#) [Networking](#) [Management](#) [Advanced](#) [Tags](#) [Review + create](#)

Define network connectivity for your virtual machine by configuring network interface card (NIC) settings. You can control ports, inbound and outbound connectivity with security group rules, or place behind an existing load balancing solution.

[Learn more ↗](#)

Network interface

When creating a virtual machine, a network interface will be created for you.

Virtual network * ⓘ	<input type="text" value="docs-ml-vnet"/> ▼ Create new
Subnet * ⓘ	<input type="text" value="Training (172.16.0.0/24)"/> ▼ Manage subnet configuration
Public IP ⓘ	<input type="text" value="None"/> ▼ Create new
NIC network security group ⓘ	<input type="radio"/> None <input type="radio"/> Basic <input checked="" type="radio"/> Advanced
<i>ⓘ This VM image has preconfigured NSG rules</i>	
Configure network security group *	<input type="text" value="(new) docs-ml-jump-nsq"/> ▼ Create new
Accelerated networking ⓘ	<input type="checkbox"/> <small>The selected image does not support accelerated networking.</small>

Load balancing

You can place this virtual machine in the backend pool of an existing Azure load balancing solution. [Learn more ↗](#)

Place this virtual machine behind an existing load balancing solution?

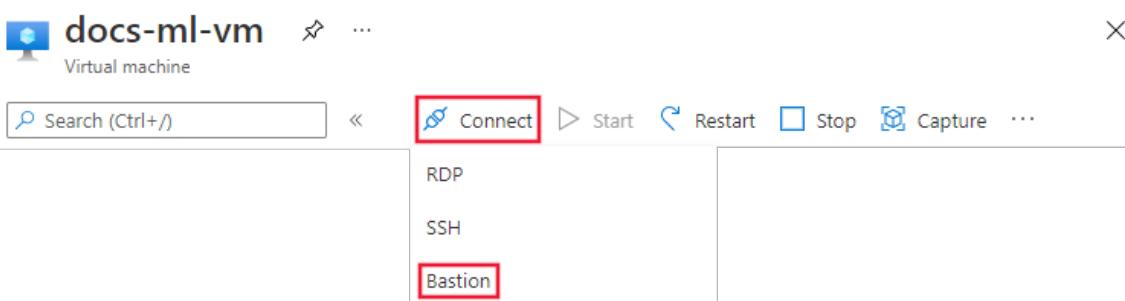
[Review + create](#)[< Previous](#)[Next : Management >](#)

4. Select **Review + create**. Verify that the information is correct, and then select **Create**.

Connect to the jump box

1. Once the virtual machine has been created, select **Go to resource**.

2. From the top of the page, select **Connect** and then **Bastion**.



3. Select **Use Bastion**, and then provide your authentication information for the virtual machine, and a connection will be established in your browser.

The screenshot shows a navigation bar with three items: RDP, SSH, and BASTION (which is underlined). Below the navigation bar is a message: "Bastion is an Azure service that allows fast, secure connections to any VM within a VNet. [Learn more](#)". At the bottom of the interface is a blue button labeled "Use Bastion", which is also highlighted with a red box.

Create a compute cluster and compute instance

A compute cluster is used by your training jobs. A compute instance provides a Jupyter Notebook experience on a shared compute resource attached to your workspace.

1. From an Azure Bastion connection to the jump box, open the **Microsoft Edge** browser on the remote desktop.
2. In the remote browser session, go to <https://ml.azure.com>. When prompted, authenticate using your Azure AD account.
3. From the **Welcome to studio!** screen, select the **Machine Learning workspace** you created earlier and then select **Get started**.

TIP

If your Azure AD account has access to multiple subscriptions or directories, use the **Directory and Subscription** dropdown to select the one that contains the workspace.

Welcome to the studio!

Select a subscription and a workspace to get started or go to the [Azure Portal](#) to create your subscription and workspace. You can switch subscriptions and workspaces at any time. [Learn more](#).

The screenshot shows the "Welcome to the studio!" screen. It includes a "Directory and Subscription" dropdown set to "Microsoft, ML-docs". Below it is a "Machine Learning workspace" dropdown containing "docs-ml-ws", which is also highlighted with a red box. There is a "Create a new workspace" link next to it. At the bottom is a large blue "Get started" button, which is also highlighted with a red box.

4. From studio, select **Compute**, **Compute clusters**, and then **+ New**.

Microsoft > mymlworkspace > Compute

Compute

Compute instances Compute clusters Inference clusters Attached computes

Scale your compute cluster from a single node to a multi node workload

Create a single or multi node compute cluster for your training, batch inferencing or reinforcement learning workloads. [Learn more](#)

[+ New](#)

[View Azure Machine Learning tutorials](#)

5. From the **Virtual Machine** dialog, select **Next** to accept the default virtual machine configuration.

Create compute cluster ⓘ

Virtual Machine Advanced Settings

Select virtual machine
Select the virtual machine size you would like to use for your compute cluster.

Location: southcentralus

Virtual machine tier ⓘ
 Dedicated Low priority

Virtual machine type ⓘ
 CPU GPU

Virtual machine size ⓘ
 Select from recommended options Select from all options

Name ↑	Category	Workload types	Available quota ⓘ
<input type="radio"/> Standard_DS11_v2 2 cores, 14GB RAM, 28GB storage	Memory optimized	Development on Notebooks (or other IDE) and light weight testing	96 cores
<input checked="" type="radio"/> Standard_DS2_v2 4 cores, 14GB RAM, 28GB storage	General purpose	Classical ML model training on small datasets	96 cores
<input type="radio"/> Standard_DS2_v2 4 cores, 28GB RAM, 56GB storage	Memory optimized	Data manipulation and training on medium-sized datasets (1-10GB)	96 cores
<input type="radio"/> Standard_D13_v2 8 cores, 56GB RAM, 400GB storage	Memory optimized	Data manipulation and training on large datasets (> 10 GB)	96 cores

[Back](#) Next [Cancel](#)

6. From the **Configure Settings** dialog, enter **cpu-cluster** as the **Compute name**. Set the **Subnet** to **Training** and then select **Create** to create the cluster.

TIP

Compute clusters dynamically scale the nodes in the cluster as needed. We recommend leaving the minimum number of nodes at 0 to reduce costs when the cluster is not in use.

Create compute cluster X

<input checked="" type="checkbox"/> Virtual Machine <input checked="" type="checkbox"/> Advanced Settings	Configure Settings Configure compute cluster settings for your selected virtual machine size. <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 10px;"> <thead> <tr> <th>Name</th> <th>Category</th> <th>Cores</th> <th>Available quota</th> <th>RAM</th> <th>Storage</th> <th>Cost/Node</th> </tr> </thead> <tbody> <tr> <td>Standard_DS3_v2</td> <td>General purpose</td> <td>4</td> <td>96 cores</td> <td>14 GB</td> <td>28 GB</td> <td>\$0.25/hr</td> </tr> </tbody> </table> Compute name * (1) <input type="text" value="my-compute"/> Minimum number of nodes * (1) <input type="text" value="0"/> Maximum number of nodes * (1) <input type="text" value="1"/> Idle seconds before scale down * (1) <input type="text" value="120"/> <input checked="" type="checkbox"/> Enable SSH access (1) Advanced settings <input checked="" type="checkbox"/> Enable virtual network (1) <div style="background-color: #ffffcc; padding: 5px; margin-top: 5px;"> (1) Your workspace is linked to a virtual network using a private endpoint connection. In order to communicate properly with the workspace, your compute resource must be provisioned in the same virtual network. </div> Virtual network * (1) <input type="text" value="ml-vnet-sentral (ml-vnet-scentral-rg)"/> Refresh virtual networks Subnet * (1) <input type="text" value="training"/> <input type="checkbox"/> No public IP (preview) (1) <input checked="" type="checkbox"/> Assign a managed identity (1)	Name	Category	Cores	Available quota	RAM	Storage	Cost/Node	Standard_DS3_v2	General purpose	4	96 cores	14 GB	28 GB	\$0.25/hr
Name	Category	Cores	Available quota	RAM	Storage	Cost/Node									
Standard_DS3_v2	General purpose	4	96 cores	14 GB	28 GB	\$0.25/hr									
Back	<input style="background-color: #0078D4; color: white; border: none; padding: 5px; width: 100px; height: 30px; font-weight: bold; border-radius: 5px;" type="button" value="Create"/>	Download a template for automation													

7. From studio, select **Compute**, **Compute instance**, and then + New.

Microsoft Azure Machine Learning

Home > Compute

Compute

Compute instances Compute clusters Inference clusters Attached computes



Get started with Azure Machine Learning notebooks and R scripts by creating a compute instance

Choose from a selection of CPU or GPU instances preconfigured with popular tools such as JupyterLab, Jupyter, and RStudio, ML packages, deep learning frameworks, and GPU drivers. [Learn more](#)

+ New

[View Azure Machine Learning tutorials](#)

8. From the **Virtual Machine** dialog, enter a unique **Computer name** and select **Next: Advanced Settings**.

Create compute instance

X

- Required Settings
- Advanced Settings

Configure required settings

Select the name and virtual machine size you would like to use for your compute instance. Please note that a compute instance can not be shared. It can only be used by a single assigned user. By default, it will be assigned to the creator and you can change this to a different user in the advanced settings section.

Compute name * ⓘ

docs-ml-ci



Location ⓘ

southcentralus

Virtual machine type ⓘ

CPU GPU

Virtual machine size ⓘ

Select from recommended options Select from all options

Total available quota: 100 cores ⓘ

Name ↑	Category	Workload types	Av... ⓘ	Cost ⓘ
<input type="radio"/> Standard_DS2_v2 2 cores, 7GB RAM, 14GB storage	General purpose	Development on Notebooks (or other IDE) and light weight testing	100 c...	
<input checked="" type="radio"/> Standard_DS3_v2 4 cores, 14GB RAM, 28GB storage	General purpose	Classical ML model training, AutoML runs, pipeline runs (default compute)	100 c...	
<input type="radio"/> Standard_DS12_v2 4 cores, 28GB RAM, 56GB storage	Memory optimized	Training on large datasets (>1GB) parallel run steps, batch inferencing	100 c...	
<input type="radio"/> Standard_F4s_v2 4 cores, 8GB RAM, 32GB storage	Compute optimized	Real-time inferencing and other latency-sensitive tasks	100 c...	

Create

Back

Next: Advanced Settings

Download a template for automation

Cancel

9. From the **Advanced Settings** dialog, set the **Subnet** to **Training**, and then select **Create**.

Create compute instance

X

- Required Settings
- Advanced Settings

Configure Settings

Configure compute instance settings for your selected virtual machine size.

Name	Category	Cores	Available quota	RAM	Storage	Cost/Hour
Standard_DS12_v2	Memory optimized	4	92 cores	28 GB	56 GB	...

Startup and shutdown schedule ⓘ

Add schedule

Enable SSH access ⓘ

Enable virtual network ⓘ

Virtual network *

myvnet (myresourcegroup)

Refresh virtual networks

Subnet *

training

Assign to another user ⓘ

Provision with setup script ⓘ

Create

Back

Next

Download a template for automation

Cancel

TIP

When you create a compute cluster or compute instance, Azure Machine Learning dynamically adds a Network Security Group (NSG). This NSG contains the following rules, which are specific to compute cluster and compute instance:

- Allow inbound TCP traffic on ports 29876-29877 from the `BatchNodeManagement` service tag.
- Allow inbound TCP traffic on port 44224 from the `AzureMachineLearning` service tag.

The following screenshot shows an example of these rules:

Filter by name						
Priority ↑↓	Name ↑↓	Port ↑↓	Protocol ↑↓	Source ↑↓	Destination ↑↓	Action ↑↓
Inbound Security Rules						
120	BatchServiceRule	29876-29877	tcp	BatchNodeManagement.Sou...	Any	Allow
149	⚠ NodeAgentRule-DenyAll	29876-29877	tcp	Any	Any	Deny
160	JupyterServerPort	44224	tcp	AzureMachineLearning	Any	Allow
170	SSH	22	tcp	Internet	Any	Deny
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBo...	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny
Outbound Security Rules						
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

For more information on creating a compute cluster and compute cluster, including how to do so with Python and the CLI, see the following articles:

- [Create a compute cluster](#)
- [Create a compute instance](#)

Configure image builds

APPLIES TO: ✓ [Azure CLI ml extension v2 \(current\)](#)

When Azure Container Registry is behind the virtual network, Azure Machine Learning can't use it to directly build Docker images (used for training and deployment). Instead, configure the workspace to use the compute cluster you created earlier. Use the following steps to create a compute cluster and configure the workspace to use it to build images:

1. Navigate to <https://shell.azure.com/> to open the Azure Cloud Shell.
2. From the Cloud Shell, use the following command to install the 2.0 CLI for Azure Machine Learning:

```
az extension add -n ml
```

3. To update the workspace to use the compute cluster to build Docker images. Replace `docs-ml-rg` with your resource group. Replace `docs-ml-ws` with your workspace. Replace `cpu-cluster` with the compute cluster to use:

```
az ml workspace update \
-n myworkspace \
-g myresourcegroup \
-i mycomputecluster
```

NOTE

You can use the same compute cluster to train models and build Docker images for the workspace.

Use the workspace

IMPORTANT

The steps in this article put Azure Container Registry behind the VNet. In this configuration, you cannot deploy a model to Azure Container Instances inside the VNet. We do not recommend using Azure Container Instances with Azure Machine Learning in a virtual network. For more information, see [Secure the inference environment](#).

As an alternative to Azure Container Instances, try Azure Machine Learning managed online endpoints. For more information, see [Enable network isolation for managed online endpoints \(preview\)](#).

At this point, you can use studio to interactively work with notebooks on the compute instance and run training jobs on the compute cluster. For a tutorial on using the compute instance and compute cluster, see [run a Python script](#).

Stop compute instance and jump box

WARNING

While it is running (started), the compute instance and jump box will continue charging your subscription. To avoid excess cost, stop them when they are not in use.

The compute cluster dynamically scales between the minimum and maximum node count set when you created it. If you accepted the defaults, the minimum is 0, which effectively turns off the cluster when not in use.

Stop the compute instance

From studio, select **Compute**, **Compute clusters**, and then select the compute instance. Finally, select **Stop** from the top of the page.

Home > Compute

Compute

Compute instances	Compute clusters	Inference clusters	Attached computes
+ New	Start	Stop	Restart
Delete	Refresh	Edit columns	Reset view
<input type="text"/> Search	Show all instances	State	All filters
<input checked="" type="checkbox"/>	Name	<input type="checkbox"/>	Applications
docs-ml-ci	Running	JupyterLab Jupyter VS Code RStudio Term	

Stop the jump box

Once it has been created, select the virtual machine in the Azure portal and then use the **Stop** button. When you are ready to use it again, use the **Start** button to start it.

The screenshot shows the Azure portal interface for a virtual machine named 'docs-ml-vm'. At the top, there's a search bar labeled 'Search (Ctrl+ /)' and several action buttons: 'Connect', 'Start', 'Restart', 'Stop' (which is highlighted with a red box), 'Capture', and more. Below the header, a message says: 'You can also configure the jump box to automatically shut down at a specific time. To do so, select Auto-shutdown, Enable, set a time, and then select Save.' The main content area shows the 'Auto-shutdown' configuration settings.

You can also configure the jump box to automatically shut down at a specific time. To do so, select **Auto-shutdown, Enable**, set a time, and then select **Save**.

This screenshot shows the 'Auto-shutdown' configuration page for the 'docs-ml-vm' virtual machine. On the left, there's a sidebar with links like Configuration, Identity, Properties, Locks, Operations, Bastion, and Auto-shutdown (which is highlighted with a red box). The main area has a 'Save' button (highlighted with a red box), 'Discard', and 'Feedback' buttons. It shows the 'Enabled' status is set to 'On' (radio button highlighted with a red box) and 'Off'. Under 'Scheduled shutdown', the time is set to '5:00:00 PM' (highlighted with a red box). A dropdown for 'Time zone' shows '(UTC-05:00) Eastern Time (US & Canada)'. There's also a section for 'Send notification before auto-shutdown?' with 'Yes' and 'No' options ('Yes' is selected).

Clean up resources

If you plan to continue using the secured workspace and other resources, skip this section.

To delete all resources created in this tutorial, use the following steps:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created in this tutorial.
3. Select **Delete resource group**.

This screenshot shows the 'Resource groups' list page in the Azure portal. At the top, there's a search bar labeled 'Search (Ctrl+ /)' and buttons for 'Create', 'Edit columns', and 'Delete resource group' (which is highlighted with a red box). Below is a table with columns for Name, Type, Status, and Last activity. One row is selected, showing 'docs-ml-rg' as the name, 'Resource group' as the type, and 'Active' as the status.

4. Enter the resource group name, then select **Delete**.

Next steps

Now that you have created a secure workspace and can access studio, learn how to [run a Python script](#) using Azure Machine Learning.

How to create a secure workspace by using template

9/22/2022 • 6 minutes to read • [Edit Online](#)

Templates provide a convenient way to create reproducible service deployments. The template defines what will be created, with some information provided by you when you use the template. For example, specifying a unique name for the Azure Machine Learning workspace.

In this tutorial, you learn how to use a [Microsoft Bicep](#) and [Hashicorp Terraform](#) template to create the following Azure resources:

- Azure Virtual Network. The following resources are secured behind this VNet:
 - Azure Machine Learning workspace
 - Azure Machine Learning compute instance
 - Azure Machine Learning compute cluster
 - Azure Storage Account
 - Azure Key Vault
 - Azure Application Insights
 - Azure Container Registry
 - Azure Bastion host
 - Azure Machine Learning Virtual Machine (Data Science Virtual Machine)
 - The **Bicep** template also creates an Azure Kubernetes Service cluster, and a separate resource group for it.

Prerequisites

Before using the steps in this article, you must have an Azure subscription. If you don't have an Azure subscription, create a [free account](#).

You must also have either a Bash or Azure PowerShell command line.

TIP

When reading this article, use the tabs in each section to select whether to view information on using Bicep or Terraform templates.

- [Bicep](#)
- [Terraform](#)

1. To install the command-line tools, see [Set up Bicep development and deployment environments](#).
2. The Bicep template used in this article is located at <https://github.com/Azure/azure-quickstart-templates/blob/master/quickstarts/microsoft.machinelearningservices/machine-learning-end-to-end-secure>. Use the following commands to clone the GitHub repo to your development environment:

TIP

If you do not have the `git` command on your development environment, you can install it from <https://git-scm.com/>.

```
git clone https://github.com/Azure/azure-quickstart-templates  
cd azure-quickstart-templates/quickstarts/microsoft.machinelearningservices/machine-learning-end-to-end-secure
```

Understanding the template

- [Bicep](#)
- [Terraform](#)

The Bicep template is made up of the `main.bicep` and the `.bicep` files in the `modules` subdirectory. The following table describes what each file is responsible for:

FILE	DESCRIPTION
<code>main.bicep</code>	Parameters and variables. Passing parameters & variables to other modules in the <code>modules</code> subdirectory.
<code>vnet.bicep</code>	Defines the Azure Virtual Network and subnets.
<code>nsg.bicep</code>	Defines the network security group rules for the VNet.
<code>bastion.bicep</code>	Defines the Azure Bastion host and subnet. Azure Bastion allows you to easily access a VM inside the VNet using your web browser.
<code>dsvmjumpbox.bicep</code>	Defines the Data Science Virtual Machine (DSVM). Azure Bastion is used to access this VM through your web browser.
<code>storage.bicep</code>	Defines the Azure Storage account used by the workspace for default storage.
<code>keyvault.bicep</code>	Defines the Azure Key Vault used by the workspace.
<code>containerregistry.bicep</code>	Defines the Azure Container Registry used by the workspace.
<code>applicationinsights.bicep</code>	Defines the Azure Application Insights instance used by the workspace.
<code>machinelearningnetworking.bicep</code>	Defines te private endpoints and DNS zones for the Azure Machine Learning workspace.
<code>Machinelearning.bicep</code>	Defines the Azure Machine Learning workspace.
<code>machinelearningcompute.bicep</code>	Defines an Azure Machine Learning compute cluster and compute instance.
<code>privateaks.bicep</code>	Defines an Azure Kubernetes Services cluster instance.

IMPORTANT

The DSVM and Azure Bastion is used as an easy way to connect to the secured workspace for this tutorial. In a production environment, we recommend using an [Azure VPN gateway](#) or [Azure ExpressRoute](#) to access the resources inside the VNet directly from your on-premises network.

Configure the template

- [Bicep](#)
- [Terraform](#)

To run the Bicep template, use the following commands from the `machine-learning-end-to-end-secure` where the `main.bicep` file is:

1. To create a new Azure Resource Group, use the following command. Replace `exampleRG` with your resource group name, and `eastus` with the Azure region you want to use:

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az group create --name exampleRG --location eastus
```

2. To run the template, use the following command:

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
  --resource-group exampleRG \
  --template-file main.bicep \
  --parameters \
  prefix=myprefix \
  dsvmJumpboxUsername=azureadmin \
  dsvmJumpboxPassword=securepassword
```

Connect to the workspace

After the template completes, use the following steps to connect to the DSVM:

1. From the [Azure portal](#), select the Azure Resource Group you used with the template. Then, select the Data Science Virtual Machine that was created by the template. If you have trouble finding it, use the filters section to filter the **Type** to **virtual machine**.

The screenshot shows the Azure Resource Group Overview page for the 'docs1202' group. The 'Overview' tab is selected. In the center, there's a search bar with 'Type == virtual machine' typed in. Below it, a table lists a single resource: 'vm-docs-irwq'. This resource has a blue icon, the name 'vm-docs-irwq', and a location of 'East US 2'. The 'Resources' tab is also visible.

- From the **Overview** section of the Virtual Machine, select **Connect**, and then select **Bastion** from the dropdown.

The screenshot shows the Azure Virtual Machine details page for 'vm-docs-irwq'. The 'Overview' tab is selected. Under the 'Connect' button, a dropdown menu is open with three options: 'RDP', 'SSH', and 'Bastion'. The 'Bastion' option is highlighted with a red box.

- When prompted, provide the **username** and **password** you specified when configuring the template and then select **Connect**.

IMPORTANT

The first time you connect to the DSVM desktop, a PowerShell window opens and begins running a script. Allow this to complete before continuing with the next step.

- From the DSVM desktop, start **Microsoft Edge** and enter <https://ml.azure.com> as the address. Sign in to your Azure subscription, and then select the workspace created by the template. The studio for your workspace is displayed.

Next steps

IMPORTANT

The Data Science Virtual Machine (DSVM) and any compute instance resources bill you for every hour that they are running. To avoid excess charges, you should stop these resources when they are not in use. For more information, see the following articles:

- [Create/manage VMs \(Linux\)](#).
- [Create/manage VMs \(Windows\)](#).
- [Create/manage compute instance](#).

To continue learning how to use the secured workspace from the DSVM, see [Tutorial: Get started with a Python](#)

script in Azure Machine Learning.

To learn more about common secure workspace configurations and input/output requirements, see [Azure Machine Learning secure workspace traffic flow](#).

Set up authentication for Azure Machine Learning resources and workflows

9/22/2022 • 10 minutes to read • [Edit Online](#)

Learn how to set up authentication to your Azure Machine Learning workspace from the Azure CLI or Azure Machine Learning SDK v2 (preview). Authentication to your Azure Machine Learning workspace is based on **Azure Active Directory** (Azure AD) for most things. In general, there are four authentication workflows that you can use when connecting to the workspace:

- **Interactive:** You use your account in Azure Active Directory to either directly authenticate, or to get a token that is used for authentication. Interactive authentication is used during *experimentation and iterative development*. Interactive authentication enables you to control access to resources (such as a web service) on a per-user basis.
- **Service principal:** You create a service principal account in Azure Active Directory, and use it to authenticate or get a token. A service principal is used when you need an *automated process to authenticate* to the service without requiring user interaction. For example, a continuous integration and deployment script that trains and tests a model every time the training code changes.
- **Azure CLI session:** You use an active Azure CLI session to authenticate. The Azure CLI extension for Machine Learning (the `m1` extension or CLI v2) is a command line tool for working with Azure Machine Learning. You can sign in to Azure via the Azure CLI on your local workstation, without storing credentials in Python code or prompting the user to authenticate. Similarly, you can reuse the same scripts as part of continuous integration and deployment pipelines, while authenticating the Azure CLI with a service principal identity.
- **Managed identity:** When using the Azure Machine Learning SDK v2 *on a compute instance or on an Azure Virtual Machine*, you can use a managed identity for Azure. This workflow allows the VM to connect to the workspace using the managed identity, without storing credentials in Python code or prompting the user to authenticate. Azure Machine Learning compute clusters can also be configured to use a managed identity to access the workspace when *training models*.

Regardless of the authentication workflow used, Azure role-based access control (Azure RBAC) is used to scope the level of access (authorization) allowed to the resources. For example, an admin or automation process might have access to create a compute instance, but not use it, while a data scientist could use it, but not delete or create it. For more information, see [Manage access to Azure Machine Learning workspace](#).

Azure AD Conditional Access can be used to further control or restrict access to the workspace for each authentication workflow. For example, an admin can allow workspace access from managed devices only.

Prerequisites

- Create an [Azure Machine Learning workspace](#).
- [Configure your development environment](#) or use a [Azure Machine Learning compute instance](#) and install the [Azure Machine Learning SDK v2](#).

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

- Install the [Azure CLI](#).

Azure Active Directory

All the authentication workflows for your workspace rely on Azure Active Directory. If you want users to authenticate using individual accounts, they must have accounts in your Azure AD. If you want to use service principals, they must exist in your Azure AD. Managed identities are also a feature of Azure AD.

For more on Azure AD, see [What is Azure Active Directory authentication](#).

Once you've created the Azure AD accounts, see [Manage access to Azure Machine Learning workspace](#) for information on granting them access to the workspace and other operations in Azure Machine Learning.

Configure a service principal

To use a service principal (SP), you must first create the SP. Then grant it access to your workspace. As mentioned earlier, Azure role-based access control (Azure RBAC) is used to control access, so you must also decide what access to grant the SP.

IMPORTANT

When using a service principal, grant it the **minimum access required for the task** it is used for. For example, you would not grant a service principal owner or contributor access if all it is used for is reading the access token for a web deployment.

The reason for granting the least access is that a service principal uses a password to authenticate, and the password may be stored as part of an automation script. If the password is leaked, having the minimum access required for a specific tasks minimizes the malicious use of the SP.

The easiest way to create an SP and grant access to your workspace is by using the [Azure CLI](#). To create a service principal and grant it access to your workspace, use the following steps:

NOTE

You must be an admin on the subscription to perform all of these steps.

1. Authenticate to your Azure subscription:

```
az login
```

If the CLI can open your default browser, it will do so and load a sign-in page. Otherwise, you need to open a browser and follow the instructions on the command line. The instructions involve browsing to <https://aka.ms/devicelogin> and entering an authorization code.

If you have multiple Azure subscriptions, you can use the `az account set -s <subscription name or ID>` command to set the subscription. For more information, see [Use multiple Azure subscriptions](#).

For other methods of authenticating, see [Sign in with Azure CLI](#).

2. Create the service principal. In the following example, an SP named **ml-auth** is created:

```
az ad sp create-for-rbac --sdk-auth --name ml-auth --role Contributor --scopes /subscriptions/<subscription id>
```

The output will be a JSON similar to the following. Take note of the `clientId`, `clientSecret`, and `tenantId` fields, as you'll need them for other steps in this article.

```
{  
    "clientId": "your-client-id",  
    "clientSecret": "your-client-secret",  
    "subscriptionId": "your-sub-id",  
    "tenantId": "your-tenant-id",  
    "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",  
    "resourceManagerEndpointUrl": "https://management.azure.com",  
    "activeDirectoryGraphResourceId": "https://graph.windows.net",  
    "sqlManagementEndpointUrl": "https://management.core.windows.net:5555",  
    "galleryEndpointUrl": "https://gallery.azure.com/",  
    "managementEndpointUrl": "https://management.core.windows.net"  
}
```

3. Retrieve the details for the service principal by using the `clientId` value returned in the previous step:

```
az ad sp show --id your-client-id
```

The following JSON is a simplified example of the output from the command. Take note of the `objectId` field, as you'll need its value for the next step.

```
{  
    "accountEnabled": "True",  
    "addIns": [],  
    "appDisplayName": "ml-auth",  
    ...  
    ...  
    ...  
    "objectId": "your-sp-object-id",  
    "objectType": "ServicePrincipal"  
}
```

4. To grant access to the workspace and other resources used by Azure Machine Learning, use the information in the following articles:

- [How to assign roles and actions in AzureML](#)
- [How to assign roles in the CLI](#)

IMPORTANT

Owner access allows the service principal to do virtually any operation in your workspace. It is used in this document to demonstrate how to grant access; in a production environment Microsoft recommends granting the service principal the minimum access needed to perform the role you intend it for. For information on creating a custom role with the access needed for your scenario, see [Manage access to Azure Machine Learning workspace](#).

Configure a managed identity

IMPORTANT

Managed identity is only supported when using the Azure Machine Learning SDK from an Azure Virtual Machine or with an Azure Machine Learning compute cluster.

Managed identity with a VM

1. Enable a [system-assigned managed identity for Azure resources on the VM](#).
2. From the [Azure portal](#), select your workspace and then select **Access Control (IAM)**.
3. Select **Add, Add Role Assignment** to open the **Add role assignment** page.
4. Select the role you want to assign the managed identity. For example, Reader. For detailed steps, see [Assign Azure roles using the Azure portal](#).

Managed identity with compute cluster

For more information, see [Set up managed identity for compute cluster](#).

Use interactive authentication

- [Python SDK v2](#)
- [Azure CLI](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

Interactive authentication uses the [Azure Identity package for Python](#). Most examples use `DefaultAzureCredential` to access your credentials. When a token is needed, it requests one using multiple identities (`EnvironmentCredential`, `ManagedIdentityCredential`, `SharedTokenCacheCredential`, `VisualStudioCodeCredential`, `AzureCliCredential`, `AzurePowerShellCredential`) in turn, stopping when one provides a token. For more information, see the [DefaultAzureCredential](#) class reference.

The following is an example of using `DefaultAzureCredential` to authenticate. If authentication using `DefaultAzureCredential` fails, a fallback of authenticating through your web browser is used instead.

```
from azure.identity import DefaultAzureCredential, InteractiveBrowserCredential

try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
    # This will open a browser page for
    credential = InteractiveBrowserCredential()
```

After the credential object has been created, the `MLClient` class is used to connect to the workspace. For example, the following code uses the `from_config()` method to load connection information:

```

try:
    ml_client = MLClient.from_config(credential=credential)
except Exception as ex:
    # NOTE: Update following workspace information to contain
    #       your subscription ID, resource group name, and workspace name
    client_config = {
        "subscription_id": "<SUBSCRIPTION_ID>",
        "resource_group": "<RESOURCE_GROUP>",
        "workspace_name": "<AZUREML_WORKSPACE_NAME>",
    }

    # write and reload from config file
    import json, os

    config_path = "../../azureml/config.json"
    os.makedirs(os.path.dirname(config_path), exist_ok=True)
    with open(config_path, "w") as fo:
        fo.write(json.dumps(client_config))
    ml_client = MLClient.from_config(credential=credential, path=config_path)

print(ml_client)

```

Use service principal authentication

- [Python SDK v2](#)
- [Azure CLI](#)

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

Authenticating with a service principal uses the [Azure Identity package for Python](#). The `DefaultAzureCredential` class looks for the following environment variables and uses the values when authenticating as the service principal:

- `AZURE_CLIENT_ID` - The client ID returned when you created the service principal.
- `AZURE_TENANT_ID` - The tenant ID returned when you created the service principal.
- `AZURE_CLIENT_SECRET` - The password/credential generated for the service principal.

TIP

During development, consider using the [python-dotenv](#) package to set these environment variables. Python-dotenv loads environment variables from `.env` files. The standard `.gitignore` file for Python automatically excludes `.env` files, so they shouldn't be checked into any GitHub repos during development.

The following example demonstrates using python-dotenv to load the environment variables from a `.env` file and then using `DefaultAzureCredential` to create the credential object:

```

from dotenv import load_dotenv

if (os.environ['ENVIRONMENT'] == 'development'):
    print("Loading environment variables from .env file")
    load_dotenv(".env")

from azure.identity import DefaultAzureCredential

credential = DefaultAzureCredential()
# Check if given credential can get token successfully.
credential.get_token("https://management.azure.com/.default")

```

After the credential object has been created, the `MLClient` class is used to connect to the workspace. For example, the following code uses the `from_config()` method to load connection information:

```
try:
    ml_client = MLClient.from_config(credential=credential)
except Exception as ex:
    # NOTE: Update following workspace information to contain
    #       your subscription ID, resource group name, and workspace name
    client_config = {
        "subscription_id": "<SUBSCRIPTION_ID>",
        "resource_group": "<RESOURCE_GROUP>",
        "workspace_name": "<AZUREML_WORKSPACE_NAME>",
    }

    # write and reload from config file
    import json, os

    config_path = ".../azureml/config.json"
    os.makedirs(os.path.dirname(config_path), exist_ok=True)
    with open(config_path, "w") as fo:
        fo.write(json.dumps(client_config))
    ml_client = MLClient.from_config(credential=credential, path=config_path)

print(ml_client)
```

The service principal can also be used to authenticate to the Azure Machine Learning [REST API](#). You use the Azure Active Directory [client credentials grant flow](#), which allow service-to-service calls for headless authentication in automated workflows.

IMPORTANT

If you are currently using Azure Active Directory Authentication Library (ADAL) to get credentials, we recommend that you [Migrate to the Microsoft Authentication Library \(MSAL\)](#). ADAL support ended June 30, 2022.

For information and samples on authenticating with MSAL, see the following articles:

- JavaScript - [How to migrate a JavaScript app from ADAL.js to MSAL.js](#).
- Node.js - [How to migrate a Node.js app from Microsoft Authentication Library to MSAL](#).
- Python - [Microsoft Authentication Library to MSAL migration guide for Python](#).

Use managed identity authentication

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

Authenticating with a managed identity uses the [Azure Identity package for Python](#). To authenticate to the workspace from a VM or compute cluster that is configured with a managed identity, use the `DefaultAzureCredential` class. This class automatically detects if a managed identity is being used, and uses the managed identity to authenticate to Azure services.

The following example demonstrates using the `DefaultAzureCredential` class to create the credential object, then using the `MLClient` class to connect to the workspace:

```

from azure.identity import DefaultAzureCredential

credential = DefaultAzureCredential()
# Check if given credential can get token successfully.
credential.get_token("https://management.azure.com/.default")

try:
    ml_client = MLClient.from_config(credential=credential)
except Exception as ex:
    # NOTE: Update following workspace information to contain
    #       your subscription ID, resource group name, and workspace name
    client_config = {
        "subscription_id": "<SUBSCRIPTION_ID>",
        "resource_group": "<RESOURCE_GROUP>",
        "workspace_name": "<AZUREML_WORKSPACE_NAME>",
    }

    # write and reload from config file
    import json, os

    config_path = "../.azureml/config.json"
    os.makedirs(os.path.dirname(config_path), exist_ok=True)
    with open(config_path, "w") as fo:
        fo.write(json.dumps(client_config))
    ml_client = MLClient.from_config(credential=credential, path=config_path)

print(ml_client)

```

Use Conditional Access

As an administrator, you can enforce [Azure AD Conditional Access policies](#) for users signing in to the workspace. For example, you can require two-factor authentication, or allow sign in only from managed devices. To use Conditional Access for Azure Machine Learning workspaces specifically, [assign the Conditional Access policy](#) to Machine Learning Cloud app.

Next steps

- [How to use secrets in training.](#)
- [How to authenticate to online endpoints.](#)

Manage access to an Azure Machine Learning workspace

9/22/2022 • 14 minutes to read • [Edit Online](#)

In this article, you learn how to manage access (authorization) to an Azure Machine Learning workspace. [Azure role-based access control \(Azure RBAC\)](#) is used to manage access to Azure resources, such as the ability to create new resources or use existing ones. Users in your Azure Active Directory (Azure AD) are assigned specific roles, which grant access to resources. Azure provides both built-in roles and the ability to create custom roles.

TIP

While this article focuses on Azure Machine Learning, individual services that Azure ML relies on provide their own RBAC settings. For example, using the information in this article, you can configure who can submit scoring requests to a model deployed as a web service on Azure Kubernetes Service. But Azure Kubernetes Service provides its own set of Azure roles. For service specific RBAC information that may be useful with Azure Machine Learning, see the following links:

- [Control access to Azure Kubernetes cluster resources](#)
- [Use Azure RBAC for Kubernetes authorization](#)
- [Use Azure RBAC for access to blob data](#)

WARNING

Applying some roles may limit UI functionality in Azure Machine Learning studio for other users. For example, if a user's role does not have the ability to create a compute instance, the option to create a compute instance will not be available in studio. This behavior is expected, and prevents the user from attempting operations that would return an access denied error.

Default roles

Azure Machine Learning workspaces have four built-in roles that are available by default. When adding users to a workspace, they can be assigned one of the built-in roles described below.

ROLE	ACCESS LEVEL
AzureML Data Scientist	Can perform all actions within an Azure Machine Learning workspace, except for creating or deleting compute resources and modifying the workspace itself.
Reader	Read-only actions in the workspace. Readers can list and view assets, including datastore credentials, in a workspace. Readers can't create or update these assets.
Contributor	View, create, edit, or delete (where applicable) assets in a workspace. For example, contributors can create an experiment, create or attach a compute cluster, submit a run, and deploy a web service.
Owner	Full access to the workspace, including the ability to view, create, edit, or delete (where applicable) assets in a workspace. Additionally, you can change role assignments.

IMPORTANT

Role access can be scoped to multiple levels in Azure. For example, someone with owner access to a workspace may not have owner access to the resource group that contains the workspace. For more information, see [How Azure RBAC works](#).

Manage workspace access

If you're an owner of a workspace, you can add and remove roles for the workspace. You can also assign roles to users. Use the following links to discover how to manage access:

- [Azure portal UI](#)
- [PowerShell](#)
- [Azure CLI](#)
- [REST API](#)
- [Azure Resource Manager templates](#)

Use Azure AD security groups to manage workspace access

You can use Azure AD security groups to manage access to workspaces. This approach has following benefits:

- Team or project leaders can manage user access to workspace as security group owners, without needing Owner role on the workspace resource directly.
- You can organize, manage and revoke users' permissions on workspace and other resources as a group, without having to manage permissions on user-by-user basis.
- Using Azure AD groups helps you to avoid reaching the [subscription limit](#) on role assignments.

To use Azure AD security groups:

1. [Create a security group](#).
2. [Add a group owner](#). This user has permissions to add or remove group members. Note that the group owner is not required to be group member, or have direct RBAC role on the workspace.
3. Assign the group an RBAC role on the workspace, such as AzureML Data Scientist, Reader or Contributor.
4. [Add group members](#). The members consequently gain access to the workspace.

Create custom role

If the built-in roles are insufficient, you can create custom roles. Custom roles might have read, write, delete, and compute resource permissions in that workspace. You can make the role available at a specific workspace level, a specific resource group level, or a specific subscription level.

NOTE

You must be an owner of the resource at that level to create custom roles within that resource.

To create a custom role, first construct a role definition JSON file that specifies the permission and scope for the role. The following example defines a custom role named "Data Scientist Custom" scoped at a specific workspace level:

`data_scientist_custom_role.json` :

```
{  
    "Name": "Data Scientist Custom",  
    "IsCustom": true,  
    "Description": "Can run experiment but can't create or delete compute.",  
    "Actions": ["*"],  
    "NotActions": [  
        "Microsoft.MachineLearningServices/workspaces/*/delete",  
        "Microsoft.MachineLearningServices/workspaces/write",  
        "Microsoft.MachineLearningServices/workspaces/computes/*/write",  
        "Microsoft.MachineLearningServices/workspaces/computes/*/delete",  
        "Microsoft.Authorization/*/write"  
    ],  
    "AssignableScopes": [  
        "/subscriptions/<subscription_id>/resourceGroups/<resource_group_name>/providers/Microsoft.MachineLearningSe  
rvices/workspaces/<workspace_name>"  
    ]  
}
```

TIP

You can change the `AssignableScopes` field to set the scope of this custom role at the subscription level, the resource group level, or a specific workspace level. The above custom role is just an example, see some suggested [custom roles for the Azure Machine Learning service](#).

This custom role can do everything in the workspace except for the following actions:

- It can't create or update a compute resource.
- It can't delete a compute resource.
- It can't add, delete, or alter role assignments.
- It can't delete the workspace.

To deploy this custom role, use the following Azure CLI command:

```
az role definition create --role-definition data_scientist_role.json
```

After deployment, this role becomes available in the specified workspace. Now you can add and assign this role in the Azure portal.

For more information on custom roles, see [Azure custom roles](#).

Azure Machine Learning operations

For more information on the operations (actions and not actions) usable with custom roles, see [Resource provider operations](#). You can also use the following Azure CLI command to list operations:

```
az provider operation show -n Microsoft.MachineLearningServices
```

List custom roles

In the Azure CLI, run the following command:

```
az role definition list --subscription <sub-id> --custom-role-only true
```

To view the role definition for a specific custom role, use the following Azure CLI command. The `<role-name>` should be in the same format returned by the command above:

```
az role definition list -n <role-name> --subscription <sub-id>
```

Update a custom role

In the Azure CLI, run the following command:

```
az role definition update --role-definition update_def.json --subscription <sub-id>
```

You need to have permissions on the entire scope of your new role definition. For example if this new role has a scope across three subscriptions, you need to have permissions on all three subscriptions.

NOTE

Role updates can take 15 minutes to an hour to apply across all role assignments in that scope.

Use Azure Resource Manager templates for repeatability

If you anticipate that you will need to recreate complex role assignments, an Azure Resource Manager template can be a big help. The [machine-learning-dependencies-role-assignment template](#) shows how role assignments can be specified in source code for reuse.

Common scenarios

The following table is a summary of Azure Machine Learning activities and the permissions required to perform them at the least scope. For example, if an activity can be performed with a workspace scope (Column 4), then all higher scope with that permission will also work automatically:

IMPORTANT

All paths in this table that start with `/` are relative paths to `Microsoft.MachineLearningServices/`:

ACTIVITY	SUBSCRIPTION-LEVEL SCOPE	RESOURCE GROUP-LEVEL SCOPE	WORKSPACE-LEVEL SCOPE
Create new workspace	Not required	Owner or contributor	N/A (becomes Owner or inherits higher scope role after creation)
Request subscription level Amlcompute quota or set workspace level quota	Owner, or contributor, or custom role allowing: <code>/locations/updateQuotas/action</code> at subscription scope	Not Authorized	Not Authorized
Create new compute cluster	Not required	Not required	Owner, contributor, or custom role allowing: <code>/workspaces/computes/write</code>
Create new compute instance	Not required	Not required	Owner, contributor, or custom role allowing: <code>/workspaces/computes/write</code>

Activity	Subscription-level scope	Resource group-level scope	Workspace-level scope
Submitting any type of run	Not required	Not required	Owner, contributor, or custom role allowing: "/workspaces/*/read", "/workspaces/environments/write", "/workspaces/experiments/runs/write", "/workspaces/metadata/artifacts/write", "/workspaces/metadata/snapshots/write", "/workspaces/environments/build/action", "/workspaces/experiments/runs/submit/action", "/workspaces/environments/readSecrets/action"
Publishing pipelines and endpoints	Not required	Not required	Owner, contributor, or custom role allowing: "/workspaces/endpoints/pipelines/*", "/workspaces/pipelinedrafts/*", "/workspaces/modules/*"
Attach an AKS resource ²	Not required	Owner or contributor on the resource group that contains AKS	
Deploying a registered model on an AKS/AI resource	Not required	Not required	Owner, contributor, or custom role allowing: "/workspaces/services/aks/write", "/workspaces/services/aci/write"
Scoring against a deployed AKS endpoint	Not required	Not required	Owner, contributor, or custom role allowing: "/workspaces/services/aks/score/action", "/workspaces/services/aks/listkeys/action" (when you are not using Azure Active Directory auth) OR "/workspaces/read" (when you are using token auth)
Accessing storage using interactive notebooks	Not required	Not required	Owner, contributor, or custom role allowing: "/workspaces/computes/read", "/workspaces/notebooks/samples/read", "/workspaces/notebooks/storage/*", "/workspaces/listStorageAccountKeys/action", "/workspaces/listNotebookAccessToken/read"
Create new custom role	Owner, contributor, or custom role allowing <code>Microsoft.Authorization/roleDefinitions/write</code>	Not required	Owner, contributor, or custom role allowing: "/workspaces/computes/write"
Create/manage online endpoints and deployments	Not required	Not required	Owner, contributor, or custom role allowing <code>Microsoft.MachineLearningServices/workspaces/onlineEndp</code>
Retrieve authentication credentials for online endpoints	Not required	Not required	Owner, contributor, or custom role allowing <code>Microsoft.MachineLearningServices/workspaces/onlineEndp</code> and <code>Microsoft.MachineLearningServices/workspaces/onlineEndp</code> .

1: If you receive a failure when trying to create a workspace for the first time, make sure that your role allows `Microsoft.MachineLearningServices/register/action`. This action allows you to register the Azure Machine Learning resource provider with your Azure subscription.

2: When attaching an AKS cluster, you also need to the [Azure Kubernetes Service Cluster Admin Role](#) on the cluster.

Create a workspace using a customer-managed key

When using a customer-managed key (CMK), an Azure Key Vault is used to store the key. The user or service principal used to create the workspace must have owner or contributor access to the key vault.

Within the key vault, the user or service principal must have create, get, delete, and purge access to the key through a key vault access policy. For more information, see [Azure Key Vault security](#).

User-assigned managed identity with Azure ML compute cluster

To assign a user assigned identity to an Azure Machine Learning compute cluster, you need write permissions to create the compute and the [Managed Identity Operator Role](#). For more information on Azure RBAC with Managed Identities, read [How to manage user assigned identity](#)

To perform MLflow operations with your Azure Machine Learning workspace, use the following scopes your custom role:

MLFLOW OPERATION	SCOPE
List all experiments in the workspace tracking store, get an experiment by id, get an experiment by name	Microsoft.MachineLearningServices/workspaces/experiments/read
Create an experiment with a name , set a tag on an experiment, restore an experiment marked for deletion	Microsoft.MachineLearningServices/workspaces/experiments/write
Delete an experiment	Microsoft.MachineLearningServices/workspaces/experiments/delete
Get a run and related data and metadata, get a list of all values for the specified metric for a given run, list artifacts for a run	Microsoft.MachineLearningServices/workspaces/experiments/runs/read
Create a new run within an experiment, delete runs, restore deleted runs, log metrics under the current run, set tags on a run, delete tags on a run, log params (key-value pair) used for a run, log a batch of metrics, params, and tags for a run, update run status	Microsoft.MachineLearningServices/workspaces/experiments/runs/write
Get registered model by name, fetch a list of all registered models in the registry, search for registered models, latest version models for each requests stage, get a registered model's version, search model versions, get URI where a model version's artifacts are stored, search for runs by experiment ids	Microsoft.MachineLearningServices/workspaces/models/read
Create a new registered model, update a registered model's name/description, rename existing registered model, create new version of the model, update a model version's description, transition a registered model to one of the stages	Microsoft.MachineLearningServices/workspaces/models/write
Delete a registered model along with all its version, delete specific versions of a registered model	Microsoft.MachineLearningServices/workspaces/models/delete

Example custom roles

Data scientist

Allows a data scientist to perform all operations inside a workspace **except**:

- Creation of compute
- Deploying models to a production AKS cluster
- Deploying a pipeline endpoint in production

`data_scientist_custom_role.json` :

```
{
  "Name": "Data Scientist Custom",
  "IsCustom": true,
  "Description": "Can run experiment but can't create or delete compute or deploy production endpoints.",
  "Actions": [
    "Microsoft.MachineLearningServices/workspaces/*/read",
    "Microsoft.MachineLearningServices/workspaces/*/action",
    "Microsoft.MachineLearningServices/workspaces/*/delete",
    "Microsoft.MachineLearningServices/workspaces/*/write"
  ],
  "NotActions": [
    "Microsoft.MachineLearningServices/workspaces/delete",
    "Microsoft.MachineLearningServices/workspaces/write",
    "Microsoft.MachineLearningServices/workspaces/computes/*/write",
    "Microsoft.MachineLearningServices/workspaces/computes/*/delete",
    "Microsoft.Authorization/*",
    "Microsoft.MachineLearningServices/workspaces/computes/listKeys/action",
    "Microsoft.MachineLearningServices/workspaces/listKeys/action",
    "Microsoft.MachineLearningServices/workspaces/services/aks/write",
    "Microsoft.MachineLearningServices/workspaces/services/aks/delete",
    "Microsoft.MachineLearningServices/workspaces/endpoints/pipelines/write"
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>"
  ]
}
```

Data scientist restricted

A more restricted role definition without wildcards in the allowed actions. It can perform all operations inside a

workspace **except**:

- Creation of compute
- Deploying models to a production AKS cluster
- Deploying a pipeline endpoint in production

`data_scientist_restricted_custom_role.json` :

```
{
    "Name": "Data Scientist Restricted Custom",
    "IsCustom": true,
    "Description": "Can run experiment but can't create or delete compute or deploy production endpoints",
    "Actions": [
        "Microsoft.MachineLearningServices/workspaces/*/read",
        "Microsoft.MachineLearningServices/workspaces/computes/start/action",
        "Microsoft.MachineLearningServices/workspaces/computes/stop/action",
        "Microsoft.MachineLearningServices/workspaces/computes/restart/action",
        "Microsoft.MachineLearningServices/workspaces/computes/applicationaccess/action",
        "Microsoft.MachineLearningServices/workspaces/notebooks/storage/read",
        "Microsoft.MachineLearningServices/workspaces/notebooks/storage/write",
        "Microsoft.MachineLearningServices/workspaces/notebooks/storage/delete",
        "Microsoft.MachineLearningServices/workspaces/notebooks/samples/read",
        "Microsoft.MachineLearningServices/workspaces/experiments/runs/write",
        "Microsoft.MachineLearningServices/workspaces/experiments/write",
        "Microsoft.MachineLearningServices/workspaces/experiments/runs/submit/action",
        "Microsoft.MachineLearningServices/workspaces/pipelinedrafts/write",
        "Microsoft.MachineLearningServices/workspaces/metadata/snapshots/write",
        "Microsoft.MachineLearningServices/workspaces/metadata/artifacts/write",
        "Microsoft.MachineLearningServices/workspaces/environments/write",
        "Microsoft.MachineLearningServices/workspaces/models/write",
        "Microsoft.MachineLearningServices/workspaces/modules/write",
        "Microsoft.MachineLearningServices/workspaces/datasets/registered/write",
        "Microsoft.MachineLearningServices/workspaces/datasets/registered/delete",
        "Microsoft.MachineLearningServices/workspaces/datasets/unregistered/write",
        "Microsoft.MachineLearningServices/workspaces/datasets/unregistered/delete",
        "Microsoft.MachineLearningServices/workspaces/computes/listNodes/action",
        "Microsoft.MachineLearningServices/workspaces/environments/build/action"
    ],
    "NotActions": [
        "Microsoft.MachineLearningServices/workspaces/computes/write",
        "Microsoft.MachineLearningServices/workspaces/write",
        "Microsoft.MachineLearningServices/workspaces/computes/delete",
        "Microsoft.MachineLearningServices/workspaces/delete",
        "Microsoft.MachineLearningServices/workspaces/computes/listKeys/action",
        "Microsoft.MachineLearningServices/workspaces/listKeys/action",
        "Microsoft.Authorization/*",
        "Microsoft.MachineLearningServices/workspaces/datasets/registered/profile/read",
        "Microsoft.MachineLearningServices/workspaces/datasets/registered/preview/read",
        "Microsoft.MachineLearningServices/workspaces/datasets/unregistered/profile/read",
        "Microsoft.MachineLearningServices/workspaces/datasets/unregistered/preview/read",
        "Microsoft.MachineLearningServices/workspaces/datasets/registered/schema/read",
        "Microsoft.MachineLearningServices/workspaces/datasets/unregistered/schema/read",
        "Microsoft.MachineLearningServices/workspaces/datastores/write",
        "Microsoft.MachineLearningServices/workspaces/datastores/delete"
    ],
    "AssignableScopes": [
        "/subscriptions/<subscription_id>"
    ]
}
```

MLflow data scientist

Allows a data scientist to perform all MLflow AzureML supported operations **except**:

- Creation of compute
- Deploying models to a production AKS cluster
- Deploying a pipeline endpoint in production

`mlflow_data_scientist_custom_role.json` :

```
{
  "Name": "MLFlow Data Scientist Custom",
  "IsCustom": true,
  "Description": "Can perform azureml mlflow integrated functionalities that includes mlflow tracking, projects, model registry",
  "Actions": [
    "Microsoft.MachineLearningServices/workspaces/experiments/read",
    "Microsoft.MachineLearningServices/workspaces/experiments/write",
    "Microsoft.MachineLearningServices/workspaces/experiments/delete",
    "Microsoft.MachineLearningServices/workspaces/experiments/runs/read",
    "Microsoft.MachineLearningServices/workspaces/experiments/runs/write",
    "Microsoft.MachineLearningServices/workspaces/models/read",
    "Microsoft.MachineLearningServices/workspaces/models/write",
    "Microsoft.MachineLearningServices/workspaces/models/delete"
  ],
  "NotActions": [
    "Microsoft.MachineLearningServices/workspaces/delete",
    "Microsoft.MachineLearningServices/workspaces/write",
    "Microsoft.MachineLearningServices/workspaces/computes/*/write",
    "Microsoft.MachineLearningServices/workspaces/computes/*/delete",
    "Microsoft.Authorization/*",
    "Microsoft.MachineLearningServices/workspaces/computes/listKeys/action",
    "Microsoft.MachineLearningServices/workspaces/listKeys/action",
    "Microsoft.MachineLearningServices/workspaces/services/aks/write",
    "Microsoft.MachineLearningServices/workspaces/services/aks/delete",
    "Microsoft.MachineLearningServices/workspaces/endpoints/pipelines/write"
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>"
  ]
}
```

MLOps

Allows you to assign a role to a service principal and use that to automate your MLOps pipelines. For example, to submit runs against an already published pipeline:

`mlops_custom_role.json` :

```
{
  "Name": "MLOps Custom",
  "IsCustom": true,
  "Description": "Can run pipelines against a published pipeline endpoint",
  "Actions": [
    "Microsoft.MachineLearningServices/workspaces/read",
    "Microsoft.MachineLearningServices/workspaces/endpoints/pipelines/read",
    "Microsoft.MachineLearningServices/workspaces/metadata/artifacts/read",
    "Microsoft.MachineLearningServices/workspaces/metadata/snapshots/read",
    "Microsoft.MachineLearningServices/workspaces/environments/read",
    "Microsoft.MachineLearningServices/workspaces/metadata/secrets/read",
    "Microsoft.MachineLearningServices/workspaces/modules/read",
    "Microsoft.MachineLearningServices/workspaces/experiments/runs/read",
    "Microsoft.MachineLearningServices/workspaces/datasets/registered/read",
    "Microsoft.MachineLearningServices/workspaces/datastores/read",
    "Microsoft.MachineLearningServices/workspaces/environments/write",
    "Microsoft.MachineLearningServices/workspaces/experiments/runs/write",
    "Microsoft.MachineLearningServices/workspaces/metadata/artifacts/write",
    "Microsoft.MachineLearningServices/workspaces/metadata/snapshots/write",
    "Microsoft.MachineLearningServices/workspaces/environments/build/action",
    "Microsoft.MachineLearningServices/workspaces/experiments/runs/submit/action"
  ],
  "NotActions": [
    "Microsoft.MachineLearningServices/workspaces/computes/write",
    "Microsoft.MachineLearningServices/workspaces/write",
    "Microsoft.MachineLearningServices/workspaces/computes/delete",
    "Microsoft.MachineLearningServices/workspaces/delete",
    "Microsoft.MachineLearningServices/workspaces/computes/listKeys/action",
    "Microsoft.MachineLearningServices/workspaces/listKeys/action",
    "Microsoft.Authorization/*"
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>"
  ]
}
```

Workspace Admin

Allows you to perform all operations within the scope of a workspace, **except**:

- Creating a new workspace
- Assigning subscription or workspace level quotas

The workspace admin also cannot create a new role. It can only assign existing built-in or custom roles within the scope of their workspace:

`workspace_admin_custom_role.json` :

```
{
  "Name": "Workspace Admin Custom",
  "IsCustom": true,
  "Description": "Can perform all operations except quota management and upgrades",
  "Actions": [
    "Microsoft.MachineLearningServices/workspaces/*/read",
    "Microsoft.MachineLearningServices/workspaces/*/action",
    "Microsoft.MachineLearningServices/workspaces/*/write",
    "Microsoft.MachineLearningServices/workspaces/*/delete",
    "Microsoft.Authorization/roleAssignments/*"
  ],
  "NotActions": [
    "Microsoft.MachineLearningServices/workspaces/write"
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>"
  ]
}
```

Data labeling

- [Data labeler](#)
- [Labeling team lead](#)
- [Vendor account manager](#)
- [Customer QA](#)
- [Vendor QA](#)

Allows you to define a role scoped only to labeling data:

`labeler_custom_role.json` :

```
{
  "Name": "Labeler Custom",
  "IsCustom": true,
  "Description": "Can label data for Labeling",
  "Actions": [
    "Microsoft.MachineLearningServices/workspaces/read",
    "Microsoft.MachineLearningServices/workspaces/labeling/projects/read",
    "Microsoft.MachineLearningServices/workspaces/labeling/projects/summary/read",
    "Microsoft.MachineLearningServices/workspaces/labeling/labels/read",
    "Microsoft.MachineLearningServices/workspaces/labeling/labels/write"
  ],
  "NotActions": [
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>"
  ]
}
```

Troubleshooting

Here are a few things to be aware of while you use Azure role-based access control (Azure RBAC):

- When you create a resource in Azure, such as a workspace, you are not directly the owner of the resource. Your role is inherited from the highest scope role that you are authorized against in that subscription. As an example if you are a Network Administrator, and have the permissions to create a Machine Learning workspace, you would be assigned the Network Administrator role against that workspace, and not the Owner role.
- To perform quota operations in a workspace, you need subscription level permissions. This means setting either subscription level quota or workspace level quota for your managed compute resources can only happen if you have write permissions at the subscription scope.
- When there are two role assignments to the same Azure Active Directory user with conflicting sections of Actions/NotActions, your operations listed in NotActions from one role might not take effect if they are also listed as Actions in another role. To learn more about how Azure parses role assignments, read [How Azure RBAC determines if a user has access to a resource](#)
- To deploy your compute resources inside a VNet, you need to explicitly have permissions for the following actions:
 - `Microsoft.Network/virtualNetworks/*/read` on the VNet resources.
 - `Microsoft.Network/virtualNetworks/subnets/join/action` on the subnet resource.
 For more information on Azure RBAC with networking, see the [Networking built-in roles](#).
- It can sometimes take up to 1 hour for your new role assignments to take effect over cached permissions across the stack.

Next steps

- Enterprise security overview
- Virtual network isolation and privacy overview
- Tutorial: Train and deploy a model
- Resource provider operations

Use Managed identities with Azure Machine Learning

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

APPLIES TO:  Azure CLI ml extension v2 (current)

[Managed identities](#) allow you to configure your workspace with the *minimum required permissions to access resources*.

When configuring Azure Machine Learning workspace in trustworthy manner, it's important to ensure that different services associated with the workspace have the correct level of access. For example, during machine learning workflow the workspace needs access to Azure Container Registry (ACR) for Docker images, and storage accounts for training data.

Furthermore, managed identities allow fine-grained control over permissions, for example you can grant or revoke access from specific compute resources to a specific ACR.

In this article, you'll learn how to use managed identities to:

- Configure and use ACR for your Azure Machine Learning workspace without having to enable admin user access to ACR.
- Access a private ACR external to your workspace, to pull base images for training or inference.
- Create workspace with user-assigned managed identity to access associated resources.

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create workspace resources](#).
- The [Azure CLI extension for Machine Learning service](#)
- The [Azure Machine Learning Python SDK](#).
- To assign roles, the login for your Azure subscription must have the [Managed Identity Operator](#) role, or other role that grants the required actions (such as [Owner](#)).
- You must be familiar with creating and working with [Managed Identities](#).

Configure managed identities

In some situations, it's necessary to disallow admin user access to Azure Container Registry. For example, the ACR may be shared and you need to disallow admin access by other users. Or, creating ACR with admin user enabled is disallowed by a subscription level policy.

IMPORTANT

When using Azure Machine Learning for inference on Azure Container Instance (ACI), admin user access on ACR is **required**. Do not disable it if you plan on deploying models to ACI for inference.

When you create ACR without enabling admin user access, managed identities are used to access the ACR to build and pull Docker images.

You can bring your own ACR with admin user disabled when you create the workspace. Alternatively, let Azure

Machine Learning create workspace ACR and disable admin user afterwards.

Bring your own ACR

If ACR admin user is disallowed by subscription policy, you should first create ACR without admin user, and then associate it with the workspace. Also, if you have existing ACR with admin user disabled, you can attach it to the workspace.

[Create ACR from Azure CLI](#) without setting `--admin-enabled` argument, or from Azure portal without enabling admin user. Then, when creating Azure Machine Learning workspace, specify the Azure resource ID of the ACR. The following example demonstrates creating a new Azure ML workspace that uses an existing ACR:

TIP

To get the value for the `--container-registry` parameter, use the `az acr show` command to show information for your ACR. The `id` field contains the resource ID for your ACR.

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

```
az ml workspace create -w <workspace name> \
-g <workspace resource group> \
-l <region> \
--container-registry /subscriptions/<subscription id>/resourceGroups/<acr resource group>/providers/Microsoft.ContainerRegistry/registries/<acr name>
```

Let Azure Machine Learning service create workspace ACR

If you don't bring your own ACR, Azure Machine Learning service will create one for you when you perform an operation that needs one. For example, submit a training job to Machine Learning Compute, build an environment, or deploy a web service endpoint. The ACR created by the workspace will have admin user enabled, and you need to disable the admin user manually.

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

1. Create a new workspace

```
az ml workspace show -n <my workspace> -g <my resource group>
```

2. Perform an action that requires ACR. For example, the [tutorial on training a model](#).

3. Get the ACR name created by the cluster:

```
az ml workspace show -w <my workspace> \
-g <my resource group> \
--query containerRegistry
```

This command returns a value similar to the following text. You only want the last portion of the text, which is the ACR instance name:

```
/subscriptions/<subscription id>/resourceGroups/<my resource group>/providers/MicrosoftContainerReggistry/registries/<ACR instance name>
```

4. Update the ACR to disable the admin user:

```
az acr update --name <ACR instance name> --admin-enabled false
```

Create compute with managed identity to access Docker images for training

To access the workspace ACR, create machine learning compute cluster with system-assigned managed identity enabled. You can enable the identity from Azure portal or Studio when creating compute, or from Azure CLI using the below. For more information, see [using managed identity with compute clusters](#).

- [Python SDK](#)
- [Azure CLI](#)
- [Studio](#)

When creating a compute cluster with the `AmlComputeProvisioningConfiguration`, use the `identity_type` parameter to set the managed identity type.

A managed identity is automatically granted ACRPull role on workspace ACR to enable pulling Docker images for training.

NOTE

If you create compute first, before workspace ACR has been created, you have to assign the ACRPull role manually.

Access base images from private ACR

By default, Azure Machine Learning uses Docker base images that come from a public repository managed by Microsoft. It then builds your training or inference environment on those images. For more information, see [What are ML environments?](#).

To use a custom base image internal to your enterprise, you can use managed identities to access your private ACR. There are two use cases:

- Use base image for training as is.
- Build Azure Machine Learning managed image with custom image as a base.

Pull Docker base image to machine learning compute cluster for training as is

Create machine learning compute cluster with system-assigned managed identity enabled as described earlier. Then, determine the principal ID of the managed identity.

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml compute show --name <cluster name> -w <workspace> -g <resource group>
```

Optionally, you can update the compute cluster to assign a user-assigned managed identity:

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml compute update --name <cluster name> --user-assigned-identities <my-identity-id>
```

To allow the compute cluster to pull the base images, grant the managed service identity ACRPull role on the private ACR

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az role assignment create --assignee <principal ID> \
--role acrpull \
--scope "/subscriptions/<subscription ID>/resourceGroups/<private ACR resource
group>/providers/Microsoft.ContainerRegistry/registries/<private ACR name>"
```

Finally, when submitting a training job, specify the base image location in the [environment definition](#).

APPLIES TO:  [Python SDK azureml v1](#)

```
from azureml.core import Environment
env = Environment(name="private-acr")
env.docker.base_image = "<ACR name>.azurecr.io/<base image repository>/<base image version>"
env.python.user_managed_dependencies = True
```

IMPORTANT

To ensure that the base image is pulled directly to the compute resource, set `user_managed_dependencies = True` and do not specify a Dockerfile. Otherwise Azure Machine Learning service will attempt to build a new Docker image and fail, because only the compute cluster has access to pull the base image from ACR.

Build Azure Machine Learning managed environment into base image from private ACR for training or inference

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

In this scenario, Azure Machine Learning service builds the training or inference environment on top of a base image you supply from a private ACR. Because the image build task happens on the workspace ACR using ACR Tasks, you must perform more steps to allow access.

1. Create **user-assigned managed identity** and grant the identity ACRPull access to the **private ACR**.
2. Grant the workspace **system-assigned managed identity** a Managed Identity Operator role on the **user-assigned managed identity** from the previous step. This role allows the workspace to assign the user-assigned managed identity to ACR Task for building the managed environment.
 - a. Obtain the principal ID of workspace system-assigned managed identity:

```
az ml workspace show -w <workspace name> -g <resource group> --query identityPrincipalId
```

- b. Grant the Managed Identity Operator role:

```
az role assignment create --assignee <principal ID> --role managedidentityoperator --scope
<user-assigned managed identity resource ID>
```

The user-assigned managed identity resource ID is Azure resource ID of the user assigned identity, in the format

```
/subscriptions/<subscription ID>/resourceGroups/<resource
group>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<user-assigned managed
identity name>
```

3. Specify the external ACR and client ID of the **user-assigned managed identity** in workspace connections by using [Workspace.set_connection method](#):

APPLIES TO:  [Python SDK azureml v1](#)

```
workspace.set_connection(  
    name="privateAcr",  
    category="ACR",  
    target = "<acr url>",  
    authType = "ManagedIdentity",  
    value={"ResourceId": "<user-assigned managed identity resource id>", "ClientId": "<user-assigned managed identity client ID>"})
```

Once the configuration is complete, you can use the base images from private ACR when building environments for training or inference. The following code snippet demonstrates how to specify the base image ACR and image name in an environment definition:

APPLIES TO:  Python SDK azureml v1

```
from azureml.core import Environment  
  
env = Environment(name="my-env")  
env.docker.base_image = "<acr url>/my-repo/my-image:latest"
```

Optionally, you can specify the managed identity resource URL and client ID in the environment definition itself by using [RegistryIdentity](#). If you use registry identity explicitly, it overrides any workspace connections specified earlier:

APPLIES TO:  Python SDK azureml v1

```
from azureml.core.container_registry import RegistryIdentity  
  
identity = RegistryIdentity()  
identity.resource_id= "<user-assigned managed identity resource ID>"  
identity.client_id=<user-assigned managed identity client ID>  
env.docker.base_image_registry.registry_identity=identity  
env.docker.base_image = "my-acr.azurecr.io/my-repo/my-image:latest"
```

Use Docker images for inference

Once you've configured ACR without admin user as described earlier, you can access Docker images for inference without admin keys from your Azure Kubernetes service (AKS). When you create or attach AKS to workspace, the cluster's service principal is automatically assigned ACRPull access to workspace ACR.

NOTE

If you bring your own AKS cluster, the cluster must have service principal enabled instead of managed identity.

Create workspace with user-assigned managed identity

When creating a workspace, you can bring your own [user-assigned managed identity](#) that will be used to access the associated resources: ACR, KeyVault, Storage, and App Insights.

IMPORTANT

When creating workspace with user-assigned managed identity, you must create the associated resources yourself, and grant the managed identity roles on those resources. Use the [role assignment ARM template](#) to make the assignments.

Use Azure CLI or Python SDK to create the workspace. When using the CLI, specify the ID using the

--primary-user-assigned-identity parameter. When using the SDK, use `primary_user_assigned_identity`. The following are examples of using the Azure CLI and Python to create a new workspace using these parameters:

Azure CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

```
az ml workspace create -w <workspace name> -g <resource group> --primary-user-assigned-identity <managed identity ARM ID>
```

Python

APPLIES TO:  Python SDK azureml v1

```
from azureml.core import Workspace

ws = Workspace.create(name="workspace name",
                      subscription_id="subscription id",
                      resource_group="resource group name",
                      primary_user_assigned_identity="managed identity ARM ID")
```

You can also use an [ARM template](#) to create a workspace with user-assigned managed identity.

For a workspace with [customer-managed keys for encryption](#), you can pass in a user-assigned managed identity to authenticate from storage to Key Vault. Use argument `user-assigned-identity-for-cmk-encryption` (CLI) or `user_assigned_identity_for_cmk_encryption` (SDK) to pass in the managed identity. This managed identity can be the same or different as the workspace primary user assigned managed identity.

Next steps

- Learn more about [enterprise security in Azure Machine Learning](#)
- Learn about [identity-based data access](#)
- Learn about [managed identities on compute cluster](#).

Secure Azure Machine Learning workspace resources using virtual networks (VNets)

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

Secure Azure Machine Learning workspace resources and compute environments using virtual networks (VNets). This article uses an example scenario to show you how to configure a complete virtual network.

TIP

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)
- [Use a firewall](#)
- [API platform network isolation](#)

For a tutorial on creating a secure workspace, see [Tutorial: Create a secure workspace](#) or [Tutorial: Create a secure workspace using a template](#).

Prerequisites

This article assumes that you have familiarity with the following topics:

- [Azure Virtual Networks](#)
- [IP networking](#)
- [Azure Machine Learning workspace with private endpoint](#)
- [Network Security Groups \(NSG\)](#)
- [Network firewalls](#)

Example scenario

In this section, you learn how a common network scenario is set up to secure Azure Machine Learning communication with private IP addresses.

The following table compares how services access different parts of an Azure Machine Learning network with and without a VNet:

SCENARIO	WORKSPACE	ASSOCIATED RESOURCES	TRAINING COMPUTE ENVIRONMENT	INFERENCE COMPUTE ENVIRONMENT
No virtual network	Public IP	Public IP	Public IP	Public IP

SCENARIO	WORKSPACE	ASSOCIATED RESOURCES	TRAINING COMPUTE ENVIRONMENT	INFERRING COMPUTE ENVIRONMENT
Public workspace, all other resources in a virtual network	Public IP	Public IP (service endpoint) - or - Private IP (private endpoint)	Public IP	Private IP
Secure resources in a virtual network	Private IP (private endpoint)	Public IP (service endpoint) - or - Private IP (private endpoint)	Private IP	Private IP

- **Workspace** - Create a private endpoint for your workspace. The private endpoint connects the workspace to the vnet through several private IP addresses.
 - **Public access** - You can optionally enable public access for a secured workspace.
- **Associated resource** - Use service endpoints or private endpoints to connect to workspace resources like Azure storage, Azure Key Vault. For Azure Container Services, use a private endpoint.
 - **Service endpoints** provide the identity of your virtual network to the Azure service. Once you enable service endpoints in your virtual network, you can add a virtual network rule to secure the Azure service resources to your virtual network. Service endpoints use public IP addresses.
 - **Private endpoints** are network interfaces that securely connect you to a service powered by Azure Private Link. Private endpoint uses a private IP address from your VNet, effectively bringing the service into your VNet.
- **Training compute access** - Access training compute targets like Azure Machine Learning Compute Instance and Azure Machine Learning Compute Clusters with public or private IP addresses.
- **Inference compute access** - Access Azure Kubernetes Services (AKS) compute clusters with private IP addresses.

The next sections show you how to secure the network scenario described above. To secure your network, you must:

1. Secure the [workspace and associated resources](#).
2. Secure the [training environment](#).
3. Secure the [inferencing environment](#).
4. Optionally: [enable studio functionality](#).
5. Configure [firewall settings](#).
6. Configure [DNS name resolution](#).

Public workspace and secured resources

If you want to access the workspace over the public internet while keeping all the associated resources secured in a virtual network, use the following steps:

1. Create an [Azure Virtual Network](#) that will contain the resources used by the workspace.
2. Use **one** of the following options to create a publicly accessible workspace:
 - Create an Azure Machine Learning workspace that **does not** use the virtual network. For more information, see [Manage Azure Machine Learning workspaces](#).
 - Create a [Private Link-enabled workspace](#) to enable communication between your VNet and workspace. Then [enable public access to the workspace](#).

3. Add the following services to the virtual network by using *either* a service endpoint or a private endpoint. Also allow trusted Microsoft services to access these services:

SERVICE	ENDPOINT INFORMATION	ALLOW TRUSTED INFORMATION
Azure Key Vault	Service endpoint Private endpoint	Allow trusted Microsoft services to bypass this firewall
Azure Storage Account	Service and private endpoint Private endpoint	Grant access to trusted Azure services
Azure Container Registry	Private endpoint	Allow trusted services

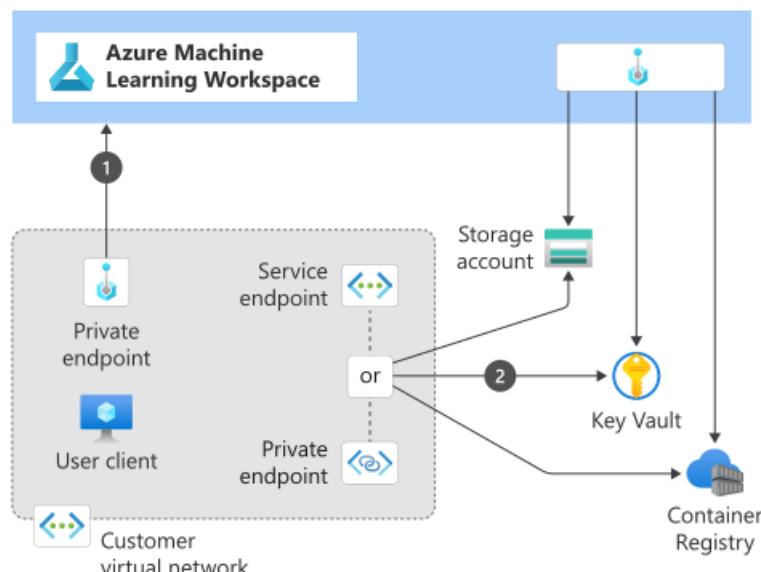
4. In properties for the Azure Storage Account(s) for your workspace, add your client IP address to the allowed list in firewall settings. For more information, see [Configure firewalls and virtual networks](#).

Secure the workspace and associated resources

Use the following steps to secure your workspace and associated resources. These steps allow your services to communicate in the virtual network.

- Create an [Azure Virtual Networks](#) that will contain the workspace and other resources. Then create a [Private Link-enabled workspace](#) to enable communication between your VNet and workspace.
- Add the following services to the virtual network by using *either* a service endpoint or a private endpoint. Also allow trusted Microsoft services to access these services:

SERVICE	ENDPOINT INFORMATION	ALLOW TRUSTED INFORMATION
Azure Key Vault	Service endpoint Private endpoint	Allow trusted Microsoft services to bypass this firewall
Azure Storage Account	Service and private endpoint Private endpoint	Grant access from Azure resource instances or Grant access to trusted Azure services
Azure Container Registry	Private endpoint	Allow trusted services



For detailed instructions on how to complete these steps, see [Secure an Azure Machine Learning workspace](#).

Limitations

Securing your workspace and associated resources within a virtual network have the following limitations:

- All resources must be behind the same VNet. However, subnets within the same VNet are allowed.

Secure the training environment

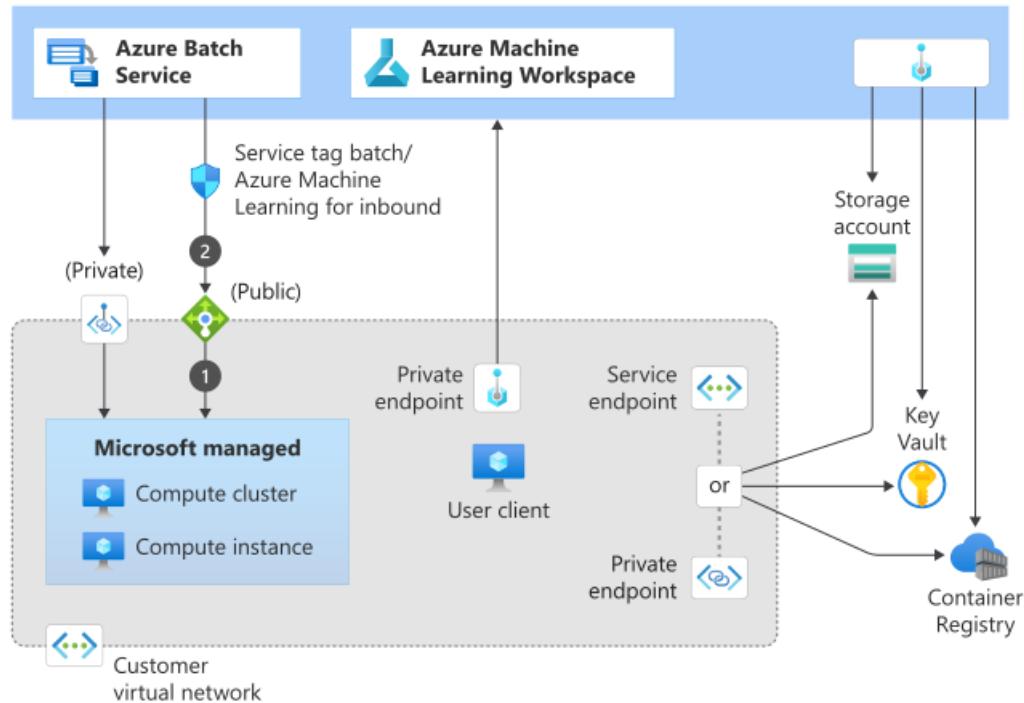
In this section, you learn how to secure the training environment in Azure Machine Learning. You also learn how Azure Machine Learning completes a training job to understand how the network configurations work together.

To secure the training environment, use the following steps:

1. Create an Azure Machine Learning [compute instance and computer cluster in the virtual network](#) to run the training job.
2. If your compute cluster or compute instance uses a public IP address, you must [Allow inbound communication](#) so that management services can submit jobs to your compute resources.

TIP

Compute cluster and compute instance can be created with or without a public IP address. If created with a public IP address, you get a load balancer with a public IP to accept the inbound access from Azure batch service and Azure Machine Learning service. You need to configure User Defined Routing (UDR) if you use a firewall. If created without a public IP, you get a private link service to accept the inbound access from Azure batch service and Azure Machine Learning service without a public IP.



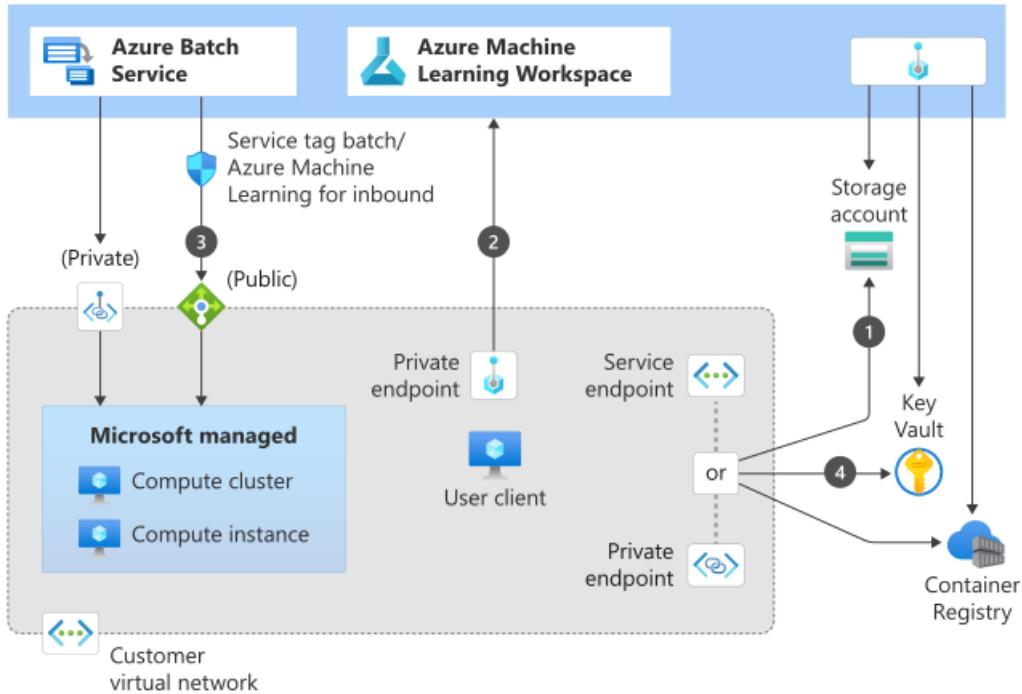
For detailed instructions on how to complete these steps, see [Secure a training environment](#).

Example training job submission

In this section, you learn how Azure Machine Learning securely communicates between services to submit a training job. This shows you how all your configurations work together to secure communication.

1. The client uploads training scripts and training data to storage accounts that are secured with a service or private endpoint.

2. The client submits a training job to the Azure Machine Learning workspace through the private endpoint.
3. Azure Batch service receives the job from the workspace. It then submits the training job to the compute environment through the public load balancer for the compute resource.
4. The compute resource receives the job and begins training. The compute resource uses information stored in key vault to access storage accounts to download training files and upload output.



Limitations

- Azure Compute Instance and Azure Compute Clusters must be in the same VNet, region, and subscription as the workspace and its associated resources.

Secure the inferencing environment

You can enable network isolation for managed online endpoints to secure the following network traffic:

- Inbound scoring requests.
- Outbound communication with the workspace, Azure Container Registry, and Azure Blob Storage.

IMPORTANT

Using network isolation for managed online endpoints is a [preview feature](#), and isn't fully supported.

For more information, see [Enable network isolation for managed online endpoints](#).

Optional: Enable public access

You can secure the workspace behind a VNet using a private endpoint and still allow access over the public internet. The initial configuration is the same as [securing the workspace and associated resources](#).

After securing the workspace with a private endpoint, use the following steps to enable clients to develop remotely using either the SDK or Azure Machine Learning studio:

1. [Enable public access](#) to the workspace.
2. [Configure the Azure Storage firewall](#) to allow communication with the IP address of clients that connect over the public internet.

Optional: enable studio functionality

If your storage is in a VNet, you must use extra configuration steps to enable full functionality in studio. By default, the following features are disabled:

- Preview data in the studio.
- Visualize data in the designer.
- Deploy a model in the designer.
- Submit an AutoML experiment.
- Start a labeling project.

To enable full studio functionality, see [Use Azure Machine Learning studio in a virtual network](#).

Limitations

[ML-assisted data labeling](#) doesn't support a default storage account behind a virtual network. Instead, use a storage account other than the default for ML assisted data labeling.

TIP

As long as it is not the default storage account, the account used by data labeling can be secured behind the virtual network.

Configure firewall settings

Configure your firewall to control traffic between your Azure Machine Learning workspace resources and the public internet. While we recommend Azure Firewall, you can use other firewall products.

For more information on firewall settings, see [Use workspace behind a Firewall](#).

Custom DNS

If you need to use a custom DNS solution for your virtual network, you must add host records for your workspace.

For more information on the required domain names and IP addresses, see [how to use a workspace with a custom DNS server](#).

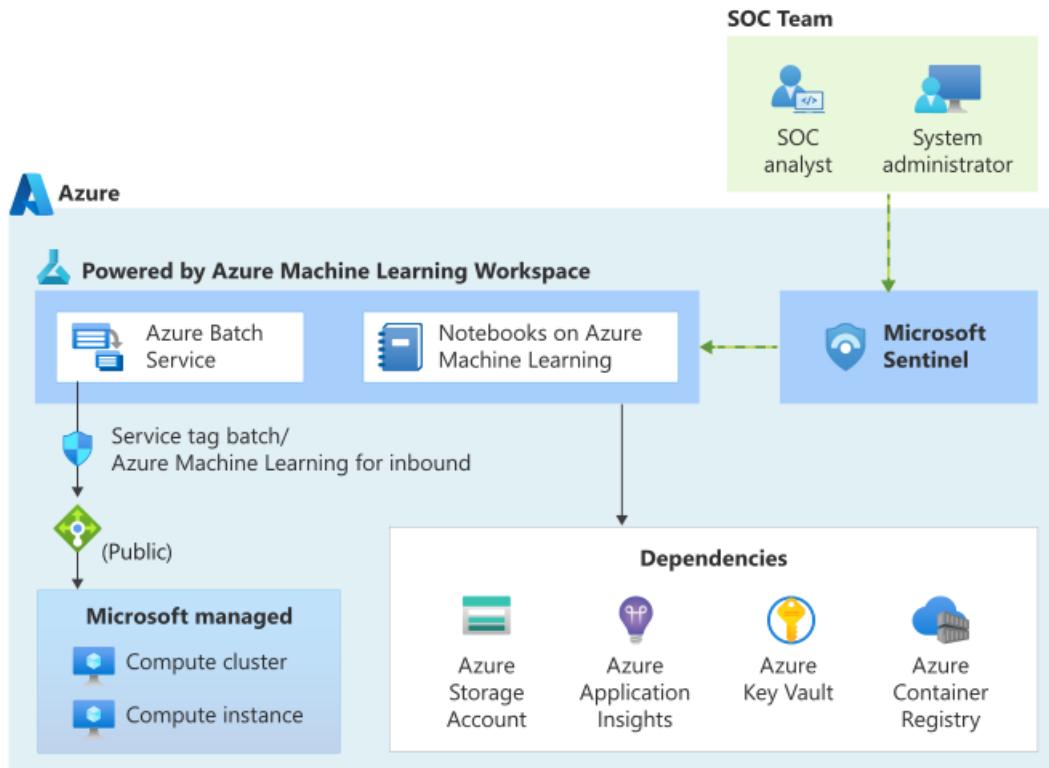
Microsoft Sentinel

Microsoft Sentinel is a security solution that can integrate with Azure Machine Learning. For example, using Jupyter notebooks provided through Azure Machine Learning. For more information, see [Use Jupyter notebooks to hunt for security threats](#).

Public access

Microsoft Sentinel can automatically create a workspace for you if you are OK with a public endpoint. In this configuration, the security operations center (SOC) analysts and system administrators connect to notebooks in your workspace through Sentinel.

For information on this process, see [Create an Azure ML workspace from Microsoft Sentinel](#)



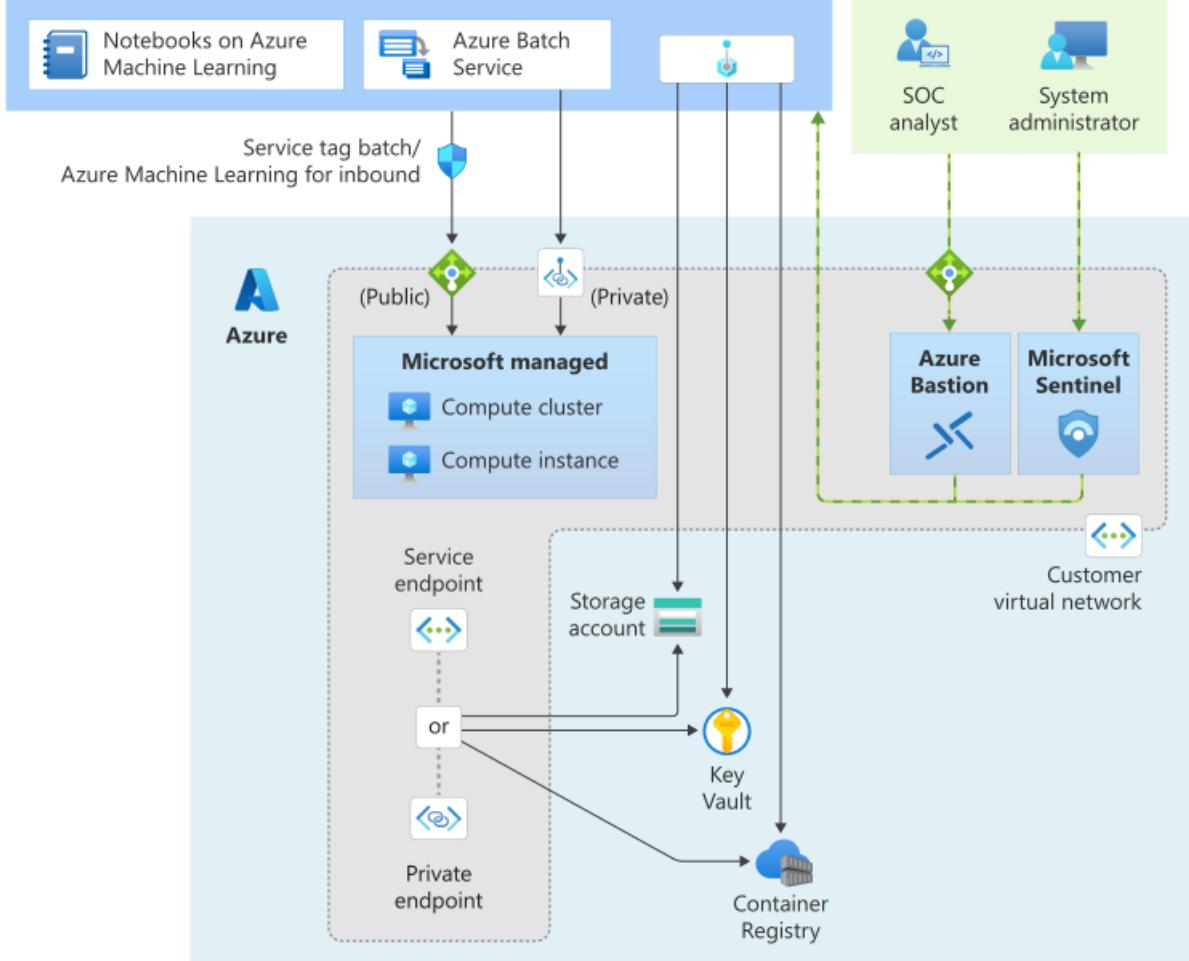
Private endpoint

If you want to secure your workspace and associated resources in a VNet, you must create the Azure Machine Learning workspace first. You must also create a virtual machine 'jump box' in the same VNet as your workspace, and enable Azure Bastion connectivity to it. Similar to the public configuration, SOC analysts and administrators can connect using Microsoft Sentinel, but some operations must be performed using Azure Bastion to connect to the VM.

For more information on this configuration, see [Create an Azure ML workspace from Microsoft Sentinel](#)



Powered by Azure Machine Learning Workspace



Next steps

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)
- [Use a firewall](#)
- [API platform network isolation](#)

Secure an Azure Machine Learning workspace with virtual networks

9/22/2022 • 16 minutes to read • [Edit Online](#)

In this article, you learn how to secure an Azure Machine Learning workspace and its associated resources in a virtual network.

TIP

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)
- [Use a firewall](#)
- [API platform network isolation](#)

For a tutorial on creating a secure workspace, see [Tutorial: Create a secure workspace](#) or [Tutorial: Create a secure workspace using a template](#).

In this article you learn how to enable the following workspaces resources in a virtual network:

- Azure Machine Learning workspace
- Azure Storage accounts
- Azure Machine Learning datastores and datasets
- Azure Key Vault
- Azure Container Registry

Prerequisites

- Read the [Network security overview](#) article to understand common virtual network scenarios and overall virtual network architecture.
- Read the [Azure Machine Learning best practices for enterprise security](#) article to learn about best practices.
- An existing virtual network and subnet to use with your compute resources.

IMPORTANT

We do not recommend using the 172.17.0.0/16 IP address range for your VNet. This is the default subnet range used by the Docker bridge network. Other ranges may also conflict depending on what you want to connect to the virtual network. For example, if you plan to connect your on-premises network to the VNet, and your on-premises network also uses the 172.16.0.0/16 range. Ultimately, it is up to **you** to plan your network infrastructure.

- To deploy resources into a virtual network or subnet, your user account must have permissions to the following actions in Azure role-based access control (Azure RBAC):

- "Microsoft.Network/virtualNetworks/join/action" on the virtual network resource.
- "Microsoft.Network/virtualNetworks/subnets/join/action" on the subnet resource.

For more information on Azure RBAC with networking, see the [Networking built-in roles](#)

Azure Container Registry

- Your Azure Container Registry must be Premium version. For more information on upgrading, see [Changing SKUs](#).
- If your Azure Container Registry uses a **private endpoint**, it must be in the same *virtual network* as the storage account and compute targets used for training or inference. If it uses a **service endpoint**, it must be in the same *virtual network* and *subnet* as the storage account and compute targets.
- Your Azure Machine Learning workspace must contain an [Azure Machine Learning compute cluster](#).

Limitations

Azure Storage Account

- If you plan to use Azure Machine Learning studio and the storage account is also in the VNet, there are extra validation requirements:
 - If the storage account uses a **service endpoint**, the workspace private endpoint and storage service endpoint must be in the same subnet of the VNet.
 - If the storage account uses a **private endpoint**, the workspace private endpoint and storage private endpoint must be in the same VNet. In this case, they can be in different subnets.

Azure Container Instances

When your Azure Machine Learning workspace is configured with a private endpoint, deploying to Azure Container Instances in a VNet is not supported. Instead, consider using a [Managed online endpoint with network isolation](#).

Azure Container Registry

When ACR is behind a virtual network, Azure Machine Learning can't use it to directly build Docker images. Instead, the compute cluster is used to build the images.

IMPORTANT

The compute cluster used to build Docker images needs to be able to access the package repositories that are used to train and deploy your models. You may need to add network security rules that allow access to public repos, [use private Python packages](#), or use [custom Docker images](#) that already include the packages.

WARNING

If your Azure Container Registry uses a private endpoint or service endpoint to communicate with the virtual network, you cannot use a managed identity with an Azure Machine Learning compute cluster.

Azure Monitor

WARNING

Azure Monitor supports using Azure Private Link to connect to a VNet. However, you must use the open Private Link mode in Azure Monitor. For more information, see [Private Link access modes: Private only vs. Open](#).

Required public internet access

Azure Machine Learning requires both inbound and outbound access to the public internet. The following tables provide an overview of what access is required and what it is for. The **protocol** for all items is **TCP**. For service tags that end in `.region`, replace `region` with the Azure region that contains your workspace. For example,

`Storage.westus`:

DIRECTION	PORTS	SERVICE TAG	PURPOSE
Inbound	29876-29877	BatchNodeManagement	Create, update, and delete of Azure Machine Learning compute instance and compute cluster. It isn't required if you use No Public IP option.
Inbound	44224	AzureMachineLearning	Create, update, and delete of Azure Machine Learning compute instance. It isn't required if you use No Public IP option.
Outbound	80, 443	AzureActiveDirectory	Authentication using Azure AD.
Outbound	443, 8787, 18881	AzureMachineLearning	Using Azure Machine Learning services.
Outbound	443	AzureResourceManager	Creation of Azure resources with Azure Machine Learning.
Outbound	443, 445 (*)	Storage.region	<p>Access data stored in the Azure Storage Account for compute cluster and compute instance. This outbound can be used to exfiltrate data. For more information, see Data exfiltration protection.</p> <p>(*) 445 is only required if you have a firewall between your virtual network for Azure ML and a private endpoint for your storage accounts.</p>
Outbound	443	AzureFrontDoor.FrontEnd * Not needed in Azure China.	<p>Global entry point for Azure Machine Learning studio. Store images and environments for AutoML.</p>
Outbound	443	MicrosoftContainerRegistry. region Note that this tag has a dependency on the AzureFrontDoor.FirstParty tag	<p>Access docker images provided by Microsoft. Setup of the Azure Machine Learning router for Azure Kubernetes Service.</p>

DIRECTION	PORTS	SERVICE TAG	PURPOSE
Outbound	443	AzureMonitor	Used to log monitoring and metrics to App Insights and Azure Monitor.
Outbound	443	Keyvault.region	Access the key vault for the Azure Batch service. Only needed if your workspace was created with the hbi_workspace flag enabled.

TIP

If you need the IP addresses instead of service tags, use one of the following options:

- Download a list from [Azure IP Ranges and Service Tags](#).
- Use the Azure CLI `az network list-service-tags` command.
- Use the Azure PowerShell `Get-AzNetworkServiceTag` command.

The IP addresses may change periodically.

IMPORTANT

When using a compute cluster that is configured for **no public IP address**, you must allow the following traffic:

- **Inbound** from source of **VirtualNetwork** and any port source, to destination of **VirtualNetwork**, and destination port of **29876, 29877**.
- **Inbound** from source **AzureLoadBalancer** and any port source to destination **VirtualNetwork** and port **44224** destination.

You may also need to allow **outbound** traffic to Visual Studio Code and non-Microsoft sites for the installation of packages required by your machine learning project. The following table lists commonly used repositories for machine learning:

HOST NAME	PURPOSE
<code>anaconda.com</code> <code>*.anaconda.com</code>	Used to install default packages.
<code>*.anaconda.org</code>	Used to get repo data.
<code>pypi.org</code>	Used to list dependencies from the default index, if any, and the index isn't overwritten by user settings. If the index is overwritten, you must also allow <code>*.pythonhosted.org</code> .
<code>cloud.r-project.org</code>	Used when installing CRAN packages for R development.
<code>*pytorch.org</code>	Used by some examples based on PyTorch.
<code>*.tensorflow.org</code>	Used by some examples based on Tensorflow.
<code>code.visualstudio.com</code>	Required to download and install VS Code desktop. This is not required for VS Code Web.

HOST NAME	PURPOSE
update.code.visualstudio.com *.vo.msecnd.net	Used to retrieve VS Code server bits that are installed on the compute instance through a setup script.
marketplace.visualstudio.com vscode.blob.core.windows.net *.gallerycdn.vsassets.io	Required to download and install VS Code extensions. These enable the remote connection to Compute Instances provided by the Azure ML extension for VS Code, see Connect to an Azure Machine Learning compute instance in Visual Studio Code for more information.
raw.githubusercontent.com/microsoft/vscode-tools-for-ai/master/azureml_remote_websocket_server/*	Used to retrieve websocket server bits, which are installed on the compute instance. The websocket server is used to transmit requests from Visual Studio Code client (desktop application) to Visual Studio Code server running on the compute instance.

When using Azure Kubernetes Service (AKS) with Azure Machine Learning, allow the following traffic to the AKS VNet:

- General inbound/outbound requirements for AKS as described in the [Restrict egress traffic in Azure Kubernetes Service](#) article.
- Outbound** to mcr.microsoft.com.
- When deploying a model to an AKS cluster, use the guidance in the [Deploy ML models to Azure Kubernetes Service](#) article.

For information on using a firewall solution, see [Use a firewall with Azure Machine Learning](#).

Secure the workspace with private endpoint

Azure Private Link lets you connect to your workspace using a private endpoint. The private endpoint is a set of private IP addresses within your virtual network. You can then limit access to your workspace to only occur over the private IP addresses. A private endpoint helps reduce the risk of data exfiltration.

For more information on configuring a private endpoint for your workspace, see [How to configure a private endpoint](#).

WARNING

Securing a workspace with private endpoints does not ensure end-to-end security by itself. You must follow the steps in the rest of this article, and the VNet series, to secure individual components of your solution. For example, if you use a private endpoint for the workspace, but your Azure Storage Account is not behind the VNet, traffic between the workspace and storage does not use the VNet for security.

Secure Azure storage accounts

Azure Machine Learning supports storage accounts configured to use either a private endpoint or service endpoint.

- [Private endpoint](#)
- [Service endpoint](#)

- In the Azure portal, select the Azure Storage Account.
- Use the information in [Use private endpoints for Azure Storage](#) to add private endpoints for the following storage resources:

- **Blob**
- **File**
- **Queue** - Only needed if you plan to use [ParallelRunStep](#) in an Azure Machine Learning pipeline.
- **Table** - Only needed if you plan to use [ParallelRunStep](#) in an Azure Machine Learning pipeline.

Create a private endpoint

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method (i)

Connect to an Azure resource in my directory.
 Connect to an Azure resource by resource ID or alias.

Subscription (i)

Resource type (i)

Resource (i)

Target sub-resource (i)
 blob
 table
 queue
 file
 web
 dfs

TIP

When configuring a storage account that is **not** the default storage, select the **Target subresource** type that corresponds to the storage account you want to add.

3. After creating the private endpoints for the storage resources, select the **Firewalls and virtual networks** tab under **Networking** for the storage account.
4. Select **Selected networks**, and then under **Resource instances**, select `Microsoft.MachineLearningServices/Workspace` as the **Resource type**. Select your workspace using **Instance name**. For more information, see [Trusted access based on system-assigned managed identity](#).

TIP

Alternatively, you can select **Allow Azure services on the trusted services list to access this storage account** to more broadly allow access from trusted services. For more information, see [Configure Azure Storage firewalls and virtual networks](#).

Firewalls and virtual networks

Private endpoint connections Custom domain

Save Discard Refresh

Allow access from

All networks Selected networks

Configure network security for your storage accounts. [Learn more](#)

Virtual networks

+ Add existing virtual network + Add new virtual network

Virtual Network	Subnet	Address range	Endpoint Status	Resource Group	Subscription
No network selected.					

Firewall

Add IP ranges to allow access from the internet or your on-premises networks. [Learn more](#).

Add your client IP address ("108.252.109.241") (i)

Address range

IP address or CIDR

Resource instances

Specify resource instances that will have access to your storage account based on their system-assigned managed identity.

Resource type	Instance name
Microsoft.MachineLearningServices/workspaces	docs-ws
Select a resource type	Select one or more instances

Exceptions

Allow Azure services on the trusted services list to access this storage account. (i)

Allow read access to storage logging from any network

Allow read access to storage metrics from any network

Network Routing

Determine how you would like to route your traffic as it travels from its source to an Azure endpoint. Microsoft routing is recommended for most customers.

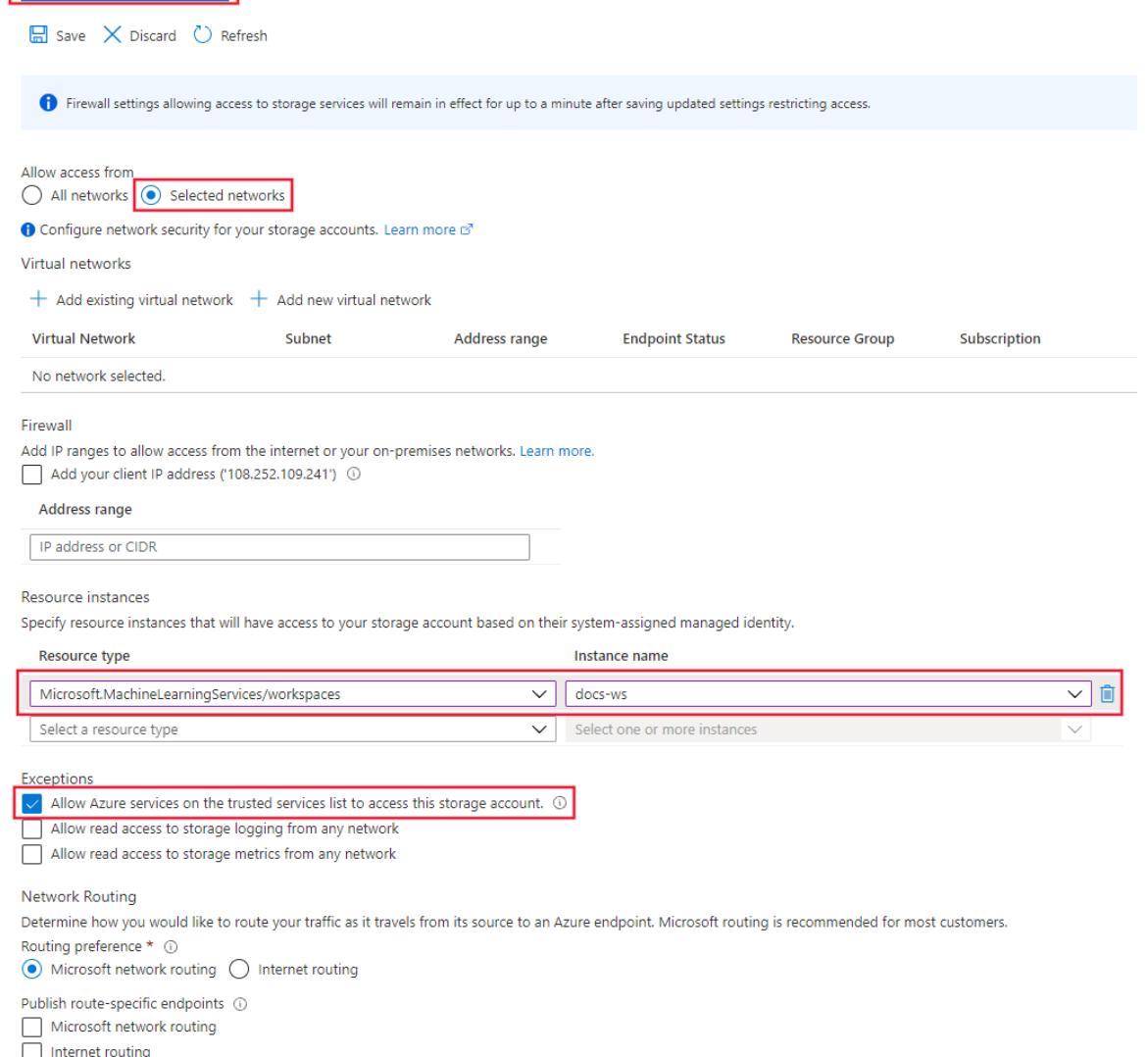
Routing preference (i)

Microsoft network routing Internet routing

Publish route-specific endpoints (i)

Microsoft network routing

Internet routing



5. Select **Save** to save the configuration.

TIP

When using a private endpoint, you can also disable public access. For more information, see [Disallow public read access](#).

Secure Azure Key Vault

Azure Machine Learning uses an associated Key Vault instance to store the following credentials:

- The associated storage account connection string
- Passwords to Azure Container Repository instances
- Connection strings to data stores

Azure key vault can be configured to use either a private endpoint or service endpoint. To use Azure Machine Learning experimentation capabilities with Azure Key Vault behind a virtual network, use the following steps:

TIP

Regardless of whether you use a private endpoint or service endpoint, the key vault must be in the same network as the private endpoint of the workspace.

- [Private endpoint](#)
- [Service endpoint](#)

For information on using a private endpoint with Azure Key Vault, see [Integrate Key Vault with Azure Private Link](#).

Enable Azure Container Registry (ACR)

TIP

If you did not use an existing Azure Container Registry when creating the workspace, one may not exist. By default, the workspace will not create an ACR instance until it needs one. To force the creation of one, train or deploy a model using your workspace before using the steps in this section.

Azure Container Registry can be configured to use a private endpoint. Use the following steps to configure your workspace to use ACR when it is in the virtual network:

1. Find the name of the Azure Container Registry for your workspace, using one of the following methods:

- [Azure CLI](#)
- [Python SDK](#)
- [Portal](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

If you've [installed the Machine Learning extension v2 for Azure CLI](#), you can use the `az ml workspace show` command to show the workspace information. The v1 extension does not return this information.

```
az ml workspace show -w yourworkspacename -g resourcegroupname --query 'container_registry'
```

This command returns a value similar to

```
"/subscriptions/{GUID}/resourceGroups/{resourcegroupname}/providers/Microsoft.ContainerRegistry/registries/{ACRname}"  
. The last part of the string is the name of the Azure Container Registry for the workspace.
```

2. Limit access to your virtual network using the steps in [Connect privately to an Azure Container Registry](#). When adding the virtual network, select the virtual network and subnet for your Azure Machine Learning resources.
3. Configure the ACR for the workspace to [Allow access by trusted services](#).
4. Create an Azure Machine Learning compute cluster. This cluster is used to build Docker images when ACR is behind a VNet. For more information, see [Create a compute cluster](#).
5. Use one of the following methods to configure the workspace to build Docker images using the compute cluster.

IMPORTANT

The following limitations apply When using a compute cluster for image builds:

- Only a CPU SKU is supported.
- If you use a compute cluster configured for no public IP address, you must provide some way for the cluster to access the public internet. Internet access is required when accessing images stored on the Microsoft Container Registry, packages installed on PyPi, Conda, etc. You need to configure User Defined Routing (UDR) to reach to a public IP to access the internet. For example, you can use the public IP of your firewall, or you can use [Virtual Network NAT](#) with a public IP. For more information, see [How to securely train in a VNet](#).

- [Azure CLI](#)
- [Python SDK](#)
- [Portal](#)

You can use the `az ml workspace update` command to set a build compute. The command is the same for both the v1 and v2 Azure CLI extensions for machine learning. In the following command, replace `myworkspace` with your workspace name, `myresourcegroup` with the resource group that contains the workspace, and `mycomputecluster` with the compute cluster name:

```
az ml workspace update --name myworkspace --resource-group myresourcegroup --image-build-compute mycomputecluster
```

TIP

When ACR is behind a VNet, you can also [disable public access](#) to it.

Datastores and datasets

The following table lists the services that you need to skip validation for:

SERVICE	SKIP VALIDATION REQUIRED?
Azure Blob storage	Yes
Azure File share	Yes
Azure Data Lake Store Gen1	No
Azure Data Lake Store Gen2	No
Azure SQL Database	Yes
PostgreSql	Yes

NOTE

Azure Data Lake Store Gen1 and Azure Data Lake Store Gen2 skip validation by default, so you don't have to do anything.

The following code sample creates a new Azure Blob datastore and sets `skip_validation=True`.

```
blob_datastore = Datastore.register_azure_blob_container(workspace=ws,  
                                                       datastore_name=blob_datastore_name,  
                                                       container_name=container_name,  
                                                       account_name=account_name,  
                                                       account_key=account_key,  
                                                       skip_validation=True ) // Set skip_validation to  
true
```

Use datasets

The syntax to skip dataset validation is similar for the following dataset types:

- Delimited file
- JSON
- Parquet
- SQL
- File

The following code creates a new JSON dataset and sets `validate=False`.

```
json_ds = Dataset.Tabular.from_json_lines_files(path=datastore_paths,  
                                                validate=False)
```

Securely connect to your workspace

To connect to a workspace that's secured behind a VNet, use one of the following methods:

- [Azure VPN gateway](#) - Connects on-premises networks to the VNet over a private connection. Connection is made over the public internet. There are two types of VPN gateways that you might use:
 - [Point-to-site](#): Each client computer uses a VPN client to connect to the VNet.
 - [Site-to-site](#): A VPN device connects the VNet to your on-premises network.
- [ExpressRoute](#) - Connects on-premises networks into the cloud over a private connection. Connection is made using a connectivity provider.
- [Azure Bastion](#) - In this scenario, you create an Azure Virtual Machine (sometimes called a jump box) inside the VNet. You then connect to the VM using Azure Bastion. Bastion allows you to connect to the VM using either an RDP or SSH session from your local web browser. You then use the jump box as your development environment. Since it is inside the VNet, it can directly access the workspace. For an example of using a jump box, see [Tutorial: Create a secure workspace](#).

IMPORTANT

When using a [VPN gateway](#) or [ExpressRoute](#), you will need to plan how name resolution works between your on-premises resources and those in the VNet. For more information, see [Use a custom DNS server](#).

If you have problems connecting to the workspace, see [Troubleshoot secure workspace connectivity](#).

Workspace diagnostics

You can run diagnostics on your workspace from Azure Machine Learning studio or the Python SDK. After diagnostics run, a list of any detected problems is returned. This list includes links to possible solutions. For more information, see [How to use workspace diagnostics](#).

Next steps

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)
- [Use a firewall](#)
- [Tutorial: Create a secure workspace](#)
- [Tutorial: Create a secure workspace using a template](#)
- [API platform network isolation](#)

Secure an Azure Machine Learning training environment with virtual networks

9/22/2022 • 21 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK `azure-ai-ml v2 (preview)`

In this article, you learn how to secure training environments with a virtual network in Azure Machine Learning. You'll learn how to secure training environments through the Azure Machine Learning **studio** and Python SDK **v2**.

TIP

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)
- [Use a firewall](#)

For a tutorial on creating a secure workspace, see [Tutorial: Create a secure workspace](#) or [Tutorial: Create a secure workspace using a template](#).

In this article you learn how to secure the following training compute resources in a virtual network:

- Azure Machine Learning compute cluster
- Azure Machine Learning compute instance
- Azure Databricks
- Virtual Machine
- HDInsight cluster

Prerequisites

- Read the [Network security overview](#) article to understand common virtual network scenarios and overall virtual network architecture.
- An existing virtual network and subnet to use with your compute resources.
- To deploy resources into a virtual network or subnet, your user account must have permissions to the following actions in Azure role-based access control (Azure RBAC):
 - "Microsoft.Network/virtualNetworks/*/read" on the virtual network resource. This permission isn't needed for Azure Resource Manager (ARM) template deployments.
 - "Microsoft.Network/virtualNetworks/subnet/join/action" on the subnet resource.

For more information on Azure RBAC with networking, see the [Networking built-in roles](#)

Azure Machine Learning compute cluster/instance

- Compute clusters and instances create the following resources. If they're unable to create these resources (for example, if there's a resource lock on the resource group) then creation, scale out, or scale in, may fail.

- IP address.
- Network Security Group (NSG).
- Load balancer.
- The virtual network must be in the same subscription as the Azure Machine Learning workspace.
- The subnet used for the compute instance or cluster must have enough unassigned IP addresses.
 - A compute cluster can dynamically scale. If there aren't enough unassigned IP addresses, the cluster will be partially allocated.
 - A compute instance only requires one IP address.
- To create a compute cluster or instance [without a public IP address](#) (a preview feature), your workspace must use a private endpoint to connect to the VNet. For more information, see [Configure a private endpoint for Azure Machine Learning workspace](#).
- If you plan to secure the virtual network by restricting traffic, see the [Required public internet access](#) section.
- The subnet used to deploy compute cluster/instance shouldn't be delegated to any other service. For example, it shouldn't be delegated to ACI.

Azure Databricks

- The virtual network must be in the same subscription and region as the Azure Machine Learning workspace.
- If the Azure Storage Account(s) for the workspace are also secured in a virtual network, they must be in the same virtual network as the Azure Databricks cluster.

Limitations

Azure Machine Learning compute cluster/instance

- If you put multiple compute instances or clusters in one virtual network, you may need to request a quota increase for one or more of your resources. The Machine Learning compute instance or cluster automatically allocates networking resources **in the resource group that contains the virtual network**. For each compute instance or cluster, the service allocates the following resources:
 - One network security group (NSG). This NSG contains the following rules, which are specific to compute cluster and compute instance:

IMPORTANT

Compute instance and compute cluster automatically create an NSG with the required rules.

If you have another NSG at the subnet level, the rules in the subnet level NSG mustn't conflict with the rules in the automatically created NSG.

To learn how the NSGs filter your network traffic, see [How network security groups filter network traffic](#).

- Allow inbound TCP traffic on ports 29876-29877 from the `BatchNodeManagement` service tag.
- Allow inbound TCP traffic on port 44224 from the `AzureMachineLearning` service tag.

The following screenshot shows an example of these rules:

<input type="text" value="Filter by name"/> Port == all Protocol == all Source == all Destination == all Action == all							
Priority ↑↓	Name ↑↓	Port ↑↓	Protocol ↑↓	Source ↑↓	Destination ↑↓	Action ↑↓	
▽ Inbound Security Rules							
120	BatchServiceRule	29876-29877	tcp	BatchNodeManagement.Sou...	Any	Allow	
149	⚠ NodeAgentRule-DenyAll	29876-29877	tcp	Any	Any	Deny	
160	JupyterServerPort	44224	tcp	AzureMachineLearning	Any	Allow	
170	SSH	22	tcp	Internet	Any	Deny	
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow	
65001	AllowAzureLoadBalancerInBo...	Any	Any	AzureLoadBalancer	Any	Allow	
65500	DenyAllInBound	Any	Any	Any	Any	Deny	
▽ Outbound Security Rules							
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow	
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow	
65500	DenyAllOutBound	Any	Any	Any	Any	Deny	

TIP

If your compute cluster or instance does not use a public IP address (a preview feature), these inbound NSG rules are not required.

- For compute cluster or instance, it's now possible to remove the public IP address (a preview feature). If you have Azure Policy assignments prohibiting Public IP creation, then deployment of the compute cluster or instance will succeed.

- One load balancer

For compute clusters, these resources are deleted every time the cluster scales down to 0 nodes and created when scaling up.

For a compute instance, these resources are kept until the instance is deleted. Stopping the instance doesn't remove the resources.

IMPORTANT

These resources are limited by the subscription's [resource quotas](#). If the virtual network resource group is locked then deletion of compute cluster/instance will fail. Load balancer cannot be deleted until the compute cluster/instance is deleted. Also please ensure there is no Azure Policy assignment which prohibits creation of network security groups.

- If you create a compute instance and plan to use the no public IP address configuration, your Azure Machine Learning workspace's managed identity must be assigned the **Reader** role for the virtual network that contains the workspace. For more information on assigning roles, see [Steps to assign an Azure role](#).
- If you have configured Azure Container Registry for your workspace behind the virtual network, you must use a compute cluster to build Docker images. If you use a compute cluster configured for no public IP address, you must provide some method for the cluster to access the public internet. Internet access is required when accessing images stored on the Microsoft Container Registry, packages installed on Pypi, Conda, etc. For more information, see [Enable Azure Container Registry](#).
- If the Azure Storage Accounts for the workspace are also in the virtual network, use the following guidance on subnet limitations:
 - If you plan to use Azure Machine Learning studio to visualize data or use designer, the storage account must be **in the same subnet as the compute instance or cluster**.
 - If you plan to use the **SDK**, the storage account can be in a different subnet.

NOTE

Adding a resource instance for your workspace or selecting the checkbox for "Allow trusted Microsoft services to access this account" is not sufficient to allow communication from the compute.

- When your workspace uses a private endpoint, the compute instance can only be accessed from inside the virtual network. If you use a custom DNS or hosts file, add an entry for `<instance-name>.<region>.instances.azureml.ms`. Map this entry to the private IP address of the workspace private endpoint. For more information, see the [custom DNS](#) article.
- Virtual network service endpoint policies don't work for compute cluster/instance system storage accounts.
- If storage and compute instance are in different regions, you may see intermittent timeouts.
- If the Azure Container Registry for your workspace uses a private endpoint to connect to the virtual network, you can't use a managed identity for the compute instance. To use a managed identity with the compute instance, don't put the container registry in the VNet.
- If you want to use Jupyter Notebooks on a compute instance:
 - Don't disable websocket communication. Make sure your network allows websocket communication to `*.instances.azureml.net` and `*.instances.azureml.ms`.
 - Make sure that your notebook is running on a compute resource behind the same virtual network and subnet as your data. When creating the compute instance, use **Advanced settings > Configure virtual network** to select the network and subnet.
- Compute clusters** can be created in a different region than your workspace. This functionality is in [preview](#), and is only available for **compute clusters**, not compute instances. When using a different region for the cluster, the following limitations apply:
 - If your workspace associated resources, such as storage, are in a different virtual network than the cluster, set up global virtual network peering between the networks. For more information, see [Virtual network peering](#).
 - You may see increased network latency and data transfer costs. The latency and costs can occur when creating the cluster, and when running jobs on it.

Guidance such as using NSG rules, user-defined routes, and input/output requirements, apply as normal when using a different region than the workspace.

WARNING

If you are using a **private endpoint-enabled workspace**, creating the cluster in a different region is not supported.

- An Azure Machine Learning workspace requires outbound access to `storage.<region>/*.blob.core.windows.net` on the public internet, where `<region>` is the Azure region of the workspace. This outbound access is required by Azure Machine Learning compute cluster and compute instance. Both are based on Azure Batch, and need to access a storage account provided by Azure Batch on the public network.

By using a Service Endpoint Policy, you can mitigate this vulnerability. This feature is currently in preview. For more information, see the [Azure Machine Learning data exfiltration prevention](#) article.

Azure Databricks

- In addition to the **databricks-private** and **databricks-public** subnets used by Azure Databricks, the

default subnet created for the virtual network is also required.

- Azure Databricks doesn't use a private endpoint to communicate with the virtual network.

For more information on using Azure Databricks in a virtual network, see [Deploy Azure Databricks in your Azure Virtual Network](#).

Azure HDInsight or virtual machine

- Azure Machine Learning supports only virtual machines that are running Ubuntu.

Required public internet access

Azure Machine Learning requires both inbound and outbound access to the public internet. The following tables provide an overview of what access is required and what it is for. The **protocol** for all items is **TCP**. For service tags that end in `.region`, replace `region` with the Azure region that contains your workspace. For example,

`Storage.westus`:

DIRECTION	PORTS	SERVICE TAG	PURPOSE
Inbound	29876-29877	BatchNodeManagement	Create, update, and delete of Azure Machine Learning compute instance and compute cluster. It isn't required if you use No Public IP option.
Inbound	44224	AzureMachineLearning	Create, update, and delete of Azure Machine Learning compute instance. It isn't required if you use No Public IP option.
Outbound	80, 443	AzureActiveDirectory	Authentication using Azure AD.
Outbound	443, 8787, 18881	AzureMachineLearning	Using Azure Machine Learning services.
Outbound	443	AzureResourceManager	Creation of Azure resources with Azure Machine Learning.
Outbound	443, 445 (*)	Storage.region	Access data stored in the Azure Storage Account for compute cluster and compute instance. This outbound can be used to exfiltrate data. For more information, see Data exfiltration protection . (*) 445 is only required if you have a firewall between your virtual network for Azure ML and a private endpoint for your storage accounts.

DIRECTION	PORTS	SERVICE TAG	PURPOSE
Outbound	443	AzureFrontDoor.FrontEnd * Not needed in Azure China.	Global entry point for Azure Machine Learning studio . Store images and environments for AutoML.
Outbound	443	MicrosoftContainerRegistry.region Note that this tag has a dependency on the AzureFrontDoor.FirstParty tag	Access docker images provided by Microsoft. Setup of the Azure Machine Learning router for Azure Kubernetes Service.
Outbound	443	AzureMonitor	Used to log monitoring and metrics to App Insights and Azure Monitor.
Outbound	443	Keyvault.region	Access the key vault for the Azure Batch service. Only needed if your workspace was created with the hbi_workspace flag enabled.

TIP

If you need the IP addresses instead of service tags, use one of the following options:

- Download a list from [Azure IP Ranges and Service Tags](#).
- Use the Azure CLI `az network list-service-tags` command.
- Use the Azure PowerShell `Get-AzNetworkServiceTag` command.

The IP addresses may change periodically.

IMPORTANT

When using a compute cluster that is configured for **no public IP address**, you must allow the following traffic:

- **Inbound** from source of **VirtualNetwork** and any port source, to destination of **VirtualNetwork**, and destination port of **29876, 29877**.
- **Inbound** from source **AzureLoadBalancer** and any port source to destination **VirtualNetwork** and port **44224** destination.

You may also need to allow **outbound** traffic to Visual Studio Code and non-Microsoft sites for the installation of packages required by your machine learning project. The following table lists commonly used repositories for machine learning:

HOST NAME	PURPOSE
anaconda.com *.anaconda.com	Used to install default packages.
*.anaconda.org	Used to get repo data.

HOST NAME	PURPOSE
pypi.org	Used to list dependencies from the default index, if any, and the index isn't overwritten by user settings. If the index is overwritten, you must also allow *.pythonhosted.org.
cloud.r-project.org	Used when installing CRAN packages for R development.
*pytorch.org	Used by some examples based on PyTorch.
*.tensorflow.org	Used by some examples based on Tensorflow.
code.visualstudio.com	Required to download and install VS Code desktop. This is not required for VS Code Web.
update.code.visualstudio.com *.vo.msecnd.net	Used to retrieve VS Code server bits that are installed on the compute instance through a setup script.
marketplace.visualstudio.com vscode.blob.core.windows.net *.gallerycdn.vsassets.io	Required to download and install VS Code extensions. These enable the remote connection to Compute Instances provided by the Azure ML extension for VS Code, see Connect to an Azure Machine Learning compute instance in Visual Studio Code for more information.
raw.githubusercontent.com/microsoft/vscode-tools-for-ai/master/azureml_remote_websocket_server/*	Used to retrieve websocket server bits, which are installed on the compute instance. The websocket server is used to transmit requests from Visual Studio Code client (desktop application) to Visual Studio Code server running on the compute instance.

When using Azure Kubernetes Service (AKS) with Azure Machine Learning, allow the following traffic to the AKS VNet:

- General inbound/outbound requirements for AKS as described in the [Restrict egress traffic in Azure Kubernetes Service](#) article.
- Outbound** to mcr.microsoft.com.
- When deploying a model to an AKS cluster, use the guidance in the [Deploy ML models to Azure Kubernetes Service](#) article.

For information on using a firewall solution, see [Use a firewall with Azure Machine Learning](#).

Compute clusters

Use the following steps to create a compute cluster in the Azure Machine Learning studio:

- Sign in to [Azure Machine Learning studio](#), and then select your subscription and workspace.
- Select **Compute** on the left, **Compute clusters** from the center, and then select + **New**.

☰

↶ Microsoft

+ New

HomeAspx

Author

- Notebooks
- Automated ML
- Designer

Assets

- Data
- Jobs
- Components
- Pipelines
- Environments
- Models
- Endpoints

Manage

- Compute
- Datastores

Home > Compute

Compute

Compute instances

Compute clusters

Inference clusters

Attached computes



Scale your compute cluster from a single node to a multi node workload

Create a single or multi node compute cluster for your training, batch inferencing or reinforcement learning workloads. [Learn more](#)

+ New

[View Azure Machine Learning tutorials](#)

3. In the **Create compute cluster** dialog, select the VM size and configuration you need and then select **Next**.

Create compute cluster [\(1\)](#)

X

Virtual Machine

▼

Select virtual machine

Select the virtual machine size you would like to use for your compute cluster.

Location *

South Central US

▼

Virtual machine priority [\(1\)](#)

Dedicated Low priority

Virtual machine type [\(1\)](#)

CPU GPU

Virtual machine size [\(1\)](#)

Select from recommended options Select from all options

Total available quota: 100 cores [\(1\)](#)

Name ↑	Category	Workload types	Av... (1)
<input type="radio"/> Standard_DS2_v2 2 cores, 7GB RAM, 14GB storage	General purpose	Development on Notebooks (or other IDE) and light weight testing	100 c...
<input checked="" type="radio"/> Standard_DS3_v2 4 cores, 14GB RAM, 28GB storage	General purpose	Classical ML model training, AutoML runs, pipeline runs (default compute)	100 c...
<input type="radio"/> Standard_DS12_v2 4 cores, 28GB RAM, 56GB storage	Memory optimi...	Training on large datasets (>1GB) parallel run steps, batch inferencing	100 c...
<input type="radio"/> Standard_F4s_v2 4 cores, 8GB RAM, 32GB storage	Compute optimi...	Real-time inferencing and other latency-sensitive tasks	100 c...

Back

Next

Cancel

4. From the **Configure Settings** section, set the **Compute name**, **Virtual network**, and **Subnet**.

Create compute cluster [\(i\)](#)

X

Settings

▼

Configure Settings

Configure compute cluster settings for your selected virtual machine size.

Name	Category	Cores	Available quota	RAM	Storage	Cost/Node
Standard_DS3_v2	General purpose	4	100 cores	14 GB	28 GB	

Compute name * [\(i\)](#)



Minimum number of nodes * [\(i\)](#)



Maximum number of nodes * [\(i\)](#)



Idle seconds before scale down * [\(i\)](#)

Enable SSH access [\(i\)](#)

Advanced settings

Enable virtual network [\(i\)](#)

Virtual network

docs-ml-vnet (docs-ml-rg) [\(i\)](#)

[\(i\)](#) Your workspace is linked to a virtual network using a private endpoint connection. In order to communicate properly with the workspace, your compute resource must be provisioned in the same virtual network.

Subnet * [\(i\)](#)

Assign a managed identity [\(i\)](#)

Back

Create

Download a template for automation

Cancel

TIP

If your workspace uses a private endpoint to connect to the virtual network, the **Virtual network** selection field is greyed out.

5. Select **Create** to create the compute cluster.

When the creation process finishes, you train your model by using the cluster in an experiment.

NOTE

You may choose to use [low-priority VMs](#) to run some or all of your workloads. See how to [create a low-priority VM](#).

No public IP for compute clusters (preview)

When you enable **No public IP**, your compute cluster doesn't use a public IP for communication with any dependencies. Instead, it communicates solely within the virtual network using Azure Private Link ecosystem and service/private endpoints, eliminating the need for a public IP entirely. No public IP removes access and discoverability of compute cluster nodes from the internet thus eliminating a significant threat vector. **No public IP** clusters help comply with no public IP policies many enterprises have.

A compute cluster with **No public IP** enabled has **no inbound communication requirements** from public internet. Specifically, neither inbound NSG rule (`BatchNodeManagement`, `AzureMachineLearning`) is required. You still need to allow inbound from source of **VirtualNetwork** and any port source, to destination of **VirtualNetwork**, and destination port of **29876**, **29877** and inbound from source **AzureLoadBalancer** and any port source to destination **VirtualNetwork** and port **44224** destination.

WARNING

By default, you do not have public internet access from No Public IP Compute Cluster. This prevents *outbound* access to required resources such as Azure Active Directory, Azure Resource Manager, Microsoft Container Registry, and other outbound resources as listed in the [Required public internet access](#) section. Or to non-Microsoft resources such as Pypi or Conda repositories. To resolve this problem, you need to configure User Defined Routing (UDR) to reach to a public IP to access the internet. For example, you can use a public IP of your firewall, or you can use [Virtual Network NAT](#) with a public IP.

No public IP clusters are dependent on [Azure Private Link](#) for Azure Machine Learning workspace. A compute cluster with **No public IP** also requires you to disable private endpoint network policies and private link service network policies. These requirements come from Azure private link service and private endpoints and aren't Azure Machine Learning specific. Follow instruction from [Disable network policies for Private Link service](#) to set the parameters `disable-private-endpoint-network-policies` and `disable-private-link-service-network-policies` on the virtual network subnet.

For **outbound connections** to work, you need to set up an egress firewall such as Azure firewall with user defined routes. For instance, you can use a firewall set up with [inbound/outbound configuration](#) and route traffic there by defining a route table on the subnet in which the compute cluster is deployed. The route table entry can set up the next hop of the private IP address of the firewall with the address prefix of 0.0.0.0/0.

You can use a service endpoint or private endpoint for your Azure container registry and Azure storage in the subnet in which cluster is deployed.

To create a no public IP address compute cluster (a preview feature) in studio, set **No public IP** checkbox in the virtual network section. You can also create no public IP compute cluster through an ARM template. In the ARM template set `enableNodePublicIP` parameter to false.

NOTE

Support for compute instances without public IP addresses is currently available and in public preview for the following regions: France Central, East Asia, West Central US, South Central US, West US 2, East US, East US 2, North Europe, West Europe, Central US, North Central US, West US, Australia East, Japan East, Japan West.

Support for compute clusters without public IP addresses is currently available and in public preview for the following regions: France Central, East Asia, West Central US, South Central US, West US 2, East US, North Europe, East US 2, Central US, West Europe, North Central US, West US, Australia East, Japan East, Japan West.

Troubleshooting

- If you get this error message during creation of cluster

`The specified subnet has PrivateLinkServiceNetworkPolicies or PrivateEndpointNetworkEndpoints enabled,`

follow the instructions from [Disable network policies for Private Link service](#) and [Disable network policies for Private Endpoint](#).

- If job execution fails with connection issues to ACR or Azure Storage, verify that customer has added ACR and Azure Storage service endpoint/private endpoints to subnet and ACR/Azure Storage allows the access from the subnet.
- To ensure that you've created a no public IP cluster, in Studio when looking at cluster details you'll see **No Public IP** property is set to **true** under resource properties.

Compute instance

For steps on how to create a compute instance deployed in a virtual network, see [Create and manage an Azure Machine Learning compute instance](#).

No public IP for compute instances (preview)

When you enable **No public IP**, your compute instance doesn't use a public IP for communication with any dependencies. Instead, it communicates solely within the virtual network using Azure Private Link ecosystem and service/private endpoints, eliminating the need for a public IP entirely. No public IP removes access and discoverability of compute instance node from the internet thus eliminating a significant threat vector. Compute instances will also do packet filtering to reject any traffic from outside virtual network. **No public IP** instances are dependent on [Azure Private Link](#) for Azure Machine Learning workspace.

WARNING

By default, you do not have public internet access from No Public IP Compute Instance. You need to configure User Defined Routing (UDR) to reach to a public IP to access the internet. For example, you can use a public IP of your firewall, or you can use [Virtual Network NAT](#) with a public IP. Specifically, you need access to Azure Active Directory, Azure Resource Manager, Microsoft Container Registry, and other outbound resources as listed in the [Required public internet access](#) section. You may also need outbound access to non-Microsoft resources such as Pypi or Conda repositories.

For **outbound connections** to work, you need to set up an egress firewall such as Azure firewall with user defined routes. For instance, you can use a firewall set up with [inbound/outbound configuration](#) and route traffic there by defining a route table on the subnet in which the compute instance is deployed. The route table entry can set up the next hop of the private IP address of the firewall with the address prefix of 0.0.0.0/0.

A compute instance with **No public IP** enabled has **no inbound communication requirements** from public internet. Specifically, neither inbound NSG rule (`BatchNodeManagement`, `AzureMachineLearning`) is required. You still need to allow inbound from source of `VirtualNetwork`, any port source, destination of `VirtualNetwork`, and destination port of **29876, 29877, 44224**.

A compute instance with **No public IP** also requires you to disable private endpoint network policies and private link service network policies. These requirements come from Azure private link service and private endpoints and aren't Azure Machine Learning specific. Follow instruction from [Disable network policies for Private Link service source IP](#) to set the parameters `disable-private-endpoint-network-policies` and `disable-private-link-service-network-policies` on the virtual network subnet.

To create a no public IP address compute instance (a preview feature) in studio, set **No public IP** checkbox in the virtual network section. You can also create no public IP compute instance through an ARM template. In the ARM template set `enableNodePublicIP` parameter to false.

Next steps:

- [Use custom DNS](#)
- [Use a firewall](#)

NOTE

Support for compute instances without public IP addresses is currently available and in public preview for the following regions: France Central, East Asia, West Central US, South Central US, West US 2, East US, East US 2, North Europe, West Europe, Central US, North Central US, West US, Australia East, Japan East, Japan West.

Support for compute clusters without public IP addresses is currently available and in public preview for the following regions: France Central, East Asia, West Central US, South Central US, West US 2, East US, North Europe, East US 2, Central US, West Europe, North Central US, West US, Australia East, Japan East, Japan West.

Inbound traffic

When using Azure Machine Learning **compute instance** (with a public IP) or **compute cluster**, allow inbound traffic from Azure Batch management and Azure Machine Learning services. Compute instance with no public IP (preview) does not require this inbound communication. A Network Security Group allowing this traffic is dynamically created for you, however you may need to also create user-defined routes (UDR) if you have a firewall. When creating a UDR for this traffic, you can use either **IP Addresses** or **service tags** to route the traffic.

IMPORTANT

Using service tags with user-defined routes is now GA. For more information, see [Virtual Network routing](#).

TIP

While a compute instance without a public IP (a preview feature) does not need a UDR for this inbound traffic, you will still need these UDRs if you also use a compute cluster or a compute instance with a public IP.

- [IP Address routes](#)
- [Service tag routes](#)

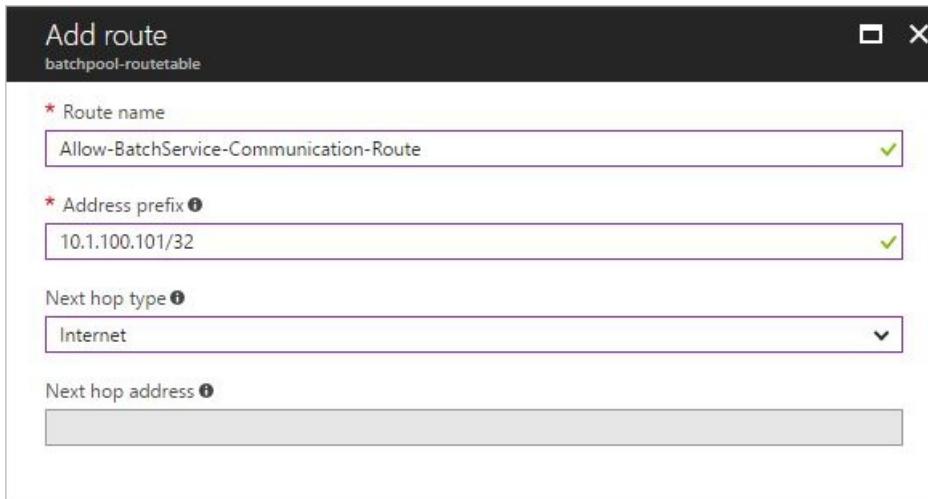
For the Azure Machine Learning service, you must add the IP address of both the **primary** and **secondary** regions. To find the secondary region, see the [Cross-region replication in Azure](#). For example, if your Azure Machine Learning service is in East US 2, the secondary region is Central US.

To get a list of IP addresses of the Batch service and Azure Machine Learning service, download the [Azure IP Ranges and Service Tags](#) and search the file for `BatchNodeManagement.<region>` and `AzureMachineLearning.<region>`, where `<region>` is your Azure region.

IMPORTANT

The IP addresses may change over time.

When creating the UDR, set the **Next hop type** to **Internet**. This means the inbound communication from Azure skips your firewall to access the load balancers with public IPs of Compute Instance and Compute Cluster. UDR is required because Compute Instance and Compute Cluster will get random public IPs at creation, and you cannot know the public IPs before creation to register them on your firewall to allow the inbound from Azure to specific IPs for Compute Instance and Compute Cluster. The following image shows an example IP address based UDR in the Azure portal:



For information on configuring UDR, see [Route network traffic with a routing table](#).

For more information on input and output traffic requirements for Azure Machine Learning, see [Use a workspace behind a firewall](#).

Azure Databricks

For specific information on using Azure Databricks with a virtual network, see [Deploy Azure Databricks in your Azure Virtual Network](#).

Virtual machine or HDInsight cluster

In this section, you learn how to use a virtual machine or Azure HDInsight cluster in a virtual network with your workspace.

Create the VM or HDInsight cluster

Create a VM or HDInsight cluster by using the Azure portal or the Azure CLI, and put the cluster in an Azure virtual network. For more information, see the following articles:

- [Create and manage Azure virtual networks for Linux VMs](#)
- [Extend HDInsight using an Azure virtual network](#)

Configure network ports

Allow Azure Machine Learning to communicate with the SSH port on the VM or cluster, configure a source entry for the network security group. The SSH port is usually port 22. To allow traffic from this source, do the following actions:

1. In the **Source** drop-down list, select **Service Tag**.
2. In the **Source service tag** drop-down list, select **AzureMachineLearning**.

* **Source**

Service Tag
Source service tag
AzureMachineLearning

* **Source port ranges**

*

* **Destination**

Any

* **Destination port ranges**

22

* **Protocol**

Any	TCP	UDP
-----	-----	-----

* **Action**

Allow	Deny
-------	------

* **Priority**

1030

* **Name**

Port_22_AML

Description

--

3. In the **Source port ranges** drop-down list, select *.
4. In the **Destination** drop-down list, select Any.
5. In the **Destination port ranges** drop-down list, select 22.
6. Under **Protocol**, select Any.
7. Under **Action**, select Allow.

Keep the default outbound rules for the network security group. For more information, see the default security rules in [Security groups](#).

If you don't want to use the default outbound rules and you do want to limit the outbound access of your virtual network, see the [required public internet access](#) section.

Attach the VM or HDInsight cluster

Attach the VM or HDInsight cluster to your Azure Machine Learning workspace. For more information, see [Manage compute resources for model training and deployment in studio](#).

Next steps

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)
- [Use a firewall](#)

Use Azure Machine Learning studio in an Azure virtual network

9/22/2022 • 7 minutes to read • [Edit Online](#)

In this article, you learn how to use Azure Machine Learning studio in a virtual network. The studio includes features like AutoML, the designer, and data labeling.

Some of the studio's features are disabled by default in a virtual network. To re-enable these features, you must enable managed identity for storage accounts you intend to use in the studio.

The following operations are disabled by default in a virtual network:

- Preview data in the studio.
- Visualize data in the designer.
- Deploy a model in the designer.
- Submit an AutoML experiment.
- Start a labeling project.

The studio supports reading data from the following datastore types in a virtual network:

- Azure Storage Account (blob & file)
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure SQL Database

In this article, you learn how to:

- Give the studio access to data stored inside of a virtual network.
- Access the studio from a resource inside of a virtual network.
- Understand how the studio impacts storage security.

TIP

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Use custom DNS](#)
- [Use a firewall](#)

For a tutorial on creating a secure workspace, see [Tutorial: Create a secure workspace](#) or [Tutorial: Create a secure workspace using a template](#).

Prerequisites

- Read the [Network security overview](#) to understand common virtual network scenarios and architecture.
- A pre-existing virtual network and subnet to use.

- An existing [Azure Machine Learning workspace with a private endpoint](#).
- An existing [Azure storage account added your virtual network](#).

Limitations

Azure Storage Account

- When the storage account is in the VNet, there are extra validation requirements when using studio:
 - If the storage account uses a **service endpoint**, the workspace private endpoint and storage service endpoint must be in the same subnet of the VNet.
 - If the storage account uses a **private endpoint**, the workspace private endpoint and storage private endpoint must be in the same VNet. In this case, they can be in different subnets.

Designer sample pipeline

There's a known issue where user cannot run sample pipeline in Designer homepage. This is the sample dataset used in the sample pipeline is Azure Global dataset, and it cannot satisfy all virtual network environment.

To resolve this issue, you can use a public workspace to run sample pipeline to get to know how to use the designer and then replace the sample dataset with your own dataset in the workspace within virtual network.

Datastore: Azure Storage Account

Use the following steps to enable access to data stored in Azure Blob and File storage:

TIP

The first step is not required for the default storage account for the workspace. All other steps are required for *any* storage account behind the VNet and used by the workspace, including the default storage account.

1. **If the storage account is the *default storage* for your workspace, skip this step.** If it is not the default, **Grant the workspace managed identity the 'Storage Blob Data Reader' role** for the Azure storage account so that it can read data from blob storage.

For more information, see the [Blob Data Reader](#) built-in role.

2. **Grant the workspace managed identity the 'Reader' role for storage private endpoints.** If your storage service uses a **private endpoint**, grant the workspace's managed identity **Reader** access to the private endpoint. The workspace's managed identity in Azure AD has the same name as your Azure Machine Learning workspace.

TIP

Your storage account may have multiple private endpoints. For example, one storage account may have separate private endpoint for blob, file, and dfs (Azure Data Lake Storage Gen2). Add the managed identity to all these endpoints.

For more information, see the [Reader](#) built-in role.

3. **Enable managed identity authentication for default storage accounts.** Each Azure Machine Learning workspace has two default storage accounts, a default blob storage account and a default file store account, which are defined when you create your workspace. You can also set new defaults in the **Datastore** management page.

The following table describes why managed identity authentication is used for your workspace default storage accounts.

STORAGE ACCOUNT	NOTES
Workspace default blob storage	<p>Stores model assets from the designer. Enable managed identity authentication on this storage account to deploy models in the designer. If managed identity authentication is disabled, the user's identity is used to access data stored in the blob.</p> <p>You can visualize and run a designer pipeline if it uses a non-default datastore that has been configured to use managed identity. However, if you try to deploy a trained model without managed identity enabled on the default datastore, deployment will fail regardless of any other datastores in use.</p>
Workspace default file store	Stores AutoML experiment assets. Enable managed identity authentication on this storage account to submit AutoML experiments.

4. **Configure datastores to use managed identity authentication.** After you add an Azure storage account to your virtual network with either a [service endpoint](#) or [private endpoint](#), you must configure your datastore to use [managed identity](#) authentication. Doing so lets the studio access data in your storage account.

Azure Machine Learning uses [datastore](#) to connect to storage accounts. When creating a new datastore, use the following steps to configure a datastore to use managed identity authentication:

- In the studio, select **Datastores**.
- To update an existing datastore, select the datastore and select **Update credentials**.

To create a new datastore, select **+ New datastore**.

- c. In the datastore settings, select Yes for Use workspace managed identity for data preview and profiling in Azure Machine Learning studio.

The screenshot shows the 'Update datastore credentials' dialog for a 'mydatastore'. The 'Authentication type' dropdown is set to 'Account key'. The 'Account key' field contains a redacted value. The 'Use workspace managed identity for data preview and profiling in Azure Machine Learning studio' checkbox is selected ('Yes'). The 'Save' button at the bottom is highlighted with a red box.

- d. In the **Networking** settings for the **Azure Storage Account**, add the **Microsoft.MachineLearningService/workspaces** **Resource type**, and set the **Instance name** to the workspace.

These steps add the workspace's managed identity as a **Reader** to the new storage service using Azure RBAC. **Reader** access allows the workspace to view the resource, but not make changes.

Datastore: Azure Data Lake Storage Gen1

When using Azure Data Lake Storage Gen1 as a datastore, you can only use POSIX-style access control lists. You can assign the workspace's managed identity access to resources just like any other security principal. For more information, see [Access control in Azure Data Lake Storage Gen1](#).

Datastore: Azure Data Lake Storage Gen2

When using Azure Data Lake Storage Gen2 as a datastore, you can use both Azure RBAC and POSIX-style access control lists (ACLs) to control data access inside of a virtual network.

To use Azure RBAC, follow the steps in the [Datastore: Azure Storage Account](#) section of this article. Data Lake Storage Gen2 is based on Azure Storage, so the same steps apply when using Azure RBAC.

To use ACLs, the workspace's managed identity can be assigned access just like any other security principal. For more information, see [Access control lists on files and directories](#).

Datastore: Azure SQL Database

To access data stored in an Azure SQL Database with a managed identity, you must create a SQL contained user that maps to the managed identity. For more information on creating a user from an external provider, see [Create contained users mapped to Azure AD identities](#).

After you create a SQL contained user, grant permissions to it by using the [GRANT T-SQL command](#).

Intermediate component output

When using the Azure Machine Learning designer intermediate component output, you can specify the output

location for any component in the designer. Use this to store intermediate datasets in separate location for security, logging, or auditing purposes. To specify output, use the following steps:

1. Select the component whose output you'd like to specify.
2. In the component settings pane that appears to the right, select **Output settings**.
3. Specify the datastore you want to use for each component output.

Make sure that you have access to the intermediate storage accounts in your virtual network. Otherwise, the pipeline will fail.

[Enable managed identity authentication](#) for intermediate storage accounts to visualize output data.

Access the studio from a resource inside the VNet

If you are accessing the studio from a resource inside of a virtual network (for example, a compute instance or virtual machine), you must allow outbound traffic from the virtual network to the studio.

For example, if you are using network security groups (NSG) to restrict outbound traffic, add a rule to a **service tag** destination of **AzureFrontDoor.Frontend**.

Firewall settings

Some storage services, such as Azure Storage Account, have firewall settings that apply to the public endpoint for that specific service instance. Usually this setting allows you to allow/disallow access from specific IP addresses from the public internet. **This is not supported** when using Azure Machine Learning studio. It is supported when using the Azure Machine Learning SDK or CLI.

TIP

Azure Machine Learning studio is supported when using the Azure Firewall service. For more information, see [Use your workspace behind a firewall](#).

Next steps

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Use custom DNS](#)
- [Use a firewall](#)

Configure a private endpoint for an Azure Machine Learning workspace

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

In this document, you learn how to configure a private endpoint for your Azure Machine Learning workspace. For information on creating a virtual network for Azure Machine Learning, see [Virtual network isolation and privacy overview](#).

Azure Private Link enables you to connect to your workspace using a private endpoint. The private endpoint is a set of private IP addresses within your virtual network. You can then limit access to your workspace to only occur over the private IP addresses. A private endpoint helps reduce the risk of data exfiltration. To learn more about private endpoints, see the [Azure Private Link](#) article.

WARNING

Securing a workspace with private endpoints does not ensure end-to-end security by itself. You must secure all of the individual components of your solution. For example, if you use a private endpoint for the workspace, but your Azure Storage Account is not behind the VNet, traffic between the workspace and storage does not use the VNet for security.

For more information on securing resources used by Azure Machine Learning, see the following articles:

- [Virtual network isolation and privacy overview](#).
- [Secure workspace resources](#).
- [Secure training environments](#).
- [Secure the inference environment](#).
- [Use Azure Machine Learning studio in a VNet](#).
- [API platform network isolation](#).

Prerequisites

- You must have an existing virtual network to create the private endpoint in.

IMPORTANT

We do not recommend using the 172.17.0.0/16 IP address range for your VNet. This is the default subnet range used by the Docker bridge network. Other ranges may also conflict depending on what you want to connect to the virtual network. For example, if you plan to connect your on-premises network to the VNet, and your on-premises network also uses the 172.16.0.0/16 range. Ultimately, it is up to **you** to plan your network infrastructure.

- [Disable network policies for private endpoints](#) before adding the private endpoint.

Limitations

- If you enable public access for a workspace secured with private endpoint and use Azure Machine Learning studio over the public internet, some features such as the designer may fail to access your data. This problem happens when the data is stored on a service that is secured behind the VNet. For example, an Azure Storage Account.

- You may encounter problems trying to access the private endpoint for your workspace if you're using Mozilla Firefox. This problem may be related to DNS over HTTPS in Mozilla Firefox. We recommend using Microsoft Edge or Google Chrome as a workaround.
- Using a private endpoint doesn't affect Azure control plane (management operations) such as deleting the workspace or managing compute resources. For example, creating, updating, or deleting a compute target. These operations are performed over the public Internet as normal. Data plane operations, such as using Azure Machine Learning studio, APIs (including published pipelines), or the SDK use the private endpoint.
- When creating a compute instance or compute cluster in a workspace with a private endpoint, the compute instance and compute cluster must be in the same Azure region as the workspace.
- When attaching an Azure Kubernetes Service cluster to a workspace with a private endpoint, the cluster must be in the same region as the workspace.
- When using a workspace with multiple private endpoints, one of the private endpoints must be in the same VNet as the following dependency services:
 - Azure Storage Account that provides the default storage for the workspace
 - Azure Key Vault for the workspace
 - Azure Container Registry for the workspace.

For example, one VNet ('services' VNet) would contain a private endpoint for the dependency services and the workspace. This configuration allows the workspace to communicate with the services. Another VNet ('clients') might only contain a private endpoint for the workspace, and be used only for communication between client development machines and the workspace.

Create a workspace that uses a private endpoint

Use one of the following methods to create a workspace with a private endpoint. Each of these methods **requires an existing virtual network**:

TIP

If you'd like to create a workspace, private endpoint, and virtual network at the same time, see [Use an Azure Resource Manager template to create a workspace for Azure Machine Learning](#).

- [Azure CLI](#)
- [Portal](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

When using the [Azure CLI extension 2.0 CLI for machine learning](#), a YAML document is used to configure the workspace. The following example demonstrates creating a new workspace using a YAML configuration:

TIP

When using private link, your workspace cannot use Azure Container Registry tasks compute for image building. The `image_build_compute` property in this configuration specifies a CPU compute cluster name to use for Docker image environment building. You can also specify whether the private link workspace should be accessible over the internet using the `public_network_access` property.

In this example, the compute referenced by `image_build_compute` will need to be created before building images.

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: mlw-privatelink-prod
location: eastus
display_name: Private Link endpoint workspace-example
description: When using private link, you must set the image_build_compute property to a cluster name to use for Docker image environment building. You can also specify whether the workspace should be accessible over the internet.
image_build_compute: cpu-compute
public_network_access: Disabled
tags:
  purpose: demonstration
```

```
az ml workspace create \
-g <resource-group-name> \
--file privatelink.yml
```

After creating the workspace, use the [Azure networking CLI commands](#) to create a private link endpoint for the workspace.

```
az network private-endpoint create \
--name <private-endpoint-name> \
--vnet-name <vnet-name> \
--subnet <subnet-name> \
--private-connection-resource-id "/subscriptions/<subscription>/resourceGroups/<resource-group-name>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>" \
--group-id amlworkspace \
--connection-name workspace -l <location>
```

To create the private DNS zone entries for the workspace, use the following commands:

```
# Add privatelink.api.azureml.ms
az network private-dns zone create \
    -g <resource-group-name> \
    --name privatelink.api.azureml.ms

az network private-dns link vnet create \
    -g <resource-group-name> \
    --zone-name privatelink.api.azureml.ms \
    --name <link-name> \
    --virtual-network <vnet-name> \
    --registration-enabled false

az network private-endpoint dns-zone-group create \
    -g <resource-group-name> \
    --endpoint-name <private-endpoint-name> \
    --name myzonegroup \
    --private-dns-zone privatelink.api.azureml.ms \
    --zone-name privatelink.api.azureml.ms

# Add privatelink.notebooks.azure.net
az network private-dns zone create \
    -g <resource-group-name> \
    --name privatelink.notebooks.azure.net

az network private-dns link vnet create \
    -g <resource-group-name> \
    --zone-name privatelink.notebooks.azure.net \
    --name <link-name> \
    --virtual-network <vnet-name> \
    --registration-enabled false

az network private-endpoint dns-zone-group add \
    -g <resource-group-name> \
    --endpoint-name <private-endpoint-name> \
    --name myzonegroup \
    --private-dns-zone privatelink.notebooks.azure.net \
    --zone-name privatelink.notebooks.azure.net
```

Add a private endpoint to a workspace

Use one of the following methods to add a private endpoint to an existing workspace:

WARNING

If you have any existing compute targets associated with this workspace, and they are not behind the same virtual network tha the private endpoint is created in, they will not work.

- [Azure CLI](#)
- [Portal](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

When using the [Azure CLI extension 2.0 CLI for machine learning](#), use the [Azure networking CLI commands](#) to create a private link endpoint for the workspace.

```
az network private-endpoint create \
--name <private-endpoint-name> \
--vnet-name <vnet-name> \
--subnet <subnet-name> \
--private-connection-resource-id "/subscriptions/<subscription>/resourceGroups/<resource-group-name>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>" \
--group-id amlworkspace \
--connection-name workspace -l <location>
```

To create the private DNS zone entries for the workspace, use the following commands:

```
# Add privatelink.api.azureml.ms
az network private-dns zone create \
-g <resource-group-name> \
--name 'privatelink.api.azureml.ms'

az network private-dns link vnet create \
-g <resource-group-name> \
--zone-name 'privatelink.api.azureml.ms' \
--name <link-name> \
--virtual-network <vnet-name> \
--registration-enabled false

az network private-endpoint dns-zone-group create \
-g <resource-group-name> \
--endpoint-name <private-endpoint-name> \
--name myzonegroup \
--private-dns-zone 'privatelink.api.azureml.ms' \
--zone-name 'privatelink.api.azureml.ms'

# Add privatelink.notebooks.azure.net
az network private-dns zone create \
-g <resource-group-name> \
--name 'privatelink.notebooks.azure.net'

az network private-dns link vnet create \
-g <resource-group-name> \
--zone-name 'privatelink.notebooks.azure.net' \
--name <link-name> \
--virtual-network <vnet-name> \
--registration-enabled false

az network private-endpoint dns-zone-group add \
-g <resource-group-name> \
--endpoint-name <private-endpoint-name> \
--name myzonegroup \
--private-dns-zone 'privatelink.notebooks.azure.net' \
--zone-name 'privatelink.notebooks.azure.net'
```

Remove a private endpoint

You can remove one or all private endpoints for a workspace. Removing a private endpoint removes the workspace from the VNet that the endpoint was associated with. Removing the private endpoint may prevent the workspace from accessing resources in that VNet, or resources in the VNet from accessing the workspace. For example, if the VNet doesn't allow access to or from the public internet.

WARNING

Removing the private endpoints for a workspace **doesn't make it publicly accessible**. To make the workspace publicly accessible, use the steps in the [Enable public access](#) section.

To remove a private endpoint, use the following information:

- [Azure CLI](#)
- [Portal](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

When using the Azure CLI extension 2.0 CLI for machine learning, use the following command to remove the private endpoint:

```
az network private-endpoint delete \
--name <private-endpoint-name> \
--resource-group <resource-group-name>
```

Enable public access

In some situations, you may want to allow someone to connect to your secured workspace over a public endpoint, instead of through the VNet. Or you may want to remove the workspace from the VNet and re-enable public access.

IMPORTANT

Enabling public access doesn't remove any private endpoints that exist. All communications between components behind the VNet that the private endpoint(s) connect to are still secured. It enables public access only to the workspace, in addition to the private access through any private endpoints.

WARNING

When connecting over the public endpoint while the workspace uses a private endpoint to communicate with other resources:

- **Some features of studio will fail to access your data.** This problem happens when the *data is stored on a service that is secured behind the VNet*. For example, an Azure Storage Account.
- Using Jupyter, JupyterLab, and RStudio on a compute instance, including running notebooks, is **not supported**.

To enable public access, use the following steps:

TIP

There are two possible properties that you can configure:

- `allow_public_access_when_behind_vnet` - used by the Python SDK v1
- `public_network_access` - used by the CLI and Python SDK v2 (preview) Each property overrides the other. For example, setting `public_network_access` will override any previous setting to `allow_public_access_when_behind_vnet`.

Microsoft recommends using `public_network_access` to enable or disable public access to a workspace.

- [Azure CLI](#)
- [Portal](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

When using the Azure CLI extension 2.0 CLI preview for machine learning, use the `az ml update` command to

enable `public_network_access` for the workspace:

```
az ml workspace update \
--set public_network_access=Enabled \
-n <workspace-name> \
-g <resource-group-name>
```

You can also enable public network access by using a YAML file. For more information, see the [workspace YAML reference](#).

Securely connect to your workspace

To connect to a workspace that's secured behind a VNet, use one of the following methods:

- [Azure VPN gateway](#) - Connects on-premises networks to the VNet over a private connection. Connection is made over the public internet. There are two types of VPN gateways that you might use:
 - [Point-to-site](#): Each client computer uses a VPN client to connect to the VNet.
 - [Site-to-site](#): A VPN device connects the VNet to your on-premises network.
- [ExpressRoute](#) - Connects on-premises networks into the cloud over a private connection. Connection is made using a connectivity provider.
- [Azure Bastion](#) - In this scenario, you create an Azure Virtual Machine (sometimes called a jump box) inside the VNet. You then connect to the VM using Azure Bastion. Bastion allows you to connect to the VM using either an RDP or SSH session from your local web browser. You then use the jump box as your development environment. Since it is inside the VNet, it can directly access the workspace. For an example of using a jump box, see [Tutorial: Create a secure workspace](#).

IMPORTANT

When using a [VPN gateway](#) or [ExpressRoute](#), you will need to plan how name resolution works between your on-premises resources and those in the VNet. For more information, see [Use a custom DNS server](#).

If you have problems connecting to the workspace, see [Troubleshoot secure workspace connectivity](#).

Multiple private endpoints

Azure Machine Learning supports multiple private endpoints for a workspace. Multiple private endpoints are often used when you want to keep different environments separate. The following are some scenarios that are enabled by using multiple private endpoints:

- Client development environments in a separate VNet.
- An Azure Kubernetes Service (AKS) cluster in a separate VNet.
- Other Azure services in a separate VNet. For example, Azure Synapse and Azure Data Factory can use a Microsoft managed virtual network. In either case, a private endpoint for the workspace can be added to the managed VNet used by those services. For more information on using a managed virtual network with these services, see the following articles:
 - [Synapse managed private endpoints](#)
 - [Azure Data Factory managed virtual network](#).

IMPORTANT

Synapse's data exfiltration protection is not supported with Azure Machine Learning.

IMPORTANT

Each VNet that contains a private endpoint for the workspace must also be able to access the Azure Storage Account, Azure Key Vault, and Azure Container Registry used by the workspace. For example, you might create a private endpoint for the services in each VNet.

Adding multiple private endpoints uses the same steps as described in the [Add a private endpoint to a workspace](#) section.

Scenario: Isolated clients

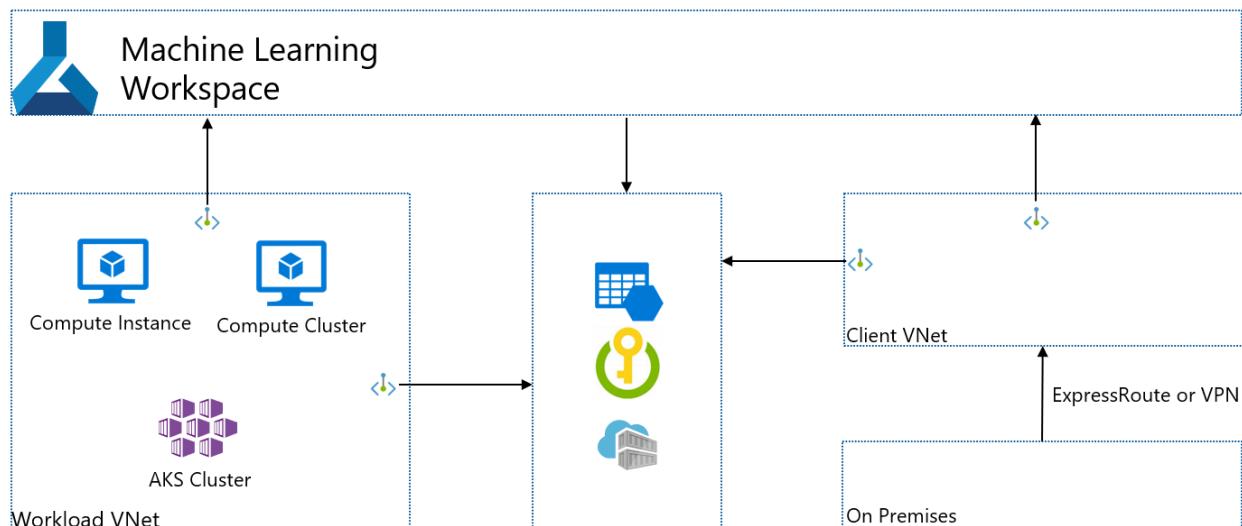
If you want to isolate the development clients, so they don't have direct access to the compute resources used by Azure Machine Learning, use the following steps:

NOTE

These steps assume that you have an existing workspace, Azure Storage Account, Azure Key Vault, and Azure Container Registry. Each of these services has a private endpoints in an existing VNet.

1. Create another VNet for the clients. This VNet might contain Azure Virtual Machines that act as your clients, or it may contain a VPN Gateway used by on-premises clients to connect to the VNet.
2. Add a new private endpoint for the Azure Storage Account, Azure Key Vault, and Azure Container Registry used by your workspace. These private endpoints should exist in the client VNet.
3. If you have another storage that is used by your workspace, add a new private endpoint for that storage. The private endpoint should exist in the client VNet and have private DNS zone integration enabled.
4. Add a new private endpoint to your workspace. This private endpoint should exist in the client VNet and have private DNS zone integration enabled.
5. Use the steps in the [Use studio in a virtual network](#) article to enable studio to access the storage account(s).

The following diagram illustrates this configuration. The **Workload VNet** contains computes created by the workspace for training & deployment. The **Client VNet** contains clients or client ExpressRoute/VPN connections. Both VNets contain private endpoints for the workspace, Azure Storage Account, Azure Key Vault, and Azure Container Registry.



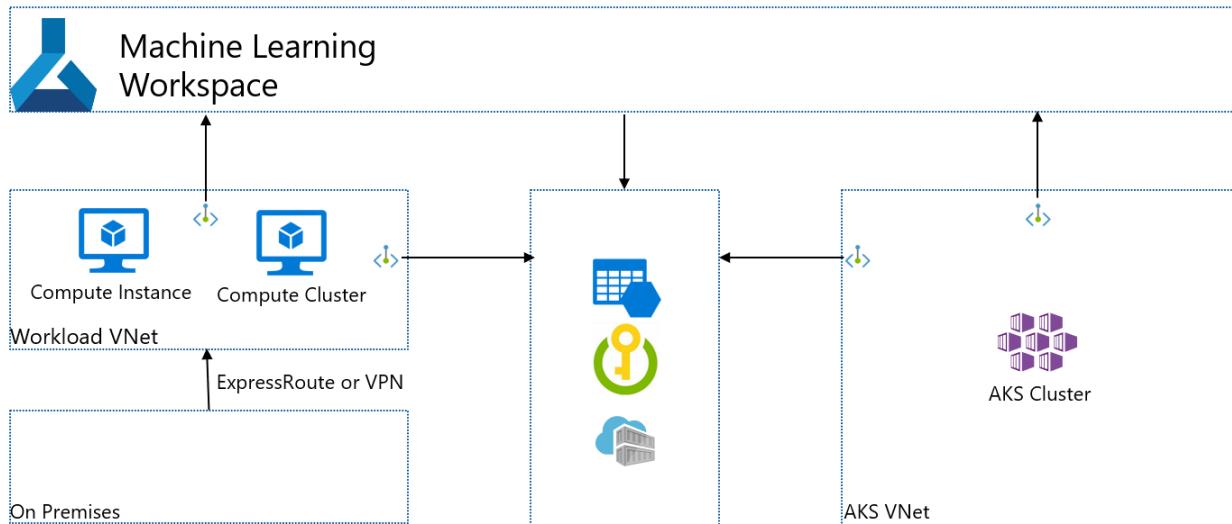
Scenario: Isolated Azure Kubernetes Service

If you want to create an isolated Azure Kubernetes Service used by the workspace, use the following steps:

NOTE

These steps assume that you have an existing workspace, Azure Storage Account, Azure Key Vault, and Azure Container Registry. Each of these services has a private endpoints in an existing VNet.

1. Create an Azure Kubernetes Service instance. During creation, AKS creates a VNet that contains the AKS cluster.
2. Add a new private endpoint for the Azure Storage Account, Azure Key Vault, and Azure Container Registry used by your workspace. These private endpoints should exist in the client VNet.
3. If you have other storage that is used by your workspace, add a new private endpoint for that storage. The private endpoint should exist in the client VNet and have private DNS zone integration enabled.
4. Add a new private endpoint to your workspace. This private endpoint should exist in the client VNet and have private DNS zone integration enabled.
5. Attach the AKS cluster to the Azure Machine Learning workspace. For more information, see [Create and attach an Azure Kubernetes Service cluster](#).



Next steps

- For more information on securing your Azure Machine Learning workspace, see the [Virtual network isolation and privacy overview](#) article.
- If you plan on using a custom DNS solution in your virtual network, see [how to use a workspace with a custom DNS server](#).
- [API platform network isolation](#)

How to use your workspace with a custom DNS server

9/22/2022 • 22 minutes to read • [Edit Online](#)

When using an Azure Machine Learning workspace with a private endpoint, there are [several ways to handle DNS name resolution](#). By default, Azure automatically handles name resolution for your workspace and private endpoint. If you instead **use your own custom DNS server**, you must manually create DNS entries or use conditional forwarders for the workspace.

IMPORTANT

This article covers how to find the fully qualified domain names (FQDN) and IP addresses for these entries if you would like to manually register DNS records in your DNS solution. Additionally this article provides architecture recommendations for how to configure your custom DNS solution to automatically resolve FQDNs to the correct IP addresses. This article does NOT provide information on configuring the DNS records for these items. Consult the documentation for your DNS software for information on how to add records.

TIP

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use a firewall](#)

Prerequisites

- An Azure Virtual Network that uses [your own DNS server](#).
- An Azure Machine Learning workspace with a private endpoint. For more information, see [Create an Azure Machine Learning workspace](#).
- Familiarity with using [Network isolation during training & inference](#).
- Familiarity with [Azure Private Endpoint DNS zone configuration](#)
- Familiarity with [Azure Private DNS](#)
- Optionally, [Azure CLI](#) or [Azure PowerShell](#).

Automated DNS server integration

Introduction

There are two common architectures to use automated DNS server integration with Azure Machine Learning:

- A custom [DNS server hosted in an Azure Virtual Network](#).
- A custom [DNS server hosted on-premises](#), connected to Azure Machine Learning through ExpressRoute.

While your architecture may differ from these examples, you can use them as a reference point. Both example architectures provide troubleshooting steps that can help you identify components that may be misconfigured.

Another option is to modify the `hosts` file on the client that is connecting to the Azure Virtual Network (VNet) that contains your workspace. For more information, see the [Host file](#) section.

Workspace DNS resolution path

Access to a given Azure Machine Learning workspace via Private Link is done by communicating with the following Fully Qualified Domains (called the workspace FQDNs) listed below:

Azure Public regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.azureml.ms
- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.cert.api.azureml.ms
- <compute instance name>.<region the workspace was created in>.instances.azureml.ms
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.
- <region>.notebooks.azure.net
- <managed online endpoint name>.<region>.inference.ml.azure.com - Used by managed online endpoints

Azure China 21Vianet regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.ml.azure.cn
- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.cert.api.ml.azure.cn
- <compute instance name>.<region the workspace was created in>.instances.azureml.cn
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.
- <region>.notebooks.chinacloudapi.cn
- <managed online endpoint name>.<region>.inference.ml.azure.cn - Used by managed online endpoints

Azure US Government regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.ml.azure.us
- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.cert.api.ml.azure.us
- <compute instance name>.<region the workspace was created in>.instances.azureml.us
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.
- <region>.notebooks.usgovcloudapi.net
- <managed online endpoint name>.<region>.inference.ml.azure.us - Used by managed online endpoints

The Fully Qualified Domains resolve to the following Canonical Names (CNAMES) called the workspace Private Link FQDNs:

Azure Public regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.privatelink.api.azureml.ms
 - ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.
 - <region>.privatelink.notebooks.azure.net
 - <managed online endpoint name>.<per-workspace globally-unique identifier>.inference.
 - <region>.privatelink.api.azureml.ms
- Used by managed online endpoints

Azure China regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.privatelink.api.ml.azure.cn
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.
- <region>.privatelink.notebooks.chinacloudapi.cn

- <managed online endpoint name>.<per-workspace globally-unique identifier>.inference.<region>.privatelink.api.ml.azure.cn
 - Used by managed online endpoints

Azure US Government regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.privatelink.api.ml.azure.us
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.<region>.privatelink.notebooks.usgovcloudapi.net
- <managed online endpoint name>.<per-workspace globally-unique identifier>.inference.<region>.privatelink.api.ml.azure.us
 - Used by managed online endpoints

The FQDNs resolve to the IP addresses of the Azure Machine Learning workspace in that region. However, resolution of the workspace Private Link FQDNs can be overridden by using a custom DNS server hosted in the virtual network. For an example of this architecture, see the [custom DNS server hosted in a vnet](#) example.

NOTE

Managed online endpoints share the workspace private endpoint. If you are manually adding DNS records to the private DNS zone `privatelink.api.azureml.ms`, an A record with wildcard `*.<per-workspace globally-unique identifier>.inference.<region>.privatelink.api.azureml.ms` should be added to route all endpoints under the workspace to the private endpoint.

Manual DNS server integration

This section discusses which Fully Qualified Domains to create A records for in a DNS Server, and which IP address to set the value of the A record to.

Retrieve Private Endpoint FQDNs

Azure Public region

The following list contains the fully qualified domain names (FQDNs) used by your workspace if it is in the Azure Public Cloud:

- <workspace-GUID>.workspace.<region>.cert.azureml.ms
- <workspace-GUID>.workspace.<region>.api.azureml.ms
- ml-<workspace-name, truncated>-<region>-<workspace-guid>.<region>.notebooks.azure.net

NOTE

The workspace name for this FQDN may be truncated. Truncation is done to keep `ml-<workspace-name, truncated>-<region>-<workspace-guid>` at 63 characters or less.

- <instance-name>.<region>.instances.azureml.ms

NOTE

- Compute instances can be accessed only from within the virtual network.
- The IP address for this FQDN is **not** the IP of the compute instance. Instead, use the private IP address of the workspace private endpoint (the IP of the `*.api.azureml.ms` entries.)

- <managed online endpoint name>.<region>.inference.ml.azure.com - Used by managed online endpoints

Azure China region

The following FQDNs are for Azure China regions:

- <workspace-GUID>.workspace.<region>.cert.api.ml.azure.cn
- <workspace-GUID>.workspace.<region>.api.ml.azure.cn
- ml-<workspace-name, truncated>-<region>-<workspace-guid>.<region>.notebooks.chinacloudapi.cn

NOTE

The workspace name for this FQDN may be truncated. Truncation is done to keep
ml-<workspace-name, truncated>-<region>-<workspace-guid> at 63 characters or less.

- <instance-name>.<region>.instances.azureml.cn
 - The IP address for this FQDN is **not** the IP of the compute instance. Instead, use the private IP address of the workspace private endpoint (the IP of the *.api.azureml.ms entries.)
- <managed online endpoint name>.<region>.inference.ml.azure.cn - Used by managed online endpoints

Azure US Government

The following FQDNs are for Azure US Government regions:

- <workspace-GUID>.workspace.<region>.cert.api.ml.azure.us
- <workspace-GUID>.workspace.<region>.api.ml.azure.us
- ml-<workspace-name, truncated>-<region>-<workspace-guid>.<region>.notebooks.usgovcloudapi.net

NOTE

The workspace name for this FQDN may be truncated. Truncation is done to keep
ml-<workspace-name, truncated>-<region>-<workspace-guid> at 63 characters or less.

- <instance-name>.<region>.instances.azureml.us
 - The IP address for this FQDN is **not** the IP of the compute instance. Instead, use the private IP address of the workspace private endpoint (the IP of the *.api.azureml.ms entries.)
- <managed online endpoint name>.<region>.inference.ml.azure.us - Used by managed online endpoints

Find the IP addresses

To find the internal IP addresses for the FQDNs in the VNet, use one of the following methods:

NOTE

The fully qualified domain names and IP addresses will be different based on your configuration. For example, the GUID value in the domain name will be specific to your workspace.

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Azure portal](#)

1. To get the ID of the private endpoint network interface, use the following command:

```
az network private-endpoint show --endpoint-name <endpoint> --resource-group <resource-group> --query 'networkInterfaces[*].id' --output table
```

2. To get the IP address and FQDN information, use the following command. Replace <resource-id> with the ID from the previous step:

```
az network nic show --ids <resource-id> --query 'ipConfigurations[*].{IPAddress: privateIpAddress, FQDNs: privateLinkConnectionProperties.fqdns}'
```

The output will be similar to the following text:

```
[  
  {  
    "FQDNs": [  
      "fb7e20a0-8891-458b-b969-55ddb3382f51.workspace.eastus.api.azureml.ms",  
      "fb7e20a0-8891-458b-b969-55ddb3382f51.workspace.eastus.cert.api.azureml.ms"  
    ],  
    "IPAddress": "10.1.0.5"  
  },  
  {  
    "FQDNs": [  
      "ml-myworkspace-eastus-fb7e20a0-8891-458b-b969-55ddb3382f51.eastus.notebooks.azure.net"  
    ],  
    "IPAddress": "10.1.0.6"  
  },  
  {  
    "FQDNs": [  
      "*.eastus.inference.ml.azure.com"  
    ],  
    "IPAddress": "10.1.0.7"  
  }  
]
```

The information returned from all methods is the same; a list of the FQDN and private IP address for the resources. The following example is from the Azure Public Cloud:

FQDN	IP ADDRESS
fb7e20a0-8891-458b-b969-55ddb3382f51.workspace.eastus.api.azureml.ms	10.1.0.5
fb7e20a0-8891-458b-b969-55ddb3382f51.workspace.eastus.cert.api.azureml.ms	10.1.0.5
ml-myworkspace-eastus-fb7e20a0-8891-458b-b969-55ddb3382f51.eastus.notebooks.azure.net	10.1.0.6
*.eastus.inference.ml.azure.com	10.1.0.7

The following table shows example IPs from Azure China regions:

FQDN	IP ADDRESS
52882c08-ead2-44aa-af65-08a75cf094bd.workspace.chinaeast2.api.ml.azure.cn	10.1.0.5

FQDN	IP ADDRESS
52882c08-ead2-44aa-af65-08a75cf094bd.workspace.chinaeast2.cert.api.ml.azure.cn	10.1.0.5
ml-mype-pltest-chinaeast2-52882c08-ead2-44aa-af65-08a75cf094bd.chinaeast2.notebooks.chinacloudapi.cn	10.1.0.6
*.chinaeast2.inference.ml.azure.cn	10.1.0.7

The following table shows example IPs from Azure US Government regions:

FQDN	IP ADDRESS
52882c08-ead2-44aa-af65-08a75cf094bd.workspace.chinaeast2.api.ml.azure.us	10.1.0.5
52882c08-ead2-44aa-af65-08a75cf094bd.workspace.chinaeast2.cert.api.ml.azure.us	10.1.0.5
ml-mype-plt-usgovvirginia-52882c08-ead2-44aa-af65-08a75cf094bd.usgovvirginia.notebooks.usgovcloudapi.net	10.1.0.6
*.usgovvirginia.inference.ml.azure.us	10.1.0.7

NOTE

Managed online endpoints share the workspace private endpoint. If you are manually adding DNS records to the private DNS zone `privatelink.api.azureml.ms`, an A record with wildcard

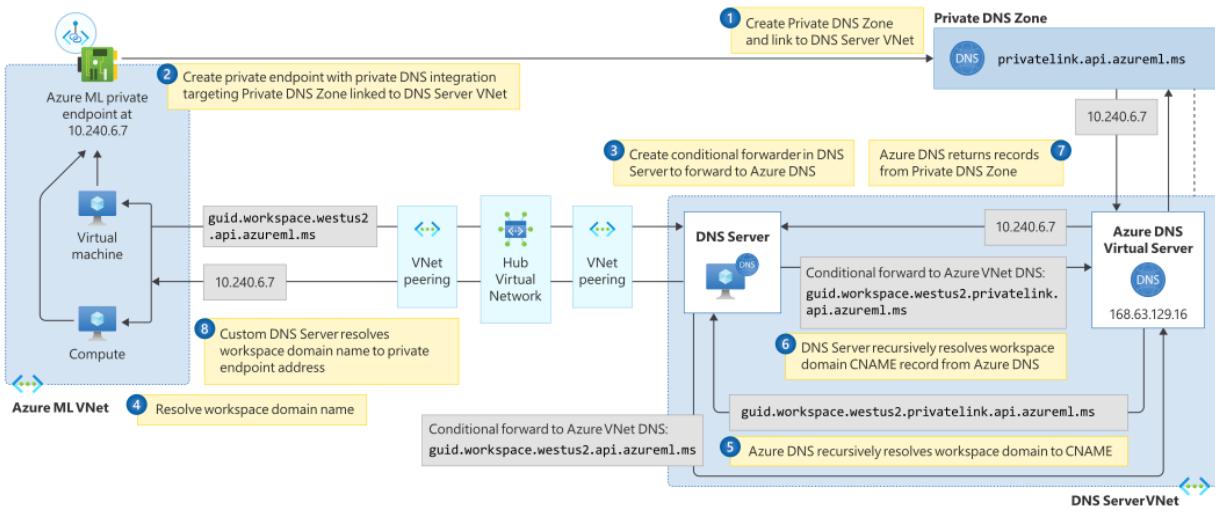
`*.<per-workspace globally-unique identifier>.inference.<region>.privatelink.api.azureml.ms` should be added to route all endpoints under the workspace to the private endpoint.

Create A records in custom DNS server

Once the list of FQDNs and corresponding IP addresses are gathered, proceed to create A records in the configured DNS Server. Refer to the documentation for your DNS server to determine how to create A records. Note it is recommended to create a unique zone for the entire FQDN, and create the A record in the root of the zone.

Example: Custom DNS Server hosted in VNet

This architecture uses the common Hub and Spoke virtual network topology. One virtual network contains the DNS server and one contains the private endpoint to the Azure Machine Learning workspace and associated resources. There must be a valid route between both virtual networks. For example, through a series of peered virtual networks.



The following steps describe how this topology works:

1. Create Private DNS Zone and link to DNS Server Virtual Network:

The first step in ensuring a Custom DNS solution works with your Azure Machine Learning workspace is to create two Private DNS Zones rooted at the following domains:

Azure Public regions:

- privatelink.api.azureml.ms
- privatelink.notebooks.azure.net

Azure China regions:

- privatelink.api.ml.azure.cn
- privatelink.notebooks.chinacloudapi.cn

Azure US Government regions:

- privatelink.api.ml.azure.us
- privatelink.notebooks.usgovcloudapi.net

NOTE

Managed online endpoints share the workspace private endpoint. If you are manually adding DNS records to the private DNS zone `privatelink.api.azureml.ms`, an A record with wildcard `*.<per-workspace globally-unique identifier>.inference.<region>.privatelink.api.azureml.ms` should be added to route all endpoints under the workspace to the private endpoint.

Following creation of the Private DNS Zone, it needs to be linked to the DNS Server Virtual Network. The Virtual Network that contains the DNS Server.

A Private DNS Zone overrides name resolution for all names within the scope of the root of the zone. This override applies to all Virtual Networks the Private DNS Zone is linked to. For example, if a Private DNS Zone rooted at `privatelink.api.azureml.ms` is linked to Virtual Network foo, all resources in Virtual Network foo that attempt to resolve `bar.workspace.westus2.privatelink.api.azureml.ms` will receive any record that is listed in the `privatelink.api.azureml.ms` zone.

However, records listed in Private DNS Zones are only returned to devices resolving domains using the default Azure DNS Virtual Server IP address. So the custom DNS Server will resolve domains for devices spread throughout your network topology. But the custom DNS Server will need to resolve Azure Machine Learning-related domains against the Azure DNS Virtual Server IP address.

2. Create private endpoint with private DNS integration targeting Private DNS Zone linked to DNS Server Virtual Network:

The next step is to create a Private Endpoint to the Azure Machine Learning workspace. The private endpoint targets both Private DNS Zones created in step 1. This ensures all communication with the workspace is done via the Private Endpoint in the Azure Machine Learning Virtual Network.

IMPORTANT

The private endpoint must have Private DNS integration enabled for this example to function correctly.

3. Create conditional forwarder in DNS Server to forward to Azure DNS:

Next, create a conditional forwarder to the Azure DNS Virtual Server. The conditional forwarder ensures that the DNS server always queries the Azure DNS Virtual Server IP address for FQDNs related to your workspace. This means that the DNS Server will return the corresponding record from the Private DNS Zone.

The zones to conditionally forward are listed below. The Azure DNS Virtual Server IP address is 168.63.129.16:

Azure Public regions:

- `api.azureml.ms`
- `notebooks.azure.net`
- `instances.azureml.ms`
- `aznbcontent.net`
- `inference.ml.azure.com` - Used by managed online endpoints

Azure China regions:

- `api.ml.azure.cn`
- `notebooks.chinacloudapi.cn`
- `instances.azureml.cn`
- `aznbcontent.net`
- `inference.ml.azure.cn` - Used by managed online endpoints

Azure US Government regions:

- `api.ml.azure.us`
- `notebooks.usgovcloudapi.net`
- `instances.azureml.us`
- `aznbcontent.net`
- `inference.ml.azure.us` - Used by managed online endpoints

IMPORTANT

Configuration steps for the DNS Server are not included here, as there are many DNS solutions available that can be used as a custom DNS Server. Refer to the documentation for your DNS solution for how to appropriately configure conditional forwarding.

4. Resolve workspace domain:

At this point, all setup is done. Now any client that uses DNS Server for name resolution and has a route to the Azure Machine Learning Private Endpoint can proceed to access the workspace. The client will first

start by querying DNS Server for the address of the following FQDNs:

Azure Public regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.azureml.ms
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.<region>.notebooks.azure.net
- <managed online endpoint name>.<region>.inference.ml.azure.com - Used by managed online endpoints

Azure China regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.ml.azure.cn
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.<region>.notebooks.chinacloudapi.cn
- <managed online endpoint name>.<region>.inference.ml.azure.cn - Used by managed online endpoints

Azure US Government regions:

- <per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.ml.azure.us
- ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.<region>.notebooks.usgovcloudapi.net
- <managed online endpoint name>.<region>.inference.ml.azure.us - Used by managed online endpoints

5. Azure DNS recursively resolves workspace domain to CNAME:

The DNS Server will resolve the FQDNs from step 4 from Azure DNS. Azure DNS will respond with one of the domains listed in step 1.

6. DNS Server recursively resolves workspace domain CNAME record from Azure DNS:

DNS Server will proceed to recursively resolve the CNAME received in step 5. Because there was a conditional forwarder setup in step 3, DNS Server will send the request to the Azure DNS Virtual Server IP address for resolution.

7. Azure DNS returns records from Private DNS zone:

The corresponding records stored in the Private DNS Zones will be returned to DNS Server, which will mean Azure DNS Virtual Server returns the IP addresses of the Private Endpoint.

8. Custom DNS Server resolves workspace domain name to private endpoint address:

Ultimately the Custom DNS Server now returns the IP addresses of the Private Endpoint to the client from step 4. This ensures that all traffic to the Azure Machine Learning workspace is via the Private Endpoint.

Troubleshooting

If you cannot access the workspace from a virtual machine or jobs fail on compute resources in the virtual network, use the following steps to identify the cause:

1. Locate the workspace FQDNs on the Private Endpoint:

Navigate to the Azure portal using one of the following links:

- [Azure Public regions](#)
- [Azure China regions](#)
- [Azure US Government regions](#)

Navigate to the Private Endpoint to the Azure Machine Learning workspace. The workspace FQDNs will be listed on the "Overview" tab.

2. Access compute resource in Virtual Network topology:

Proceed to access a compute resource in the Azure Virtual Network topology. This will likely require accessing a Virtual Machine in a Virtual Network that is peered with the Hub Virtual Network.

3. Resolve workspace FQDNs:

Open a command prompt, shell, or PowerShell. Then for each of the workspace FQDNs, run the following command:

```
nslookup <workspace FQDN>
```

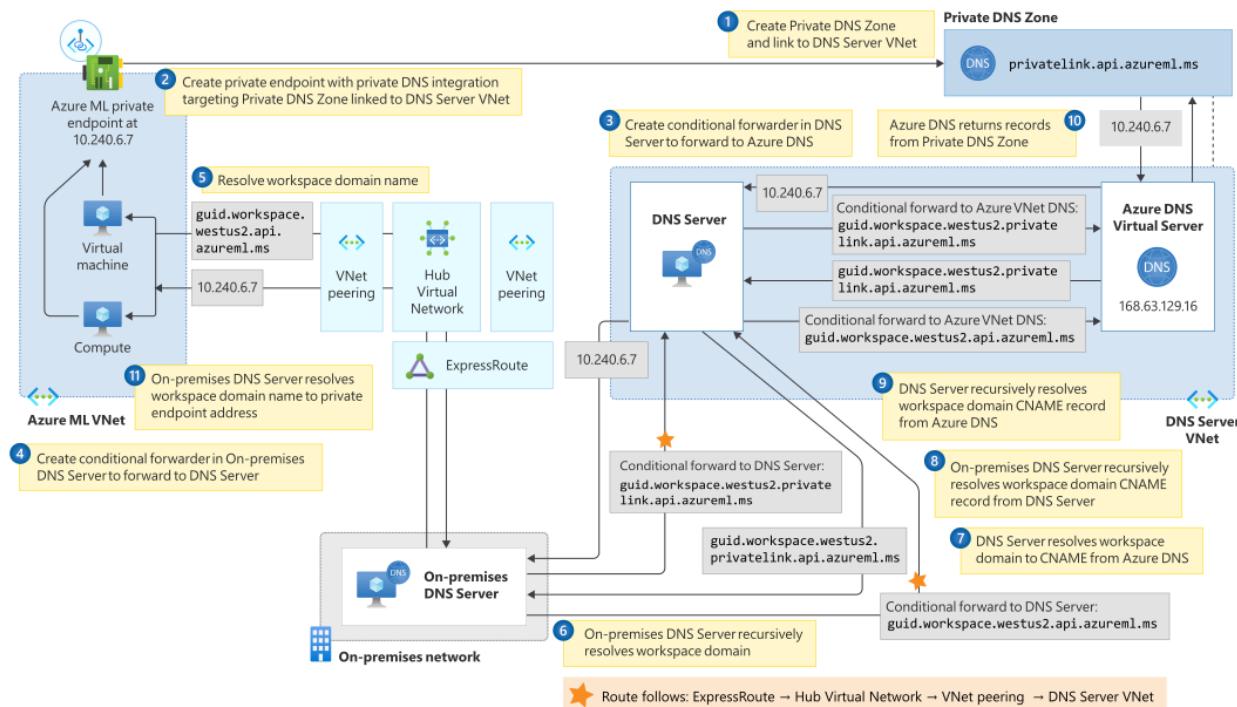
The result of each nslookup should return one of the two private IP addresses on the Private Endpoint to the Azure Machine Learning workspace. If it does not, then there is something misconfigured in the custom DNS solution.

Possible causes:

- The compute resource running the troubleshooting commands is not using DNS Server for DNS resolution
- The Private DNS Zones chosen when creating the Private Endpoint are not linked to the DNS Server VNet
- Conditional forwarders to Azure DNS Virtual Server IP were not configured correctly

Example: Custom DNS Server hosted on-premises

This architecture uses the common Hub and Spoke virtual network topology. ExpressRoute is used to connect from your on-premises network to the Hub virtual network. The Custom DNS server is hosted on-premises. A separate virtual network contains the private endpoint to the Azure Machine Learning workspace and associated resources. With this topology, there needs to be another virtual network hosting a DNS server that can send requests to the Azure DNS Virtual Server IP address.



The following steps describe how this topology works:

1. Create Private DNS Zone and link to DNS Server Virtual Network:

The first step in ensuring a Custom DNS solution works with your Azure Machine Learning workspace is to create two Private DNS Zones rooted at the following domains:

Azure Public regions:

- `privatelink.api.azureml.ms`
- `privatelink.notebooks.azure.net`

Azure China regions:

- `privatelink.api.ml.azure.cn`
- `privatelink.notebooks.chinacloudapi.cn`

Azure US Government regions:

- `privatelink.api.ml.azure.us`
- `privatelink.notebooks.usgovcloudapi.net`

NOTE

Managed online endpoints share the workspace private endpoint. If you are manually adding DNS records to the private DNS zone `privatelink.api.azureml.ms`, an A record with wildcard `*.<per-workspace globally-unique identifier>.inference.<region>.privatelink.api.azureml.ms` should be added to route all endpoints under the workspace to the private endpoint.

Following creation of the Private DNS Zone, it needs to be linked to the DNS Server VNet – the Virtual Network that contains the DNS Server.

NOTE

The DNS Server in the virtual network is separate from the On-premises DNS Server.

A Private DNS Zone overrides name resolution for all names within the scope of the root of the zone. This override applies to all Virtual Networks the Private DNS Zone is linked to. For example, if a Private DNS Zone rooted at `privatelink.api.azureml.ms` is linked to Virtual Network foo, all resources in Virtual Network foo that attempt to resolve `bar.workspace.westus2.privatelink.api.azureml.ms` will receive any record that is listed in the `privatelink.api.azureml.ms` zone.

However, records listed in Private DNS Zones are only returned to devices resolving domains using the default Azure DNS Virtual Server IP address. The Azure DNS Virtual Server IP address is only valid within the context of a Virtual Network. When using an on-premises DNS server, it is not able to query the Azure DNS Virtual Server IP address to retrieve records.

To get around this behavior, create an intermediary DNS Server in a virtual network. This DNS server can query the Azure DNS Virtual Server IP address to retrieve records for any Private DNS Zone linked to the virtual network.

While the On-premises DNS Server will resolve domains for devices spread throughout your network topology, it will resolve Azure Machine Learning-related domains against the DNS Server. The DNS Server will resolve those domains from the Azure DNS Virtual Server IP address.

2. Create private endpoint with private DNS integration targeting Private DNS Zone linked to DNS Server Virtual Network:

The next step is to create a Private Endpoint to the Azure Machine Learning workspace. The private endpoint targets both Private DNS Zones created in step 1. This ensures all communication with the workspace is done via the Private Endpoint in the Azure Machine Learning Virtual Network.

IMPORTANT

The private endpoint must have Private DNS integration enabled for this example to function correctly.

3. Create conditional forwarder in DNS Server to forward to Azure DNS:

Next, create a conditional forwarder to the Azure DNS Virtual Server. The conditional forwarder ensures that the DNS server always queries the Azure DNS Virtual Server IP address for FQDNs related to your workspace. This means that the DNS Server will return the corresponding record from the Private DNS Zone.

The zones to conditionally forward are listed below. The Azure DNS Virtual Server IP address is 168.63.129.16.

Azure Public regions:

- `api.azureml.ms`
- `notebooks.azure.net`
- `instances.azureml.ms`
- `aznbcontent.net`
- `inference.ml.azure.com` - Used by managed online endpoints

Azure China regions:

- `api.ml.azure.cn`
- `notebooks.chinacloudapi.cn`
- `instances.azureml.cn`
- `aznbcontent.net`
- `inference.ml.azure.cn` - Used by managed online endpoints

Azure US Government regions:

- `api.ml.azure.us`
- `notebooks.usgovcloudapi.net`
- `instances.azureml.us`
- `aznbcontent.net`
- `inference.ml.azure.us` - Used by managed online endpoints

IMPORTANT

Configuration steps for the DNS Server are not included here, as there are many DNS solutions available that can be used as a custom DNS Server. Refer to the documentation for your DNS solution for how to appropriately configure conditional forwarding.

4. Create conditional forwarder in On-premises DNS Server to forward to DNS Server:

Next, create a conditional forwarder to the DNS Server in the DNS Server Virtual Network. This forwarder is for the zones listed in step 1. This is similar to step 3, but, instead of forwarding to the Azure DNS Virtual Server IP address, the On-premises DNS Server will be targeting the IP address of the DNS Server. As the On-premises DNS Server is not in Azure, it is not able to directly resolve records in Private DNS Zones. In this case the DNS Server proxies requests from the On-premises DNS Server to the Azure DNS Virtual Server IP. This allows the On-premises DNS Server to retrieve records in the Private DNS Zones linked to the DNS Server Virtual Network.

The zones to conditionally forward are listed below. The IP addresses to forward to are the IP addresses

of your DNS Servers:

Azure Public regions:

- `api.azureml.ms`
- `notebooks.azure.net`
- `instances.azureml.ms`
- `inference.ml.azure.com` - Used by managed online endpoints

Azure China regions:

- `api.ml.azure.cn`
- `notebooks.chinacloudapi.cn`
- `instances.azureml.cn`
- `inference.ml.azure.cn` - Used by managed online endpoints

Azure US Government regions:

- `api.ml.azure.us`
- `notebooks.usgovcloudapi.net`
- `instances.azureml.us`
- `inference.ml.azure.us` - Used by managed online endpoints

IMPORTANT

Configuration steps for the DNS Server are not included here, as there are many DNS solutions available that can be used as a custom DNS Server. Refer to the documentation for your DNS solution for how to appropriately configure conditional forwarding.

5. Resolve workspace domain:

At this point, all setup is done. Any client that uses on-premises DNS Server for name resolution, and has a route to the Azure Machine Learning Private Endpoint, can proceed to access the workspace.

The client will first start by querying On-premises DNS Server for the address of the following FQDNs:

Azure Public regions:

- `<per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.azureml.ms`
- `ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.<region>.notebooks.azure.net`
- `<managed online endpoint name>.<region>.inference.ml.azure.com` - Used by managed online endpoints

Azure China regions:

- `<per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.ml.azure.cn`
- `ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.<region>.notebooks.chinacloudapi.cn`
- `<managed online endpoint name>.<region>.inference.ml.azure.cn` - Used by managed online endpoints

Azure US Government regions:

- `<per-workspace globally-unique identifier>.workspace.<region the workspace was created in>.api.ml.azure.us`
- `ml-<workspace-name, truncated>-<region>-<per-workspace globally-unique identifier>.<region>.notebooks.usgovcloudapi.net`
- `<managed online endpoint name>.<region>.inference.ml.azure.us` - Used by managed online endpoints

6. On-premises DNS server recursively resolves workspace domain:

The on-premises DNS Server will resolve the FQDNs from step 5 from the DNS Server. Because there is a conditional forwarder (step 4), the on-premises DNS Server will send the request to the DNS Server for resolution.

7. DNS Server resolves workspace domain to CNAME from Azure DNS:

The DNS server will resolve the FQDNs from step 5 from the Azure DNS. Azure DNS will respond with one of the domains listed in step 1.

8. On-premises DNS Server recursively resolves workspace domain CNAME record from DNS Server:

On-premises DNS Server will proceed to recursively resolve the CNAME received in step 7. Because there was a conditional forwarder setup in step 4, On-premises DNS Server will send the request to DNS Server for resolution.

9. DNS Server recursively resolves workspace domain CNAME record from Azure DNS:

DNS Server will proceed to recursively resolve the CNAME received in step 7. Because there was a conditional forwarder setup in step 3, DNS Server will send the request to the Azure DNS Virtual Server IP address for resolution.

10. Azure DNS returns records from Private DNS zone:

The corresponding records stored in the Private DNS Zones will be returned to DNS Server, which will mean the Azure DNS Virtual Server returns the IP addresses of the Private Endpoint.

11. On-premises DNS Server resolves workspace domain name to private endpoint address:

The query from On-premises DNS Server to DNS Server in step 8 ultimately returns the IP addresses associated with the Private Endpoint to the Azure Machine Learning workspace. These IP addresses are returned to the original client, which will now communicate with the Azure Machine Learning workspace over the Private Endpoint configured in step 1.

Example: Hosts file

The `hosts` file is a text document that Linux, macOS, and Windows all use to override name resolution for the local computer. The file contains a list of IP addresses and the corresponding host name. When the local computer tries to resolve a host name, if the host name is listed in the `hosts` file, the name is resolved to the corresponding IP address.

IMPORTANT

The `hosts` file only overrides name resolution for the local computer. If you want to use a `hosts` file with multiple computers, you must modify it individually on each computer.

The following table lists the location of the `hosts` file:

OPERATING SYSTEM	LOCATION
Linux	<code>/etc/hosts</code>
macOS	<code>/etc/hosts</code>
Windows	<code>%SystemRoot%\System32\drivers\etc\hosts</code>

TIP

The name of the file is `hosts` with no extension. When editing the file, use administrator access. For example, on Linux or macOS you might use `sudo vi`. On Windows, run notepad as an administrator.

The following is an example of `hosts` file entries for Azure Machine Learning:

```
# For core Azure Machine Learning hosts  
10.1.0.5    fb7e20a0-8891-458b-b969-55ddb3382f51.workspace.eastus.api.azureml.ms  
10.1.0.5    fb7e20a0-8891-458b-b969-55ddb3382f51.workspace.cert.api.azureml.ms  
10.1.0.6    ml-myworkspace-eastus-fb7e20a0-8891-458b-b969-55ddb3382f51.eastus.notebooks.azure.net  
  
# For a managed online/batch endpoint named 'mymanagedendpoint'  
10.1.0.7    mymanagedendpoint.eastus.inference.ml.azure.com  
  
# For a compute instance named 'mycomputeinstance'  
10.1.0.5    mycomputeinstance.eastus.instances.azureml.ms
```

For more information on the `hosts` file, see [https://wikipedia.org/wiki/Hosts_\(file\)](https://wikipedia.org/wiki/Hosts_(file)).

Dependency services DNS resolution

The services that your workspace relies on may also be secured using a private endpoint. If so, then you may need to create a custom DNS record if you need to directly communicate with the service. For example, if you want to directly work with the data in an Azure Storage Account used by your workspace.

NOTE

Some services have multiple private-endpoints for sub-services or features. For example, an Azure Storage Account may have individual private endpoints for Blob, File, and DFS. If you need to access both Blob and File storage, then you must enable resolution for each specific private endpoint.

For more information on the services and DNS resolution, see [Azure Private Endpoint DNS configuration](#).

Troubleshooting

If after running through the above steps you are unable to access the workspace from a virtual machine or jobs fail on compute resources in the Virtual Network containing the Private Endpoint to the Azure Machine learning workspace, follow the below steps to try to identify the cause.

1. Locate the workspace FQDNs on the Private Endpoint:

Navigate to the Azure portal using one of the following links:

- [Azure Public regions](#)
- [Azure China regions](#)
- [Azure US Government regions](#)

Navigate to the Private Endpoint to the Azure Machine Learning workspace. The workspace FQDNs will be listed on the "Overview" tab.

2. Access compute resource in Virtual Network topology:

Proceed to access a compute resource in the Azure Virtual Network topology. This will likely require accessing a Virtual Machine in a Virtual Network that is peered with the Hub Virtual Network.

3. Resolve workspace FQDNs:

Open a command prompt, shell, or PowerShell. Then for each of the workspace FQDNs, run the following command:

```
nslookup <workspace FQDN>
```

The result of each nslookup should yield one of the two private IP addresses on the Private Endpoint to the Azure Machine Learning workspace. If it does not, then there is something misconfigured in the custom DNS solution.

Possible causes:

- The compute resource running the troubleshooting commands is not using DNS Server for DNS resolution
- The Private DNS Zones chosen when creating the Private Endpoint are not linked to the DNS Server VNet
- Conditional forwarders from DNS Server to Azure DNS Virtual Server IP were not configured correctly
- Conditional forwarders from On-premises DNS Server to DNS Server were not configured correctly

Next steps

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use a firewall](#)

For information on integrating Private Endpoints into your DNS configuration, see [Azure Private Endpoint DNS configuration](#).

Configure inbound and outbound network traffic

9/22/2022 • 18 minutes to read • [Edit Online](#)

In this article, learn about the network communication requirements when securing Azure Machine Learning workspace in a virtual network (VNet). Including how to configure Azure Firewall to control access to your Azure Machine Learning workspace and the public internet. To learn more about securing Azure Machine Learning, see [Enterprise security for Azure Machine Learning](#).

NOTE

The information in this article applies to Azure Machine Learning workspace configured with a private endpoint.

TIP

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)

Well-known ports

The following are well-known ports used by services listed in this article. If a port range is used in this article and isn't listed in this section, it's specific to the service and may not have published information on what it's used for:

PORT	DESCRIPTION
80	Unsecured web traffic (HTTP)
443	Secured web traffic (HTTPS)
445	SMB traffic used to access file shares in Azure File storage
8787	Used when connecting to RStudio on a compute instance

Required public internet access

Azure Machine Learning requires both inbound and outbound access to the public internet. The following tables provide an overview of what access is required and what it is for. The **protocol** for all items is **TCP**. For service tags that end in `.region`, replace `region` with the Azure region that contains your workspace. For example, `Storage.westus`:

DIRECTION	PORTS	SERVICE TAG	PURPOSE
Inbound	29876-29877	BatchNodeManagement	Create, update, and delete of Azure Machine Learning compute instance and compute cluster. It isn't required if you use No Public IP option.
Inbound	44224	AzureMachineLearning	Create, update, and delete of Azure Machine Learning compute instance. It isn't required if you use No Public IP option.
Outbound	80, 443	AzureActiveDirectory	Authentication using Azure AD.
Outbound	443, 8787, 18881	AzureMachineLearning	Using Azure Machine Learning services.
Outbound	443	AzureResourceManager	Creation of Azure resources with Azure Machine Learning.
Outbound	443, 445 (*)	Storage.region	<p>Access data stored in the Azure Storage Account for compute cluster and compute instance. This outbound can be used to exfiltrate data. For more information, see Data exfiltration protection.</p> <p>(*) 445 is only required if you have a firewall between your virtual network for Azure ML and a private endpoint for your storage accounts.</p>
Outbound	443	AzureFrontDoor:FrontEnd * Not needed in Azure China.	<p>Global entry point for Azure Machine Learning studio. Store images and environments for AutoML.</p>
Outbound	443	MicrosoftContainerRegistry.region Note that this tag has a dependency on the AzureFrontDoor.FirstParty tag	Access docker images provided by Microsoft. Setup of the Azure Machine Learning router for Azure Kubernetes Service.
Outbound	443	AzureMonitor	Used to log monitoring and metrics to App Insights and Azure Monitor.

DIRECTION	PORTS	SERVICE TAG	PURPOSE
Outbound	443	Keyvault.region	Access the key vault for the Azure Batch service. Only needed if your workspace was created with the hbi_workspace flag enabled.

TIP

If you need the IP addresses instead of service tags, use one of the following options:

- Download a list from [Azure IP Ranges and Service Tags](#).
- Use the Azure CLI `az network list-service-tags` command.
- Use the Azure PowerShell `Get-AzNetworkServiceTag` command.

The IP addresses may change periodically.

IMPORTANT

When using a compute cluster that is configured for **no public IP address**, you must allow the following traffic:

- **Inbound** from source of **VirtualNetwork** and any port source, to destination of **VirtualNetwork**, and destination port of **29876, 29877**.
- **Inbound** from source **AzureLoadBalancer** and any port source to destination **VirtualNetwork** and port **44224** destination.

You may also need to allow **outbound** traffic to Visual Studio Code and non-Microsoft sites for the installation of packages required by your machine learning project. The following table lists commonly used repositories for machine learning:

HOST NAME	PURPOSE
anaconda.com *.anaconda.com	Used to install default packages.
*.anaconda.org	Used to get repo data.
pypi.org	Used to list dependencies from the default index, if any, and the index isn't overwritten by user settings. If the index is overwritten, you must also allow *.pythonhosted.org .
cloud.r-project.org	Used when installing CRAN packages for R development.
*pytorch.org	Used by some examples based on PyTorch.
*.tensorflow.org	Used by some examples based on Tensorflow.
code.visualstudio.com	Required to download and install VS Code desktop. This is not required for VS Code Web.
update.code.visualstudio.com *.vo.msecnd.net	Used to retrieve VS Code server bits that are installed on the compute instance through a setup script.

HOST NAME	PURPOSE
marketplace.visualstudio.com vscode.blob.core.windows.net *.gallerycdn.vsassets.io	Required to download and install VS Code extensions. These enable the remote connection to Compute Instances provided by the Azure ML extension for VS Code, see Connect to an Azure Machine Learning compute instance in Visual Studio Code for more information.
raw.githubusercontent.com/microsoft/vscode-tools-for-ai/master/azureml_remote_websocket_server/*	Used to retrieve websocket server bits, which are installed on the compute instance. The websocket server is used to transmit requests from Visual Studio Code client (desktop application) to Visual Studio Code server running on the compute instance.

When using Azure Kubernetes Service (AKS) with Azure Machine Learning, allow the following traffic to the AKS VNet:

- General inbound/outbound requirements for AKS as described in the [Restrict egress traffic in Azure Kubernetes Service](#) article.
- Outbound** to mcr.microsoft.com.
- When deploying a model to an AKS cluster, use the guidance in the [Deploy ML models to Azure Kubernetes Service](#) article.

Azure Firewall

IMPORTANT

Azure Firewall provides security *for Azure Virtual Network resources*. Some Azure Services, such as Azure Storage Accounts, have their own firewall settings that *apply to the public endpoint for that specific service instance*. The information in this document is specific to Azure Firewall.

For information on service instance firewall settings, see [Use studio in a virtual network](#).

- For **inbound** traffic to Azure Machine Learning compute cluster and compute instance, use [user-defined routes \(UDRs\)](#) to skip the firewall.
- For **outbound** traffic, create **network** and **application** rules.

These rule collections are described in more detail in [What are some Azure Firewall concepts](#).

Inbound configuration

When using Azure Machine Learning **compute instance** (with a public IP) or **compute cluster**, allow inbound traffic from Azure Batch management and Azure Machine Learning services. Compute instance with no public IP (preview) does not require this inbound communication. A Network Security Group allowing this traffic is dynamically created for you, however you may need to also create user-defined routes (UDR) if you have a firewall. When creating a UDR for this traffic, you can use either **IP Addresses** or **service tags** to route the traffic.

IMPORTANT

Using service tags with user-defined routes is now GA. For more information, see [Virtual Network routing](#).

TIP

While a compute instance without a public IP (a preview feature) does not need a UDR for this inbound traffic, you will still need these UDRs if you also use a compute cluster or a compute instance with a public IP.

- [IP Address routes](#)
- [Service tag routes](#)

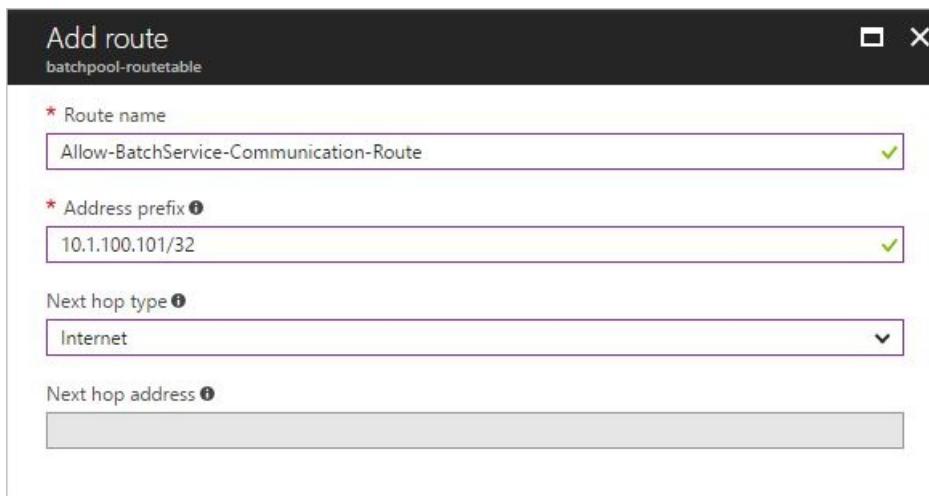
For the Azure Machine Learning service, you must add the IP address of both the **primary** and **secondary** regions. To find the secondary region, see the [Cross-region replication in Azure](#). For example, if your Azure Machine Learning service is in East US 2, the secondary region is Central US.

To get a list of IP addresses of the Batch service and Azure Machine Learning service, download the [Azure IP Ranges and Service Tags](#) and search the file for `BatchNodeManagement.<region>` and `AzureMachineLearning.<region>`, where `<region>` is your Azure region.

IMPORTANT

The IP addresses may change over time.

When creating the UDR, set the **Next hop type to Internet**. This means the inbound communication from Azure skips your firewall to access the load balancers with public IPs of Compute Instance and Compute Cluster. UDR is required because Compute Instance and Compute Cluster will get random public IPs at creation, and you cannot know the public IPs before creation to register them on your firewall to allow the inbound from Azure to specific IPs for Compute Instance and Compute Cluster. The following image shows an example IP address based UDR in the Azure portal:



For information on configuring UDR, see [Route network traffic with a routing table](#).

Outbound configuration

1. Add **Network rules**, allowing traffic **to** and **from** the following service tags:

SERVICE TAG	PROTOCOL	PORT
AzureActiveDirectory	TCP	80, 443
AzureMachineLearning	TCP	443, 8787, 18881
AzureResourceManager	TCP	443

Service Tag	Protocol	Port
Storage.region	TCP	443
AzureFrontDoor.FrontEnd * Not needed in Azure China.	TCP	443
AzureContainerRegistry.region	TCP	443
MicrosoftContainerRegistry.region Note that this tag has a dependency on the AzureFrontDoor.FirstParty tag	TCP	443
AzureKeyVault.region	TCP	443

TIP

- AzureContainerRegistry.region is only needed for custom Docker images. Including small modifications (such as additional packages) to base images provided by Microsoft.
- MicrosoftContainerRegistry.region is only needed if you plan on using the *default Docker images provided by Microsoft*, and *enabling user-managed dependencies*.
- AzureKeyVault.region is only needed if your workspace was created with the `hbi_workspace` flag enabled.
- For entries that contain `region`, replace with the Azure region that you're using. For example, `AzureContainerRegistry.westus`.

2. Add Application rules for the following hosts:

NOTE

This is not a complete list of the hosts required for all hosts you may need to communicate with, only the most commonly used. For example, if you need access to a GitHub repository or other host, you must identify and add the required hosts for that scenario.

Host Name	Purpose
anaconda.com *.anaconda.com	Used to install default packages.
*.anaconda.org	Used to get repo data.
pypi.org	Used to list dependencies from the default index, if any, and the index isn't overwritten by user settings. If the index is overwritten, you must also allow <code>*.pythonhosted.org</code> .
cloud.r-project.org	Used when installing CRAN packages for R development.
*pytorch.org	Used by some examples based on PyTorch.
*tensorflow.org	Used by some examples based on Tensorflow.

HOST NAME	PURPOSE
*vscode.dev *vscode-unpkg.net *vscode-cdn.net *vscodeexperiments.azureedge.net default.exp-tas.com	Required to access vscode.dev (Visual Studio Code for the Web)
code.visualstudio.com	Required to download and install VS Code desktop. This is not required for VS Code Web.
update.code.visualstudio.com .vo.msecnd.net	Used to retrieve VS Code server bits that are installed on the compute instance through a setup script.
marketplace.visualstudio.com vscode.blob.core.windows.net .gallerycdn.vsassets.io	Required to download and install VS Code extensions. These enable the remote connection to Compute Instances provided by the Azure ML extension for VS Code, see Connect to an Azure Machine Learning compute instance in Visual Studio Code for more information.
raw.githubusercontent.com/microsoft/vscode-tools-for-ai/master/azureml_remote_websocket_server/*	Used to retrieve websocket server bits that are installed on the compute instance. The websocket server is used to transmit requests from Visual Studio Code client (desktop application) to Visual Studio Code server running on the compute instance.
dc.applicationinsights.azure.com	Used to collect metrics and diagnostics information when working with Microsoft support.
dc.applicationinsights.microsoft.com	Used to collect metrics and diagnostics information when working with Microsoft support.
dc.services.visualstudio.com	Used to collect metrics and diagnostics information when working with Microsoft support.

For **Protocol:Port**, select use **http**, **https**.

For more information on configuring application rules, see [Deploy and configure Azure Firewall](#).

- To restrict outbound traffic for models deployed to Azure Kubernetes Service (AKS), see the [Restrict egress traffic in Azure Kubernetes Service](#) and [Deploy ML models to Azure Kubernetes Service](#) articles.

Kubernetes Compute

[Kubernetes Cluster](#) running behind an outbound proxy server or firewall needs extra network configuration. Configure the [Azure Arc network requirements](#) needed by Azure Arc agents. The following outbound URLs are also required for Azure Machine Learning,

OUTBOUND ENDPOINT	PORT	DESCRIPTION	TRAINING	INFERENCE
*.kusto.windows.net *.table.core.windows.net *.queue.core.windows.net	https:443	Required to upload system logs to Kusto.	✓	✓

OUTBOUND ENDPOINT	PORT	DESCRIPTION	TRAINING	INFERENCE
*.azurecr.io	https:443	Azure container registry, required to pull docker images used for machine learning workloads.	✓	✓
*.blob.core.windows.net	https:443	Azure blob storage, required to fetch machine learning project scripts,data or models, and upload job logs/outputs.	✓	✓
*.workspace.<region>.api.azureml.ms *<region>.experiments.azureml.net <region>.api.azureml.ms	https:443	Azure Machine Learning service API.	✓	✓
pypi.org	https:443	Python package index, to install pip packages used for training job environment initialization.	✓	N/A
archive.ubuntu.com security.ubuntu.com ppa.launchpad.net	http:80	Required to download the necessary security patches.	✓	N/A

NOTE

<region> is the lowercase full spelling of Azure Region, for example, eastus, southeastasia.

Other firewalls

The guidance in this section is generic, as each firewall has its own terminology and specific configurations. If you have questions, check the documentation for the firewall you're using.

If not configured correctly, the firewall can cause problems using your workspace. There are various host names that are used both by the Azure Machine Learning workspace. The following sections list hosts that are required for Azure Machine Learning.

Dependencies API

You can also use the Azure Machine Learning REST API to get a list of hosts and ports that you must allow **outbound** traffic to. To use this API, use the following steps:

1. Get an authentication token. The following command demonstrates using the [Azure CLI](#) to get an authentication token and subscription ID:

```
TOKEN=$(az account get-access-token --query accessToken -o tsv)
SUBSCRIPTION=$(az account show --query id -o tsv)
```

2. Call the API. In the following command, replace the following values:

- Replace `<region>` with the Azure region your workspace is in. For example, `westus2`.
- Replace `<resource-group>` with the resource group that contains your workspace.
- Replace `<workspace-name>` with the name of your workspace.

```
az rest --method GET \
    --url
"https://<region>.api.azureml.ms/rp/workspaces/subscriptions/$SUBSCRIPTION/resourceGroups/<resource-
group>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-
name>/outboundNetworkDependenciesEndpoints?api-version=2018-03-01-preview" \
    --header Authorization="Bearer $TOKEN"
```

The result of the API call is a JSON document. The following snippet is an excerpt of this document:

```
{
  "value": [
    {
      "properties": {
        "category": "Azure Active Directory",
        "endpoints": [
          {
            "domainName": "login.microsoftonline.com",
            "endpointDetails": [
              {
                "port": 80
              },
              {
                "port": 443
              }
            ]
          }
        ]
      }
    },
    {
      "properties": {
        "category": "Azure portal",
        "endpoints": [
          {
            "domainName": "management.azure.com",
            "endpointDetails": [
              {
                "port": 443
              }
            ]
          }
        ]
      }
    },
    ...
  ]
}
```

Microsoft hosts

The hosts in the following tables are owned by Microsoft, and provide services required for the proper functioning of your workspace. The tables list hosts for the Azure public, Azure Government, and Azure China 21Vianet regions.

IMPORTANT

Azure Machine Learning uses Azure Storage Accounts in your subscription and in Microsoft-managed subscriptions.

Where applicable, the following terms are used to differentiate between them in this section:

- **Your storage:** The Azure Storage Account(s) in your subscription, which is used to store your data and artifacts such as models, training data, training logs, and Python scripts.>
- **Microsoft storage:** The Azure Machine Learning compute instance and compute clusters rely on Azure Batch, and must access storage located in a Microsoft subscription. This storage is used only for the management of the compute instances. None of your data is stored here.

General Azure hosts

- [Azure public](#)
- [Azure Government](#)
- [Azure China 21Vianet](#)

REQUIRED FOR	HOSTS	PROTOCOL	PORTS
Azure Active Directory	login.microsoftonline.com	TCP	80, 443
Azure portal	management.azure.com	TCP	443
Azure Resource Manager	management.azure.com	TCP	443

Azure Machine Learning hosts

IMPORTANT

In the following table, replace `<storage>` with the name of the default storage account for your Azure Machine Learning workspace.

- [Azure public](#)
- [Azure Government](#)
- [Azure China 21Vianet](#)

REQUIRED FOR	HOSTS	PROTOCOL	PORTS
Azure Machine Learning studio	ml.azure.com	TCP	443
API	*.azureml.ms	TCP	443
API	*.azureml.net	TCP	443
Model management	*.modelmanagement.azure.ml.net	TCP	443
Integrated notebook	*.notebooks.azure.net	TCP	443

REQUIRED FOR	HOSTS	PROTOCOL	PORTS
Integrated notebook	<storage>.file.core.windows.net	TCP	443, 445
Integrated notebook	<storage>.dfs.core.windows.net	TCP	443
Integrated notebook	<storage>.blob.core.windows.net	TCP	443
Integrated notebook	graph.microsoft.com	TCP	443
Integrated notebook	*.aznbcontent.net	TCP	443
AutoML NLP, Vision	automlresources-prod.azureedge.net	TCP	443
AutoML NLP, Vision	aka.ms	TCP	443

NOTE

AutoML NLP, Vision are currently only supported in Azure public regions.

Azure Machine Learning compute instance and compute cluster hosts

TIP

- The host for **Azure Key Vault** is only needed if your workspace was created with the `hbi_workspace` flag enabled.
- Ports 8787 and 18881 for **compute instance** are only needed when your Azure Machine workspace has a private endpoint.
- In the following table, replace `<storage>` with the name of the default storage account for your Azure Machine Learning workspace.
- Websocket communication must be allowed to the compute instance. If you block websocket traffic, Jupyter notebooks won't work correctly.

- [Azure public](#)
- [Azure Government](#)
- [Azure China 21Vianet](#)

REQUIRED FOR	HOSTS	PROTOCOL	PORTS
Compute cluster/instance	graph.windows.net	TCP	443
Compute instance	*.instances.azureml.net	TCP	443
Compute instance	*.instances.azureml.ms	TCP	443, 8787, 18881
Microsoft storage access	*.blob.core.windows.net	TCP	443
Microsoft storage access	*.table.core.windows.net	TCP	443

REQUIRED FOR	HOSTS	PROTOCOL	PORTS
Microsoft storage access	*.queue.core.windows.net	TCP	443
Your storage account	<storage>.file.core.windows.net	TCP	443, 445
Your storage account	<storage>.blob.core.windows.net	TCP	443
Azure Key Vault	*.vault.azure.net	TCP	443

Docker images maintained by Azure Machine Learning

REQUIRED FOR	HOSTS	PROTOCOL	PORTS
Microsoft Container Registry	mcr.microsoft.com *.data.mcr.microsoft.com	TCP	443
Azure Machine Learning pre-built images	viennaglobal.azurecr.io	TCP	443

TIP

- **Azure Container Registry** is required for any custom Docker image. This includes small modifications (such as additional packages) to base images provided by Microsoft.
- **Microsoft Container Registry** is only needed if you plan on using the *default Docker images provided by Microsoft*, and *enabling user-managed dependencies*.
- If you plan on using federated identity, follow the [Best practices for securing Active Directory Federation Services](#) article.

Also, use the information in the [inbound configuration](#) section to add IP addresses for `BatchNodeManagement` and `AzureMachineLearning`.

For information on restricting access to models deployed to AKS, see [Restrict egress traffic in Azure Kubernetes Service](#).

Monitoring, metrics, and diagnostics

To support logging of metrics and other monitoring information to Azure Monitor and Application Insights, allow outbound traffic to the following hosts:

NOTE

The information logged to these hosts is also used by Microsoft Support to be able to diagnose any problems you run into with your workspace.

- `dc.applicationinsights.azure.com`
- `dc.applicationinsights.microsoft.com`
- `dc.services.visualstudio.com`
- `*.in.applicationinsights.azure.com`

For a list of IP addresses for these hosts, see [IP addresses used by Azure Monitor](#).

Python hosts

The hosts in this section are used to install Python packages, and are required during development, training, and deployment.

NOTE

This is not a complete list of the hosts required for all Python resources on the internet, only the most commonly used. For example, if you need access to a GitHub repository or other host, you must identify and add the required hosts for that scenario.

HOST NAME	PURPOSE
anaconda.com *.anaconda.com	Used to install default packages.
*.anaconda.org	Used to get repo data.
pypi.org	Used to list dependencies from the default index, if any, and the index isn't overwritten by user settings. If the index is overwritten, you must also allow *.pythonhosted.org.
*pytorch.org	Used by some examples based on PyTorch.
*.tensorflow.org	Used by some examples based on Tensorflow.

R hosts

The hosts in this section are used to install R packages, and are required during development, training, and deployment.

NOTE

This is not a complete list of the hosts required for all R resources on the internet, only the most commonly used. For example, if you need access to a GitHub repository or other host, you must identify and add the required hosts for that scenario.

HOST NAME	PURPOSE
cloud.r-project.org	Used when installing CRAN packages.

Visual Studio Code hosts

The hosts in this section are used to install Visual Studio Code packages to establish a remote connection between Visual Studio Code and compute instances in your Azure Machine Learning workspace.

NOTE

This is not a complete list of the hosts required for all Visual Studio Code resources on the internet, only the most commonly used. For example, if you need access to a GitHub repository or other host, you must identify and add the required hosts for that scenario.

HOST NAME	PURPOSE
*vscode.dev *vscode-unpkg.net *vscode-cdn.net *vscodeexperiments.azureedge.net default.exp-tas.com	Required to access vscode.dev (Visual Studio Code for the Web)
code.visualstudio.com	Required to download and install VS Code desktop. This is not required for VS Code Web.
update.code.visualstudio.com .vo.msecnd.net	Used to retrieve VS Code server bits that are installed on the compute instance through a setup script.
marketplace.visualstudio.com vscode.blob.core.windows.net .gallerycdn.vsassets.io	Required to download and install VS Code extensions. These enable the remote connection to Compute Instances provided by the Azure ML extension for VS Code, see Connect to an Azure Machine Learning compute instance in Visual Studio Code for more information.
raw.githubusercontent.com/microsoft/vscode-tools-for-ai/master/azureml_remote_websocket_server/*	Used to retrieve websocket server bits that are installed on the compute instance. The websocket server is used to transmit requests from Visual Studio Code client (desktop application) to Visual Studio Code server running on the compute instance.

Next steps

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the workspace resources](#)
- [Secure the training environment](#)
- [Secure the inference environment](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)

For more information on configuring Azure Firewall, see [Tutorial: Deploy and configure Azure Firewall using the Azure portal](#).

Azure Machine Learning data exfiltration prevention (Preview)

9/22/2022 • 5 minutes to read • [Edit Online](#)

Azure Machine Learning has several inbound and outbound dependencies. Some of these dependencies can expose a data exfiltration risk by malicious agents within your organization. This document explains how to minimize data exfiltration risk by limiting inbound and outbound requirements.

- **Inbound:** Azure Machine Learning compute instance and compute cluster have two inbound requirements: the `batchnodemanagement` (ports 29876-29877) and `azurermachinelearning` (port 44224) service tags. You can control this inbound traffic by using a network security group. It's difficult to disguise Azure service IPs, so there's low data exfiltration risk. You can also configure the compute to not use a public IP, which removes inbound requirements.
- **Outbound:** If malicious agents don't have write access to outbound destination resources, they can't use that outbound for data exfiltration. Azure Active Directory, Azure Resource Manager, Azure Machine Learning, and Microsoft Container Registry belong to this category. On the other hand, Storage and `AzureFrontDoor.frontend` can be used for data exfiltration.
 - **Storage Outbound:** This requirement comes from compute instance and compute cluster. A malicious agent can use this outbound rule to exfiltrate data by provisioning and saving data in their own storage account. You can remove data exfiltration risk by using an Azure Service Endpoint Policy and Azure Batch's simplified node communication architecture.
 - **AzureFrontDoor.frontend outbound:** Azure Front Door is required by the Azure Machine Learning studio UI and AutoML. To narrow down the list of possible outbound destinations to just those required by Azure ML, allowlist the following fully qualified domain names (FQDN) on your firewall.
 - `ml.azure.com`
 - `automlresources-prod.azureedge.net`

Prerequisites

- An Azure subscription
- An Azure Virtual Network (VNet)
- An Azure Machine Learning workspace with a private endpoint that connects to the VNet.
 - The storage account used by the workspace must also connect to the VNet using a private endpoint.

Limitations

- Data exfiltration prevention isn't supported with an Azure Machine Learning compute cluster or compute instance configured for no public IP.

1. Opt in to the preview

IMPORTANT

Before opting in to this preview, you must have created a workspace and a compute instance on the subscription you plan to use. You can delete the compute instance and/or workspace after creating them.

Use the form at <https://forms.office.com/r/1TraBek7LV> to opt in to this Azure Machine Learning preview. Microsoft will contact you once your subscription has been allowlisted to the preview.

TIP

It may take one to two weeks to allowlist your subscription.

2. Allow inbound & outbound network traffic

Inbound

IMPORTANT

The following information **modifies** the guidance provided in the [Inbound traffic](#) section of the "Secure training environment with virtual networks" article.

Inbound traffic from the service tag `BatchNodeManagement.<region>` or equivalent IP addresses is **not required**.

Outbound

IMPORTANT

The following information is **in addition** to the guidance provided in the [Secure training environment with virtual networks](#) and [Configure inbound and outbound network traffic](#) articles.

Select the configuration that you're using:

- [Network security group](#)
- [Firewall](#)

Allow outbound traffic over **TCP port 443** to the following service tags. Replace `<region>` with the Azure region that contains your compute cluster or instance:

- `BatchNodeManagement.<region>`
- `Storage.<region>` - A Service Endpoint Policy will be applied in a later step to limit outbound traffic.

3. Enable storage endpoint for the subnet

1. From the [Azure portal](#), select the **Azure Virtual Network** for your Azure ML workspace.
2. From the left of the page, select **Subnets** and then select the subnet that contains your compute cluster-instance resources.
3. In the form that appears, expand the **Services** dropdown and then **enable Microsoft.Storage**. Select **Save** to save these changes.

4. Create the service endpoint policy

1. From the [Azure portal](#), add a new **Service Endpoint Policy**. On the **Basics** tab, provide the required information and then select **Next**.

2. On the **Policy definitions** tab, perform the following actions:

- Select + **Add a resource**, and then provide the following information:

TIP

- At least one storage account resource must be listed in the policy.
- If you are adding multiple storage accounts, and the *default storage account* for your workspace is configured with a private endpoint, you do not need to include it in the policy.

- Service:** Microsoft.Storage
- Scope:** Select the scope. For example, select **Single account** if you want to limit the network traffic to one storage account.
- Subscription:** The Azure subscription that contains the storage account.
- Resource group:** The resource group that contains the storage account.
- Resource:** The storage account.

Select **Add** to add the resource information.

- Select + **Add an alias**, and then select `/services/Azure/MachineLearning` as the **Server Alias** value. Select **Add** to add the alias.

3. Select **Review + Create**, and then select **Create**.

5. Curated environments

When using Azure ML curated environments, make sure to use the latest environment version. The container registry for the environment must also be `mcr.microsoft.com`. To check the container registry, use the following steps:

1. From [Azure ML studio](#), select your workspace and then select **Environments**.

2. Verify that the **Azure container registry** begins with a value of `mcr.microsoft.com`.

IMPORTANT

If the container registry is `viennaglobal.azurecr.io` you cannot use the curated environment with the data exfiltration preview. Try upgrading to the latest version of the curated environment.

3. When using `mcr.microsoft.com`, you must also allow outbound configuration to the following resources.

Select the configuration option that you're using:

- [Network security group](#)
- [Firewall](#)

Allow outbound traffic over **TCP port 443** to the following service tags. Replace `<region>` with the Azure region that contains your compute cluster or instance.

- `MicrosoftContainerRegistry.<region>`
- `AzureFrontDoor.FirstParty`

Next steps

For more information, see the following articles:

- [How to configure inbound and outbound network traffic](#)

- Azure Batch simplified node communication

Network Isolation Change with Our New API Platform on Azure Resource Manager

9/22/2022 • 4 minutes to read • [Edit Online](#)

In this article, you'll learn about network isolation changes with our new v2 API platform on Azure Resource Manager (ARM) and its effect on network isolation.

Prerequisites

- The [Azure Machine Learning Python SDK](#) or [Azure CLI extension for machine learning v1](#).

IMPORTANT

The v1 extension (`azure-cli-ml`) version must be 1.41.0 or greater. Use the `az version` command to view version information.

What is the new API platform on Azure Resource Manager (ARM)

There are two types of operations used by the v1 and v2 APIs, **Azure Resource Manager (ARM)** and **Azure Machine Learning workspace**.

With the v1 API, most operations used the workspace. For v2, we've moved most operations to use public ARM.

API VERSION	PUBLIC ARM	INSIDE WORKSPACE VIRTUAL NETWORK
v1	Workspace and compute create, update, and delete (CRUD) operations.	Other operations such as experiments.
v2	Most operations such as workspace, compute, datastore, dataset, job, environment, code, component, endpoints.	Remaining operations.

The v2 API provides a consistent API in one place. You can more easily use Azure role-based access control and Azure Policy for resources with the v2 API because it's based on Azure Resource Manager.

The Azure Machine Learning CLI v2 uses our new v2 API platform. New features such as [managed online endpoints](#) are only available using the v2 API platform.

What are the network isolation changes with V2

As mentioned in the previous section, there are two types of operations; with ARM and with the workspace. With the **legacy v1 API**, most operations used the workspace. With the v1 API, adding a private endpoint to the workspace provided network isolation for everything except CRUD operations on the workspace or compute resources.

With the **new v2 API**, most operations use ARM. So enabling a private endpoint on your workspace doesn't provide the same level of network isolation. Operations that use ARM communicate over public networks, and include any metadata (such as your resource IDs) or parameters used by the operation. For example, the [create](#)

or update job api sends metadata, and [parameters](#).

IMPORTANT

For most people, using the public ARM communications is OK:

- Public ARM communications is the standard for management operations with Azure services. For example, creating an Azure Storage Account or Azure Virtual Network uses ARM.
- The Azure Machine Learning operations do not expose data in your storage account (or other storage in the VNet) on public networks. For example, a training job that runs on a compute cluster in the VNet, and uses data from a storage account in the VNet, would securely access the data directly using the VNet.
- All communication with public ARM is encrypted using TLS 1.2.

If you need time to evaluate the new v2 API before adopting it in your enterprise solutions, or have a company policy that prohibits sending communication over public networks, you can enable the *v1_legacy_mode* parameter. When enabled, this parameter disables the v2 API for your workspace.

WARNING

Enabling *v1_legacy_mode* may prevent you from using features provided by the v2 API. For example, some features of Azure Machine Learning studio may be unavailable.

Scenarios and Required Actions

WARNING

The *v1_legacy_mode* parameter is available now, but the v2 API blocking functionality will be enforced starting the week of May 15th, 2022.

- If you don't plan on using a private endpoint with your workspace, you don't need to enable parameter.
- If you're OK with operations communicating with public ARM, you don't need to enable the parameter.
- You only need to enable the parameter if you're using a private endpoint with the workspace *and* don't want to allow operations with ARM over public networks.

Once we implement the parameter, it will be retroactively applied to existing workspaces using the following logic:

- If you have **an existing workspace with a private endpoint**, the flag will be **true**.
- If you have **an existing workspace without a private endpoint** (public workspace), the flag will be **false**.

After the parameter has been implemented, the default value of the flag depends on the underlying REST API version used when you create a workspace (with a private endpoint):

- If the API version is **older** than `2022-05-01`, then the flag is **true** by default.
- If the API version is `2022-05-01` or **newer**, then the flag is **false** by default.

IMPORTANT

If you want to use the v2 API with your workspace, you must set the *v1_legacy_mode* parameter to **false**.

How to update v1_legacy_mode parameter

WARNING

The *v1_legacy_mode* parameter is available now, but the v2 API blocking functionality will be enforced starting the week of May 15th, 2022.

To update v1_legacy_mode, use the following steps:

- [Python SDK](#)
- [Azure CLI extension v1](#)

To disable v1_legacy_mode, use [Workspace.update](#) and set `v1_legacy_mode=false`.

```
from azureml.core import Workspace

ws = Workspace.from_config()
ws.update(v1_legacy_mode=False)
```

IMPORTANT

Note that it takes about 30 minutes to an hour or more for changing v1_legacy_mode parameter from **true** to **false** to be reflected in the workspace. Therefore, if you set the parameter to **false** but receive an error that the parameter is **true** in a subsequent operation, please try after a few more minutes.

Next steps

- [Use a private endpoint with Azure Machine Learning workspace.](#)
- [Create private link for managing Azure resources.](#)

Failover for business continuity and disaster recovery

9/22/2022 • 11 minutes to read • [Edit Online](#)

To maximize your uptime, plan ahead to maintain business continuity and prepare for disaster recovery with Azure Machine Learning.

Microsoft strives to ensure that Azure services are always available. However, unplanned service outages may occur. We recommend having a disaster recovery plan in place for handling regional service outages. In this article, you'll learn how to:

- Plan for a multi-regional deployment of Azure Machine Learning and associated resources.
- Maximize chances to recover logs, notebooks, docker images, and other metadata.
- Design for high availability of your solution.
- Initiate a failover to another region.

IMPORTANT

Azure Machine Learning itself does not provide automatic failover or disaster recovery. Backup and restore of workspace metadata such as run history is unavailable.

In case you have accidentally deleted your workspace or corresponding components, this article also provides you with currently supported recovery options.

Understand Azure services for Azure Machine Learning

Azure Machine Learning depends on multiple Azure services. Some of these services are provisioned in your subscription. You're responsible for the high-availability configuration of these services. Other services are created in a Microsoft subscription and are managed by Microsoft.

Azure services include:

- **Azure Machine Learning infrastructure:** A Microsoft-managed environment for the Azure Machine Learning workspace.
- **Associated resources:** Resources provisioned in your subscription during Azure Machine Learning workspace creation. These resources include Azure Storage, Azure Key Vault, Azure Container Registry, and Application Insights.
 - Default storage has data such as model, training log data, and dataset.
 - Key Vault has credentials for Azure Storage, Container Registry, and data stores.
 - Container Registry has a Docker image for training and inferencing environments.
 - Application Insights is for monitoring Azure Machine Learning.
- **Compute resources:** Resources you create after workspace deployment. For example, you might create a compute instance or compute cluster to train a Machine Learning model.
 - Compute instance and compute cluster: Microsoft-managed model development environments.
 - Other resources: Microsoft computing resources that you can attach to Azure Machine Learning, such as Azure Kubernetes Service (AKS), Azure Databricks, Azure Container Instances, and Azure HDInsight. You're responsible for configuring high-availability settings for these resources.

- **Other data stores:** Azure Machine Learning can mount other data stores such as Azure Storage, Azure Data Lake Storage, and Azure SQL Database for training data. These data stores are provisioned within your subscription. You're responsible for configuring their high-availability settings.

The following table shows the Azure services are managed by Microsoft and which are managed by you. It also indicates the services that are highly available by default.

SERVICE	MANAGED BY	HIGH AVAILABILITY BY DEFAULT
Azure Machine Learning infrastructure	Microsoft	
Associated resources		
Azure Storage	You	
Key Vault	You	✓
Container Registry	You	
Application Insights	You	NA
Compute resources		
Compute instance	Microsoft	
Compute cluster	Microsoft	
Other compute resources such as AKS, Azure Databricks, Container Instances, HDInsight	You	
Other data stores such as Azure Storage, SQL Database, Azure Database for PostgreSQL, Azure Database for MySQL, Azure Databricks File System	You	

The rest of this article describes the actions you need to take to make each of these services highly available.

Plan for multi-regional deployment

A multi-regional deployment relies on creation of Azure Machine Learning and other resources (infrastructure) in two Azure regions. If a regional outage occurs, you can switch to the other region. When planning on where to deploy your resources, consider:

- **Regional availability:** Use regions that are close to your users. To check regional availability for Azure Machine Learning, see [Azure products by region](#).
- **Azure paired regions:** Paired regions coordinate platform updates and prioritize recovery efforts where needed. For more information, see [Azure paired regions](#).
- **Service availability:** Decide whether the resources used by your solution should be hot/hot, hot/warm, or hot/cold.
 - **Hot/hot:** Both regions are active at the same time, with one region ready to begin use immediately.

- **Hot/warm:** Primary region active, secondary region has critical resources (for example, deployed models) ready to start. Non-critical resources would need to be manually deployed in the secondary region.
- **Hot/cold:** Primary region active, secondary region has Azure Machine Learning and other resources deployed, along with needed data. Resources such as models, model deployments, or pipelines would need to be manually deployed.

TIP

Depending on your business requirements, you may decide to treat different Azure Machine Learning resources differently. For example, you may want to use hot/hot for deployed models (inference), and hot/cold for experiments (training).

Azure Machine Learning builds on top of other services. Some services can be configured to replicate to other regions. Others you must manually create in multiple regions. The following table provides a list of services, who is responsible for replication, and an overview of the configuration:

AZURE SERVICE	GEO-REPLICATED BY	CONFIGURATION
Machine Learning workspace	You	Create a workspace in the selected regions.
Machine Learning compute	You	Create the compute resources in the selected regions. For compute resources that can dynamically scale, make sure that both regions provide sufficient compute quota for your needs.
Key Vault	Microsoft	Use the same Key Vault instance with the Azure Machine Learning workspace and resources in both regions. Key Vault automatically fails over to a secondary region. For more information, see Azure Key Vault availability and redundancy .
Container Registry	Microsoft	Configure the Container Registry instance to geo-replicate registries to the paired region for Azure Machine Learning. Use the same instance for both workspace instances. For more information, see Geo-replication in Azure Container Registry .
Storage Account	You	Azure Machine Learning does not support default storage-account failover using geo-redundant storage (GRS), geo-zone-redundant storage (GZRS), read-access geo-redundant storage (RA-GRS), or read-access geo-zone-redundant storage (RA-GZRS). Create a separate storage account for the default storage of each workspace. Create separate storage accounts or services for other data storage. For more information, see Azure Storage redundancy .

Application Insights	You	Create Application Insights for the workspace in both regions. To adjust the data-retention period and details, see Data collection, retention, and storage in Application Insights .
----------------------	-----	---

To enable fast recovery and restart in the secondary region, we recommend the following development practices:

- Use Azure Resource Manager templates. Templates are 'infrastructure-as-code', and allow you to quickly deploy services in both regions.
- To avoid drift between the two regions, update your continuous integration and deployment pipelines to deploy to both regions.
- When automating deployments, include the configuration of workspace attached compute resources such as Azure Kubernetes Service.
- Create role assignments for users in both regions.
- Create network resources such as Azure Virtual Networks and private endpoints for both regions. Make sure that users have access to both network environments. For example, VPN and DNS configurations for both virtual networks.

Compute and data services

Depending on your needs, you may have more compute or data services that are used by Azure Machine Learning. For example, you may use Azure Kubernetes Services or Azure SQL Database. Use the following information to learn how to configure these services for high availability.

Compute resources

- **Azure Kubernetes Service:** See [Best practices for business continuity and disaster recovery in Azure Kubernetes Service \(AKS\)](#) and [Create an Azure Kubernetes Service \(AKS\) cluster that uses availability zones](#). If the AKS cluster was created by using the Azure Machine Learning Studio, SDK, or CLI, cross-region high availability is not supported.
- **Azure Databricks:** See [Regional disaster recovery for Azure Databricks clusters](#).
- **Container Instances:** An orchestrator is responsible for failover. See [Azure Container Instances and container orchestrators](#).
- **HDInsight:** See [High availability services supported by Azure HDInsight](#).

Data services

- **Azure Blob container / Azure Files / Data Lake Storage Gen2:** See [Azure Storage redundancy](#).
- **Data Lake Storage Gen1:** See [High availability and disaster recovery guidance for Data Lake Storage Gen1](#).

- **SQL Database:** See [High availability for Azure SQL Database and SQL Managed Instance](#).
- **Azure Database for PostgreSQL:** See [High availability concepts in Azure Database for PostgreSQL - Single Server](#).
- **Azure Database for MySQL:** See [Understand business continuity in Azure Database for MySQL](#).
- **Azure Databricks File System:** See [Regional disaster recovery for Azure Databricks clusters](#).

TIP

If you provide your own customer-managed key to deploy an Azure Machine Learning workspace, Azure Cosmos DB is also provisioned within your subscription. In that case, you're responsible for configuring its high-availability settings. See [High availability with Azure Cosmos DB](#).

Design for high availability

Deploy critical components to multiple regions

Determine the level of business continuity that you are aiming for. The level may differ between the components of your solution. For example, you may want to have a hot/hot configuration for production pipelines or model deployments, and hot/cold for experimentation.

Manage training data on isolated storage

By keeping your data storage isolated from the default storage the workspace uses for logs, you can:

- Attach the same storage instances as datastores to the primary and secondary workspaces.
- Make use of geo-replication for data storage accounts and maximize your uptime.

Manage machine learning assets as code

NOTE

Backup and restore of workspace metadata such as run history, models and environments is unavailable. Specifying assets and configurations as code using YAML specs, will help you re-create assets across workspaces in case of a disaster.

Jobs in Azure Machine Learning are defined by a job specification. This specification includes dependencies on input artifacts that are managed on a workspace-instance level, including environments, datasets, and compute. For multi-region job submission and deployments, we recommend the following practices:

- Manage your code base locally, backed by a Git repository.
 - Export important notebooks from Azure Machine Learning studio.
 - Export pipelines authored in studio as code.

NOTE

Pipelines created in studio designer cannot currently be exported as code.

- Manage configurations as code.
 - Avoid hardcoded references to the workspace. Instead, configure a reference to the workspace instance using a [config file](#) and use [Workspace.from_config\(\)](#) to initialize the workspace. To automate the process, use the [Azure CLI extension for machine learning](#) command `az ml folder attach`.
 - Use job submission helpers such as [ScriptRunConfig](#) and [Pipeline](#).
 - Use [Environments.save_to_directory\(\)](#) to save your environment definitions.
 - Use a Dockerfile if you use custom Docker images.

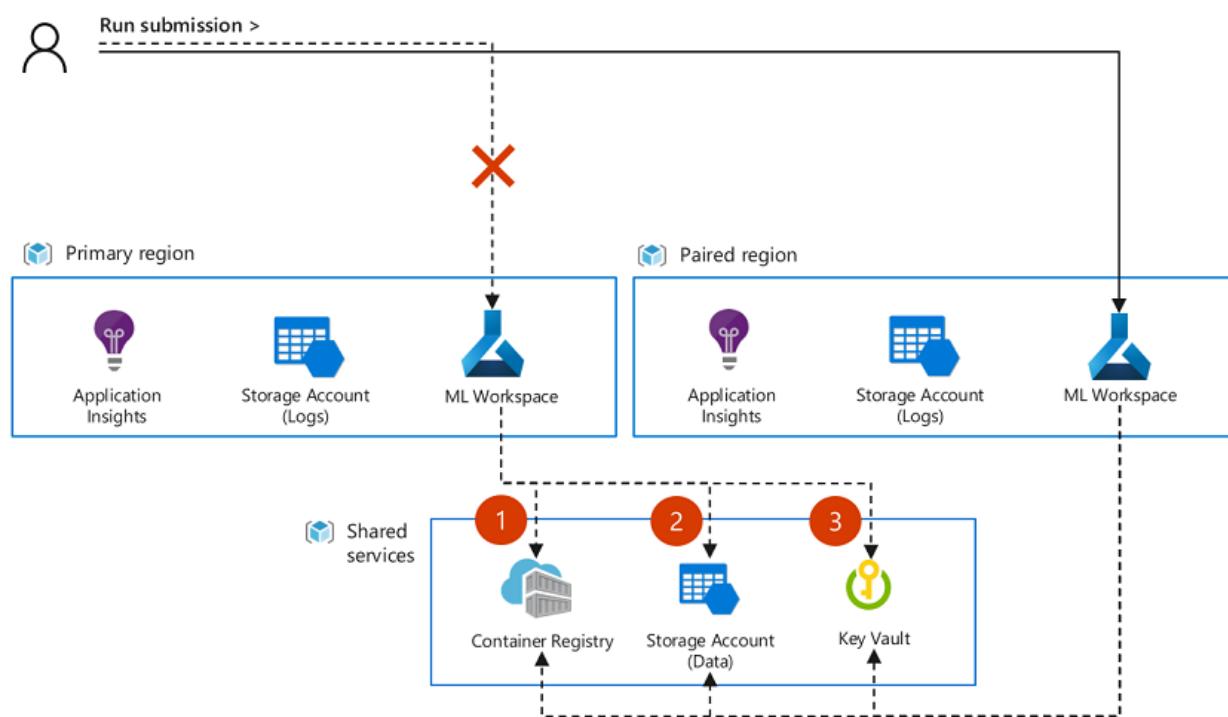
- Use the [Dataset](#) class to define the collection of data [paths](#) used by your solution.
- Use the [Inferenceconfig](#) class to deploy models as inference endpoints.

Initiate a failover

Continue work in the failover workspace

When your primary workspace becomes unavailable, you can switch over the secondary workspace to continue experimentation and development. Azure Machine Learning does not automatically submit jobs to the secondary workspace if there is an outage. Update your code configuration to point to the new workspace resource. We recommend to avoiding hardcoding workspace references. Instead, use a [workspace config file](#) to minimize manual user steps when changing workspaces. Make sure to also update any automation, such as continuous integration and deployment pipelines to the new workspace.

Azure Machine Learning cannot sync or recover artifacts or metadata between workspace instances. Dependent on your application deployment strategy, you might have to move artifacts or recreate experimentation inputs such as dataset objects in the failover workspace in order to continue job submission. In case you have configured your primary workspace and secondary workspace resources to share associated resources with geo-replication enabled, some objects might be directly available to the failover workspace. For example, if both workspaces share the same docker images, configured datastores, and Azure Key Vault resources. The following diagram shows a configuration where two workspaces share the same images (1), datastores (2), and Key Vault (3).



NOTE

Any jobs that are running when a service outage occurs will not automatically transition to the secondary workspace. It is also unlikely that the jobs will resume and finish successfully in the primary workspace once the outage is resolved. Instead, these jobs must be resubmitted, either in the secondary workspace or in the primary (once the outage is resolved).

Moving artifacts between workspaces

Depending on your recovery approach, you may need to copy artifacts such as dataset and model objects between the workspaces to continue your work. Currently, the portability of artifacts between workspaces is limited. We recommend managing artifacts as code where possible so that they can be recreated in the failover

instance.

The following artifacts can be exported and imported between workspaces by using the [Azure CLI extension for machine learning](#):

ARTIFACT	EXPORT	IMPORT
Models	<code>az ml model download --model-id {ID} --target-dir {PATH}</code>	<code>az ml model register --name {NAME} --path {PATH}</code>
Environments	<code>az ml environment download -n {NAME} -d {PATH}</code>	<code>az ml environment register -d {PATH}</code>
Azure ML pipelines (code-generated)	<code>az ml pipeline get --path {PATH}</code>	<code>az ml pipeline create --name {NAME} -y {PATH}</code>

TIP

- **Registered datasets** cannot be downloaded or moved. This includes datasets generated by Azure ML, such as intermediate pipeline datasets. However datasets that refer to a shared file location that both workspaces can access, or where the underlying data storage is replicated, can be registered on both workspaces. Use the [az ml dataset register](#) to register a dataset.
- **Job outputs** are stored in the default storage account associated with a workspace. While job outputs might become inaccessible from the studio UI in the case of a service outage, you can directly access the data through the storage account. For more information on working with data stored in blobs, see [Create, download, and list blobs with Azure CLI](#).

Recovery options

Workspace deletion

If you accidentally deleted your workspace it is currently not possible to recover it. However you are able to retrieve your existing notebooks from the corresponding storage if you follow these steps:

- In the [Azure portal](#) navigate to the storage account that was linked to the deleted Azure Machine Learning workspace.
- In the Data storage section on the left, click on **File shares**.
- Your notebooks are located on the file share with the name that contains your workspace ID.

The screenshot shows the Azure Storage File Shares blade for a storage account named 'testworkspace5647443388'. The blade has a sidebar with links like Overview, Activity log, Tags, and Storage Explorer (preview). The main area displays 'File share settings' with 'Active Directory: Not configured', 'Soft delete: 7 days', and 'Maximum capacity: 5 TiB'. A search bar at the top right says 'Search file shares by prefix (case-sensitive)' and has a 'Show deleted shares' toggle switch which is turned on. Below this, a table lists three deleted file shares:

Name	Modified	Tier	Quota
azurerm-filestore-9a05819f-08dd-4a0f-ab8b-c3e5801586b3	10/19/2021, 1:48:54 PM	Transaction optimized	5 TiB
code-391ff5ac-6576-460f-ba4d-7e03433c68b6	10/19/2021, 1:52:46 PM	Transaction optimized	5 TiB
codeshare-9a05819f-08dd-4a0f-ab8b-c3e5801586b3	10/19/2021, 1:48:54 PM	Transaction optimized	5 TiB

Next steps

To learn about repeatable infrastructure deployments with Azure Machine Learning, use an [Azure Resource](#)

Manager template.

Regenerate storage account access keys

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v1 Python SDK azureml v1

Learn how to change the access keys for Azure Storage accounts used by Azure Machine Learning. Azure Machine Learning can use storage accounts to store data or trained models.

For security purposes, you may need to change the access keys for an Azure Storage account. When you regenerate the access key, Azure Machine Learning must be updated to use the new key. Azure Machine Learning may be using the storage account for both model storage and as a datastore.

IMPORTANT

Credentials registered with datastores are saved in your Azure Key Vault associated with the workspace. If you have [soft-delete](#) enabled for your Key Vault, this article provides instructions for updating credentials. If you unregister the datastore and try to re-register it under the same name, this action will fail. See [Turn on Soft Delete for an existing key vault](#) for how to enable soft delete in this scenario.

Prerequisites

- An Azure Machine Learning workspace. For more information, see the [Create workspace resources](#) article.
- The [Azure Machine Learning SDK](#).
- The [Azure Machine Learning CLI extension v1](#).

NOTE

The code snippets in this document were tested with version 1.0.83 of the Python SDK.

What needs to be updated

Storage accounts can be used by the Azure Machine Learning workspace (storing logs, models, snapshots, etc.) and as a datastore. The process to update the workspace is a single Azure CLI command, and can be ran after updating the storage key. The process of updating datastores is more involved, and requires discovering what datastores are currently using the storage account and then re-registering them.

IMPORTANT

Update the workspace using the Azure CLI, and the datastores using Python, at the same time. Updating only one or the other is not sufficient, and may cause errors until both are updated.

To discover the storage accounts that are used by your datastores, use the following code:

```

import azureml.core
from azureml.core import Workspace, Datastore

ws = Workspace.from_config()

default_ds = ws.get_default_datastore()
print("Default datastore: " + default_ds.name + ", storage account name: " +
      default_ds.account_name + ", container name: " + default_ds.container_name)

datastores = ws.datastores
for name, ds in datastores.items():
    if ds.datastore_type == "AzureBlob":
        print("Blob store - datastore name: " + name + ", storage account name: " +
              ds.account_name + ", container name: " + ds.container_name)
    if ds.datastore_type == "AzureFile":
        print("File share - datastore name: " + name + ", storage account name: " +
              ds.account_name + ", container name: " + ds.container_name)

```

This code looks for any registered datastores that use Azure Storage and lists the following information:

- Datastore name: The name of the datastore that the storage account is registered under.
- Storage account name: The name of the Azure Storage account.
- Container: The container in the storage account that is used by this registration.

It also indicates whether the datastore is for an Azure Blob or an Azure File share, as there are different methods to re-register each type of datastore.

If an entry exists for the storage account that you plan on regenerating access keys for, save the datastore name, storage account name, and container name.

Update the access key

To update Azure Machine Learning to use the new key, use the following steps:

IMPORTANT

Perform all steps, updating both the workspace using the CLI, and datastores using Python. Updating only one or the other may cause errors until both are updated.

1. Regenerate the key. For information on regenerating an access key, see [Manage storage account access keys](#). Save the new key.
2. The Azure Machine Learning workspace will automatically synchronize the new key and begin using it after an hour. To force the workspace to sync to the new key immediately, use the following steps:
 - a. To sign in to the Azure subscription that contains your workspace by using the following Azure CLI command:

```
az login
```

TIP

After logging in, you see a list of subscriptions associated with your Azure account. The subscription information with `isDefault: true` is the currently activated subscription for Azure CLI commands. This subscription must be the same one that contains your Azure Machine Learning workspace. You can find the subscription ID from the [Azure portal](#) by visiting the overview page for your workspace. You can also use the SDK to get the subscription ID from the workspace object. For example,

```
Workspace.from_config().subscription_id .
```

To select another subscription, use the `az account set -s <subscription name or ID>` command and specify the subscription name or ID to switch to. For more information about subscription selection, see [Use multiple Azure Subscriptions](#).

- b. To update the workspace to use the new key, use the following command. Replace `myworkspace` with your Azure Machine Learning workspace name, and replace `myresourcegroup` with the name of the Azure resource group that contains the workspace.

```
az ml workspace sync-keys -w myworkspace -g myresourcegroup
```

TIP

If you get an error message stating that the `ml` extension isn't installed, use the following command to install it:

```
az extension add -n azure-cli-ml
```

This command automatically syncs the new keys for the Azure storage account used by the workspace.

3. You can re-register datastore(s) that use the storage account via the SDK or [the Azure Machine Learning studio](#).

- a. To **re-register datastores via the Python SDK**, use the values from the [What needs to be updated](#) section and the key from step 1 with the following code.

Since `overwrite=True` is specified, this code overwrites the existing registration and updates it to use the new key.

```
# Re-register the blob container
ds_blob = Datastore.register_azure_blob_container(workspace=ws,
                                                    datastore_name='your datastore name',
                                                    container_name='your container name',
                                                    account_name='your storage account name',
                                                    account_key='new storage account key',
                                                    overwrite=True)

# Re-register file shares
ds_file = Datastore.register_azure_file_share(workspace=ws,
                                               datastore_name='your datastore name',
                                               file_share_name='your container name',
                                               account_name='your storage account name',
                                               account_key='new storage account key',
                                               overwrite=True)
```

- b. To **re-register datastores via the studio**, select **Datastores** from the left pane of the studio.

- a. Select which datastore you want to update.
- b. Select the **Update credentials** button on the top left.
- c. Use your new access key from step 1 to populate the form and click **Save**.

If you are updating credentials for your **default datastore**, complete this step and repeat step 2b to resync your new key with the default datastore of the workspace.

Next steps

For more information on registering datastores, see the [Datastore](#) class reference.

Manage Azure Machine Learning workspaces in the portal or with the Python SDK (v2)

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

In this article, you create, view, and delete [Azure Machine Learning workspaces](#) for Azure Machine Learning, using the [Azure portal](#) or the [SDK for Python](#).

As your needs change or requirements for automation increase you can also manage workspaces [using the CLI](#), or [via the VS Code extension](#).

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- If using the Python SDK:
 1. [Install the SDK v2](#).
 2. Provide your subscription details

```
# Enter details of your subscription
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
```

3. Get a handle to the subscription. `ml_client` will be used in all the Python code in this article.

```
# get a handle to the subscription

from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

ml_client = MLClient(DefaultAzureCredential(), subscription_id, resource_group)
```

- (Optional) If you have multiple accounts, add the tenant ID of the Azure Active Directory you wish to use into the `DefaultAzureCredential`. Find your tenant ID from the [Azure portal](#) under [Azure Active Directory, External Identities](#).

```
DefaultAzureCredential(interactive_browser_tenant_id="<TENANT_ID>")
```

- (Optional) If you're working on a [sovereign cloud](#)**, specify the sovereign cloud to authenticate with into the `DefaultAzureCredential` ..

```
from azure.identity import AzureAuthorityHosts
DefaultAzureCredential(authority=AzureAuthorityHosts.AZURE_GOVERNMENT))
```

Limitations

- When creating a new workspace, you can either automatically create services needed by the workspace

or use existing services. If you want to use **existing services from a different Azure subscription** than the workspace, you must register the Azure Machine Learning namespace in the subscription that contains those services. For example, creating a workspace in subscription A that uses a storage account from subscription B, the Azure Machine Learning namespace must be registered in subscription B before you can use the storage account with the workspace.

The resource provider for Azure Machine Learning is **Microsoft.MachineLearningServices**. For information on how to see if it is registered and how to register it, see the [Azure resource providers and types](#) article.

IMPORTANT

This only applies to resources provided during workspace creation; Azure Storage Accounts, Azure Container Register, Azure Key Vault, and Application Insights.

- By default, creating a workspace also creates an Azure Container Registry (ACR). Since ACR doesn't currently support unicode characters in resource group names, use a resource group that doesn't contain these characters.
- Azure Machine Learning doesn't support hierarchical namespace (Azure Data Lake Storage Gen2 feature) for the workspace's default storage account.

TIP

An Azure Application Insights instance is created when you create the workspace. You can delete the Application Insights instance after cluster creation if you want. Deleting it limits the information gathered from the workspace, and may make it more difficult to troubleshoot problems. **If you delete the Application Insights instance created by the workspace, you cannot re-create it without deleting and recreating the workspace.**

For more information on using this Application Insights instance, see [Monitor and collect data from Machine Learning web service endpoints](#).

Create a workspace

You can create a workspace [directly in Azure Machine Learning studio](#), with limited options available. Or use one of the methods below for more control of options.

- [Python SDK](#)
- [Portal](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

- **Default specification.** By default, dependent resources and the resource group will be created automatically. This code creates a workspace named `myworkspace` and a resource group named `myresourcegroup` in `eastus2`.

```

# Creating a unique workspace name with current datetime to avoid conflicts
from azure.ai.ml.entities import Workspace
import datetime

basic_workspace_name = "mlw-basic-prod-" + datetime.datetime.now().strftime(
    "%Y%m%d%H%M"
)

ws_basic = Workspace(
    name=basic_workspace_name,
    location="eastus",
    display_name="Basic workspace-example",
    description="This example shows how to create a basic workspace",
    hbi_workspace=False,
    tags=dict(purpose="demo"),
)
ml_client.workspaces.begin_create(ws_basic)

```

- **Use existing Azure resources.** You can also create a workspace that uses existing Azure resources with the Azure resource ID format. Find the specific Azure resource IDs in the Azure portal or with the SDK. This example assumes that the resource group, storage account, key vault, App Insights, and container registry already exist.

```

# Creating a unique workspace name with current datetime to avoid conflicts
import datetime
from azure.ai.ml.entities import Workspace

basic_ex_workspace_name = "mlw-basicex-prod-" + datetime.datetime.now().strftime(
    "%Y%m%d%H%M"
)

ws_with_existing = Workspace(
    name=basic_ex_workspace_name,
    location="eastus",
    display_name="Bring your own dependent resources-example",
    description="This sample specifies a workspace configuration with existing dependent resources",

storage_account="/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft
.Storage/storageAccounts/<STORAGE_ACCOUNT>",

container_registry="/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Micros
oft.ContainerRegistry/registries/<CONTAINER_REGISTRY>",

key_vault="/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.KeyVa
ult/vaults/<KEY_VAULT>",

application_insights="/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Micr
osoft.insights/components/<APP_INSIGHTS>",
tags=dict(purpose="demonstration"),
)

# uncomment this line after providing details of subscription, resource group and other details above
# ml_client.begin_create_or_update(ws_with_existing)

```

For more information, see [Workspace SDK reference](#).

If you have problems in accessing your subscription, see [Set up authentication for Azure Machine Learning resources and workflows](#), as well as the [Authentication in Azure Machine Learning](#) notebook.

Networking

IMPORTANT

For more information on using a private endpoint and virtual network with your workspace, see [Network isolation and privacy](#).

- [Python SDK](#)
- [Portal](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

```
# Creating a unique workspace name with current datetime to avoid conflicts
import datetime
from azure.ai.ml.entities import Workspace

basic_private_link_workspace_name = (
    "mlw-privatelink-prod-" + datetime.datetime.now().strftime("%Y%m%d%H%M")
)

ws_private_link = Workspace(
    name=basic_private_link_workspace_name,
    location="eastus",
    display_name="Private Link endpoint workspace-example",
    description="When using private link, you must set the image_build_compute property to a cluster name to use for Docker image environment building. You can also specify whether the workspace should be accessible over the internet.",
    image_build_compute="cpu-compute",
    public_network_access="Disabled",
    tags=dict(purpose="demonstration"),
)
ml_client.workspaces.begin_create(ws_private_link)
```

This class requires an existing virtual network.

Advanced

By default, metadata for the workspace is stored in an Azure Cosmos DB instance that Microsoft maintains. This data is encrypted using Microsoft-managed keys.

To limit the data that Microsoft collects on your workspace, select **High business impact workspace** in the portal, or set `hbi_workspace=true` in Python. For more information on this setting, see [Encryption at rest](#).

IMPORTANT

Selecting high business impact can only be done when creating a workspace. You cannot change this setting after workspace creation.

Use your own data encryption key

You can provide your own key for data encryption. Doing so creates the Azure Cosmos DB instance that stores metadata in your Azure subscription. For more information, see [Customer-managed keys](#).

Use the following steps to provide your own key:

IMPORTANT

Before following these steps, you must first perform the following actions:

Follow the steps in [Configure customer-managed keys](#) to:

- Register the Azure Cosmos DB provider
- Create and configure an Azure Key Vault
- Generate a key

- [Python SDK](#)
- [Portal](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

```
from azure.ai.ml.entities import Workspace, CustomerManagedKey

# specify the workspace details
ws = Workspace(
    name="my_workspace",
    location="eastus",
    display_name="My workspace",
    description="This example shows how to create a workspace",
    customer_managed_key=CustomerManagedKey(
        key_vault="/subscriptions/<SUBSCRIPTION_ID>/resourcegroups/<RESOURCE_GROUP>/providers/microsoft.keyvault/vaults/<VAULT_NAME>"
            key_uri=""
        )
        tags=dict(purpose="demo")
    )
    ml_client.workspaces.begin_create(ws)
```

Download a configuration file

If you'll be running your code on a [compute instance](#), skip this step. The compute instance will create and store copy of this file for you.

If you plan to use code on your local environment that references this workspace, download the file:

1. Select your workspace in [Azure studio](#)
2. At the top right, select the workspace name, then select **Download config.json**

Place the file into the directory structure with your Python scripts or Jupyter Notebooks. It can be in the same directory, a subdirectory named `.azureml`, or in a parent directory. When you create a compute instance, this file is added to the correct directory on the VM for you.

Connect to a workspace

When running machine learning tasks using the SDK, you require a `MLClient` object that specifies the connection to your workspace. You can create an `MLClient` object from parameters, or with a configuration file.

APPLIES TO: [Python SDK azureml v1](#)

- **With a configuration file:** This code will read the contents of the configuration file to find your workspace. You'll get a prompt to sign in if you aren't already authenticated.

```
from azure.ai.ml import MLClient

# read the config from the current directory
ws_from_config = MLClient.from_config()
```

- **From parameters:** There's no need to have a `config.json` file available if you use this approach.

```
from azure.ai.ml import MLClient
from azure.ai.ml.entities import Workspace
from azure.identity import DefaultAzureCredential

# uncomment this line after providing details of subscription, resource group and workspace name
# ws = MLClient(DefaultAzureCredential(), subscription_id='<SUBSCRIPTION_ID>',
# resource_group_name='<RESOURCE_GROUP>', workspace_name='<AML_WORKSPACE_NAME>')
```

If you have problems in accessing your subscription, see [Set up authentication for Azure Machine Learning resources and workflows](#), as well as the [Authentication in Azure Machine Learning](#) notebook.

Find a workspace

See a list of all the workspaces you can use.

You can also search for workspace inside studio. See [Search for Azure Machine Learning assets \(preview\)](#).

- [Python SDK](#)
- [Portal](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

```
from azure.ai.ml import MLClient
from azure.ai.ml.entities import Workspace
from azure.identity import DefaultAzureCredential

# Enter details of your subscription
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"

my_ml_client = MLClient(DefaultAzureCredential(), subscription_id, resource_group)
```

```
for ws in my_ml_client.workspaces.list():
    print(ws.name, ":", ws.location, ":", ws.description)
```

To get details of a specific workspace:

```
ws = my_ml_client.workspaces.get("<AML_WORKSPACE_NAME>")
# uncomment this line after providing a workspace name above
# print(ws.location, ":", ws.resource_group)
```

Delete a workspace

When you no longer need a workspace, delete it.

WARNING

Once an Azure Machine Learning workspace has been deleted, it cannot be recovered.

If you accidentally deleted your workspace, you may still be able to retrieve your notebooks. For details, see [Failover for business continuity and disaster recovery](#).

- [Python SDK](#)
- [Portal](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

```
ml_client.workspaces.begin_delete(name=ws_basic.name, delete_dependent_resources=True)
```

The default action isn't to delete resources associated with the workspace, that is, container registry, storage account, key vault, and application insights. Set `delete_dependent_resources` to True to delete these resources as well.

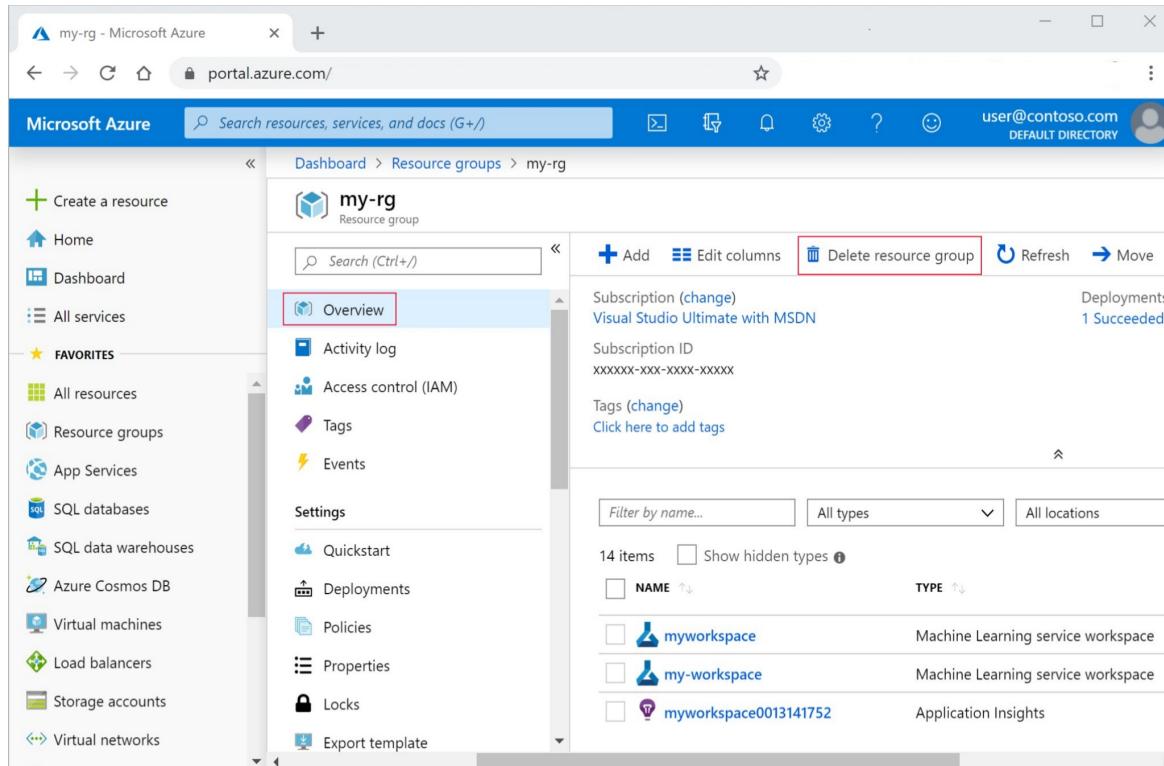
Clean up resources

IMPORTANT

The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.
2. From the list, select the resource group that you created.
3. Select **Delete resource group**.



The screenshot shows the Microsoft Azure portal interface. The user is in the 'Resource groups' section, specifically viewing the 'my-rg' resource group. The 'Overview' tab is selected. At the top right of the blade, there is a red box around the 'Delete resource group' button. The blade displays subscription information (Visual Studio Ultimate with MSDN), deployment status (1 Succeeded), and a list of resources including 'myworkspace', 'my-workspace', and 'myworkspace0013141752'. The left sidebar shows various service categories like Home, Dashboard, All services, and Favorites.

4. Enter the resource group name. Then select **Delete**.

Troubleshooting

- **Supported browsers in Azure Machine Learning studio:** We recommend that you use the most up-to-date browser that's compatible with your operating system. The following browsers are supported:
 - Microsoft Edge (The new Microsoft Edge, latest version. Not Microsoft Edge legacy)
 - Safari (latest version, Mac only)
 - Chrome (latest version)
 - Firefox (latest version)
- **Azure portal:**
 - If you go directly to your workspace from a share link from the SDK or the Azure portal, you can't view the standard **Overview** page that has subscription information in the extension. In this scenario, you also can't switch to another workspace. To view another workspace, go directly to [Azure Machine Learning studio](#) and search for the workspace name.
 - All assets (Data, Experiments, Computes, and so on) are available only in [Azure Machine Learning studio](#). They're *not* available from the Azure portal.
 - Attempting to export a template for a workspace from the Azure portal may return an error similar to the following text:

Could not get resource of the type <type>. Resources of this type will not be exported. As a workaround, use one of the templates provided at <https://github.com/Azure/azure-quickstart-templates/tree/master/quickstarts/microsoft.machinelearningservices> as the basis for your template.

Workspace diagnostics

You can run diagnostics on your workspace from Azure Machine Learning studio or the Python SDK. After diagnostics run, a list of any detected problems is returned. This list includes links to possible solutions. For more information, see [How to use workspace diagnostics](#).

Resource provider errors

When creating an Azure Machine Learning workspace, or a resource used by the workspace, you may receive an error similar to the following messages:

- No registered resource provider found for location {location}
- The subscription is not registered to use namespace {resource-provider-namespace}

Most resource providers are automatically registered, but not all. If you receive this message, you need to register the provider mentioned.

The following table contains a list of the resource providers required by Azure Machine Learning:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.MachineLearningServices	Creating the Azure Machine Learning workspace.
Microsoft.Storage	Azure Storage Account is used as the default storage for the workspace.
Microsoft.ContainerRegistry	Azure Container Registry is used by the workspace to build Docker images.
Microsoft.KeyVault	Azure Key Vault is used by the workspace to store secrets.
Microsoft.Notebooks/NotebookProxies	Integrated notebooks on Azure Machine Learning compute instance.
Microsoft.ContainerService	If you plan on deploying trained models to Azure Kubernetes Services.

If you plan on using a customer-managed key with Azure Machine Learning, then the following service providers must be registered:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.DocumentDB/databaseAccounts	Azure CosmosDB instance that logs metadata for the workspace.
Microsoft.Search/searchServices	Azure Search provides indexing capabilities for the workspace.

For information on registering resource providers, see [Resolve errors for resource provider registration](#).

Deleting the Azure Container Registry

The Azure Machine Learning workspace uses Azure Container Registry (ACR) for some operations. It will automatically create an ACR instance when it first needs one.

WARNING

Once an Azure Container Registry has been created for a workspace, do not delete it. Doing so will break your Azure Machine Learning workspace.

Examples

Examples in this article come from [workspace.ipynb](#).

Next steps

Once you have a workspace, learn how to [Train and deploy a model](#).

To learn more about planning a workspace for your organization's requirements, see [Organize and set up Azure Machine Learning](#).

- If you need to move a workspace to another Azure subscription, see [How to move a workspace](#).

For information on how to keep your Azure ML up to date with the latest security updates, see [Vulnerability management](#).

Manage Azure Machine Learning workspaces using Azure CLI

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

In this article, you learn how to create and manage Azure Machine Learning workspaces using the Azure CLI. The Azure CLI provides commands for managing Azure resources and is designed to get you working quickly with Azure, with an emphasis on automation. The machine learning extension to the CLI provides commands for working with Azure Machine Learning resources.

Prerequisites

- An [Azure subscription](#). If you don't have one, try the [free or paid version of Azure Machine Learning](#).
- To use the CLI commands in this document from your [local environment](#), you need the [Azure CLI](#).

If you use the [Azure Cloud Shell](#), the CLI is accessed through the browser and lives in the cloud.

Limitations

- When creating a new workspace, you can either automatically create services needed by the workspace or use existing services. If you want to use [existing services from a different Azure subscription](#) than the workspace, you must register the Azure Machine Learning namespace in the subscription that contains those services. For example, creating a workspace in subscription A that uses a storage account from subscription B, the Azure Machine Learning namespace must be registered in subscription B before you can use the storage account with the workspace.

The resource provider for Azure Machine Learning is [Microsoft.MachineLearningServices](#). For information on how to see if it is registered and how to register it, see the [Azure resource providers and types](#) article.

IMPORTANT

This only applies to resources provided during workspace creation; Azure Storage Accounts, Azure Container Register, Azure Key Vault, and Application Insights.

TIP

An Azure Application Insights instance is created when you create the workspace. You can delete the Application Insights instance after cluster creation if you want. Deleting it limits the information gathered from the workspace, and may make it more difficult to troubleshoot problems. [If you delete the Application Insights instance created by the workspace, you cannot re-create it without deleting and recreating the workspace.](#)

For more information on using this Application Insights instance, see [Monitor and collect data from Machine Learning web service endpoints](#).

Secure CLI communications

Some of the Azure CLI commands communicate with Azure Resource Manager over the internet. This communication is secured using HTTPS/TLS 1.2.

With the Azure Machine Learning CLI extension v2 ('ml'), all of the commands communicate with the Azure Resource Manager. This includes operational data such as YAML parameters and metadata. If your Azure Machine Learning workspace is public (that is, not behind a virtual network), then there's no extra configuration required. Communications are secured using HTTPS/TLS 1.2.

If your Azure Machine Learning workspace uses a private endpoint and virtual network and you're using CLI v2, choose one of the following configurations to use:

- If you're **OK** with the CLI v2 communication over the public internet, use the following `--public-network-access` parameter for the `az ml workspace update` command to enable public network access. For example, the following command updates a workspace for public network access:

```
az ml workspace update --name myworkspace --public-network-access enabled
```

- If you are **not OK** with the CLI v2 communication over the public internet, you can use an Azure Private Link to increase security of the communication. Use the following links to secure communications with Azure Resource Manager by using Azure Private Link.

1. [Secure your Azure Machine Learning workspace inside a virtual network using a private endpoint.](#)
2. [Create a Private Link for managing Azure resources.](#)
3. [Create a private endpoint](#) for the Private Link created in the previous step.

IMPORTANT

To configure the private link for Azure Resource Manager, you must be the *subscription owner* for the Azure subscription, and an *owner* or *contributor* of the root management group. For more information, see [Create a private link for managing Azure resources](#).

For more information on CLI v2 communication, see [Install and set up the CLI](#).

Connect the CLI to your Azure subscription

IMPORTANT

If you are using the Azure Cloud Shell, you can skip this section. The cloud shell automatically authenticates you using the account you log into your Azure subscription.

There are several ways that you can authenticate to your Azure subscription from the CLI. The most simple is to interactively authenticate using a browser. To authenticate interactively, open a command line or terminal and use the following command:

```
az login
```

If the CLI can open your default browser, it will do so and load a sign-in page. Otherwise, you need to open a browser and follow the instructions on the command line. The instructions involve browsing to <https://aka.ms/devicelogin> and entering an authorization code.

TIP

After logging in, you see a list of subscriptions associated with your Azure account. The subscription information with `isDefault: true` is the currently activated subscription for Azure CLI commands. This subscription must be the same one that contains your Azure Machine Learning workspace. You can find the subscription ID from the [Azure portal](#) by visiting the overview page for your workspace. You can also use the SDK to get the subscription ID from the workspace object. For example, `Workspace.from_config().subscription_id`.

To select another subscription, use the `az account set -s <subscription name or ID>` command and specify the subscription name or ID to switch to. For more information about subscription selection, see [Use multiple Azure Subscriptions](#).

For other methods of authenticating, see [Sign in with Azure CLI](#).

Create a resource group

The Azure Machine Learning workspace must be created inside a resource group. You can use an existing resource group or create a new one. To **create a new resource group**, use the following command. Replace `<resource-group-name>` with the name to use for this resource group. Replace `<location>` with the Azure region to use for this resource group:

NOTE

You should select a region where Azure Machine Learning is available. For information, see [Products available by region](#).

```
az group create --name <resource-group-name> --location <location>
```

The response from this command is similar to the following JSON. You can use the output values to locate the created resources or parse them as input to subsequent CLI steps for automation.

```
{
  "id": "/subscriptions/<subscription-GUID>/resourceGroups/<resourcegroupname>",
  "location": "<location>",
  "managedBy": null,
  "name": "<resource-group-name>",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": null
}
```

For more information on working with resource groups, see [az group](#).

Create a workspace

When you deploy an Azure Machine Learning workspace, various other services are [required as dependent associated resources](#). When you use the CLI to create the workspace, the CLI can either create new associated resources on your behalf or you could attach existing resources.

IMPORTANT

When attaching your own storage account, make sure that it meets the following criteria:

- The storage account is *not* a premium account (Premium_LRS and Premium_GRS)
- Both Azure Blob and Azure File capabilities enabled
- Hierarchical Namespace (ADLS Gen 2) is disabled These requirements are only for the *default* storage account used by the workspace.

When attaching Azure container registry, you must have the [admin account](#) enabled before it can be used with an Azure Machine Learning workspace.

- [Create with new resources](#)
- [Bring existing resources](#)

To create a new workspace where the **services are automatically created**, use the following command:

```
az ml workspace create -n <workspace-name> -g <resource-group-name>
```

IMPORTANT

When attaching existing resources, you don't have to specify all. You can specify one or more. For example, you can specify an existing storage account and the workspace will create the other resources.

The output of the workspace creation command is similar to the following JSON. You can use the output values to locate the created resources or parse them as input to subsequent CLI steps.

```
{
  "applicationInsights": "/subscriptions/<service-GUID>/resourcegroups/<resource-group-name>/providers/microsoft.insights/components/<application-insight-name>",
  "containerRegistry": "/subscriptions/<service-GUID>/resourcegroups/<resource-group-name>/providers/microsoft.containerregistry/registries/<acr-name>",
  "creationTime": "2019-08-30T20:24:19.6984254+00:00",
  "description": "",
  "friendlyName": "<workspace-name>",
  "id": "/subscriptions/<service-GUID>/resourceGroups/<resource-group-name>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>",
  "identityPrincipalId": "<GUID>",
  "identityTenantId": "<GUID>",
  "identityType": "SystemAssigned",
  "keyVault": "/subscriptions/<service-GUID>/resourcegroups/<resource-group-name>/providers/microsoft.keyvault/vaults/<key-vault-name>",
  "location": "<location>",
  "name": "<workspace-name>",
  "resourceGroup": "<resource-group-name>",
  "storageAccount": "/subscriptions/<service-GUID>/resourcegroups/<resource-group-name>/providers/microsoft.storage/storageaccounts/<storage-account-name>",
  "type": "Microsoft.MachineLearningServices/workspaces",
  "workspaceid": "<GUID>"
}
```

Advanced configurations

Configure workspace for private network connectivity

Dependent on your use case and organizational requirements, you can choose to configure Azure Machine

Learning using private network connectivity. You can use the Azure CLI to deploy a workspace and a Private link endpoint for the workspace resource. For more information on using a private endpoint and virtual network (VNet) with your workspace, see [Virtual network isolation and privacy overview](#). For complex resource configurations, also refer to template based deployment options including [Azure Resource Manager](#).

When using private link, your workspace can't use Azure Container Registry to build docker images. Hence, you must set the `image_build_compute` property to a CPU compute cluster name to use for Docker image environment building. You can also specify whether the private link workspace should be accessible over the internet using the `public_network_access` property.

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: mlw-privatelink-prod
location: eastus
display_name: Private Link endpoint workspace-example
description: When using private link, you must set the image_build_compute property to a cluster name to use for Docker image environment building. You can also specify whether the workspace should be accessible over the internet.
image_build_compute: cpu-compute
public_network_access: Disabled
tags:
  purpose: demonstration
```

```
az ml workspace create -g <resource-group-name> --file privatelink.yml
```

After creating the workspace, use the [Azure networking CLI commands](#) to create a private link endpoint for the workspace.

```
az network private-endpoint create \
  --name <private-endpoint-name> \
  --vnet-name <vnet-name> \
  --subnet <subnet-name> \
  --private-connection-resource-id "/subscriptions/<subscription>/resourceGroups/<resource-group-name>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>" \
  --group-id amlworkspace \
  --connection-name workspace -l <location>
```

To create the private DNS zone entries for the workspace, use the following commands:

```

# Add privatelink.api.azureml.ms
az network private-dns zone create \
    -g <resource-group-name> \
    --name 'privatelink.api.azureml.ms'

az network private-dns link vnet create \
    -g <resource-group-name> \
    --zone-name 'privatelink.api.azureml.ms' \
    --name <link-name> \
    --virtual-network <vnet-name> \
    --registration-enabled false

az network private-endpoint dns-zone-group create \
    -g <resource-group-name> \
    --endpoint-name <private-endpoint-name> \
    --name myzonegroup \
    --private-dns-zone 'privatelink.api.azureml.ms' \
    --zone-name 'privatelink.api.azureml.ms'

# Add privatelink.notebooks.azure.net
az network private-dns zone create \
    -g <resource-group-name> \
    --name 'privatelink.notebooks.azure.net'

az network private-dns link vnet create \
    -g <resource-group-name> \
    --zone-name 'privatelink.notebooks.azure.net' \
    --name <link-name> \
    --virtual-network <vnet-name> \
    --registration-enabled false

az network private-endpoint dns-zone-group add \
    -g <resource-group-name> \
    --endpoint-name <private-endpoint-name> \
    --name myzonegroup \
    --private-dns-zone 'privatelink.notebooks.azure.net' \
    --zone-name 'privatelink.notebooks.azure.net'

```

Customer-managed key and high business impact workspace

By default, metadata for the workspace is stored in an Azure Cosmos DB instance that Microsoft maintains. This data is encrypted using Microsoft-managed keys. Instead of using the Microsoft-managed key, you can also provide your own key. Doing so creates an extra set of resources in your Azure subscription to store your data.

To learn more about the resources that are created when you bring your own key for encryption, see [Data encryption with Azure Machine Learning](#).

Use the `customer_managed_key` parameter and containing `key_vault` and `key_uri` parameters, to specify the resource ID and uri of the key within the vault.

To limit the data that Microsoft collects on your workspace, you can additionally specify the `hbi_workspace` property.

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: mlw-cmkexample-prod
location: eastus
display_name: Customer managed key encryption-example
description: This configurations shows how to create a workspace that uses customer-managed keys for encryption.
customer_managed_key:
  key_vault:
    /subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.KeyVault/vaults/<KEY_VAULT>
    key_uri: https://<KEY_VAULT>.vault.azure.net/keys/<KEY_NAME>/<KEY_VERSION>
tags:
  purpose: demonstration
```

Then, you can reference this configuration file as part of the workspace creation CLI command.

```
az ml workspace create -g <resource-group-name> --file cmk.yml
```

NOTE

Authorize the **Machine Learning App** (in Identity and Access Management) with contributor permissions on your subscription to manage the data encryption additional resources.

NOTE

Azure Cosmos DB is **not** used to store information such as model performance, information logged by experiments, or information logged from your model deployments. For more information on monitoring these items, see the [Monitoring and logging](#) section of the architecture and concepts article.

IMPORTANT

Selecting high business impact can only be done when creating a workspace. You cannot change this setting after workspace creation.

For more information on customer-managed keys and high business impact workspace, see [Enterprise security for Azure Machine Learning](#).

Using the CLI to manage workspaces

Get workspace information

To get information about a workspace, use the following command:

```
az ml workspace show -n <workspace-name> -g <resource-group-name>
```

For more information, see the [az ml workspace show](#) documentation.

Update a workspace

To update a workspace, use the following command:

```
az ml workspace update -n <workspace-name> -g <resource-group-name>
```

For more information, see the [az ml workspace update](#) documentation.

Sync keys for dependent resources

If you change access keys for one of the resources used by your workspace, it takes around an hour for the workspace to synchronize to the new key. To force the workspace to sync the new keys immediately, use the following command:

```
az ml workspace sync-keys -n <workspace-name> -g <resource-group-name>
```

For more information on changing keys, see [Regenerate storage access keys](#).

For more information on the sync-keys command, see [az ml workspace sync-keys](#).

Delete a workspace

WARNING

Once an Azure Machine Learning workspace has been deleted, it cannot be recovered.

To delete a workspace after it's no longer needed, use the following command:

```
az ml workspace delete -n <workspace-name> -g <resource-group-name>
```

IMPORTANT

Deleting a workspace does not delete the application insight, storage account, key vault, or container registry used by the workspace.

You can also delete the resource group, which deletes the workspace and all other Azure resources in the resource group. To delete the resource group, use the following command:

```
az group delete -g <resource-group-name>
```

For more information, see the [az ml workspace delete](#) documentation.

If you accidentally deleted your workspace, are still able to retrieve your notebooks. For more information, see the [workspace deletion](#) section of the disaster recovery article.

Troubleshooting

Resource provider errors

When creating an Azure Machine Learning workspace, or a resource used by the workspace, you may receive an error similar to the following messages:

- No registered resource provider found for location {location}
- The subscription is not registered to use namespace {resource-provider-namespace}

Most resource providers are automatically registered, but not all. If you receive this message, you need to register the provider mentioned.

The following table contains a list of the resource providers required by Azure Machine Learning:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.MachineLearningServices	Creating the Azure Machine Learning workspace.
Microsoft.Storage	Azure Storage Account is used as the default storage for the workspace.
Microsoft.ContainerRegistry	Azure Container Registry is used by the workspace to build Docker images.
Microsoft.KeyVault	Azure Key Vault is used by the workspace to store secrets.
Microsoft.Notebooks/NotebookProxies	Integrated notebooks on Azure Machine Learning compute instance.
Microsoft.ContainerService	If you plan on deploying trained models to Azure Kubernetes Services.

If you plan on using a customer-managed key with Azure Machine Learning, then the following service providers must be registered:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.DocumentDB/databaseAccounts	Azure CosmosDB instance that logs metadata for the workspace.
Microsoft.Search/searchServices	Azure Search provides indexing capabilities for the workspace.

For information on registering resource providers, see [Resolve errors for resource provider registration](#).

Moving the workspace

WARNING

Moving your Azure Machine Learning workspace to a different subscription, or moving the owning subscription to a new tenant, is not supported. Doing so may cause errors.

Deleting the Azure Container Registry

The Azure Machine Learning workspace uses Azure Container Registry (ACR) for some operations. It will automatically create an ACR instance when it first needs one.

WARNING

Once an Azure Container Registry has been created for a workspace, do not delete it. Doing so will break your Azure Machine Learning workspace.

Next steps

For more information on the Azure CLI extension for machine learning, see the [az ml](#) documentation.

To check for problems with your workspace, see [How to use workspace diagnostics](#).

To learn how to move a workspace to a new Azure subscription, see [How to move a workspace](#).

For information on how to keep your Azure ML up to date with the latest security updates, see [Vulnerability management](#).

Use an Azure Resource Manager template to create a workspace for Azure Machine Learning

9/22/2022 • 15 minutes to read • [Edit Online](#)

In this article, you learn several ways to create an Azure Machine Learning workspace using Azure Resource Manager templates. A Resource Manager template makes it easy to create resources as a single, coordinated operation. A template is a JSON document that defines the resources that are needed for a deployment. It may also specify deployment parameters. Parameters are used to provide input values when using the template.

For more information, see [Deploy an application with Azure Resource Manager template](#).

Prerequisites

- An [Azure subscription](#). If you do not have one, try the [free or paid version of Azure Machine Learning](#).
- To use a template from a CLI, you need either [Azure PowerShell](#) or the [Azure CLI](#).

Limitations

- When creating a new workspace, you can either automatically create services needed by the workspace or use existing services. If you want to use **existing services from a different Azure subscription** than the workspace, you must register the Azure Machine Learning namespace in the subscription that contains those services. For example, creating a workspace in subscription A that uses a storage account from subscription B, the Azure Machine Learning namespace must be registered in subscription B before you can use the storage account with the workspace.

The resource provider for Azure Machine Learning is **Microsoft.MachineLearningServices**. For information on how to see if it is registered and how to register it, see the [Azure resource providers and types](#) article.

IMPORTANT

This only applies to resources provided during workspace creation; Azure Storage Accounts, Azure Container Register, Azure Key Vault, and Application Insights.

Multiple workspaces in the same VNet

The template doesn't support multiple Azure Machine Learning workspaces deployed in the same VNet. This is because the template creates new DNS zones during deployment.

If you want to create a template that deploys multiple workspaces in the same VNet, set this up manually (using the Azure Portal or CLI) and then [use the Azure portal to generate a template](#).

Workspace Resource Manager template

The Azure Resource Manager template used throughout this document can be found in the [microsoft.machinelearningservices/machine-learning-workspace-vnet](#) directory of the Azure quickstart templates GitHub repository.

This template creates the following Azure services:

- Azure Storage Account

- Azure Key Vault
- Azure Application Insights
- Azure Container Registry
- Azure Machine Learning workspace

The resource group is the container that holds the services. The various services are required by the Azure Machine Learning workspace.

The example template has two **required** parameters:

- The **location** where the resources will be created.

The template will use the location you select for most resources. The exception is the Application Insights service, which is not available in all of the locations that the other services are. If you select a location where it is not available, the service will be created in the South Central US location.

- The **workspaceName**, which is the friendly name of the Azure Machine Learning workspace.

NOTE

The workspace name is case-insensitive.

The names of the other services are generated randomly.

TIP

While the template associated with this document creates a new Azure Container Registry, you can also create a new workspace without creating a container registry. One will be created when you perform an operation that requires a container registry. For example, training or deploying a model.

You can also reference an existing container registry or storage account in the Azure Resource Manager template, instead of creating a new one. When doing so, you must either [use a managed identity](#) (preview), or [enable the admin account](#) for the container registry.

WARNING

Once an Azure Container Registry has been created for a workspace, do not delete it. Doing so will break your Azure Machine Learning workspace.

For more information on templates, see the following articles:

- [Author Azure Resource Manager templates](#)
- [Deploy an application with Azure Resource Manager templates](#)
- [Microsoft.MachineLearningServices resource types](#)

Deploy template

To deploy your template you have to create a resource group.

See the [Azure portal](#) section if you prefer using the graphical user interface.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az group create --name "examplegroup" --location "eastus"
```

Once your resource group is successfully created, deploy the template with the following command:

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
--name "exampledeployment" \
--resource-group "examplegroup" \
--template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.machinelearningservices/machine-learning-workspace-vnet/azuredeploy.json" \
--parameters workspaceName="exampleworkspace" location="eastus"
```

By default, all of the resources created as part of the template are new. However, you also have the option of using existing resources. By providing additional parameters to the template, you can use existing resources. For example, if you want to use an existing storage account set the **storageAccountOption** value to **existing** and provide the name of your storage account in the **storageAccountName** parameter.

IMPORTANT

If you want to use an existing Azure Storage account, it cannot be a premium account (Premium_LRS and Premium_GRS). It also cannot have a hierarchical namespace (used with Azure Data Lake Storage Gen2). Neither premium storage or hierarchical namespace are supported with the default storage account of the workspace. Neither premium storage or hierarchical namespaces are supported with the *default* storage account of the workspace. You can use premium storage or hierarchical namespace with *non-default* storage accounts.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
--name "exampledeployment" \
--resource-group "examplegroup" \
--template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.machinelearningservices/machine-learning-workspace-vnet/azuredeploy.json" \
--parameters workspaceName="exampleworkspace" \
location="eastus" \
storageAccountOption="existing" \
storageAccountName="existingstorageaccountname"
```

Deploy an encrypted workspace

The following example template demonstrates how to create a workspace with three settings:

- Enable high confidentiality settings for the workspace. This creates a new Cosmos DB instance.
- Enable encryption for the workspace.
- Uses an existing Azure Key Vault to retrieve customer-managed keys. Customer-managed keys are used to create a new Cosmos DB instance for the workspace.

IMPORTANT

Once a workspace has been created, you cannot change the settings for confidential data, encryption, key vault ID, or key identifiers. To change these values, you must create a new workspace using the new values.

For more information, see [Customer-managed keys](#).

IMPORTANT

There are some specific requirements your subscription must meet before using this template:

- You must have an existing Azure Key Vault that contains an encryption key.
- The Azure Key Vault must be in the same region where you plan to create the Azure Machine Learning workspace.
- You must specify the ID of the Azure Key Vault and the URI of the encryption key.

For steps on creating the vault and key, see [Configure customer-managed keys](#).

To get the values for the `cmk_keyvault` (ID of the Key Vault) and the `resource_cmk_uri` (key URI) parameters needed by this template, use the following steps:

1. To get the Key Vault ID, use the following command:

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az keyvault show --name <keyvault-name> --query 'id' --output tsv
```

This command returns a value similar to

```
/subscriptions/{subscription-guid}/resourceGroups/<resource-group-name>/providers/Microsoft.KeyVault/vaults/<keyvault-name>
```

2. To get the value for the URI for the customer managed key, use the following command:

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az keyvault key show --vault-name <keyvault-name> --name <key-name> --query 'key.kid' --output tsv
```

This command returns a value similar to <https://mykeyvault.vault.azure.net/keys/mykey/{guid}>.

IMPORTANT

Once a workspace has been created, you cannot change the settings for confidential data, encryption, key vault ID, or key identifiers. To change these values, you must create a new workspace using the new values.

To enable use of Customer Managed Keys, set the following parameters when deploying the template:

- `encryption_status` to Enabled.
- `cmk_keyvault` to the `cmk_keyvault` value obtained in previous steps.
- `resource_cmk_uri` to the `resource_cmk_uri` value obtained in previous steps.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
    --name "exampledeployment" \
    --resource-group "examplegroup" \
    --template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.machinelearningservices/machine-learning-workspace-
vnet/azuredeploy.json" \
    --parameters workspaceName="exampleworkspace" \
    location="eastus" \
    encryption_status="Enabled" \
    cmk_keyvault="/subscriptions/{subscription-guid}/resourceGroups/<resource-group-
name>/providers/Microsoft.KeyVault/vaults/<keyvault-name>" \
    resource_cmk_uri="https://mykeyvault.vault.azure.net/keys/mykey/{guid}" \
```

When using a customer-managed key, Azure Machine Learning creates a secondary resource group which contains the Cosmos DB instance. For more information, see [encryption at rest - Cosmos DB](#).

An additional configuration you can provide for your data is to set the **confidential_data** parameter to **true**. Doing so, does the following:

- Starts encrypting the local scratch disk for Azure Machine Learning compute clusters, providing you have not created any previous clusters in your subscription. If you have previously created a cluster in the subscription, open a support ticket to have encryption of the scratch disk enabled for your compute clusters.
- Cleans up the local scratch disk between jobs.
- Securely passes credentials for the storage account, container registry, and SSH account from the execution layer to your compute clusters by using key vault.
- Enables IP filtering to ensure the underlying batch pools cannot be called by any external services other than AzureMachineLearningService.

IMPORTANT

Once a workspace has been created, you cannot change the settings for confidential data, encryption, key vault ID, or key identifiers. To change these values, you must create a new workspace using the new values.

For more information, see [encryption at rest](#).

Deploy workspace behind a virtual network

By setting the `vnetOption` parameter value to either `new` or `existing`, you are able to create the resources used by a workspace behind a virtual network.

IMPORTANT

For container registry, only the 'Premium' sku is supported.

IMPORTANT

Application Insights does not support deployment behind a virtual network.

Only deploy workspace behind private endpoint

If your associated resources are not behind a virtual network, you can set the `privateEndpointType` parameter to `AutoApproval` or `ManualApproval` to deploy the workspace behind a private endpoint. This can be done for both new and existing workspaces. When updating an existing workspace, fill in the template parameters with the information from the existing workspace.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
    --name "exampledeployment" \
    --resource-group "examplegroup" \
    --template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.machinelearningservices/machine-learning-workspace-
vnet/azuredeploy.json" \
    --parameters workspaceName="exampleworkspace" \
    location="eastus" \
    privateEndpointType="AutoApproval"
```

Use a new virtual network

To deploy a resource behind a new virtual network, set the `vnetOption` to `new` along with the virtual network settings for the respective resource. The deployment below shows how to deploy a workspace with the storage account resource behind a new virtual network.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
    --name "exampledeployment" \
    --resource-group "examplegroup" \
    --template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.machinelearningservices/machine-learning-workspace-
vnet/azuredeploy.json" \
    --parameters workspaceName="exampleworkspace" \
    location="eastus" \
    vnetOption="new" \
    vnetName="examplevnet" \
    storageAccountBehindVNet="true" \
    privateEndpointType="AutoApproval"
```

Alternatively, you can deploy multiple or all dependent resources behind a virtual network.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
--name "exampledeployment" \
--resource-group "examplegroup" \
--template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.machinelearningservices/machine-learning-workspace-vnet/azuredploy.json" \
--parameters workspaceName="exampleworkspace" \
location="eastus" \
vnetOption="new" \
vnetName="examplevnet" \
storageAccountBehindVNet="true" \
keyVaultBehindVNet="true" \
containerRegistryBehindVNet="true" \
containerRegistryOption="new" \
containerRegistrySku="Premium" \
privateEndpointType="AutoApproval"
```

Use an existing virtual network & resources

To deploy a workspace with existing associated resources you have to set the **vnetOption** parameter to **existing** along with subnet parameters. However, you need to create service endpoints in the virtual network for each of the resources **before** deployment. Like with new virtual network deployments, you can have one or all of your resources behind a virtual network.

IMPORTANT

Subnet should have `Microsoft.Storage` service endpoint

IMPORTANT

Subnets do not allow creation of private endpoints. Disable private endpoint to enable subnet.

1. Enable service endpoints for the resources.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az network vnet subnet update --resource-group "examplegroup" --vnet-name "examplevnet" --name "examplesubnet" --service-endpoints "Microsoft.Storage"
az network vnet subnet update --resource-group "examplegroup" --vnet-name "examplevnet" --name "examplesubnet" --service-endpoints "Microsoft.KeyVault"
az network vnet subnet update --resource-group "examplegroup" --vnet-name "examplevnet" --name "examplesubnet" --service-endpoints "Microsoft.ContainerRegistry"
```

2. Deploy the workspace

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
--name "exampledeployment" \
--resource-group "examplegroup" \
--template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.machinelearningservices/machine-learning-workspace-vnet/azuredeploy.json" \
--parameters workspaceName="exampleworkspace" \
location="eastus" \
vnetOption="existing" \
vnetName="examplevnet" \
vnetResourceGroupName="examplegroup" \
storageAccountBehindVNet="true" \
keyVaultBehindVNet="true" \
containerRegistryBehindVNet="true" \
containerRegistryOption="new" \
containerRegistrySku="Premium" \
subnetName="examplesubnet" \
subnetOption="existing"
privateEndpointType="AutoApproval"
```

Use the Azure portal

1. Follow the steps in [Deploy resources from custom template](#). When you arrive at the **Select a template** screen, choose the **quickstarts** entry. When it appears, select the link labeled "Click here to open template repository". This link takes you to the `quickstarts` directory in the Azure quickstart templates repository.
2. In the list of quickstart templates, select `microsoft.machinelearningservices`. Finally, select `Deploy to Azure`.
3. When the template appears, provide the following required information and any other parameters depending on your deployment scenario.
 - Subscription: Select the Azure subscription to use for these resources.
 - Resource group: Select or create a resource group to contain the services.
 - Region: Select the Azure region where the resources will be created.
 - Workspace name: The name to use for the Azure Machine Learning workspace that will be created. The workspace name must be between 3 and 33 characters. It may only contain alphanumeric characters and '-'.
 - Location: Select the location where the resources will be created.
4. Select **Review + create**.
5. In the **Review + create** screen, agree to the listed terms and conditions and select **Create**.

For more information, see [Deploy resources from custom template](#).

Troubleshooting

Resource provider errors

When creating an Azure Machine Learning workspace, or a resource used by the workspace, you may receive an error similar to the following messages:

- `No registered resource provider found for location {location}`
- `The subscription is not registered to use namespace {resource-provider-namespace}`

Most resource providers are automatically registered, but not all. If you receive this message, you need to register the provider mentioned.

The following table contains a list of the resource providers required by Azure Machine Learning:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.MachineLearningServices	Creating the Azure Machine Learning workspace.
Microsoft.Storage	Azure Storage Account is used as the default storage for the workspace.
Microsoft.ContainerRegistry	Azure Container Registry is used by the workspace to build Docker images.
Microsoft.KeyVault	Azure Key Vault is used by the workspace to store secrets.
Microsoft.Notebooks/NotebookProxies	Integrated notebooks on Azure Machine Learning compute instance.
Microsoft.ContainerService	If you plan on deploying trained models to Azure Kubernetes Services.

If you plan on using a customer-managed key with Azure Machine Learning, then the following service providers must be registered:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.DocumentDB/databaseAccounts	Azure CosmosDB instance that logs metadata for the workspace.
Microsoft.Search/searchServices	Azure Search provides indexing capabilities for the workspace.

For information on registering resource providers, see [Resolve errors for resource provider registration](#).

Azure Key Vault access policy and Azure Resource Manager templates

When you use an Azure Resource Manager template to create the workspace and associated resources (including Azure Key Vault), multiple times. For example, using the template multiple times with the same parameters as part of a continuous integration and deployment pipeline.

Most resource creation operations through templates are idempotent, but Key Vault clears the access policies each time the template is used. Clearing the access policies breaks access to the Key Vault for any existing workspace that is using it. For example, Stop/Create functionalities of Azure Notebooks VM may fail.

To avoid this problem, we recommend one of the following approaches:

- Do not deploy the template more than once for the same parameters. Or delete the existing resources before using the template to recreate them.
- Examine the Key Vault access policies and then use these policies to set the `accessPolicies` property of the template. To view the access policies, use the following Azure CLI command:

```
az keyvault show --name mykeyvault --resource-group myresourcegroup --query properties.accessPolicies
```

For more information on using the `accessPolicies` section of the template, see the [AccessPolicyEntry object reference](#).

- Check if the Key Vault resource already exists. If it does, do not recreate it through the template. For

example, to use the existing Key Vault instead of creating a new one, make the following changes to the template:

- Add a parameter that accepts the ID of an existing Key Vault resource:

```
"keyVaultId":{  
    "type": "string",  
    "metadata": {  
        "description": "Specify the existing Key Vault ID."  
    }  
}
```

- Remove the section that creates a Key Vault resource:

```
{  
    "type": "Microsoft.KeyVault/vaults",  
    "apiVersion": "2018-02-14",  
    "name": "[variables('keyVaultName')]",  
    "location": "[parameters('location')]",  
    "properties": {  
        "tenantId": "[variables('tenantId')]",  
        "sku": {  
            "name": "standard",  
            "family": "A"  
        },  
        "accessPolicies": [  
        ]  
    }  
},
```

- Remove the `"[resourceId('Microsoft.KeyVault/vaults', variables('keyVaultName'))]"`, line from the `dependsOn` section of the workspace. Also Change the `keyVault` entry in the `properties` section of the workspace to reference the `keyVaultId` parameter:

```
{  
    "type": "Microsoft.MachineLearningServices/workspaces",  
    "apiVersion": "2019-11-01",  
    "name": "[parameters('workspaceName')]",  
    "location": "[parameters('location')]",  
    "dependsOn": [  
        "[resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName'))]",  
        "[resourceId('Microsoft.Insights/components', variables('applicationInsightsName'))]"  
    ],  
    "identity": {  
        "type": "systemAssigned"  
    },  
    "sku": {  
        "tier": "[parameters('sku')]",  
        "name": "[parameters('sku')]"  
    },  
    "properties": {  
        "friendlyName": "[parameters('workspaceName')]",  
        "keyVault": "[parameters('keyVaultId')]",  
        "applicationInsights": "  
[resourceId('Microsoft.Insights/components', variables('applicationInsightsName'))]",  
        "storageAccount": "  
[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]"  
    }  
}
```

After these changes, you can specify the ID of the existing Key Vault resource when running the template.

The template will then reuse the Key Vault by setting the `keyVault` property of the workspace to its ID.

To get the ID of the Key Vault, you can reference the output of the original template job or use the Azure CLI. The following command is an example of using the Azure CLI to get the Key Vault resource ID:

```
az keyvault show --name mykeyvault --resource-group myresourcegroup --query id
```

This command returns a value similar to the following text:

```
/subscriptions/{subscription-guid}/resourceGroups/myresourcegroup/providers/Microsoft.KeyVault/vaults/mykeyvault
```

Next steps

- [Deploy resources with Resource Manager templates and Resource Manager REST API.](#)
- [Creating and deploying Azure resource groups through Visual Studio.](#)
- [For other templates related to Azure Machine Learning, see the Azure Quickstart Templates repository.](#)
- [How to use workspace diagnostics.](#)
- [Move an Azure Machine Learning workspace to another subscription.](#)

Manage Azure Machine Learning workspaces using Terraform

9/22/2022 • 10 minutes to read • [Edit Online](#)

In this article, you learn how to create and manage an Azure Machine Learning workspace using Terraform configuration files. [Terraform](#)'s template-based configuration files enable you to define, create, and configure Azure resources in a repeatable and predictable manner. Terraform tracks resource state and is able to clean up and destroy resources.

A Terraform configuration is a document that defines the resources that are needed for a deployment. It may also specify deployment variables. Variables are used to provide input values when using the configuration.

Prerequisites

- An [Azure subscription](#). If you don't have one, try the [free or paid version of Azure Machine Learning](#).
- An installed version of the [Azure CLI](#).
- Configure Terraform: follow the directions in this article and the [Terraform and configure access to Azure](#) article.

Limitations

- When creating a new workspace, you can either automatically create services needed by the workspace or use existing services. If you want to use [existing services from a different Azure subscription](#) than the workspace, you must register the Azure Machine Learning namespace in the subscription that contains those services. For example, creating a workspace in subscription A that uses a storage account from subscription B, the Azure Machine Learning namespace must be registered in subscription B before you can use the storage account with the workspace.

The resource provider for Azure Machine Learning is [Microsoft.MachineLearningServices](#). For information on how to see if it is registered and how to register it, see the [Azure resource providers and types](#) article.

IMPORTANT

This only applies to resources provided during workspace creation; Azure Storage Accounts, Azure Container Register, Azure Key Vault, and Application Insights.

TIP

An Azure Application Insights instance is created when you create the workspace. You can delete the Application Insights instance after cluster creation if you want. Deleting it limits the information gathered from the workspace, and may make it more difficult to troubleshoot problems. [If you delete the Application Insights instance created by the workspace, you cannot re-create it without deleting and recreating the workspace.](#)

For more information on using this Application Insights instance, see [Monitor and collect data from Machine Learning web service endpoints](#).

Declare the Azure provider

Create the Terraform configuration file that declares the Azure provider:

1. Create a new file named `main.tf`. If working with Azure Cloud Shell, use bash:

```
code main.tf
```

2. Paste the following code into the editor:

main.tf:

```
terraform {  
    required_version = ">=1.0"  
  
    required_providers {  
        azurerm = {  
            source  = "hashicorp/azurerm"  
            version = "=2.76.0"  
        }  
    }  
}  
  
provider "azurerm" {  
    features {}  
}  
  
data "azurerm_client_config" "current" {}  
  
resource "azurerm_resource_group" "default" {  
    name      = "rg-${var.name}-${var.environment}"  
    location = var.location  
}
```

3. Save the file (**<Ctrl>S**) and exit the editor (**<Ctrl>Q**).

Deploy a workspace

The following Terraform configurations can be used to create an Azure Machine Learning workspace. When you create an Azure Machine Learning workspace, various other services are required as dependencies. The template also specifies these [associated resources to the workspace](#). Depending on your needs, you can choose to use the template that creates resources with either public or private network connectivity.

- [Public network connectivity](#)
- [Private network connectivity](#)

Some resources in Azure require globally unique names. Before deploying your resources using the following templates, set the `name` variable to a value that is unique.

variables.tf:

```
variable "name" {
  type      = string
  description = "Name of the deployment"
}

variable "environment" {
  type      = string
  description = "Name of the environment"
  default    = "dev"
}

variable "location" {
  type      = string
  description = "Location of the resources"
  default    = "East US"
}
```

workspace.tf:

```

# Dependent resources for Azure Machine Learning
resource "azurerm_application_insights" "default" {
    name          = "appi-${var.name}-${var.environment}"
    location      = azurerm_resource_group.default.location
    resource_group_name = azurerm_resource_group.default.name
    application_type = "web"
}

resource "azurerm_key_vault" "default" {
    name          = "kv-${var.name}-${var.environment}"
    location      = azurerm_resource_group.default.location
    resource_group_name = azurerm_resource_group.default.name
    tenant_id     = data.azurerm_client_config.current.tenant_id
    sku_name      = "premium"
    purge_protection_enabled = false
}

resource "azurerm_storage_account" "default" {
    name          = "st${var.name}${var.environment}"
    location      = azurerm_resource_group.default.location
    resource_group_name = azurerm_resource_group.default.name
    account_tier   = "Standard"
    account_replication_type = "GRS"
}

resource "azurerm_container_registry" "default" {
    name          = "cr${var.name}${var.environment}"
    location      = azurerm_resource_group.default.location
    resource_group_name = azurerm_resource_group.default.name
    sku           = "Premium"
    admin_enabled = true
}

# Machine Learning workspace
resource "azurerm_machine_learning_workspace" "default" {
    name          = "mlw-${var.name}-${var.environment}"
    location      = azurerm_resource_group.default.location
    resource_group_name = azurerm_resource_group.default.name
    application_insights_id = azurerm_application_insights.default.id
    key_vault_id   = azurerm_key_vault.default.id
    storage_account_id = azurerm_storage_account.default.id
    container_registry_id = azurerm_container_registry.default.id

    identity {
        type = "SystemAssigned"
    }
}

```

Troubleshooting

Resource provider errors

When creating an Azure Machine Learning workspace, or a resource used by the workspace, you may receive an error similar to the following messages:

- No registered resource provider found for location {location}
- The subscription is not registered to use namespace {resource-provider-namespace}

Most resource providers are automatically registered, but not all. If you receive this message, you need to register the provider mentioned.

The following table contains a list of the resource providers required by Azure Machine Learning:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.MachineLearningServices	Creating the Azure Machine Learning workspace.
Microsoft.Storage	Azure Storage Account is used as the default storage for the workspace.
Microsoft.ContainerRegistry	Azure Container Registry is used by the workspace to build Docker images.
Microsoft.KeyVault	Azure Key Vault is used by the workspace to store secrets.
Microsoft.Notebooks/NotebookProxies	Integrated notebooks on Azure Machine Learning compute instance.
Microsoft.ContainerService	If you plan on deploying trained models to Azure Kubernetes Services.

If you plan on using a customer-managed key with Azure Machine Learning, then the following service providers must be registered:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.DocumentDB/databaseAccounts	Azure CosmosDB instance that logs metadata for the workspace.
Microsoft.Search/searchServices	Azure Search provides indexing capabilities for the workspace.

For information on registering resource providers, see [Resolve errors for resource provider registration](#).

Next steps

- To learn more about Terraform support on Azure, see [Terraform on Azure documentation](#).
- For details on the Terraform Azure provider and Machine Learning module, see [Terraform Registry Azure Resource Manager Provider](#).
- To find "quick start" template examples for Terraform, see [Azure Terraform QuickStart Templates](#):
 - [101: Machine learning workspace and compute](#) – the minimal set of resources needed to get started with Azure ML.
 - [201: Machine learning workspace, compute, and a set of network components for network isolation](#) – all resources that are needed to create a production-pilot environment for use with HBI data.
 - [202: Similar to 201, but with the option to bring existing network components..](#)
 - [301: Machine Learning workspace \(Secure Hub and Spoke with Firewall\)](#).
- To learn more about network configuration options, see [Secure Azure Machine Learning workspace resources using virtual networks \(VNets\)](#).
- For alternative Azure Resource Manager template-based deployments, see [Deploy resources with Resource Manager templates and Resource Manager REST API](#).
- For information on how to keep your Azure ML up to date with the latest security updates, see [Vulnerability management](#).

Create, run, and delete Azure ML resources using REST

9/22/2022 • 10 minutes to read • [Edit Online](#)

There are several ways to manage your Azure ML resources. You can use the [portal](#), [command-line interface](#), or [Python SDK](#). Or, you can choose the REST API. The REST API uses HTTP verbs in a standard way to create, retrieve, update, and delete resources. The REST API works with any language or tool that can make HTTP requests. REST's straightforward structure often makes it a good choice in scripting environments and for MLOps automation.

In this article, you learn how to:

- Retrieve an authorization token
- Create a properly-formatted REST request using service principal authentication
- Use GET requests to retrieve information about Azure ML's hierarchical resources
- Use PUT and POST requests to create and modify resources
- Use PUT requests to create Azure ML workspaces
- Use DELETE requests to clean up resources

Prerequisites

- An [Azure subscription](#) for which you have administrative rights. If you don't have such a subscription, try the [free or paid personal subscription](#)
- An [Azure Machine Learning Workspace](#).
- Administrative REST requests use service principal authentication. Follow the steps in [Set up authentication for Azure Machine Learning resources and workflows](#) to create a service principal in your workspace
- The `curl` utility. The `curl` program is available in the [Windows Subsystem for Linux](#) or any UNIX distribution.
In PowerShell, `curl` is an alias for `Invoke-WebRequest` and `curl -d "key=val" -X POST uri` becomes
`Invoke-WebRequest -Body "key=val" -Method POST -Uri uri`.

Retrieve a service principal authentication token

Administrative REST requests are authenticated with an OAuth2 implicit flow. This authentication flow uses a token provided by your subscription's service principal. To retrieve this token, you'll need:

- Your tenant ID (identifying the organization to which your subscription belongs)
- Your client ID (which will be associated with the created token)
- Your client secret (which you should safeguard)

You should have these values from the response to the creation of your service principal. Getting these values is discussed in [Set up authentication for Azure Machine Learning resources and workflows](#). If you're using your company subscription, you might not have permission to create a service principal. In that case, you should use either a [free or paid personal subscription](#).

To retrieve a token:

1. Open a terminal window
2. Enter the following code at the command line
3. Substitute your own values for `<YOUR-TENANT-ID>`, `<YOUR-CLIENT-ID>`, and `<YOUR-CLIENT-SECRET>`. Throughout

this article, strings surrounded by angle brackets are variables you'll have to replace with your own appropriate values.

4. Run the command

```
curl -X POST https://login.microsoftonline.com/<YOUR-TENANT-ID>/oauth2/token \
-d "grant_type=client_credentials&resource=https%3A%2F%2Fmanagement.azure.com%2F&client_id=<YOUR-CLIENT-
ID>&client_secret=<YOUR-CLIENT-SECRET>" \
```

The response should provide an access token good for one hour:

```
{
  "token_type": "Bearer",
  "expires_in": "3599",
  "ext_expires_in": "3599",
  "expires_on": "1578523094",
  "not_before": "1578519194",
  "resource": "https://management.azure.com/",
  "access_token": "YOUR-ACCESS-TOKEN"
}
```

Make note of the token, as you'll use it to authenticate all administrative requests. You'll do so by setting an Authorization header in all requests:

```
curl -h "Authorization:Bearer <YOUR-ACCESS-TOKEN>" ...more args...
```

NOTE

The value starts with the string "Bearer " including a single space before you add the token.

Get a list of resource groups associated with your subscription

To retrieve the list of resource groups associated with your subscription, run:

```
curl https://management.azure.com/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups?api-version=2021-04-01
-H "Authorization:Bearer <YOUR-ACCESS-TOKEN>"
```

Across Azure, many REST APIs are published. Each service provider updates their API on their own cadence, but does so without breaking existing programs. The service provider uses the `api-version` argument to ensure compatibility.

IMPORTANT

The `api-version` argument varies from service to service. For the Machine Learning Service, for instance, the current API version is `2022-05-01`. To find the latest API version for other Azure services, see the [Azure REST API reference](#) for the specific service.

All REST calls should set the `api-version` argument to the expected value. You can rely on the syntax and semantics of the specified version even as the API continues to evolve. If you send a request to a provider without the `api-version` argument, the response will contain a human-readable list of supported values.

The above call will result in a compacted JSON response of the form:

```
{  
    "value": [  
        {  
            "id": "/subscriptions/12345abc-abbc-1b2b-1234-57ab575a5a5a/resourceGroups/RG1",  
            "name": "RG1",  
            "type": "Microsoft.Resources/resourceGroups",  
            "location": "westus2",  
            "properties": {  
                "provisioningState": "Succeeded"  
            }  
        },  
        {  
            "id": "/subscriptions/12345abc-abbc-1b2b-1234-57ab575a5a5a/resourceGroups/RG2",  
            "name": "RG2",  
            "type": "Microsoft.Resources/resourceGroups",  
            "location": "eastus",  
            "properties": {  
                "provisioningState": "Succeeded"  
            }  
        }  
    ]  
}
```

Drill down into workspaces and their resources

To retrieve the set of workspaces in a resource group, run the following, replacing <YOUR-SUBSCRIPTION-ID>, <YOUR-RESOURCE-GROUP>, and <YOUR-ACCESS-TOKEN>:

```
curl https://management.azure.com/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-  
GROUP>/providers/Microsoft.MachineLearningServices/workspaces/?api-version=2022-05-01 \  
-H "Authorization:Bearer <YOUR-ACCESS-TOKEN>"
```

Again you'll receive a JSON list, this time containing a list, each item of which details a workspace:

```
{
  "id": "/subscriptions/12345abc-abbc-1b2b-1234-
57ab575a5a5a/resourceGroups/DeepLearningResourceGroup/providers/Microsoft.MachineLearningServices/workspaces
/my-workspace",
  "name": "my-workspace",
  "type": "Microsoft.MachineLearningServices/workspaces",
  "location": "centralus",
  "tags": {},
  "etag": null,
  "properties": {
    "friendlyName": "",
    "description": "",
    "creationTime": "2020-01-03T19:56:09.7588299+00:00",
    "storageAccount": "/subscriptions/12345abc-abbc-1b2b-1234-
57ab575a5a5a/resourcegroups/DeepLearningResourceGroup/providers/microsoft.storage/storageaccounts/myworkspac
e0275623111",
    "containerRegistry": null,
    "keyVault": "/subscriptions/12345abc-abbc-1b2b-1234-
57ab575a5a5a/resourcegroups/DeepLearningResourceGroup/providers/microsoft.keyvault/vaults/myworkspace2525649
324",
    "applicationInsights": "/subscriptions/12345abc-abbc-1b2b-1234-
57ab575a5a5a/resourcegroups/DeepLearningResourceGroup/providers/microsoft.insights/components/myworkspace205
3523719",
    "hbiWorkspace": false,
    "workspaceId": "cba12345-abab-abab-abab-ababab123456",
    "subscriptionState": null,
    "subscriptionStatusChangeTimeStampUtc": null,
    "discoveryUrl": "https://centralus.experiments.azureml.net/discovery"
  },
  "identity": {
    "type": "SystemAssigned",
    "principalId": "abcdef1-abab-1234-1234-abababab123456",
    "tenantId": "1fedcba-abab-1234-1234-abababab123456"
  },
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  }
}
```

To work with resources within a workspace, you'll switch from the general [management.azure.com](#) server to a REST API server specific to the location of the workspace. Note the value of the `discoveryUrl` key in the above JSON response. If you GET that URL, you'll receive a response something like:

```
{
  "api": "https://centralus.api.azureml.ms",
  "catalog": "https://catalog.cortanaanalytics.com",
  "experimentation": "https://centralus.experiments.azureml.net",
  "gallery": "https://gallery.cortanaintelligence.com/project",
  "history": "https://centralus.experiments.azureml.net",
  "hyperdrive": "https://centralus.experiments.azureml.net",
  "labeling": "https://centralus.experiments.azureml.net",
  "modelmanagement": "https://centralus.modelmanagement.azureml.net",
  "pipelines": "https://centralus.aether.ms",
  "studiocoreservices": "https://centralus.studioservice.azureml.com"
}
```

The value of the `api` response is the URL of the server that you'll use for more requests. To list experiments, for instance, send the following command. Replace `REGIONAL-API-SERVER` with the value of the `api` response (for instance, `centralus.api.azureml.ms`). Also replace `YOUR-SUBSCRIPTION-ID`, `YOUR-RESOURCE-GROUP`, `YOUR-WORKSPACE-NAME`, and `YOUR-ACCESS-TOKEN` as usual:

```
curl https://<REGIONAL-API-SERVER>/history/v1.0/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>/ \
providers/Microsoft.MachineLearningServices/workspaces/<YOUR-WORKSPACE-NAME>/experiments?api-version=2022-05-01 \
-H "Authorization:Bearer <YOUR-ACCESS-TOKEN>"
```

Similarly, to retrieve registered models in your workspace, send:

```
curl https://<REGIONAL-API-SERVER>/modelmanagement/v1.0/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>/ \
providers/Microsoft.MachineLearningServices/workspaces/<YOUR-WORKSPACE-NAME>/models?api-version=2022-05-01 \
-H "Authorization:Bearer <YOUR-ACCESS-TOKEN>"
```

Notice that to list experiments the path begins with `history/v1.0` while to list models, the path begins with `modelmanagement/v1.0`. The REST API is divided into several operational groups, each with a distinct path.

AREA	PATH
Artifacts	/rest/api/azureml
Data stores	/azure/machine-learning/how-to-access-data
Hyperparameter tuning	hyperdrive/v1.0/
Models	modelmanagement/v1.0/
Run history	execution/v1.0/ and history/v1.0/

You can explore the REST API using the general pattern of:

URL COMPONENT	EXAMPLE
https://	
REGIONAL-API-SERVER/	centralus.api.azureml.ms/
operations-path/	history/v1.0/
subscriptions/YOUR-SUBSCRIPTION-ID/	subscriptions/abcde123-abab-abab-1234-0123456789abc/
resourceGroups/YOUR-RESOURCE-GROUP/	resourceGroups/MyResourceGroup/
providers/operation-provider/	providers/Microsoft.MachineLearningServices/
provider-resource-path/	workspaces/MyWorkspace/experiments/FirstExperiment/runs/1/
operations-endpoint/	artifacts/metadata/

Create and modify resources using PUT and POST requests

In addition to resource retrieval with the GET verb, the REST API supports the creation of all the resources necessary to train, deploy, and monitor ML solutions.

Training and running ML models require compute resources. You can list the compute resources of a workspace with:

```
curl https://management.azure.com/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>/providers/Microsoft.MachineLearningServices/workspaces/<YOUR-WORKSPACE-NAME>/computes?api-version=2022-05-01
\ -H "Authorization:Bearer <YOUR-ACCESS-TOKEN>"
```

To create or overwrite a named compute resource, you'll use a PUT request. In the following, in addition to the now-familiar replacements of `<YOUR-SUBSCRIPTION-ID>`, `<YOUR-RESOURCE-GROUP>`, `<YOUR-WORKSPACE-NAME>`, and `<YOUR-ACCESS-TOKEN>`, replace `<YOUR-COMPUTE-NAME>`, and values for `location`, `vmSize`, `vmPriority`, `scaleSettings`, `adminUserName`, and `adminUserPassword`. As specified in the reference at [Machine Learning Compute - Create Or Update SDK Reference](#), the following command creates a dedicated, single-node Standard_D1 (a basic CPU compute resource) that will scale down after 30 minutes:

```
curl -X PUT \
  'https://management.azure.com/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>/providers/Microsoft.MachineLearningServices/workspaces/<YOUR-WORKSPACE-NAME>/computes/<YOUR-COMPUTE-NAME>?api-version=2022-05-01' \
  -H 'Authorization:Bearer <YOUR-ACCESS-TOKEN>' \
  -H 'Content-Type: application/json' \
  -d '{
    "location": "eastus",
    "properties": {
      "computeType": "AmlCompute",
      "properties": {
        "vmSize": "Standard_D1",
        "vmPriority": "Dedicated",
        "scaleSettings": {
          "maxNodeCount": 1,
          "minNodeCount": 0,
          "nodeIdleTimeBeforeScaleDown": "PT30M"
        }
      }
    },
    "userAccountCredentials": {
      "adminUserName": "<ADMIN_USERNAME>",
      "adminUserPassword": "<ADMIN_PASSWORD>"
    }
}'
```

NOTE

In Windows terminals you may have to escape the double-quote symbols when sending JSON data. That is, text such as `"location"` becomes `\\"location\\"`.

A successful request will get a `201 created` response, but note that this response simply means that the provisioning process has begun. You'll need to poll (or use the portal) to confirm its successful completion.

Create a workspace using REST

Every Azure ML workspace has a dependency on four other Azure resources: an Azure Container Registry resource, Azure Key Vault, Azure Application Insights, and an Azure Storage account. You can't create a workspace until these resources exist. Consult the REST API reference for the details of creating each such resource.

To create a workspace, PUT a call similar to the following to `management.azure.com`. While this call requires you to

set a large number of variables, it's structurally identical to other calls that this article has discussed.

```
curl -X PUT \
'https://management.azure.com/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\ \
/providers/Microsoft.MachineLearningServices/workspaces/<YOUR-NEW-WORKSPACE-NAME>?api-version=2022-05-01' \
-H 'Authorization: Bearer <YOUR-ACCESS-TOKEN>' \
-H 'Content-Type: application/json' \
-d '{
  "location": "AZURE-LOCATION",
  "identity" : {
    "type" : "systemAssigned"
  },
  "properties": {
    "friendlyName" : "<YOUR-WORKSPACE-FRIENDLY-NAME>",
    "description" : "<YOUR-WORKSPACE-DESCRIPTION>",
    "containerRegistry" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\ \
providers/Microsoft.ContainerRegistry/registries/<YOUR-REGISTRY-NAME>",
    "keyVault" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\ \
/providers/Microsoft.Keyvault/vaults/<YOUR-KEYVAULT-NAME>",
    "applicationInsights" : "subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\ \
providers/Microsoft.insights/components/<YOUR-APPLICATION-INSIGHTS-NAME>",
    "storageAccount" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\ \
providers/Microsoft.Storage/storageAccounts/<YOUR-STORAGE-ACCOUNT-NAME>"
  }
}'
```

You should receive a `202 Accepted` response and, in the returned headers, a `Location` URI. You can GET this URI for information on the deployment, including helpful debugging information if there's a problem with one of your dependent resources (for instance, if you forgot to enable admin access on your container registry).

Create a workspace using a user-assigned managed identity

When creating workspace, you can specify a user-assigned managed identity that will be used to access the associated resources: ACR, KeyVault, Storage, and App Insights. To create a workspace with user-assigned managed identity, use the below request body.

```

curl -X PUT \
  'https://management.azure.com/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\ 
/providers/Microsoft.MachineLearningServices/workspaces/<YOUR-NEW-WORKSPACE-NAME>?api-version=2022-05-01' \
-H 'Authorization: Bearer <YOUR-ACCESS-TOKEN>' \
-H 'Content-Type: application/json' \
-d '{
  "location": "AZURE-LOCATION",
  "identity": {
    "type": "SystemAssigned,UserAssigned",
    "userAssignedIdentities": {
      "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\\ 
providers/Microsoft.ManagedIdentity/userAssignedIdentities/<YOUR-MANAGED-IDENTITY>": {}
    }
  },
  "properties": {
    "friendlyName" : "<YOUR-WORKSPACE-FRIENDLY-NAME>",
    "description" : "<YOUR-WORKSPACE-DESCRIPTION>",
    "containerRegistry" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\\ 
providers/Microsoft.ContainerRegistry/registries/<YOUR-REGISTRY-NAME>",
    "keyVault" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\\ 
/providers/Microsoft.Keyvault/vaults/<YOUR-KEYVAULT-NAME>",
    "applicationInsights" : "subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\\ 
providers/Microsoft.insights/components/<YOUR-APPLICATION-INSIGHTS-NAME>",
    "storageAccount" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP>\\ 
providers/Microsoft.Storage/storageAccounts/<YOUR-STORAGE-ACCOUNT-NAME>"
  }
}'

```

Create a workspace using customer-managed encryption keys

By default, metadata for the workspace is stored in an Azure Cosmos DB instance that Microsoft maintains. This data is encrypted using Microsoft-managed keys. Instead of using the Microsoft-managed key, you can also provide your own key. Doing so creates an [another set of resources](#) in your Azure subscription to store your data.

To create a workspace that uses your keys for encryption, you need to meet the following prerequisites:

- The Azure Machine Learning service principal must have contributor access to your Azure subscription.
- You must have an existing Azure Key Vault that contains an encryption key.
- The Azure Key Vault must exist in the same Azure region where you'll create the Azure Machine Learning workspace.
- The Azure Key Vault must have soft delete and purge protection enabled to protect against data loss if there was accidental deletion.
- You must have an access policy in Azure Key Vault that grants get, wrap, and unwrap access to the Azure Cosmos DB application.

To create a workspace that uses a user-assigned managed identity and customer-managed keys for encryption, use the below request body. When using a user-assigned managed identity for the workspace, also set the `userAssignedIdentity` property to the resource ID of the managed identity.

```

curl -X PUT \
  'https://management.azure.com/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP> \
/providers/Microsoft.MachineLearningServices/workspaces/<YOUR-NEW-WORKSPACE-NAME>?api-version=2022-05-01' \
-H 'Authorization: Bearer <YOUR-ACCESS-TOKEN>' \
-H 'Content-Type: application/json' \
-d '{
  "location": "eastus2euap",
  "identity": {
    "type": "SystemAssigned"
  },
  "properties": {
    "friendlyName": "<YOUR-WORKSPACE-FRIENDLY-NAME>",
    "description": "<YOUR-WORKSPACE-DESCRIPTION>",
    "containerRegistry" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP> \
/providers/Microsoft.ContainerRegistry/registries/<YOUR-REGISTRY-NAME>",
    "keyVault" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP> \
/providers/Microsoft.Keyvault/vaults/<YOUR-KEYVAULT-NAME>",
    "applicationInsights" : "subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP> \
/providers/Microsoft.insights/components/<YOUR-APPLICATION-INSIGHTS-NAME>",
    "storageAccount" : "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP> \
/providers/Microsoft.Storage/storageAccounts/<YOUR-STORAGE-ACCOUNT-NAME>",
    "encryption": {
      "status": "Enabled",
      "identity": {
        "userAssignedIdentity": null
      },
      "keyVaultProperties": {
        "keyVaultArmId": "/subscriptions/<YOUR-SUBSCRIPTION-ID>/resourceGroups/<YOUR-RESOURCE-GROUP> \
/providers/Microsoft.KeyVault/vaults/<YOUR-VAULT>",
        "keyIdentifier": "https://<YOUR-VAULT>.vault.azure.net/keys/<YOUR-KEY>/<YOUR-KEY-VERSION>",
        "identityClientId": ""
      }
    },
    "hbiWorkspace": false
  }
}'

```

Delete resources you no longer need

Some, but not all, resources support the DELETE verb. Check the [API Reference](#) before committing to the REST API for deletion use-cases. To delete a model, for instance, you can use:

```

curl
-X DELETE \
'https://<REGIONAL-API-SERVER>/modelmanagement/v1.0/subscriptions/<YOUR-SUBSCRIPTION- \
ID>/resourceGroups/<YOUR-RESOURCE-GROUP>/providers/Microsoft.MachineLearningServices/workspaces/<YOUR- \
WORKSPACE-NAME>/models/<YOUR-MODEL-ID>?api-version=2022-05-01' \
-H 'Authorization: Bearer <YOUR-ACCESS-TOKEN>'

```

Troubleshooting

Resource provider errors

When creating an Azure Machine Learning workspace, or a resource used by the workspace, you may receive an error similar to the following messages:

- No registered resource provider found for location {location}
- The subscription is not registered to use namespace {resource-provider-namespace}

Most resource providers are automatically registered, but not all. If you receive this message, you need to register the provider mentioned.

The following table contains a list of the resource providers required by Azure Machine Learning:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.MachineLearningServices	Creating the Azure Machine Learning workspace.
Microsoft.Storage	Azure Storage Account is used as the default storage for the workspace.
Microsoft.ContainerRegistry	Azure Container Registry is used by the workspace to build Docker images.
Microsoft.KeyVault	Azure Key Vault is used by the workspace to store secrets.
Microsoft.Notebooks/NotebookProxies	Integrated notebooks on Azure Machine Learning compute instance.
Microsoft.ContainerService	If you plan on deploying trained models to Azure Kubernetes Services.

If you plan on using a customer-managed key with Azure Machine Learning, then the following service providers must be registered:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.DocumentDB/databaseAccounts	Azure CosmosDB instance that logs metadata for the workspace.
Microsoft.Search/searchServices	Azure Search provides indexing capabilities for the workspace.

For information on registering resource providers, see [Resolve errors for resource provider registration](#).

Moving the workspace

WARNING

Moving your Azure Machine Learning workspace to a different subscription, or moving the owning subscription to a new tenant, is not supported. Doing so may cause errors.

Deleting the Azure Container Registry

The Azure Machine Learning workspace uses Azure Container Registry (ACR) for some operations. It will automatically create an ACR instance when it first needs one.

WARNING

Once an Azure Container Registry has been created for a workspace, do not delete it. Doing so will break your Azure Machine Learning workspace.

Next steps

- Explore the complete [AzureML REST API reference](#).
- Learn how to use the designer to [Predict automobile price with the designer](#).
- Explore [Azure Machine Learning with Jupyter notebooks](#).

Move Azure Machine Learning workspaces between subscriptions (preview)

9/22/2022 • 3 minutes to read • [Edit Online](#)

As the requirements of your machine learning application change, you may need to move your workspace to a different Azure subscription. For example, you may need to move the workspace in the following situations:

- Promote workspace from test subscription to production subscription.
- Change the design and architecture of your application.
- Move workspace to a subscription with more available quota.
- Move workspace to a subscription with different cost center.

Moving the workspace enables you to migrate the workspace and its contents as a single, automated step. The following table describes the workspace contents that are moved:

WORKSPACE CONTENTS	MOVED WITH WORKSPACE
Datastores	Yes
Datasets	No
Experiment jobs	Yes
Environments	Yes
Models and other assets stored in the workspace	Yes
Compute resources	No
Endpoints	No

IMPORTANT

Workspace move is currently in public preview. This preview is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Prerequisites

- An Azure Machine Learning workspace in the source subscription. For more information, see [Create workspace resources](#).
- You must have permissions to manage resources in both source and target subscriptions. For example, Contributor or Owner role at the **subscription** level. For more information on roles, see [Azure roles](#)
- The destination subscription must be registered for required resource providers. The following table contains a list of the resource providers required by Azure Machine Learning:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.MachineLearningServices	Creating the Azure Machine Learning workspace.
Microsoft.Storage	Azure Storage Account is used as the default storage for the workspace.
Microsoft.ContainerRegistry	Azure Container Registry is used by the workspace to build Docker images.
Microsoft.KeyVault	Azure Key Vault is used by the workspace to store secrets.
Microsoft.Notebooks/NotebookProxies	Integrated notebooks on Azure Machine Learning compute instance.
Microsoft.ContainerService	If you plan on deploying trained models to Azure Kubernetes Services.

If you plan on using a customer-managed key with Azure Machine Learning, then the following service providers must be registered:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.DocumentDB/databaseAccounts	Azure Cosmos DB instance that logs metadata for the workspace.
Microsoft.Search/searchServices	Azure Search provides indexing capabilities for the workspace.

For information on registering resource providers, see [Resolve errors for resource provider registration](#).

- The [Azure CLI](#).

TIP

The move operation does not use the Azure CLI extension for machine learning.

Limitations

- Workspace move is not meant for replicating workspaces, or moving individual assets such as models or datasets from one workspace to another.
- Workspace move doesn't support migration across Azure regions or Azure Active Directory tenants.
- The workspace mustn't be in use during the move operation. Verify that all experiment jobs, data profiling jobs, and labeling projects have completed. Also verify that inference endpoints aren't being invoked.
- The workspace will become unavailable during the move.
- Before to the move, you must delete or detach computes and inference endpoints from the workspace.
- Datastores may still show the old subscription information after the move.

Prepare and validate the move

1. In Azure CLI, set the subscription to that of your origin workspace

```
az account set -s origin-sub-id
```

2. Verify that the origin workspace isn't being used. Check that any experiment jobs, data profiling jobs, or labeling projects have completed. Also verify that inferencing endpoints aren't being invoked.
3. Delete or detach any computes from the workspace, and delete any inferencing endpoints. Moving computes and endpoints isn't supported. Also note that the workspace will become unavailable during the move.
4. Create a destination resource group in the new subscription. This resource group will contain the workspace after the move. The destination must be in the same region as the origin.

```
az group create -g destination-rg -l my-region --subscription destination-sub-id
```

5. The following command demonstrates how to validate the move operation for workspace. You can include associated resources such as storage account, container registry, key vault, and application insights into the move by adding them to the `resources` list. The validation may take several minutes. In this command, `origin-rg` is the origin resource group, while `destination-rg` is the destination. The subscription IDs are represented by `origin-sub-id` and `destination-sub-id`, while the workspace is `origin-workspace-name`:

```
az resource invoke-action --action validateMoveResources --ids "/subscriptions/origin-sub-id/resourceGroups/origin-rg" --request-body "{ \"resources\": [\"/subscriptions/origin-sub-id/resourceGroups/origin-rg/providers/Microsoft.MachineLearningServices/workspaces/origin-workspace-name\"], \"targetResourceGroup\": \"/subscriptions/destination-sub-id/resourceGroups/destination-rg\" }"
```

Move the workspace

Once the validation has succeeded, move the workspace. You may also include any associated resources into move operation by adding them to the `ids` parameter. This operation may take several minutes.

```
az resource move --destination-group destination-rg --destination-subscription-id destination-sub-id --ids "/subscriptions/origin-sub-id/resourceGroups/origin-rg/providers/Microsoft.MachineLearningServices/workspaces/origin-workspace-name"
```

After the move has completed, recreate any computes and redeploy any web service endpoints at the new location.

Next steps

- Learn about [resource move](#)

Link Azure Synapse Analytics and Azure Machine Learning workspaces and attach Apache Spark pools(preview)

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, you learn how to create a linked service that links your [Azure Synapse Analytics](#) workspace and [Azure Machine Learning workspace](#).

With your Azure Machine Learning workspace linked with your Azure Synapse workspace, you can attach an Apache Spark pool, powered by Azure Synapse Analytics, as a dedicated compute for data wrangling at scale or conduct model training all from the same Python notebook.

You can link your ML workspace and Synapse workspace via the [Python SDK](#) or the [Azure Machine Learning studio](#).

You can also link workspaces and attach a Synapse Spark pool with a single [Azure Resource Manager \(ARM\) template](#).

IMPORTANT

The Azure Machine Learning and Azure Synapse integration is in public preview. The functionalities presented from the `azureml-synapse` package are [experimental](#) preview features, and may change at any time.

Prerequisites

- [Create an Azure Machine Learning workspace](#).
- [Create a Synapse workspace in Azure portal](#).
- [Create Apache Spark pool using Azure portal, web tools, or Synapse Studio](#)
- Install the [Azure Machine Learning Python SDK](#)
- Access to the [Azure Machine Learning studio](#).

Link workspaces with the Python SDK

IMPORTANT

To link to the Synapse workspace successfully, you must be granted the **Owner** role of the Synapse workspace. Check your access in the [Azure portal](#).

If you are not an **Owner** and are only a **Contributor** to the Synapse workspace, you can only use existing linked services. See how to [Retrieve and use an existing linked service](#).

The following code employs the `LinkedService` and `SynapseWorkspaceLinkedServiceConfiguration` classes to,

- Link your machine learning workspace, `ws` with your Azure Synapse workspace.

- Register your Synapse workspace with Azure Machine Learning as a linked service.

```

import datetime
from azureml.core import Workspace, LinkedService, SynapseWorkspaceLinkedServiceConfiguration

# Azure Machine Learning workspace
ws = Workspace.from_config()

#link configuration
synapse_link_config = SynapseWorkspaceLinkedServiceConfiguration(
    subscription_id=ws.subscription_id,
    resource_group= 'your resource group',
    name='mySynapseWorkspaceName')

# Link workspaces and register Synapse workspace in Azure Machine Learning
linked_service = LinkedService.register(workspace = ws,
                                         name = 'synapselink1',
                                         linked_service_config = synapse_link_config)

```

IMPORTANT

A managed identity, `system_assigned_identity_principal_id`, is created for each linked service. This managed identity must be granted the **Synapse Apache Spark Administrator** role of the Synapse workspace before you start your Synapse session. [Assign the Synapse Apache Spark Administrator role to the managed identity in the Synapse Studio](#).

To find the `system_assigned_identity_principal_id` of a specific linked service, use
`LinkedService.get('<your-mlworkspace-name>', '<linked-service-name>')`.

Manage linked services

View all the linked services associated with your machine learning workspace.

```
LinkedService.list(ws)
```

To unlink your workspaces, use the `unregister()` method

```
linked_service.unregister()
```

Link workspaces via studio

Link your machine learning workspace and Synapse workspace via the Azure Machine Learning studio with the following steps:

1. Sign in to the [Azure Machine Learning studio](#).
2. Select **Linked Services** in the **Manage** section of the left pane.
3. Select **Add integration**.
4. On the **Link workspace** form, populate the fields

FIELD	DESCRIPTION
Name	Provide a name for your linked service. This name is what will be used to reference to this particular linked service.

FIELD	DESCRIPTION
Subscription name	Select the name of your subscription that's associated with your machine learning workspace.
Synapse workspace	Select the Synapse workspace you want to link to.

5. Select **Next** to open the **Select Spark pools (optional)** form. On this form, you select which Synapse Spark pool to attach to your workspace
6. Select **Next** to open the **Review** form and check your selections.
7. Select **Create** to complete the linked service creation process.

Get an existing linked service

Before you can attach a dedicated compute for data wrangling, you must have an ML workspace that's linked to an Azure Synapse Analytics workspace, this is referred to as a linked service.

To retrieve and use an existing linked service, requires **User or Contributor** permissions to the **Azure Synapse Analytics workspace**.

This example retrieves an existing linked service, `synapseslink1`, from the workspace, `ws`, with the `get()` method.

```
from azureml.core import LinkedService

linked_service = LinkedService.get(ws, 'synapseslink1')
```

Attach Synapse Spark pool as a compute

Once you retrieve the linked service, attach a Synapse Apache Spark pool as a dedicated compute resource for your data wrangling tasks.

You can attach Apache Spark pools via,

- Azure Machine Learning studio
- [Azure Resource Manager \(ARM\) templates](#)
- The Azure Machine Learning Python SDK

Attach a pool via the studio

Follow these steps:

1. Sign in to the [Azure Machine Learning studio](#).
2. Select **Linked Services** in the **Manage** section of the left pane.
3. Select your Synapse workspace.
4. Select **Attached Spark pools** on the top left.
5. Select **Attach**.
6. Select your Apache Spark pool from the list and provide a name.
 - a. This list identifies the available Synapse Spark pools that can be attached to your compute.
 - b. To create a new Synapse Spark pool, see [Create Apache Spark pool with the Synapse Studio](#)
7. Select **Attach selected**.

Attach a pool with the Python SDK

You can also employ the **Python SDK** to attach an Apache Spark pool.

The follow code,

1. Configures the `SynapseCompute` with,
 - a. The `LinkedService`, `linked_service` that you either created or retrieved in the previous step.
 - b. The type of compute target you want to attach, `SynapseSpark`
 - c. The name of the Apache Spark pool. This must match an existing Apache Spark pool that is in your Azure Synapse Analytics workspace.
2. Creates a machine learning `ComputeTarget` by passing in,
 - a. The machine learning workspace you want to use, `ws`
 - b. The name you'd like to refer to the compute within the Azure Machine Learning workspace.
 - c. The `attach_configuration` you specified when configuring your Synapse Compute.
 - a. The call to `ComputeTarget.attach()` is asynchronous, so the sample blocks until the call completes.

```
from azureml.core.compute import SynapseCompute, ComputeTarget

attach_config = SynapseCompute.attach_configuration(linked_service, #Linked synapse workspace alias
                                                    type='SynapseSpark', #Type of assets to attach
                                                    pool_name=synapse_spark_pool_name) #Name of Synapse
spark pool

synapse_compute = ComputeTarget.attach(workspace= ws,
                                       name= synapse_compute_name,
                                       attach_configuration= attach_config
                                       )

synapse_compute.wait_for_completion()
```

Verify the Apache Spark pool is attached.

```
ws.compute_targets['Synapse Spark pool alias']
```

Next steps

- [How to data wrangle with Azure Synapse \(preview\).](#)
- [How to use Apache Spark in your machine learning pipeline with Azure Synapse \(preview\)](#)
- [Train a model.](#)
- [How to securely integrate Azure Synapse and Azure Machine Learning workspaces.](#)

How to securely integrate Azure Machine Learning and Azure Synapse

9/22/2022 • 5 minutes to read • [Edit Online](#)

In this article, learn how to securely integrate with Azure Machine Learning from Azure Synapse. This integration enables you to use Azure Machine Learning from notebooks in your Azure Synapse workspace. Communication between the two workspaces is secured using an Azure Virtual Network.

TIP

You can also perform integration in the opposite direction, using Azure Synapse spark pool from Azure Machine Learning. For more information, see [Link Azure Synapse and Azure Machine Learning](#).

Prerequisites

- An Azure subscription.
- An Azure Machine Learning workspace with a private endpoint connection to a virtual network. The following workspace dependency services must also have a private endpoint connection to the virtual network:
 - Azure Storage Account

TIP

For the storage account there are three separate private endpoints; one each for blob, file, and dfs.

- Azure Key Vault
- Azure Container Registry

A quick and easy way to build this configuration is to use a [Microsoft Bicep or HashiCorp Terraform template](#).

- An Azure Synapse workspace in a **managed** virtual network, using a **managed** private endpoint. For more information, see [Azure Synapse Analytics Managed Virtual Network](#).

WARNING

The Azure Machine Learning integration is not currently supported in Synapse Workspaces with data exfiltration protection. When configuring your Azure Synapse workspace, do **not** enable data exfiltration protection. For more information, see [Azure Synapse Analytics Managed Virtual Network](#).

NOTE

The steps in this article make the following assumptions:

- The Azure Synapse workspace is in a different resource group than the Azure Machine Learning workspace.
- The Azure Synapse workspace uses a **managed virtual network**. The managed virtual network secures the connectivity between Azure Synapse and Azure Machine Learning. It does **not** restrict access to the Azure Synapse workspace. You will access the workspace over the public internet.

Understanding the network communication

In this configuration, Azure Synapse uses a **managed** private endpoint and virtual network. The managed virtual network and private endpoint secures the internal communications from Azure Synapse to Azure Machine Learning by restricting network traffic to the virtual network. It does **not** restrict communication between your client and the Azure Synapse workspace.

Azure Machine Learning doesn't provide managed private endpoints or virtual networks, and instead uses a **user-managed** private endpoint and virtual network. In this configuration, both internal and client/service communication is restricted to the virtual network. For example, if you wanted to directly access the Azure Machine Learning studio from outside the virtual network, you would use one of the following options:

- Create an Azure Virtual Machine inside the virtual network and use Azure Bastion to connect to it. Then connect to Azure Machine Learning from the VM.
- Create a VPN gateway or use ExpressRoute to connect clients to the virtual network.

Since the Azure Synapse workspace is publicly accessible, you can connect to it without having to create things like a VPN gateway. The Synapse workspace securely connects to Azure Machine Learning over the virtual network. Azure Machine Learning and its resources are secured within the virtual network.

When adding data sources, you can also secure those behind the virtual network. For example, securely connecting to an Azure Storage Account or Data Lake Store Gen 2 through the virtual network.

For more information, see the following articles:

- [Azure Synapse Analytics Managed Virtual Network](#)
- [Secure Azure Machine Learning workspace resources using virtual networks](#).
- [Connect to a secure Azure storage account from your Synapse workspace](#).

Configure Azure Synapse

IMPORTANT

Before following these steps, you need an Azure Synapse workspace that is configured to use a managed virtual network.

For more information, see [Azure Synapse Analytics Managed Virtual Network](#).

1. From Azure Synapse Studio, [Create a new Azure Machine Learning linked service](#).
2. After creating and publishing the linked service, select **Manage**, **Managed private endpoints**, and then **+ New** in Azure Synapse Studio.

Name	Provisioning status	Approval status	VNet name	Possible ports	Linked resource ID
synapse-ws-custstgacct-...	Succeeded	Approved	default	1	/subscriptions/abcdef01-2345-6789-0abc-def012345678/resourceGroups/...
synapse-ws-sql-mysyna...	Succeeded	Approved	default	1	/subscriptions/abcdef01-2345-6789-0abc-def012345678/resourceGroups/...
synapse-ws-sq OnDemand...	Succeeded	Approved	default	0	/subscriptions/abcdef01-2345-6789-0abc-def012345678/resourceGroups/...

3. From the **New managed private endpoint** page, search for **Azure Machine Learning** and select the tile.

New managed private endpoint

Azure Machine Learning

Azure Machine Learning

Continue Cancel

4. When prompted to select the Azure Machine Learning workspace, use the **Azure subscription** and **Azure Machine Learning workspace** you added previously as a linked service. Select **Create** to create the endpoint.

New managed private endpoint (Azure Machine Learning)

Choose a name for your managed private endpoint. This name cannot be updated later.

Name *****
AzureMLService1

Description

Azure Machine Learning workspace selection method ⓘ
 From Azure subscription Enter manually

Azure subscription ⓘ
ML-docs

Azure Machine Learning workspace name *****
mymlworkspace

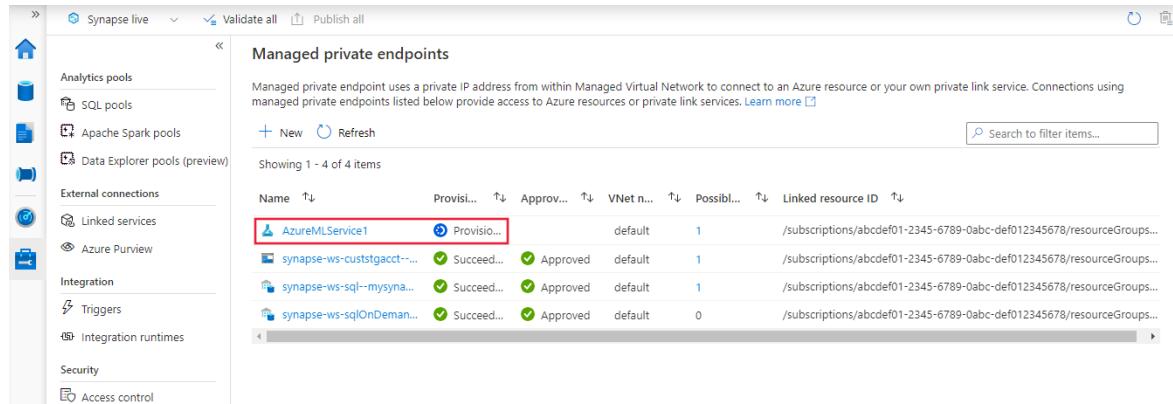
After creation, a private endpoint request will be generated that must get approved by an owner of the data source.

Create **Back** **Cancel**

5. The endpoint will be listed as **Provisioning** until it has been created. Once created, the **Approval** column will list a status of **Pending**. You'll approve the endpoint in the [Configure Azure Machine Learning](#) section.

NOTE

In the following screenshot, a managed private endpoint has been created for the Azure Data Lake Storage Gen 2 associated with this Synapse workspace. For information on how to create an Azure Data Lake Storage Gen 2 and enable a private endpoint for it, see [Provision and secure a linked service with Managed VNet](#).



Name	Provis...	Approv...	VNet n...	Possibl...	Linked resource ID
AzureMLService1	Provisio...		default	1	/subscriptions/abcdef01-2345-6789-0abc-def012345678/resourceGroups...
synapse-ws-custgacct...	Succes...	Approved	default	1	/subscriptions/abcdef01-2345-6789-0abc-def012345678/resourceGroups...
synapse-ws-sql--mysyna...	Succes...	Approved	default	1	/subscriptions/abcdef01-2345-6789-0abc-def012345678/resourceGroups...
synapse-ws-sqjOnDemand...	Succes...	Approved	default	0	/subscriptions/abcdef01-2345-6789-0abc-def012345678/resourceGroups...

Create a Spark pool

To verify that the integration between Azure Synapse and Azure Machine Learning is working, you'll use an Apache Spark pool. For information on creating one, see [Create a Spark pool](#).

Configure Azure Machine Learning

- From the [Azure portal](#), select your **Azure Machine Learning workspace**, and then select **Networking**.

2. Select **Private endpoints**, and then select the endpoint you created in the previous steps. It should have a status of **Pending**. Select **Approve** to approve the endpoint connection.

The screenshot shows the 'Networking' section of the Azure portal for a workspace named 'mymlworkspace'. Under 'Private endpoint connections', there is a table with columns: Connection name, Connection state, Private endpoint, and Description. One row is selected, showing 'mymlworkspace.a7dda26f-da84...' with a status of 'Pending'. The 'Approve' button is highlighted with a red box.

3. From the left of the page, select **Access control (IAM)**. Select **+ Add**, and then select **Role assignment**.

The screenshot shows the 'Access control (IAM)' section of the Azure portal for 'mymlworkspace'. The 'Access control (IAM)' menu item is selected and highlighted with a red box. A dropdown menu is open over the '+ Add' button, with 'Add role assignment' highlighted with a red box.

4. Select **Contributor**, and then select **Next**.

The screenshot shows the 'Add role assignment' page. The 'Contributor' role is selected and highlighted with a red box. At the bottom, the 'Next' button is highlighted with a red box.

Name ↑↓	Description ↑↓	Type ↑↓	Category ↑↓	Details
Owner	Grants full access to manage all resources, including the ability to assign roles in Azure RBAC.	BuiltinRole	General	View
Contributor	Grants full access to manage all resources, but does not allow you to assign roles in Azure RBAC, ma...	BuiltinRole	General	View
Reader	View all resources, but does not allow you to make any changes.	BuiltinRole	General	View
Azure Service Deploy Release Manage...	Contributor role for services deploying through Azure Service Deploy.	CustomRole	None	View
AzureML Data Scientist	Can perform all actions within an Azure Machine Learning workspace, except for creating or deleting...	BuiltinRole	None	View
AzureML Metrics Writer (preview)	Lets you write metrics to AzureML workspace	BuiltinRole	Preview	View
Log Analytics Contributor	Log Analytics Contributor can read all monitoring data and edit monitoring settings. Editing monitor...	BuiltinRole	Analytics	View
Log Analytics Reader	Log Analytics Reader can view and search all monitoring data as well as and view monitoring setting...	BuiltinRole	Analytics	View

5. Select **User, group, or service principal**, and then + **Select members**. Enter the name of the identity created earlier, select it, and then use the **Select** button.

6. Select **Review + assign**, verify the information, and then select the **Review + assign** button.

TIP

It may take several minutes for the Azure Machine Learning workspace to update the credentials cache. Until it has been updated, you may receive errors when trying to access the Azure Machine Learning workspace from Synapse.

Verify connectivity

1. From Azure Synapse Studio, select **Develop**, and then **+ Notebook**.

2. In the **Attach to** field, select the Apache Spark pool for your Azure Synapse workspace, and enter the following code in the first cell:

```
from notebookutils.mssparkutils import azureML

# getWorkspace() takes the linked service name,
# not the Azure Machine Learning workspace name.
ws = azureML.getWorkspace("AzureMLService1")

print(ws.name)
```

This code snippet connects to the linked workspace, and then prints the workspace info. In the printed output, the value displayed is the name of the Azure Machine Learning workspace, not the linked service name that was used in the `getWorkspace()` call. For more information on using the `ws` object, see the [Workspace class reference](#).

Next steps

- [Quickstart: Create a new Azure Machine Learning linked service in Synapse.](#)
- [Link Azure Synapse Analytics and Azure Machine Learning workspaces.](#)

How to use workspace diagnostics

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

Azure Machine Learning provides a diagnostic API that can be used to identify problems with your workspace. Errors returned in the diagnostics report include information on how to resolve the problem.

You can use the workspace diagnostics from the Azure Machine Learning studio or Python SDK.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Quickstart: Create workspace resources](#) article to create one.
- To install the Python SDK v2, use the following command:

```
pip install --pre azure-ai-ml
```

For more information, see [Install the Python SDK v2 for Azure Machine Learning \(preview\)](#).

Diagnostics from studio

From [Azure Machine Learning studio](#) or the Python SDK, you can run diagnostics on your workspace to check your setup. To run diagnostics, select the '?' icon from the upper right corner of the page. Then select **Run workspace diagnostics**.



Help

X

Tutorial

Visit the tutorial page to learn more about Azure Machine Learning concepts and authoring experience.

[View tutorial](#)

[Launch guided tour](#)

Diagnostics

[Run workspace diagnostics](#)

Support

Create a new support request to get assistance with billing, subscription, technical (including advisory) or quota management issues in Azure Portal.

[New support request](#)

Resources

[Documentation](#)

[Cheat Sheet](#)

[Privacy + Terms](#)

After diagnostics run, a list of any detected problems is returned. This list includes links to possible solutions.

Diagnostics from Python

The following snippet demonstrates how to use workspace diagnostics from Python

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

```
from azure.ai.ml import MLClient
from azure.ai.ml.entities import Workspace
from azure.identity import DefaultAzureCredential

subscription_id = '<your-subscription-id>'
resource_group = '<your-resource-group-name>'
workspace = '<your-workspace-name>'

ml_client = MLClient(DefaultAzureCredential(), subscription_id, resource_group)
resp = ml_client.workspaces.begin_diagnose(workspace)
print(resp)
```

The response is a JSON document that contains information on any problems detected with the workspace. The following JSON is an example response:

```
{  
    "value": {  
        "user_defined_route_results": [],  
        "network_security_rule_results": [],  
        "resource_lock_results": [],  
        "dns_resolution_results": [{  
            "code": "CustomDnsInUse",  
            "level": "Warning",  
            "message": "It is detected VNet '/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Network/virtualNetworks/<virtual-network-name>' of private endpoint '/subscriptions/<subscription-id>/resourceGroups/larrygroup0916/providers/Microsoft.Network/privateEndpoints/<workspace-private-endpoint>' is not using Azure default DNS. You need to configure your DNS server and check https://learn.microsoft.com/azure/machine-learning/how-to-custom-dns to make sure the custom DNS is set up correctly."  
        }],  
        "storage_account_results": [],  
        "key_vault_results": [],  
        "container_registry_results": [],  
        "application_insights_results": [],  
        "other_results": []  
    }  
}
```

If no problems are detected, an empty JSON document is returned.

For more information, see the [Workspace](#) reference.

Next steps

- [How to manage workspaces in portal or SDK](#)

Use customer-managed keys with Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

In the [customer-managed keys concepts article](#), you learned about the encryption capabilities that Azure Machine Learning provides. Now learn how to use customer-managed keys with Azure Machine Learning.

Customer-managed keys are used with the following services that Azure Machine Learning relies on:

SERVICE	WHAT IT'S USED FOR
Azure Cosmos DB	Stores metadata for Azure Machine Learning
Azure Cognitive Search	Stores workspace metadata for Azure Machine Learning
Azure Storage Account	Stores workspace metadata for Azure Machine Learning
Azure Container Instance	Hosting trained models as inference endpoints
Azure Kubernetes Service	Hosting trained models as inference endpoints

TIP

- Azure Cosmos DB, Cognitive Search, and Storage Account are secured using the same key. You can use a different key for Azure Kubernetes Service and Container Instance.
- To use a customer-managed key with Azure Cosmos DB, Cognitive Search, and Storage Account, the key is provided when you create your workspace. The key(s) used with Azure Container Instance and Kubernetes Service are provided when configuring those resources.

Prerequisites

- An Azure subscription.
- The following Azure resource providers must be registered:

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.MachineLearningServices	Creating the Azure Machine Learning workspace.
Microsoft.Storage.Azure	Storage Account is used as the default storage for the workspace.
Microsoft.KeyVault	Azure Key Vault is used by the workspace to store secrets.
Microsoft.DocumentDB/databaseAccounts	Azure Cosmos DB instance that logs metadata for the workspace.

RESOURCE PROVIDER	WHY IT'S NEEDED
Microsoft.Search/searchServices	Azure Search provides indexing capabilities for the workspace.

For information on registering resource providers, see [Resolve errors for resource provider registration](#).

Limitations

- The customer-managed key for resources the workspace depends on can't be updated after workspace creation.
- Resources managed by Microsoft in your subscription can't transfer ownership to you.
- You can't delete Microsoft-managed resources used for customer-managed keys without also deleting your workspace.
- The key vault that contains your customer-managed key must be in the same Azure subscription as the Azure Machine Learning workspace

IMPORTANT

When using a customer-managed key, the costs for your subscription will be higher because of the additional resources in your subscription. To estimate the cost, use the [Azure pricing calculator](#).

Create Azure Key Vault

To create the key vault, see [Create a key vault](#). When creating Azure Key Vault, you must enable **soft delete** and **purge protection**.

IMPORTANT

The key vault must be in the same Azure subscription that will contain your Azure Machine Learning workspace.

Create a key

TIP

If you have problems creating the key, it may be caused by Azure role-based access controls that have been applied in your subscription. Make sure that the security principal (user, managed identity, service principal, etc.) you are using to create the key has been assigned the **Contributor** role for the key vault instance. You must also configure an **Access policy** in key vault that grants the security principal **Create**, **Get**, **Delete**, and **Purge** authorization.

If you plan to use a user-assigned managed identity for your workspace, the managed identity must also be assigned these roles and access policies.

For more information, see the following articles:

- [Provide access to key vault keys, certificates, and secrets](#)
- [Assign a key vault access policy](#)
- [Use managed identities with Azure Machine Learning](#)

- From the [Azure portal](#), select the key vault instance. Then select **Keys** from the left.

- Select **+ Generate/import** from the top of the page. Use the following values to create a key:

- Set Options to Generate.**

- Enter a **Name** for the key. The name should be something that identifies what the planned use is. For example, `my-cosmos-key`.
- Set **Key type** to RSA.
- We recommend selecting at least 3072 for the **RSA key size**.
- Leave **Enabled** set to yes.

Optionally you can set an activation date, expiration date, and tags.

3. Select **Create** to create the key.

Allow Azure Cosmos DB to access the key

1. To configure the key vault, select it in the [Azure portal](#) and then select **Access policies** from the left menu.
2. To create permissions for Azure Cosmos DB, select + **Create** at the top of the page. Under **Key permissions**, select **Get, Unwrap Key**, and **Wrap key** permissions.
3. Under **Principal**, search for **Azure Cosmos DB** and then select it. The principal ID for this entry is `a232010e-820c-4083-83bb-3ace5fc29d0b` for all regions other than Azure Government. For Azure Government, the principal ID is `57506a73-e302-42a9-b869-6f12d9ec29e9`.
4. Select **Review + Create**, and then select **Create**.

Create a workspace that uses a customer-managed key

Create an Azure Machine Learning workspace. When creating the workspace, you must select the **Azure Key Vault** and the **key**. Depending on how you create the workspace, you specify these resources in different ways:

WARNING

The key vault that contains your customer-managed key must be in the same Azure subscription as the workspace.

- **Azure portal**: Select the key vault and key from a dropdown input box when configuring the workspace.
- **SDK, REST API, and Azure Resource Manager templates**: Provide the Azure Resource Manager ID of the key vault and the URL for the key. To get these values, use the [Azure CLI](#) and the following commands:

```
# Replace `mykv` with your key vault name.
# Replace `mykey` with the name of your key.

# Get the Azure Resource Manager ID of the key vault
az keyvault show --name mykv --query id
# Get the URL for the key
az keyvault key show --vault-name mykv -n mykey --query key.kid
```

The key vault ID value will be similar to

`/subscriptions/{GUID}/resourceGroups/{resource-group-name}/providers/Microsoft.KeyVault/vaults/mykv`.

The URL for the key will be similar to `https://mykv.vault.azure.net/keys/mykey/{GUID}`.

For examples of creating the workspace with a customer-managed key, see the following articles:

CREATION METHOD	ARTICLE
CLI	Create a workspace with Azure CLI
Azure portal/ Python SDK	Create and manage a workspace

CREATION METHOD	ARTICLE
Azure Resource Manager template	Create a workspace with a template
REST API	Create, run, and delete Azure ML resources with REST

Once the workspace has been created, you'll notice that Azure resource group is created in your subscription. This group is in addition to the resource group for your workspace. This resource group will contain the Microsoft-managed resources that your key is used with. The resource group will be named using the formula of <Azure Machine Learning workspace resource group name><GUID>. It will contain an Azure Cosmos DB instance, Azure Storage Account, and Azure Cognitive Search.

TIP

- The **Request Units** for the Azure Cosmos DB instance automatically scale as needed.
- If your Azure Machine Learning workspace uses a private endpoint, this resource group will also contain a Microsoft-managed Azure Virtual Network. This VNet is used to secure communications between the managed services and the workspace. You **cannot provide your own VNet for use with the Microsoft-managed resources**. You also **cannot modify the virtual network**. For example, you cannot change the IP address range that it uses.

IMPORTANT

If your subscription does not have enough quota for these services, a failure will occur.

WARNING

Don't delete the resource group that contains this Azure Cosmos DB instance, or any of the resources automatically created in this group. If you need to delete the resource group or Microsoft-managed services in it, you must delete the Azure Machine Learning workspace that uses it. The resource group resources are deleted when the associated workspace is deleted.

For more information on customer-managed keys with Cosmos DB, see [Configure customer-managed keys for your Azure Cosmos DB account](#).

Azure Container Instance

When **deploying** a trained model to an Azure Container instance (ACI), you can encrypt the deployed resource using a customer-managed key. For information on generating a key, see [Encrypt data with a customer-managed key](#).

To use the key when deploying a model to Azure Container Instance, create a new deployment configuration using `AciWebservice.deploy_configuration()`. Provide the key information using the following parameters:

- `cmk_vault_base_url` : The URL of the key vault that contains the key.
- `cmk_key_name` : The name of the key.
- `cmk_key_version` : The version of the key.

For more information on creating and using a deployment configuration, see the following articles:

- [AciWebservice.deploy_configuration\(\)](#) reference
- [Where and how to deploy](#)
- [Deploy a model to Azure Container Instances](#)

For more information on using a customer-managed key with ACI, see [Encrypt data with a customer-managed key](#).

Azure Kubernetes Service

You may encrypt a deployed Azure Kubernetes Service resource using customer-managed keys at any time. For more information, see [Bring your own keys with Azure Kubernetes Service](#).

This process allows you to encrypt both the Data and the OS Disk of the deployed virtual machines in the Kubernetes cluster.

IMPORTANT

This process only works with AKS K8s version 1.17 or higher.

Next steps

- [Customer-managed keys with Azure Machine Learning](#)
- [Create a workspace with Azure CLI](#) |
- [Create and manage a workspace](#) |
- [Create a workspace with a template](#) |
- [Create, run, and delete Azure ML resources with REST](#) |

Create and manage an Azure Machine Learning compute instance

9/22/2022 • 19 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#) [Python SDK azure-ai-ml v2 \(preview\)](#)

Learn how to create and manage a [compute instance](#) in your Azure Machine Learning workspace.

Use a compute instance as your fully configured and managed development environment in the cloud. For development and testing, you can also use the instance as a [training compute target](#) or for an [inference target](#). A compute instance can run multiple jobs in parallel and has a job queue. As a development environment, a compute instance can't be shared with other users in your workspace.

In this article, you learn how to:

- [Create a compute instance](#)
- [Manage](#) (start, stop, restart, delete) a compute instance
- [Create a schedule](#) to automatically start and stop the compute instance (preview)

You can also [use a setup script \(preview\)](#) to create the compute instance with your own custom environment.

Compute instances can run jobs securely in a [virtual network environment](#), without requiring enterprises to open up SSH ports. The job executes in a containerized environment and packages your model dependencies in a Docker container.

NOTE

This article shows CLI v2 in the sections below. If you are still using CLI v1, see [Create an Azure Machine Learning compute cluster CLI v1](#).

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- The [Azure CLI extension for Machine Learning service \(v2\)](#), [Azure Machine Learning Python SDK \(v2\)](#), or the [Azure Machine Learning Visual Studio Code extension](#).
- If using the Python SDK, [set up your development environment with a workspace](#). Once your environment is set up, attach to the workspace in your Python script:

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

Run this code to connect to your Azure ML workspace.

Replace your Subscription ID, Resource Group name and Workspace name in the code below. To find these values:

1. Sign in to [Azure Machine Learning studio](#).
2. Open the workspace you wish to use.
3. In the upper right Azure Machine Learning studio toolbar, select your workspace name.
4. Copy the value for workspace, resource group and subscription ID into the code.

5. If you're using a notebook inside studio, you'll need to copy one value, close the area and paste, then come back for the next one.

```
# Enter details of your AML workspace
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
workspace = "<AML_WORKSPACE_NAME>"
```

```
# get a handle to the workspace
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

ml_client = MLClient(
    DefaultAzureCredential(), subscription_id, resource_group, workspace
)
```

`ml_client` is a handler to the workspace that you'll use to manage other resources and jobs.

Create

IMPORTANT

Items marked (preview) below are currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Time estimate: Approximately 5 minutes.

Creating a compute instance is a one time process for your workspace. You can reuse the compute as a development workstation or as a compute target for training. You can have multiple compute instances attached to your workspace.

The dedicated cores per region per VM family quota and total regional quota, which applies to compute instance creation, is unified and shared with Azure Machine Learning training compute cluster quota. Stopping the compute instance doesn't release quota to ensure you'll be able to restart the compute instance. It isn't possible to change the virtual machine size of compute instance once it's created.

The following example demonstrates how to create a compute instance:

- [Python SDK](#)
- [Azure CLI](#)
- [Studio](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

```
# Compute Instances need to have a unique name across the region.
# Here we create a unique name with current datetime
from azure.ai.ml.entities import ComputeInstance, AmlCompute
import datetime

ci_basic_name = "basic-ci" + datetime.datetime.now().strftime("%Y%m%d%H%M")
ci_basic = ComputeInstance(name=ci_basic_name, size="STANDARD_DS3_V2")
ml_client.begin_create_or_update(ci_basic)
```

For more information on the classes, methods, and parameters used in this example, see the following reference documents:

- [AmlCompute class](#)
- [ComputeInstance class](#)

Configure auto-stop (preview)

To avoid getting charged for a compute instance that is switched on but inactive, you can configure auto-stop.

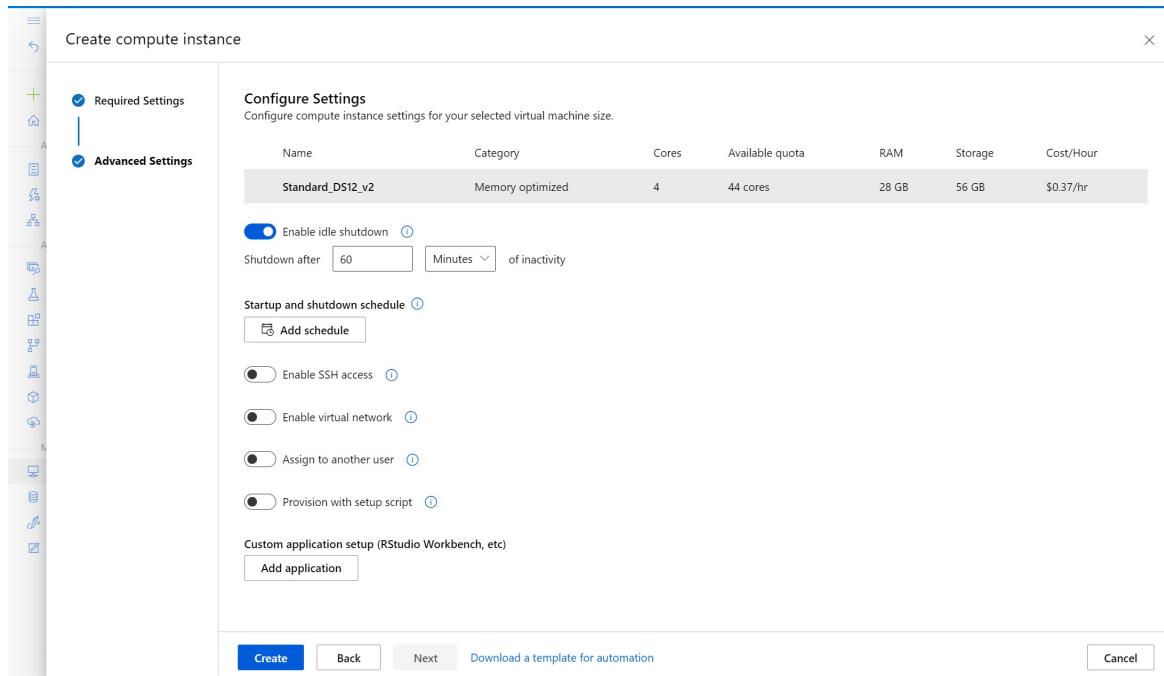
A compute instance is considered inactive if the below conditions are met:

- No active Jupyter Kernel sessions (which translates to no Notebooks usage via Jupyter, JupyterLab or Interactive notebooks)
- No active Jupyter terminal sessions
- No active AzureML runs or experiments
- No SSH connections
- No VS code connections; you must close your VS Code connection for your compute instance to be considered inactive. Sessions are auto-terminated if VS code detects no activity for 3 hours.

Activity on custom applications installed on the compute instance isn't considered. There are also some basic bounds around inactivity time periods; CI must be inactive for a minimum of 15 mins and a maximum of three days.

This setting can be configured during CI creation or for existing Cls via the following interfaces:

- AzureML Studio



- REST API

Endpoint:

```
POST
https://management.azure.com/subscriptions/{SUB_ID}/resourceGroups/{RG_NAME}/providers/Microsoft.MachineLearningServices/workspaces/{WS_NAME}/computes/{CI_NAME}/updateIdleShutdownSetting?api-version=2021-07-01
```

Body:

```
{
    "idleTimeBeforeShutdown": "PT30M" // this must be a string in ISO 8601 format
}
```

- CLIV2 (YAML): only configurable during new CI creation

```
# Note that this is just a snippet for the idle shutdown property. Refer to the "Create" Azure CLI section for more information.
idle_time_before_shutdown_minutes: 30
```

- Python SDKv2: only configurable during new CI creation

```
ComputeInstance(name=ci_basic_name, size="STANDARD_DS3_v2", idle_time_before_shutdown_minutes="30")
```

- ARM Templates: only configurable during new CI creation

```
// Note that this is just a snippet for the idle shutdown property in an ARM template
{
    "idleTimeBeforeShutdown": "PT30M" // this must be a string in ISO 8601 format
}
```

Azure policy support

Administrators can use a built-in [Azure Policy](#) definition to enforce auto-stop on all compute instances in a given

subscription/resource-group.

1. Navigate to Azure Policy in the Azure portal.
2. Under "Definitions", look for the idle shutdown policy.

The screenshot shows the Azure Policy Definitions blade. A policy named 'Azure Machine Learning Compute Instance should have idle shutdown.' is selected. The blade includes tabs for 'Assign', 'Edit definition', 'Duplicate definition', 'Delete definition', and 'Export definition'. Below the tabs, there's a section for 'Essentials' with fields for Name, Description, Available Effects, Category, Definition location, Definition ID, Type, and Mode. The policy details are as follows:

Name	Azure Machine Learning Compute Instance should have idle shutdown.
Description	Having an idle shutdown schedule reduces cost by shutting down computes that are idle after a period of inactivity.
Available Effects	Audit, Deny, Disabled
Category	Machine Learning
Definition location	--
Definition ID	/providers/Microsoft.Authorization/policyDefinitions/679dddf89-ab8f-48a5-9029-e760540...
Type	Built-in
Mode	All

3. Assign policy to the necessary scope.

You can also create your own custom Azure policy. For example, if the below policy is assigned, all new compute instances will have auto-stop configured with a 60-minute inactivity period.

```
{
  "mode": "All",
  "policyRule": {
    "if": {
      "allOf": [
        {
          "field": "type",
          "equals": "Microsoft.MachineLearningServices/workspaces/computes"
        },
        {
          "field": "Microsoft.MachineLearningServices/workspaces/computes/computeType",
          "equals": "ComputeInstance"
        },
        {
          "anyOf": [
            {
              "field": "Microsoft.MachineLearningServices/workspaces/computes/idleTimeBeforeShutdown",
              "exists": false
            },
            {
              "value": ""
            }
          ]
        }
      ],
      "empty(field('Microsoft.MachineLearningServices/workspaces/computes/idleTimeBeforeShutdown'))",
      "equals": true
    }
  },
  "then": {
    "effect": "append",
    "details": [
      {
        "field": "Microsoft.MachineLearningServices/workspaces/computes/idleTimeBeforeShutdown",
        "value": "PT60M"
      }
    ]
  },
  "parameters": {}
}
```

Create on behalf of (preview)

As an administrator, you can create a compute instance on behalf of a data scientist and assign the instance to them with:

- Studio, using the [Advanced settings](#)
- [Azure Resource Manager template](#). For details on how to find the TenantID and ObjectId needed in this template, see [Find identity object IDs for authentication configuration](#). You can also find these values in the Azure Active Directory portal.
- REST API

The data scientist you create the compute instance for needs the following [be Azure role-based access control \(Azure RBAC\)](#) permissions:

- *Microsoft.MachineLearningServices/workspaces/computes/start/action*
- *Microsoft.MachineLearningServices/workspaces/computes/stop/action*
- *Microsoft.MachineLearningServices/workspaces/computes/restart/action*
- *Microsoft.MachineLearningServices/workspaces/computes/applicationaccess/action*
- *Microsoft.MachineLearningServices/workspaces/computes/updateSchedules/action*

The data scientist can start, stop, and restart the compute instance. They can use the compute instance for:

- Jupyter
- JupyterLab
- RStudio
- Integrated notebooks

Schedule automatic start and stop (preview)

Define multiple schedules for auto-shutdown and auto-start. For instance, create a schedule to start at 9 AM and stop at 6 PM from Monday-Thursday, and a second schedule to start at 9 AM and stop at 4 PM for Friday. You can create a total of four schedules per compute instance.

Schedules can also be defined for [create on behalf of](#) compute instances. You can create a schedule that creates the compute instance in a stopped state. Stopped compute instances are particularly useful when you create a compute instance on behalf of another user.

Create a schedule in studio

1. [Fill out the form](#).
2. On the second page of the form, open **Show advanced settings**.
3. Select **Add schedule** to add a new schedule.

Create compute instance

Required Settings

Advanced Settings

Configure Settings

Configure compute instance settings for your selected vir

Name	Category
Standard_DS3_v2	General purpo

Enable SSH access [\(i\)](#)

Enable virtual network [\(i\)](#)

Assign to another user [\(i\)](#)

Provision with setup script [\(i\)](#)

Startup and shutdown schedule [\(i\)](#)

[Add schedule](#)

4. Select Start compute instance or Stop compute instance.

5. Select the Time zone.

6. Select the Startup time or Shutdown time.

7. Select the days when this schedule is active.

Startup and shutdown schedule [\(i\)](#)

Schedule 1 [Edit](#)

Start compute instance Stop compute instance

Time zone [\(UTC\) Coordinated Universal Time](#)

Shutdown time [08:00 PM](#)

Active days

Sunday Monday Tuesday Wednesday Thursday Friday Saturday

[Add schedule](#)

[Create](#) [Back](#) [Next](#) [Download a template for automation](#) [Cancel](#)

8. Select Add schedule again if you want to create another schedule.

Once the compute instance is created, you can view, edit, or add new schedules from the compute instance details section.

NOTE

Timezone labels don't account for day light savings. For instance, (UTC+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna is actually UTC+02:00 during day light savings.

Create a schedule with CLI

APPLIES TO:  Azure CLI ml extension v2 (current)

```
az ml compute create -f create-instance.yml
```

Where the file *create-instance.yml* is:

```
$schema: https://azuremlschemas.azureedge.net/latest/computeInstance.schema.json
name: schedule-example-i
type: computeinstance
size: STANDARD_DS3_V2
schedules:
  compute_start_stop:
    - action: stop
      trigger:
        type: cron
        start_time: "2021-03-10T21:21:07"
        time_zone: Pacific Standard Time
        expression: 0 18 * * *
```

Create a schedule with a Resource Manager template

You can schedule the automatic start and stop of a compute instance by using a Resource Manager [template](#).

In the Resource Manager template, add:

```
"schedules": "[parameters('schedules')]"
```

Then use either cron or LogicApps expressions to define the schedule that starts or stops the instance in your parameter file:

```

"schedules": {
  "value": {
    "computeStartStop": [
      {
        "triggerType": "Cron",
        "cron": {
          "timeZone": "UTC",
          "expression": "0 18 * * *"
        },
        "action": "Stop",
        "status": "Enabled"
      },
      {
        "triggerType": "Cron",
        "cron": {
          "timeZone": "UTC",
          "expression": "0 8 * * *"
        },
        "action": "Start",
        "status": "Enabled"
      },
      {
        "triggerType": "Recurrence",
        "recurrence": {
          "frequency": "Day",
          "interval": 1,
          "timeZone": "UTC",
          "schedule": {
            "hours": [17],
            "minutes": [0]
          }
        },
        "action": "Stop",
        "status": "Enabled"
      }
    ]
  }
}

```

- Action can have value of "Start" or "Stop".
- For trigger type of `Recurrence` use the same syntax as logic app, with this [recurrence schema](#).
- For trigger type of `cron`, use standard cron syntax:

```

// Crontab expression format:
//
// * * * * *
// - - - -
// | | | |
// | | | +---- day of week (0 - 6) (Sunday=0)
// | | +----- month (1 - 12)
// | +-------- day of month (1 - 31)
// | +----- hour (0 - 23)
// +----- min (0 - 59)
//
// Star (*) in the value field above means all legal values as in
// braces for that column. The value column can have a * or a list
// of elements separated by commas. An element is either a number in
// the ranges shown above or two numbers in the range separated by a
// hyphen (meaning an inclusive range).

```

Azure Policy support to default a schedule

Use Azure Policy to enforce a shutdown schedule exists for every compute instance in a subscription or default

to a schedule if nothing exists. Following is a sample policy to default a shutdown schedule at 10 PM PST.

```
{
  "mode": "All",
  "policyRule": {
    "if": {
      "allOf": [
        {
          "field": "Microsoft.MachineLearningServices/workspaces/computes/computeType",
          "equals": "ComputeInstance"
        },
        {
          "field": "Microsoft.MachineLearningServices/workspaces/computes/schedules",
          "exists": "false"
        }
      ]
    },
    "then": {
      "effect": "append",
      "details": [
        {
          "field": "Microsoft.MachineLearningServices/workspaces/computes/schedules",
          "value": {
            "computeStartStop": [
              {
                "triggerType": "Cron",
                "cron": {
                  "startTime": "2021-03-10T21:21:07",
                  "timeZone": "Pacific Standard Time",
                  "expression": "0 22 * * *"
                },
                "action": "Stop",
                "status": "Enabled"
              }
            ]
          }
        }
      ]
    }
  }
}
```

Add custom applications such as RStudio (preview)

You can set up other applications, such as RStudio, when creating a compute instance. Follow these steps in studio to set up a custom application on your compute instance

1. Fill out the form to [create a new compute instance](#)
2. Select **Next: Advanced Settings**
3. Select **Add application** under the **Custom application setup (RStudio Workbench, etc.)** section

Required Settings

Advanced Settings

Configure Settings
Configure compute instance settings for your selected virtual machine size.

Name	Category	Cores	Available quota	RAM	Storage	Cost/Hour
Standard_DS12_v2	Memory optimized	4	56 cores	28 GB	56 GB	\$0.30/hr

Startup and shutdown schedule [\(i\)](#)

Enable SSH access [\(i\)](#)

Enable virtual network [\(i\)](#)

Assign to another user [\(i\)](#)

Provision with setup script [\(i\)](#)

Assign a managed identity [\(i\)](#)

Custom application setup (RStudio Workbench, etc)

Setup RStudio Workbench

RStudio is one of the most popular IDEs among R developers for ML and data science projects. You can easily set up RStudio Workbench to run on your compute instance, using your own RStudio license, and access the rich feature set that RStudio Workbench offers.

1. Follow the steps listed above to **Add application** when creating your compute instance.
2. Select **RStudio Workbench (bring your own license)** in the **Application** dropdown and enter your RStudio Workbench license key in the **License key** field. You can get your RStudio Workbench license or trial license [from RStudio](#).
3. Select **Create** to add RStudio Workbench application to your compute instance.

Custom application setup (RStudio Workbench, etc)

rstudio-workbench2

Application

Target port * [\(i\)](#) **Published port * [\(i\)](#)**

Docker image *

License key *

Create Back Next [Download a template for automation](#) Cancel

IMPORTANT

If using a private link workspace, ensure that the docker image and ghcr.io are accessible. Also, use a published port in the range 8704-8993.

NOTE

- Support for accessing your workspace file store from RStudio is not yet available.
- When accessing multiple instances of RStudio, if you see a "400 Bad Request. Request Header Or Cookie Too Large" error, use a new browser or access from a browser in incognito mode.
- Shiny applications are not currently supported on RStudio Workbench.

Setup RStudio open source

To use RStudio open source, set up a custom application as follows:

1. Follow the steps listed above to **Add application** when creating your compute instance.
2. Select **Custom Application** on the **Application** dropdown
3. Configure the **Application name** you would like to use.
4. Set up the application to run on **Target port** `8787` - the docker image for RStudio open source listed below needs to run on this Target port.
5. Set up the application to be accessed on **Published port** `8787` - you can configure the application to be accessed on a different Published port if you wish.
6. Point the **Docker image** to `ghcr.io/azure/rocker-rstudio-ml-verse:latest`.
7. Use **Bind mounts** to add access to the files in your default storage account:
 - Specify `/home/azureuser/cloudfiles` for **Host path**.
 - Specify `/home/azureuser/cloudfiles` for the **Container path**.
 - Select **Add** to add this mounting. Because the files are mounted, changes you make to them will be available in other compute instances and applications.
8. Select **Create** to set up RStudio as a custom application on your compute instance.

Custom application setup (RStudio Workbench, etc)

custom-application

Application

Custom Application

Application name *

custom-application

Target port * (i) **Published port * (i)**

8787 8787

Docker image *

ghcr.io/azure/rocker-rstudio-ml-verse:latest

Environment variables

Name : Value Add

(i) No Environment variables

Bind mounts

Host path : Container path Add

(i) No Bind mounts

Add application

Create Back Next Download a template for automation Cancel

IMPORTANT

If using a private link workspace, ensure that the docker image and ghcr.io are accessible. Also, use a published port in the range 8704-8993.

Setup other custom applications

Set up other custom applications on your compute instance by providing the application on a Docker image.

1. Follow the steps listed above to **Add application** when creating your compute instance.
2. Select **Custom Application** on the **Application** dropdown.

3. Configure the **Application name**, the **Target port** you wish to run the application on, the **Published port** you wish to access the application on and the **Docker image** that contains your application.
4. Optionally, add **Environment variables** you wish to use for your application.
5. Use **Bind mounts** to add access to the files in your default storage account:
 - Specify **/home/azureuser/cloudfiles** for **Host path**.
 - Specify **/home/azureuser/cloudfiles** for the **Container path**.
 - Select **Add** to add this mounting. Because the files are mounted, changes you make to them will be available in other compute instances and applications.
6. Select **Create** to set up the custom application on your compute instance.

Custom application setup (RStudio Workbench, etc)

custom-application

Application

Custom Application

Application name *

custom-application

Target port * ⓘ **Published port * ⓘ**

A port is required A port is required

Docker image *

A docker image is required

Environment variables

Name	:	Value	Add
------	---	-------	-----

No Environment variables

Bind mounts

Host path	:	Container path	Add
-----------	---	----------------	-----

No Bind mounts

Add application

Create **Back** **Next** **Cancel**

IMPORTANT

If using a private link workspace, ensure that the docker image and ghcr.io are accessible. Also, use a published port in the range 8704-8993.

Accessing custom applications in studio

Access the custom applications that you set up in studio:

1. On the left, select **Compute**.
2. On the **Compute instance** tab, see your applications under the **Applications** column.

Compute

Compute instances Compute clusters Inference clusters Attached computes

+ New **⟳ Refresh** **⟳ Start** **⟲ Stop** **⟳ Restart** **ⓧ Delete** **📅 Schedule** **>Edit columns** **⟲ Reset view** **☰ View quota**

Search

Name	State	Applications ⓘ	Size ↑	Created on	Assigned to
MyComputeInstance	Running	JupyterLab Jupyter VS Code rstudio-workbench Terminal Notebook	STANDARD_DS12_V2	May 10, 2022 7:51 PM	

NOTE

It might take a few minutes after setting up a custom application until you can access it via the links above. The amount of time taken will depend on the size of the image used for your custom application. If you see a 502 error message when trying to access the application, wait for some time for the application to be set up and try again.

Manage

Start, stop, restart, and delete a compute instance. A compute instance doesn't automatically scale down, so make sure to stop the resource to prevent ongoing charges. Stopping a compute instance deallocates it. Then start it again when you need it. While stopping the compute instance stops the billing for compute hours, you'll still be billed for disk, public IP, and standard load balancer.

You can [create a schedule](#) for the compute instance to automatically start and stop based on a time and day of week.

TIP

The compute instance has 120GB OS disk. If you run out of disk space, [use the terminal](#) to clear at least 1-2 GB before you stop or restart the compute instance. Please do not stop the compute instance by issuing sudo shutdown from the terminal. The temp disk size on compute instance depends on the VM size chosen and is mounted on /mnt.

- [Python SDK](#)
- [Azure CLI](#)
- [Studio](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

In the examples below, the name of the compute instance is stored in the variable `ci_basic_name`.

- Get status

```
from azure.ai.ml.entities import ComputeInstance, AmlCompute

# Get compute
ci_basic_state = ml_client.compute.get(ci_basic_name)
```

- Stop

```
from azure.ai.ml.entities import ComputeInstance, AmlCompute

# Stop compute
ml_client.compute.begin_stop(ci_basic_name)
```

- Start

```
from azure.ai.ml.entities import ComputeInstance, AmlCompute

# Start compute
ml_client.compute.begin_start(ci_basic_name)
```

- Restart

```
from azure.ai.ml.entities import ComputeInstance, AmlCompute  
  
# Restart compute  
ml_client.compute.begin_restart(ci_basic_name)
```

- Delete

```
from azure.ai.ml.entities import ComputeInstance, AmlCompute  
  
ml_client.compute.begin_delete(ci_basic_name)
```

Azure RBAC allows you to control which users in the workspace can create, delete, start, stop, restart a compute instance. All users in the workspace contributor and owner role can create, delete, start, stop, and restart compute instances across the workspace. However, only the creator of a specific compute instance, or the user assigned if it was created on their behalf, is allowed to access Jupyter, JupyterLab, and RStudio on that compute instance. A compute instance is dedicated to a single user who has root access. That user has access to Jupyter/JupyterLab/RStudio running on the instance. Compute instance will have single-user sign-in and all actions will use that user's identity for Azure RBAC and attribution of experiment jobs. SSH access is controlled through public/private key mechanism.

These actions can be controlled by Azure RBAC:

- *Microsoft.MachineLearningServices/workspaces/computes/read*
- *Microsoft.MachineLearningServices/workspaces/computes/write*
- *Microsoft.MachineLearningServices/workspaces/computes/delete*
- *Microsoft.MachineLearningServices/workspaces/computes/start/action*
- *Microsoft.MachineLearningServices/workspaces/computes/stop/action*
- *Microsoft.MachineLearningServices/workspaces/computes/restart/action*
- *Microsoft.MachineLearningServices/workspaces/computes/updateSchedules/action*

To create a compute instance, you'll need permissions for the following actions:

- *Microsoft.MachineLearningServices/workspaces/computes/write*
- *Microsoft.MachineLearningServices/workspaces/checkComputeNameAvailability/action*

Next steps

- [Access the compute instance terminal](#)
- [Create and manage files](#)
- [Update the compute instance to the latest VM image](#)
- [Submit a training job](#)

Create an Azure Machine Learning compute cluster

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#) [Python SDK azure-ai-ml v2 \(preview\)](#)

Learn how to create and manage a [compute cluster](#) in your Azure Machine Learning workspace.

You can use Azure Machine Learning compute cluster to distribute a training or batch inference process across a cluster of CPU or GPU compute nodes in the cloud. For more information on the VM sizes that include GPUs, see [GPU-optimized virtual machine sizes](#).

In this article, learn how to:

- Create a compute cluster
- Lower your compute cluster cost
- Set up a [managed identity](#) for the cluster

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- The [Azure CLI extension for Machine Learning service \(v2\)](#), [Azure Machine Learning Python SDK](#), or the [Azure Machine Learning Visual Studio Code extension](#).
- If using the Python SDK, [set up your development environment with a workspace](#). Once your environment is set up, attach to the workspace in your Python script:

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

Run this code to connect to your Azure ML workspace.

Replace your Subscription ID, Resource Group name and Workspace name in the code below. To find these values:

1. Sign in to [Azure Machine Learning studio](#).
2. Open the workspace you wish to use.
3. In the upper right Azure Machine Learning studio toolbar, select your workspace name.
4. Copy the value for workspace, resource group and subscription ID into the code.
5. If you're using a notebook inside studio, you'll need to copy one value, close the area and paste, then come back for the next one.

```
# Enter details of your AML workspace
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
workspace = "<AML_WORKSPACE_NAME>"
```

```
# get a handle to the workspace
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

ml_client = MLClient(
    DefaultAzureCredential(), subscription_id, resource_group, workspace
)
```

`ml_client` is a handler to the workspace that you'll use to manage other resources and jobs.

What is a compute cluster?

Azure Machine Learning compute cluster is a managed-compute infrastructure that allows you to easily create a single or multi-node compute. The compute cluster is a resource that can be shared with other users in your workspace. The compute scales up automatically when a job is submitted, and can be put in an Azure Virtual Network. Compute cluster supports **no public IP (preview)** deployment as well in virtual network. The compute executes in a containerized environment and packages your model dependencies in a [Docker container](#).

Compute clusters can run jobs securely in a [virtual network environment](#), without requiring enterprises to open up SSH ports. The job executes in a containerized environment and packages your model dependencies in a Docker container.

Limitations

- Some of the scenarios listed in this document are marked as **preview**. Preview functionality is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).
- Compute clusters can be created in a different region than your workspace. This functionality is in **preview**, and is only available for **compute clusters**, not compute instances. This preview isn't available if you're using a private endpoint-enabled workspace.

WARNING

When using a compute cluster in a different region than your workspace or datastores, you may see increased network latency and data transfer costs. The latency and costs can occur when creating the cluster, and when running jobs on it.

- We currently support only creation (and not updating) of clusters through [ARM templates](#). For updating compute, we recommend using the SDK, Azure CLI or UX for now.
- Azure Machine Learning Compute has default limits, such as the number of cores that can be allocated. For more information, see [Manage and request quotas for Azure resources](#).
- Azure allows you to place *locks* on resources, so that they can't be deleted or are read only. **Do not apply resource locks to the resource group that contains your workspace**. Applying a lock to the resource group that contains your workspace will prevent scaling operations for Azure ML compute clusters. For more information on locking resources, see [Lock resources to prevent unexpected changes](#).

TIP

Clusters can generally scale up to 100 nodes as long as you have enough quota for the number of cores required. By default clusters are setup with inter-node communication enabled between the nodes of the cluster to support MPI jobs for example. However you can scale your clusters to 1000s of nodes by simply [raising a support ticket](#), and requesting to allow list your subscription, or workspace, or a specific cluster for disabling inter-node communication.

Create

Time estimate: Approximately 5 minutes.

Azure Machine Learning Compute can be reused across runs. The compute can be shared with other users in the workspace and is retained between runs, automatically scaling nodes up or down based on the number of runs submitted, and the `max_nodes` set on your cluster. The `min_nodes` setting controls the minimum nodes available.

The dedicated cores per region per VM family quota and total regional quota, which applies to compute cluster creation, is unified and shared with Azure Machine Learning training compute instance quota.

IMPORTANT

To avoid charges when no jobs are running, **set the minimum nodes to 0**. This setting allows Azure Machine Learning to de-allocate the nodes when they aren't in use. Any value larger than 0 will keep that number of nodes running, even if they are not in use.

The compute autoscales down to zero nodes when it isn't used. Dedicated VMs are created to run your jobs as needed.

- [Python SDK](#)
- [Azure CLI](#)
- [Studio](#)

To create a persistent Azure Machine Learning Compute resource in Python, specify the `size` and `max_instances` properties. Azure Machine Learning then uses smart defaults for the other properties.

- `size`*: The VM family of the nodes created by Azure Machine Learning Compute.
- `*max_instances`: The max number of nodes to autoscale up to when you run a job on Azure Machine Learning Compute.

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

```
from azure.ai.ml.entities import AmlCompute

cluster_basic = AmlCompute(
    name="basic-example",
    type="amlcompute",
    size="STANDARD_DS3_V2",
    location="westus",
    min_instances=0,
    max_instances=2,
    idle_time_before_scale_down=120,
)
ml_client.begin_create_or_update(cluster_basic)
```

You can also configure several advanced properties when you create Azure Machine Learning Compute. The properties allow you to create a persistent cluster of fixed size, or within an existing Azure Virtual Network in

your subscription. See the [AmlCompute class](#) for details.

WARNING

When setting the `location` parameter, if it is a different region than your workspace or datastores you may see increased network latency and data transfer costs. The latency and costs can occur when creating the cluster, and when running jobs on it.

Lower your compute cluster cost

You may also choose to use [low-priority VMs](#) to run some or all of your workloads. These VMs don't have guaranteed availability and may be preempted while in use. You'll have to restart a preempted job.

Use any of these ways to specify a low-priority VM:

- [Python SDK](#)
- [Azure CLI](#)
- [Studio](#)

APPLIES TO:  [Python SDK azureml v1](#)

```
from azure.ai.ml.entities import AmlCompute

cluster_low_pri = AmlCompute(
    name="low-pri-example",
    size="STANDARD_DS3_v2",
    min_instances=0,
    max_instances=2,
    idle_time_before_scale_down=120,
    tier="low_priority",
)
ml_client.begin_create_or_update(cluster_low_pri)
```

Set up managed identity

Azure Machine Learning compute clusters also support [managed identities](#) to authenticate access to Azure resources without including credentials in your code. There are two types of managed identities:

- A **system-assigned managed identity** is enabled directly on the Azure Machine Learning compute cluster. The life cycle of a system-assigned identity is directly tied to the compute cluster. If the compute cluster is deleted, Azure automatically cleans up the credentials and the identity in Azure AD.
- A **user-assigned managed identity** is a standalone Azure resource provided through Azure Managed Identity service. You can assign a user-assigned managed identity to multiple resources, and it persists for as long as you want. This managed identity needs to be created beforehand and then passed as the `identity_id` as a required parameter.

NOTE

Managed identity is supported on [compute clusters](#), but is not supported on [compute instances](#).

- [Python SDK](#)
- [Azure CLI](#)
- [Studio](#)

NOTE

Azure Machine Learning compute clusters support only **one system-assigned identity or multiple user-assigned identities**, not both concurrently.

Managed identity usage

The **default managed identity** is the system-assigned managed identity or the first user-assigned managed identity.

During a run there are two applications of an identity:

1. The system uses an identity to set up the user's storage mounts, container registry, and datastores.
 - In this case, the system will use the default-managed identity.
2. The user applies an identity to access resources from within the code for a submitted run
 - In this case, provide the *client_id* corresponding to the managed identity you want to use to retrieve a credential.
 - Alternatively, get the user-assigned identity's client ID through the *DEFAULT_IDENTITY_CLIENT_ID* environment variable.

For example, to retrieve a token for a datastore with the default-managed identity:

```
client_id = os.environ.get('DEFAULT_IDENTITY_CLIENT_ID')
credential = ManagedIdentityCredential(client_id=client_id)
token = credential.get_token('https://storage.azure.com/')
```

Troubleshooting

There's a chance that some users who created their Azure Machine Learning workspace from the Azure portal before the GA release might not be able to create AmlCompute in that workspace. You can either raise a support request against the service or create a new workspace through the portal or the SDK to unblock yourself immediately.

Stuck at resizing

If your Azure Machine Learning compute cluster appears stuck at resizing (0 -> 0) for the node state, this may be caused by Azure resource locks.

Azure allows you to place *locks* on resources, so that they cannot be deleted or are read only. **Locking a resource can lead to unexpected results**. Some operations that don't seem to modify the resource actually require actions that are blocked by the lock.

With Azure Machine Learning, applying a delete lock to the resource group for your workspace will prevent scaling operations for Azure ML compute clusters. To work around this problem we recommend **removing** the lock from resource group and instead applying it to individual items in the group.

IMPORTANT

Do not apply the lock to the following resources:

RESOURCE NAME	RESOURCE TYPE
<GUID>-azurebatch-cloudservicenetworksecuritygroup	Network security group
<GUID>-azurebatch-cloudservicepublicip	Public IP address
<GUID>-azurebatch-cloudserviceloadbalancer	Load balancer

These resources are used to communicate with, and perform operations such as scaling on, the compute cluster. Removing the resource lock from these resources should allow autoscaling for your compute clusters.

For more information on resource locking, see [Lock resources to prevent unexpected changes](#).

Next steps

Use your compute cluster to:

- [Submit a training run](#)
- [Run batch inference](#).

Manage compute resources for model training and deployment in studio

9/22/2022 • 3 minutes to read • [Edit Online](#)

In this article, learn how to manage the compute resources you use for model training and deployment in Azure Machine studio.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today
- An [Azure Machine Learning workspace](#)

What's a compute target?

With Azure Machine Learning, you can train your model on a variety of resources or environments, collectively referred to as *compute targets*). A compute target can be a local machine or a cloud resource, such as an Azure Machine Learning Compute, Azure HDInsight, or a remote virtual machine.

View compute targets

To see all compute targets for your workspace, use the following steps:

1. Navigate to [Azure Machine Learning studio](#).
2. Under **Manage**, select **Compute**.
3. Select tabs at the top to show each type of compute target.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The left sidebar has a 'Compute' section highlighted with a red box. The main area shows a tab bar with 'Compute instances' selected, also highlighted with a red box. Below the tabs is a 3D icon representing compute resources. To the right of the icon is a call-to-action button: 'Get started with Azure Machine Learning notebooks and R scripts by creating a compute instance'. At the bottom of the main area, there is descriptive text about choosing from preconfigured CPU or GPU instances and links to 'Learn more' and 'View Azure Machine Learning tutorials'.

Compute instance and clusters

You can create compute instances and compute clusters in your workspace, using the Azure Machine Learning SDK, CLI, or studio:

- [Compute instance](#)
- [Compute cluster](#)

In addition, you can use the [VS Code extension](#) to create compute instances and compute clusters in your workspace.

Kubernetes cluster

For information on configuring and attaching a Kubrnetes cluster to your workspace, see [Configure Kubernetes cluster for Azure Machine Learning](#).

Other compute targets

To use VMs created outside the Azure Machine Learning workspace, you must first attach them to your workspace. Attaching the compute resource makes it available to your workspace.

1. Navigate to [Azure Machine Learning studio](#).
2. Under **Manage**, select **Compute**.
3. In the tabs at the top, select **Attached compute** to attach a compute target for **training**. Or select **Inference clusters** to attach an AKS cluster for **inferencing**.
4. Select **+New**, then select the type of compute to attach. Not all compute types can be attached from Azure Machine Learning studio.
5. Fill out the form and provide values for the required properties.

NOTE

Microsoft recommends that you use SSH keys, which are more secure than passwords. Passwords are vulnerable to brute force attacks. SSH keys rely on cryptographic signatures. For information on how to create SSH keys for use with Azure Virtual Machines, see the following documents:

- [Create and use SSH keys on Linux or macOS](#)
- [Create and use SSH keys on Windows](#)

6. Select **Attach**.

NOTE

To create and attach a compute target for training on Azure Arc-enabled Kubernetes cluster, see [Configure Azure Arc-enabled Machine Learning](#)

IMPORTANT

To attach an Azure Kubernetes Services (AKS) or Azure Arc-enabled Kubernetes cluster, you must be subscription owner or have permission to access AKS cluster resources under the subscription. Otherwise, the cluster list on "attach new compute" page will be blank.

To detach your compute use the following steps:

1. In Azure Machine Learning studio, select **Compute**, **Attached compute**, and the compute you wish to

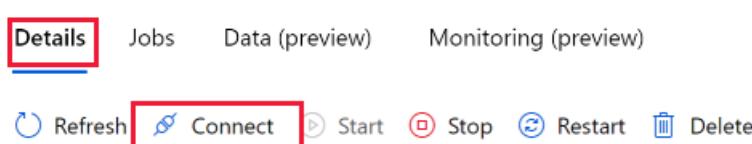
remove.

2. Use the **Detach** link to detach your compute.

Connect with SSH access

After you create a compute with SSH access enabled, use these steps for access.

1. Find the compute in your workspace resources:
 - a. On the left, select **Compute**.
 - b. Use the tabs at the top to select **Compute instance** or **Compute cluster** to find your machine.
2. Select the compute name in the list of resources.
3. Find the connection string:
 - For a **compute instance**, select **Connect** at the top of the **Details** section.



- For a **compute cluster**, select **Nodes** at the top, then select the **Connection string** in the table for your node.

A screenshot of the Azure portal showing the 'Nodes' tab selected for a compute cluster. The table includes columns for Node ID, State, Current run ID, and Connection string. The 'Connection string' column for the first node is highlighted with a red box and contains the value 'ssh azureuser@20.81.59.6...'. Other rows show 'tvm..._dbbef9f541f09a59a3dbbc8626c1883...' and 'Idle'.

Node ID	State	Current run ID	Connection string
tvm..._dbbef9f541f09a59a3dbbc8626c1883...	Idle		ssh azureuser@20.81.59.6...

4. Copy the connection string.
5. For Windows, open PowerShell or a command prompt:
 - a. Go into the directory or folder where your key is stored
 - b. Add the -i flag to the connection string to locate the private key and point to where it is stored:

```
ssh -i <keyname.pem> azureuser@... (rest of connection string)
```

6. For Linux users, follow the steps from [Create and use an SSH key pair for Linux VMs in Azure](#)
7. For SCP use:

```
scp -i key.pem -P {port} {fileToCopyFromLocal } azureuser@yourComputeInstancePublicIP:{destination}
```

Next steps

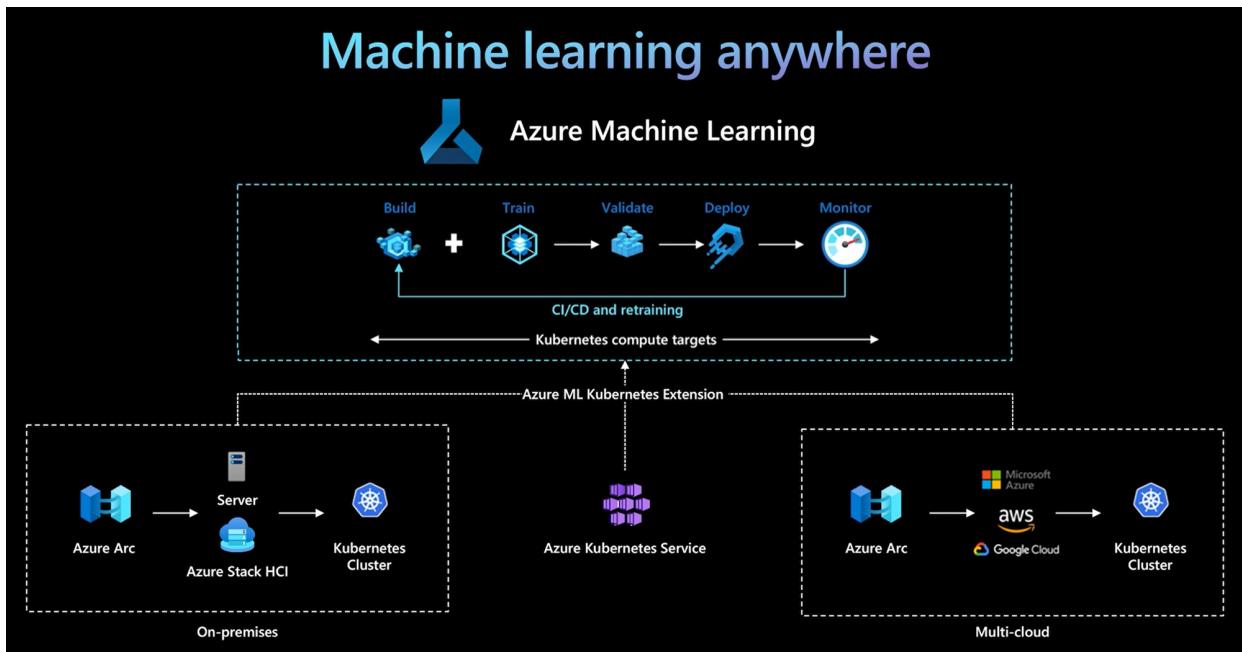
- Use the compute resource to [submit a training run](#).
- Learn how to [efficiently tune hyperparameters](#) to build better models.
- Once you have a trained model, learn [how and where to deploy models](#).
- [Use Azure Machine Learning with Azure Virtual Networks](#)

Introduction to Kubernetes compute target in AzureML

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO: ✓ Azure CLI ml extension v2 (current) ✓ Python SDK azure-ai-ml v2 (preview)

With AzureML CLI/Python SDK v2, AzureML introduced a new compute target - Kubernetes compute target. You can easily enable an existing **Azure Kubernetes Service** (AKS) cluster or **Azure Arc-enabled Kubernetes** (Arc Kubernetes) cluster to become a Kubernetes compute target in AzureML, and use it to train or deploy models.



In this article, you learn about:

- How it works
- Usage scenarios
- Recommended best practices
- KubernetesCompute and legacy AksCompute

How it works

AzureML Kubernetes compute supports two kinds of Kubernetes cluster:

- **AKS cluster** in Azure. With your self-managed AKS cluster in Azure, you can gain security and controls to meet compliance requirement and flexibility to manage teams' ML workload.
- **Arc Kubernetes cluster** outside of Azure. With Arc Kubernetes cluster, you can train or deploy models in any infrastructure on-premises, across multicloud, or the edge.

With a simple cluster extension deployment on AKS or Arc Kubernetes cluster, Kubernetes cluster is seamlessly supported in AzureML to run training or inference workload. It's easy to enable and use an existing Kubernetes cluster for AzureML workload with the following simple steps:

1. Prepare an [Azure Kubernetes Service cluster](#) or [Arc Kubernetes cluster](#).
2. [Deploy the AzureML extension](#).

3. [Attach Kubernetes cluster to your Azure ML workspace](#).
4. Use the Kubernetes compute target from CLI v2, SDK v2, and the Studio UI.

IT-operation team. The IT-operation team is responsible for the first 3 steps above: prepare an AKS or Arc Kubernetes cluster, deploy Azure ML cluster extension, and attach Kubernetes cluster to Azure ML workspace. In addition to these essential compute setup steps, IT-operation team also uses familiar tools such as Azure CLI or kubectl to take care of the following tasks for the data-science team:

- Network and security configurations, such as outbound proxy server connection or Azure firewall configuration, inference router (azureml-fe) setup, SSL/TLS termination, and VNET configuration.
- Create and manage instance types for different ML workload scenarios and gain efficient compute resource utilization.
- Trouble shooting workload issues related to Kubernetes cluster.

Data-science team. Once the IT-operations team finishes compute setup and compute target(s) creation, the data-science team can discover a list of available compute targets and instance types in AzureML workspace. These compute resources can be used for training or inference workload. Data science specifies compute target name and instance type name using their preferred tools or APIs such as AzureML CLI v2, Python SDK v2 (preview), or Studio UI.

Kubernetes usage scenarios

With Arc Kubernetes cluster, you can build, train, and deploy models in any infrastructure on-premises and across multicloud using Kubernetes. This opens some new use patterns previously not possible in cloud setting environment. The following table provides a summary of the new use patterns enabled by AzureML Kubernetes compute:

USAGE PATTERN	LOCATION OF DATA	MOTIVATION	INFRA SETUP & AZURE ML IMPLEMENTATION
Train model in cloud, deploy model on-premises	Cloud	Make use of cloud compute. Either because of elastic compute needs or special hardware such as a GPU. Model must be deployed on-premises because of security, compliance, or latency requirements	1. Azure managed compute in cloud. 2. Customer managed Kubernetes on-premises. 3. Fully automated MLOps in hybrid mode, including training and model deployment steps transitioning seamlessly from cloud to on-premises and vice versa. 4. Repeatable, with all assets tracked properly. Model retrained when necessary, and model deployment updated automatically after retraining.

USAGE PATTERN	LOCATION OF DATA	MOTIVATION	INFRA SETUP & AZURE ML IMPLEMENTATION
Train model on-premises, deploy model in cloud	On-premises	Data must remain on-premises due to data-residency requirements. Deploy model in the cloud for global service access or for compute elasticity for scale and throughput.	<ol style="list-style-type: none"> 1. Azure managed compute in cloud. 2. Customer managed Kubernetes on-premises. 3. Fully automated MLOps in hybrid mode, including training and model deployment steps transitioning seamlessly from cloud to on-premises and vice versa. 4. Repeatable, with all assets tracked properly. Model retrained when necessary, and model deployment updated automatically after retraining.
Bring your own AKS in Azure	Cloud	More security and controls. All private IP machine learning to prevent data exfiltration.	<ol style="list-style-type: none"> 1. AKS cluster behind an Azure VNet. 2. Create private endpoints in the same VNet for AzureML workspace and its associated resources. 3. Fully automated MLOps.
Full ML lifecycle on-premises	On-premises	Secure sensitive data or proprietary IP, such as ML models and code/scripts.	<ol style="list-style-type: none"> 1. Outbound proxy server connection on-premises. 2. Azure ExpressRoute and Azure Arc private link to Azure resources. 3. Customer managed Kubernetes on-premises. 4. Fully automated MLOps.

Recommended best practices

Separation of responsibilities between the IT-operations team and data-science team. As we mentioned above, managing your own compute and infrastructure for ML workload is a complicated task, and it's best to be done by IT-operations team so data-science team can focus on ML models for organizational efficiency.

Create and manage instance types for different ML workload scenarios. Each ML workload uses different amounts of compute resources such as CPU/GPU and memory. AzureML implements instance type as Kubernetes custom resource definition (CRD) with properties of nodeSelector and resource request/limit. With a carefully curated list of instance types, IT-operations can target ML workload on specific node(s) and manage compute resource utilization efficiently.

Multiple AzureML workspaces share the same Kubernetes cluster. You can attach Kubernetes cluster multiple times to the same AzureML workspace or different AzureML workspaces, creating multiple compute targets in one workspace or multiple workspaces. Since many customers organize data science projects around AzureML workspace, multiple data science projects can now share the same Kubernetes cluster. This significantly reduces ML infrastructure management overheads and IT cost saving.

Team/project workload isolation using Kubernetes namespace. When you attach Kubernetes cluster to AzureML workspace, you can specify a Kubernetes namespace for the compute target. All workloads run by the

compute target will be placed under the specified namespace.

KubernetesCompute and legacy AksCompute

With AzureML CLI/Python SDK v1, you can deploy models on AKS using AksCompute target. Both KubernetesCompute target and AksCompute target support AKS integration, however they support it differently. Following table shows key differences.

CAPABILITIES	AKS INTEGRATION WITH AKSCOMPUTE (LEGACY)	AKS INTEGRATION WITH KUBERNETESCOMPUTE
CLI/SDK v1	Yes	No
CLI/SDK v2	No	Yes
Training	No	Yes
Real-time inference	Yes	Yes
Batch inference	No	Yes
Real-time inference new features	No new features development	Active roadmap

With these key differences and overall AzureML evolves to use SDK/CLI v2, AzureML recommends you to use Kubernetes compute target to deploy models if you decide to use AKS for model deployment.

Next steps

- [Step 1: Deploy AzureML extension](#)
- [Step 2: Attach Kubernetes cluster to workspace](#)
- [Create and manage instance types](#)

Other resources

- [Kubernetes version and region availability](#)
- [Work with custom data storage](#)

Examples

All AzureML examples can be found in <https://github.com/Azure/azureml-examples.git>.

For any AzureML example, you only need to update the compute target name to your Kubernetes compute target, then you're all done.

- Explore training job samples with CLI v2 - <https://github.com/Azure/azureml-examples/tree/main/cli/jobs>
- Explore model deployment with online endpoint samples with CLI v2 - <https://github.com/Azure/azureml-examples/tree/main/cli/endpoints/online/kubernetes>
- Explore batch endpoint samples with CLI v2 - <https://github.com/Azure/azureml-examples/tree/main/cli/endpoints/batch>
- Explore training job samples with SDK v2 -<https://github.com/Azure/azureml-examples/tree/main/sdk/jobs>
- Explore model deployment with online endpoint samples with SDK v2 -<https://github.com/Azure/azureml-examples/tree/main/sdk/endpoints/online/kubernetes>

Deploy AzureML extension on AKS or Arc Kubernetes cluster

9/22/2022 • 12 minutes to read • [Edit Online](#)

To enable your AKS or Arc Kubernetes cluster to run training jobs or inference workloads, you must first deploy the AzureML extension on an AKS or Arc Kubernetes cluster. The AzureML extension is built on the [cluster extension for AKS](#) and [cluster extension or Arc Kubernetes](#), and its lifecycle can be managed easily with Azure CLI [k8s-extension](#).

In this article, you can learn:

- Prerequisites
- Limitations
- Review AzureML extension config settings
- AzureML extension deployment scenarios
- Verify AzureML extension deployment
- Review AzureML extension components
- Manage AzureML extension

Prerequisites

- An AKS cluster is up and running in Azure.
- Or an Arc Kubernetes cluster is up and running. Follow instructions in [connect existing Kubernetes cluster to Azure Arc](#).
 - If the cluster is an Azure RedHat OpenShift Service (ARO) cluster or OpenShift Container Platform (OCP) cluster, you must satisfy other prerequisite steps as documented in the [Reference for configuring Kuberentes cluster](#) article.
- The Kubernetes cluster must have minimum of 4 vCPU cores and 8-GB memory.
- Cluster running behind an outbound proxy server or firewall needs extra [network configurations](#)
- Install or upgrade Azure CLI to version 2.24.0 or higher.
- Install or upgrade Azure CLI extension `k8s-extension` to version 1.2.3 or higher.

Limitations

- [Using a service principal with AKS is not supported](#) by Azure Machine Learning. The AKS cluster must use a [system-assigned managed identity](#) instead.
- [Disabling local accounts](#) for AKS is not supported by Azure Machine Learning. When the AKS Cluster is deployed, local accounts are enabled by default.
- If your AKS cluster has an [Authorized IP range enabled to access the API server](#), enable the AzureML control plane IP ranges for the AKS cluster. The AzureML control plane is deployed across paired regions. Without access to the API server, the machine learning pods can't be deployed. Use the [IP ranges](#) for both the [paired regions](#) when enabling the IP ranges in an AKS cluster.
- If you've previously followed the steps from [AzureML AKS v1 document](#) to create or attach your AKS as inference cluster, use the following link to [clean up the legacy azureml-fe related resources](#) before you continue the next step.

Review AzureML extension configuration settings

You can use AzureML CLI command `k8s-extension create` to deploy AzureML extension. CLI

`k8s-extension create` allows you to specify a set of configuration settings in `key=value` format using `--config` or `--config-protected` parameter. Following is the list of available configuration settings to be specified during AzureML extension deployment.

CONFIGURATION SETTING KEY NAME	DESCRIPTION	TRAINING	INFERENCE	TRAINING AND INFERENCE
<code>enableTraining</code>	<code>True</code> or <code>False</code> , default <code>False</code> . Must be set to <code>True</code> for AzureML extension deployment with Machine Learning model training and batch scoring support.	✓	N/A	✓
<code>enableInference</code>	<code>True</code> or <code>False</code> , default <code>False</code> . Must be set to <code>True</code> for AzureML extension deployment with Machine Learning inference support.	N/A	✓	✓
<code>allowInsecureConnection</code>	<code>True</code> or <code>False</code> , default <code>False</code> . Can be set to <code>True</code> to use inference HTTP endpoints for development or test purposes.	N/A	Optional	Optional
<code>inferenceRouterService</code>	<code>loadBalancer</code> , <code>nodePort</code> or <code>clusterIP</code> . Required if <code>enableInference=True</code> .	N/A	✓	✓
<code>internalLoadBalancerProtocol</code>	This config is only applicable for Azure Kubernetes Service(AKS) cluster now. Set to <code>azure</code> to allow the inference router using internal load balancer.	N/A	Optional	Optional

CONFIGURATION SETTING KEY NAME	DESCRIPTION	TRAINING	INFERENCE	TRAINING AND INFERENCE
<code>sslSecret</code>	The name of Kubernetes secret in <code>azureml</code> namespace to store <code>cert.pem</code> (PEM-encoded TLS/SSL cert) and <code>key.pem</code> (PEM-encoded TLS/SSL key), required for inference HTTPS endpoint support, when <code>allowInsecureConnections</code> is set to False. You can find a sample YAML definition of <code>sslSecret</code> here . Use this config or combination of <code>sslCertPemFile</code> and <code>sslKeyPemFile</code> protected config settings.	N/A	Optional	Optional
<code>sslCname</code>	An TLS/SSL CName is used by inference HTTPS endpoint. Required if <code>allowInsecureConnections=False</code>	N/A	Optional	Optional
<code>inferenceRouterHA</code>	<code>True</code> or <code>False</code> , default <code>True</code> . By default, AzureML extension will deploy three inference router replicas for high availability, which requires at least three worker nodes in a cluster. Set to <code>False</code> if your cluster has fewer than three worker nodes, in this case only one inference router service is deployed.	N/A	Optional	Optional

CONFIGURATION SETTING KEY NAME	DESCRIPTION	TRAINING	INFERENCE	TRAINING AND INFERENCE
<code>nodeSelector</code>	<p>By default, the deployed kubernetes resources are randomly deployed to one or more nodes of the cluster, and daemonset resources are deployed to ALL nodes. If you want to restrict the extension deployment to specific nodes with label <code>key1=value1</code> and <code>key2=value2</code>, use</p> <pre><code>nodeSelector.key1=value1 , nodeSelector.key2=value2</code></pre> <p>correspondingly.</p>	Optional	Optional	Optional
<code>installNvidiaDevicePlu</code>	<p><code>True</code> or <code>False</code>, default <code>False</code>.</p> <p>NVIDIA Device Plugin is required for ML workloads on NVIDIA GPU hardware. By default, AzureML extension deployment won't install NVIDIA Device Plugin regardless Kubernetes cluster has GPU hardware or not. User can specify this setting to <code>True</code>, to install it, but make sure to fulfill Prerequisites.</p>	Optional	Optional	Optional
<code>installPromOp</code>	<p><code>True</code> or <code>False</code>, default <code>True</code>.</p> <p>AzureML extension needs prometheus operator to manage prometheus. Set to <code>False</code> to reuse existing prometheus operator. Compatible kube-prometheus-stack helm chart versions are from 9.3.4 to 30.0.1.</p>	Optional	Optional	Optional

CONFIGURATION SETTING KEY NAME	DESCRIPTION	TRAINING	INFERENCE	TRAINING AND INFERENCE
<code>installVolcano</code>	<code>True</code> or <code>False</code> , default <code>True</code> . AzureML extension needs volcano scheduler to schedule the job. Set to <code>False</code> to reuse existing volcano scheduler. Supported volcano scheduler versions are 1.4, 1.5.	Optional	N/A	Optional
<code>installDcgmExporter</code>	<code>True</code> or <code>False</code> , default <code>False</code> . Dcgm-exporter can expose GPU metrics for AzureML workloads, which can be monitored in Azure portal. Set <code>installDcgmExporter</code> to <code>True</code> to install dcgm-exporter. But if you want to utilize your own dcgm-exporter, refer to DCGM exporter	Optional	Optional	Optional
CONFIGURATION PROTECTED SETTING KEY NAME	DESCRIPTION	TRAINING	INFERENCE	TRAINING AND INFERENCE
<code>sslCertPemFile</code> , <code>sslKeyPemFile</code>	Path to TLS/SSL certificate and key file (PEM-encoded), required for AzureML extension deployment with inference HTTPS endpoint support, when <code>allowInsecureConnections</code> is set to False. Note PEM file with pass phrase protected isn't supported	N/A	Optional	Optional

As you can see from above configuration settings table, the combinations of different configuration settings allow you to deploy AzureML extension for different ML workload scenarios:

- For training job and batch inference workload, specify `enableTraining=True`
- For inference workload only, specify `enableInference=True`
- For all kinds of ML workload, specify both `enableTraining=True` and `enableInference=True`

If you plan to deploy AzureML extension for real-time inference workload and want to specify `enableInference=True`, pay attention to following configuration settings related to real-time inference workload:

- `azureml-fe` router service is required for real-time inference support and you need to specify `inferenceRouterServiceType` config setting for `azureml-fe`. `azureml-fe` can be deployed with one of following `inferenceRouterServiceType` :
 - Type `LoadBalancer`. Exposes `azureml-fe` externally using a cloud provider's load balancer. To specify this value, ensure that your cluster supports load balancer provisioning. Note most on-premises Kubernetes clusters might not support external load balancer.
 - Type `NodePort`. Exposes `azureml-fe` on each Node's IP at a static port. You'll be able to contact `azureml-fe`, from outside of cluster, by requesting `<NodeIP>:<NodePort>`. Using `NodePort` also allows you to set up your own load balancing solution and TLS/SSL termination for `azureml-fe`.
 - Type `ClusterIP`. Exposes `azureml-fe` on a cluster-internal IP and it makes `azureml-fe` only reachable from within the cluster. For `azureml-fe` to serve inference requests coming outside of cluster, it requires you to set up your own load balancing solution and TLS/SSL termination for `azureml-fe`.
- To ensure high availability of `azureml-fe` routing service, AzureML extension deployment by default creates three replicas of `azureml-fe` for clusters having three nodes or more. If your cluster has less than 3 nodes, set `inferenceLoadbalancerHA=False`.
- You also want to consider using **HTTPS** to restrict access to model endpoints and secure the data that clients submit. For this purpose, you would need to specify either `sslSecret` config setting or combination of `sslKeyPemFile` and `sslCertPemFile` config-protected settings.
- By default, AzureML extension deployment expects config settings for **HTTPS** support. For development or testing purposes, **HTTP** support is conveniently provided through config setting `allowInsecureConnections=True`.

AzureML extension deployment - CLI examples and Azure portal

- [Azure CLI](#)
- [Azure portal](#)

To deploy AzureML extension with CLI, use `az k8s-extension create` command passing in values for the mandatory parameters.

We list four typical extension deployment scenarios for reference. To deploy extension for your production usage, carefully read the complete list of [configuration settings](#).

- **Use AKS cluster in Azure for a quick proof of concept to run all kinds of ML workload, i.e., to run training jobs or to deploy models as online/batch endpoints**

For AzureML extension deployment on AKS cluster, make sure to specify `managedClusters` value for `--cluster-type` parameter. Run the following Azure CLI command to deploy AzureML extension:

```
az k8s-extension create --name <extension-name> --extension-type Microsoft.AzureML.Kubernetes --config enableTraining=True enableInference=True inferenceRouterServiceType=LoadBalancer allowInsecureConnections=True inferenceLoadbalancerHA=False --cluster-type managedClusters --cluster-name <your-AKS-cluster-name> --resource-group <your-RG-name> --scope cluster
```

- **Use Arc Kubernetes cluster outside of Azure for a quick proof of concept, to run training jobs only**

For AzureML extension deployment on [Arc Kubernetes](#) cluster, you would need to specify `connectedClusters` value for `--cluster-type` parameter. Run the following Azure CLI command to deploy AzureML extension:

```
az k8s-extension create --name <extension-name> --extension-type Microsoft.AzureML.Kubernetes --config enableTraining=True --cluster-type connectedClusters --cluster-name <your-connected-cluster-name> --resource-group <your-RG-name> --scope cluster
```

- **Enable an AKS cluster in Azure for production training and inference workload** For AzureML extension deployment on AKS, make sure to specify `managedClusters` value for `--cluster-type` parameter. Assuming your cluster has more than three nodes, and you'll use an Azure public load balancer and HTTPS for inference workload support. Run the following Azure CLI command to deploy AzureML extension:

```
az k8s-extension create --name <extension-name> --extension-type Microsoft.AzureML.Kubernetes --config enableTraining=True enableInference=True inferenceRouterServiceType=LoadBalancer sslCname=<ssl cname> --config-protected sslCertPemFile=<file-path-to-cert-PEM> sslKeyPemFile=<file-path-to-cert-KEY> --cluster-type managedClusters --cluster-name <your-AKS-cluster-name> --resource-group <your-RG-name> --scope cluster
```

- **Enable an Arc Kubernetes cluster anywhere for production training and inference workload using NVIDIA GPUs**

For AzureML extension deployment on [Arc Kubernetes](#) cluster, make sure to specify `connectedClusters` value for `--cluster-type` parameter. Assuming your cluster has more than three nodes, you'll use a NodePort service type and HTTPS for inference workload support, run following Azure CLI command to deploy AzureML extension:

```
az k8s-extension create --name <extension-name> --extension-type Microsoft.AzureML.Kubernetes --config enableTraining=True enableInference=True inferenceRouterServiceType=NodePort sslCname=<ssl cname> installNvidiaDevicePlugin=True installDcgmExporter=True --config-protected sslCertPemFile=<file-path-to-cert-PEM> sslKeyPemFile=<file-path-to-cert-KEY> --cluster-type connectedClusters --cluster-name <your-connected-cluster-name> --resource-group <your-RG-name> --scope cluster
```

Verify AzureML extension deployment

1. Run the following CLI command to check AzureML extension details:

```
az k8s-extension show --name <extension-name> --cluster-type connectedClusters --cluster-name <your-connected-cluster-name> --resource-group <resource-group>
```

2. In the response, look for "name" and "provisioningState": "Succeeded". Note it might show "provisioningState": "Pending" for the first few minutes.
3. If the provisioningState shows Succeeded, run the following command on your machine with the kubeconfig file pointed to your cluster to check that all pods under "azureml" namespace are in 'Running' state:

```
kubectl get pods -n azureml
```

Review AzureML extension component

Upon AzureML extension deployment completes, you can use `kubectl get deployments -n azureml` to see list of resources created in the cluster. It usually consists a subset of following resources per configuration settings specified.

RESOURCE NAME	RESOURCE TYPE	TRAINING	INFERENCE	TRAINING AND INFERENCE	DESCRIPTION	COMMUNICATION WITH CLOUD
relayserver	Kubernetes deployment	✓	✓	✓	Relayserver is only created for Arc Kubernetes cluster, and not in AKS cluster. Relayserver works with Azure Relay to communicate with the cloud services.	Receive the request of job creation, model deployment from cloud service; sync the job status with cloud service.
gateway	Kubernetes deployment	✓	✓	✓	The gateway is used to communicate and send data back and forth.	Send nodes and cluster resource information to cloud services.
aml-operator	Kubernetes deployment	✓	N/A	✓	Manage the lifecycle of training jobs.	Token exchange with the cloud token service for authentication and authorization of Azure Container Registry.
metrics-controller-manager	Kubernetes deployment	✓	✓	✓	Manage the configuration for Prometheus	N/A
{EXTENSION-NAME}-kube-state-metrics	Kubernetes deployment	✓	✓	✓	Export the cluster-related metrics to Prometheus.	N/A
{EXTENSION-NAME}-prometheus-operator	Kubernetes deployment	Optional	Optional	Optional	Provide Kubernetes native deployment and management of Prometheus and related monitoring components.	N/A

Resource Name	Resource Type	Training	Inference	Training and Inference	Description	Communication with Cloud
amlarc-identity-controller	Kubernetes deployment	N/A	✓	✓	Request and renew Azure Blob/Azure Container Registry token through managed identity.	Token exchange with the cloud token service for authentication and authorization of Azure Container Registry and Azure Blob used by inference/model deployment.
amlarc-identity-proxy	Kubernetes deployment	N/A	✓	✓	Request and renew Azure Blob/Azure Container Registry token through managed identity.	Token exchange with the cloud token service for authentication and authorization of Azure Container Registry and Azure Blob used by inference/model deployment.
azureml-fe-v2	Kubernetes deployment	N/A	✓	✓	The front-end component that routes incoming inference requests to deployed services.	Send service logs to Azure Blob.
inference-operator-controller-manager	Kubernetes deployment	N/A	✓	✓	Manage the lifecycle of inference endpoints.	N/A
volcano-admission	Kubernetes deployment	Optional	N/A	Optional	Volcano admission webhook.	N/A

RESOURCE NAME	RESOURCE TYPE	TRAINING	INFERENCE	TRAINING AND INFERENCE	DESCRIPTION	COMMUNICATION WITH CLOUD
volcano-controllers	Kubernetes deployment	Optional	N/A	Optional	Manage the lifecycle of Azure Machine Learning training job pods.	N/A
volcano-scheduler	Kubernetes deployment	Optional	N/A	Optional	Used to perform in-cluster job scheduling.	N/A
fluent-bit	Kubernetes daemonset	✓	✓	✓	Gather the components' system log.	Upload the components' system log to cloud.
{EXTENSION-NAME}-dcgm-exporter	Kubernetes daemonset	Optional	Optional	Optional	dcmg-exporter exposes GPU metrics for Prometheus.	N/A
nvidia-device-plugin-daemonset	Kubernetes daemonset	Optional	Optional	Optional	nvidia-device-plugin-daemonset exposes GPUs on each node of your cluster	N/A
prometheus-prom-prometheus	Kubernetes statefulset	✓	✓	✓	Gather and send job metrics to cloud.	Send job metrics like cpu/gpu/memory utilization to cloud.

IMPORTANT

- Azure Relay resource is under the same resource group as the Arc cluster resource. It is used to communicate with the Kubernetes cluster and modifying them will break attached compute targets.
- By default, the kubernetes deployment resources are randomly deployed to 1 or more nodes of the cluster, and daemonset resources are deployed to ALL nodes. If you want to restrict the extension deployment to specific nodes, use `nodeSelector` configuration setting described as below.

NOTE

- {EXTENSION-NAME}: is the extension name specified with `az k8s-extension create --name` CLI command.

Manage AzureML extension

Update, list, show and delete an AzureML extension.

- For AKS cluster without Azure Arc connected, refer to [Usage of AKS extensions](#).
- For Azure Arc-enabled Kubernetes, refer to [Usage of cluster extensions](#).

Next steps

- [Step 2: Attach Kubernetes cluster to workspace](#)
- [Create and manage instance types](#)
- [AzureML inference router and connectivity requirements](#)
- [Secure AKS inferencing environment](#)

Attach a Kubernetes cluster to AzureML workspace

9/22/2022 • 3 minutes to read • [Edit Online](#)

Once AzureML extension is deployed on AKS or Arc Kubernetes cluster, you can attach the Kubernetes cluster to AzureML workspace and create compute targets for ML professionals to use.

Some key considerations when attaching Kubernetes cluster to AzureML workspace:

- If you need to access Azure resource seamlessly from your training script, you can specify a managed identity for Kubernetes compute target during attach operation.
- If you plan to have different compute target for different project/team, you can specify Kubernetes namespace for the compute target to isolate workload among different teams/projects.
- For the same Kubernetes cluster, you can attach it to the same workspace multiple times and create multiple compute targets for different project/team/workload.
- For the same Kubernetes cluster, you can also attach it to multiple workspaces, and the multiple workspaces can share the same Kubernetes cluster.

Prerequisite

Azure Machine Learning workspace defaults to having a system-assigned managed identity to access Azure ML resources. The steps are completed if the system assigned default setting is on.

Otherwise, if a user-assigned managed identity is specified in Azure Machine Learning workspace creation, the following role assignments need to be granted to the managed identity manually before attaching the compute.

AZURE RESOURCE NAME	ROLE TO BE ASSIGNED	DESCRIPTION
Azure Relay	Azure Relay Owner	Only applicable for Arc-enabled Kubernetes cluster. Azure Relay isn't created for AKS cluster without Arc connected.
Azure Arc-enabled Kubernetes or AKS	Reader	Applicable for both Arc-enabled Kubernetes cluster and AKS cluster.

Azure Relay resource is created during the extension deployment under the same Resource Group as the Arc-enabled Kubernetes cluster.

- [Azure CLI](#)
- [Python SDK](#)
- [Studio](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

The following commands show how to attach an AKS and Azure Arc-enabled Kubernetes cluster, and use it as a compute target with managed identity enabled.

AKS cluster

```
az ml compute attach --resource-group <resource-group-name> --workspace-name <workspace-name> --type  
Kubernetes --name k8s-compute --resource-id "/subscriptions/<subscription-id>/resourceGroups/<resource-  
group-name>/providers/Microsoft.ContainerService/managedclusters/<cluster-name>" --identity-type  
SystemAssigned --namespace <Kubernetes namespace to run AzureML workloads> --no-wait
```

Arc Kubernetes cluster

```
az ml compute attach --resource-group <resource-group-name> --workspace-name <workspace-name> --type  
Kubernetes --name amlarc-compute --resource-id "/subscriptions/<subscription-id>/resourceGroups/<resource-  
group-name>/providers/Microsoft.Kubernetes/connectedClusters/<cluster-name>" --user-assigned-identities  
"subscriptions/<subscription-id>/resourceGroups/<resource-group-  
name>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<identity-name>" --no-wait
```

Set the `--type` argument to `Kubernetes`. Use the `identity_type` argument to enable `SystemAssigned` or `UserAssigned` managed identities.

IMPORTANT

`--user-assigned-identities` is only required for `UserAssigned` managed identities. Although you can provide a list of comma-separated user managed identities, only the first one is used when you attach your cluster.

Compute attach won't create the Kubernetes namespace automatically or validate whether the kubernetes namespace existed. You need to verify that the specified namespace exists in your cluster, otherwise, any AzureML workloads submitted to this compute will fail.

Next steps

- [Create and manage instance types](#)
- [AzureML inference router and connectivity requirements](#)
- [Secure AKS inferencing environment](#)

Create and manage instance types for efficient compute resource utilization

9/22/2022 • 3 minutes to read • [Edit Online](#)

What are instance types?

Instance types are an Azure Machine Learning concept that allows targeting certain types of compute nodes for training and inference workloads. For an Azure VM, an example for an instance type is `STANDARD_D2_V3`.

In Kubernetes clusters, instance types are represented in a custom resource definition (CRD) that is installed with the AzureML extension. Instance types are represented by two elements in AzureML extension: `nodeSelector` and `resources`.

In short, a `nodeSelector` lets you specify which node a pod should run on. The node must have a corresponding label. In the `resources` section, you can set the compute resources (CPU, memory and NVIDIA GPU) for the pod.

Default instance type

By default, a `defaultinstancetype` with following definition is created when you attach Kuberenetus cluster to AzureML workspace:

- No `nodeSelector` is applied, meaning the pod can get scheduled on any node.
- The workload's pods are assigned default resources with 0.6 cpu cores, 1536Mi memory and 0 GPU:

```
resources:  
  requests:  
    cpu: "0.6"  
    memory: "1536Mi"  
  limits:  
    cpu: "0.6"  
    memory: "1536Mi"  
    nvidia.com/gpu: null
```

NOTE

- The default instance type purposefully uses little resources. To ensure all ML workloads run with appropriate resources, for example GPU resource, it is highly recommended to create custom instance types.
- `defaultinstancetype` will not appear as an `InstanceType` custom resource in the cluster when running the command `kubectl get instancetype`, but it will appear in all clients (UI, CLI, SDK).
- `defaultinstancetype` can be overridden with a custom instance type definition having the same name as `defaultinstancetype` (see [Create custom instance types](#) section)

Create custom instance types

To create a new instance type, create a new custom resource for the instance type CRD. For example:

```
kubectl apply -f my_instance_type.yaml
```

With `my_instance_type.yaml`:

```
apiVersion: amlarc.azureml.com/v1alpha1
kind: InstanceType
metadata:
  name: myinstancetype
spec:
  nodeSelector:
    mylabel: mylabelvalue
  resources:
    limits:
      cpu: "1"
      nvidia.com/gpu: 1
      memory: "2Gi"
    requests:
      cpu: "700m"
      memory: "1500Mi"
```

The following steps will create an instance type with the labeled behavior:

- Pods will be scheduled only on nodes with label `mylabel: mylabelvalue`.
- Pods will be assigned resource requests of `700m` CPU and `1500Mi` memory.
- Pods will be assigned resource limits of `1` CPU, `2Gi` memory and `1` NVIDIA GPU.

NOTE

- NVIDIA GPU resources are only specified in the `limits` section as integer values. For more information, see the Kubernetes [documentation](#).
- CPU and memory resources are string values.
- CPU can be specified in millicores, for example `100m`, or in full numbers, for example `"1"` is equivalent to `1000m`.
- Memory can be specified as a full number + suffix, for example `1024Mi` for 1024 MiB.

It's also possible to create multiple instance types at once:

```
kubectl apply -f my_instance_type_list.yaml
```

With `my_instance_type_list.yaml`:

```
apiVersion: amlarc.azureml.com/v1alpha1
kind: InstanceTypeList
items:
- metadata:
  name: cpusmall
  spec:
    resources:
      requests:
        cpu: "100m"
        memory: "100Mi"
    limits:
      cpu: "1"
      nvidia.com/gpu: 0
      memory: "1Gi"

- metadata:
  name: defaultinstancetype
  spec:
    resources:
      requests:
        cpu: "1"
        memory: "1Gi"
    limits:
      cpu: "1"
      nvidia.com/gpu: 0
      memory: "1Gi"
```

The above example creates two instance types: `cpusmall` and `defaultinstancetype`. This `defaultinstancetype` definition will override the `defaultinstancetype` definition created when Kubernetes cluster was attached to AzureML workspace.

If a training or inference workload is submitted without an instance type, it uses the `defaultinstancetype`. To specify a default instance type for a Kubernetes cluster, create an instance type with name `defaultinstancetype`. It will automatically be recognized as the default.

Select instance type to submit training job

To select an instance type for a training job using CLI (V2), specify its name as part of the `resources` properties section in job YAML. For example:

```
command: python -c "print('Hello world!')"
environment:
  image: library/python:latest
compute: azureml:<compute_target_name>
resources:
  instance_type: <instance_type_name>
```

In the above example, replace `<compute_target_name>` with the name of your Kubernetes compute target and `<instance_type_name>` with the name of the instance type you wish to select. If there's no `instance_type` property specified, the system will use `defaultinstancetype` to submit job.

Select instance type to deploy model

To select an instance type for a model deployment using CLI (V2), specify its name for `instance_type` property in deployment YAML. For example:

```
name: blue
app_insights_enabled: true
endpoint_name: <endpoint name>
model:
  path: ./model/sklearn_mnist_model.pkl
code_configuration:
  code: ./script/
  scoring_script: score.py
instance_type: <instance type name>
environment:
  conda_file: file:./model/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1
```

In the above example, replace `<instance_type_name>` with the name of the instance type you wish to select. If there's no `instance_type` property specified, the system will use `defaultinstancetype` to deploy model.

Next steps

- [AzureML inference router and connectivity requirements](#)
- [Secure AKS inferencing environment](#)

AzureML inference router and connectivity requirements

9/22/2022 • 5 minutes to read • [Edit Online](#)

AzureML inference router is a critical component for real-time inference with Kubernetes cluster. In this article, you can learn about:

- What is AzureML inference router
- How autoscaling works
- How to configure and meet inference request performance (# of requests per second and latency)
- Connectivity requirements for AKS inferencing cluster

What is AzureML inference router

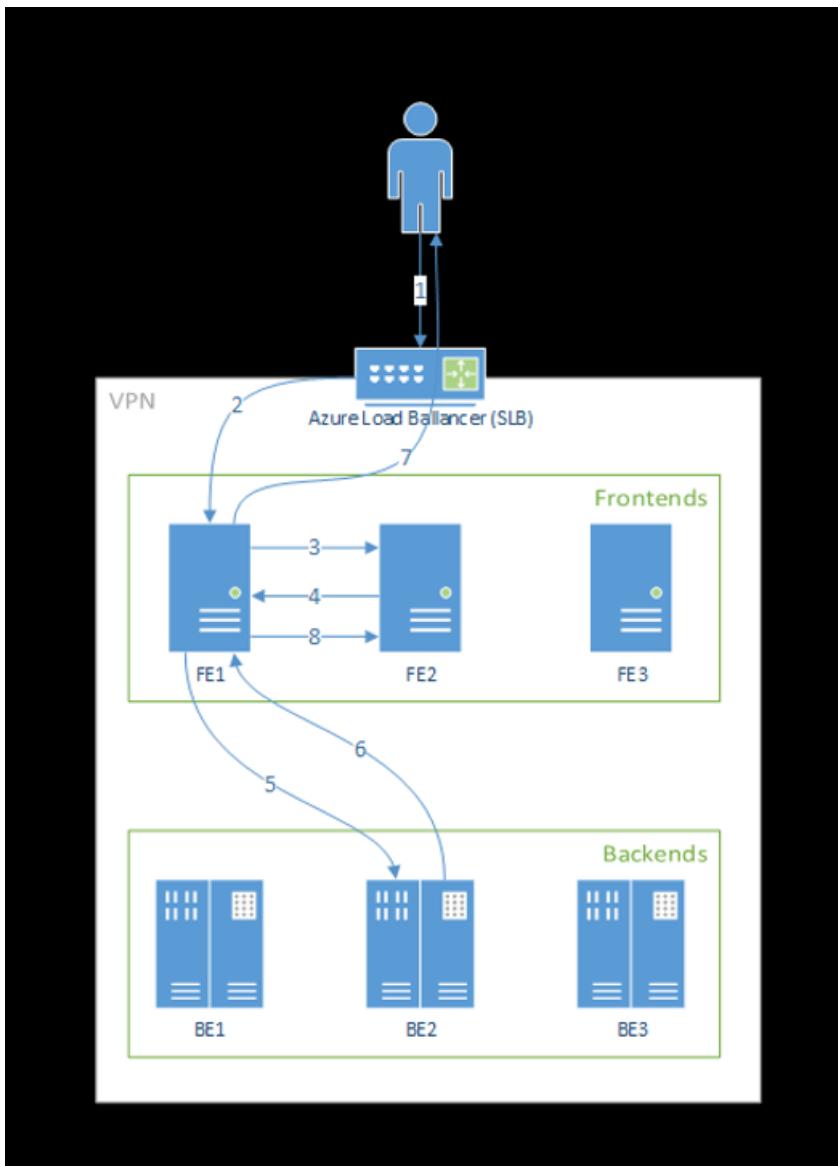
AzureML inference router is the front-end component (`azureml-fe`) which is deployed on AKS or Arc Kubernetes cluster at AzureML extension deployment time. It has following functions:

- Routes incoming inference requests from cluster load balancer or ingress controller to corresponding model pods.
- Load-balance all incoming inference requests with smart coordinated routing.
- manages model pods auto-scaling.
- Fault-tolerant and failover capability, ensuring inference requests is always served for critical business application.

The following steps are how requests are processed by the front-end:

1. Client sends request to the load balancer.
2. Load balancer sends to one of the front-ends.
3. The front-end locates the service router (the front-end instance acting as coordinator) for the service.
4. The service router selects a back-end and returns it to the front-end.
5. The front-end forwards the request to the back-end.
6. After the request has been processed, the back-end sends a response to the front-end component.
7. The front-end propagates the response back to the client.
8. The front-end informs the service router that the back-end has finished processing and is available for other requests.

The following diagram illustrates this flow:



As you can see from above diagram, by default 3 `azureml-fe` instances are created during AzureML extension deployment, one instance acts as coordinating role, and the other instances serve incoming inference requests. The coordinating instance has all information about model pods and makes decision about which model pod to serve incoming request, while the serving `azureml-fe` instances are responsible for routing the request to selected model pod and propagate the response back to the original user.

Autoscaling

AzureML inference router handles autoscaling for all model deployments on the Kubernetes cluster. Since all inference requests go through it, it has the necessary data to automatically scale the deployed model(s).

IMPORTANT

- Do not enable Kubernetes Horizontal Pod Autoscaler (HPA) for model deployments. Doing so would cause the two auto-scaling components to compete with each other. Azureml-fe is designed to auto-scale models deployed by AzureML, where HPA would have to guess or approximate model utilization from a generic metric like CPU usage or a custom metric configuration.
- Azureml-fe does not scale the number of nodes in an AKS cluster, because this could lead to unexpected cost increases. Instead, it scales the number of replicas for the model within the physical cluster boundaries. If you need to scale the number of nodes within the cluster, you can manually scale the cluster or [configure the AKS cluster autoscaler](#).

Autoscaling can be controlled by `scale_settings` property in deployment YAML. The following example demonstrates how to enable autoscaling:

```
# deployment yaml
# other properties skipped
scale_setting:
  type: target_utilization
  min_instances: 3
  max_instances: 15
  target_utilization_percentage: 70
  polling_interval: 10
# other deployment properties continue
```

Decisions to scale up/down is based off of utilization of the current container replicas. The number of replicas that are busy (processing a request) divided by the total number of current replicas is the current utilization. If this number exceeds `target_utilization_percentage`, then more replicas are created. If it's lower, then replicas are reduced. By default, the target utilization is 70%.

Decisions to add replicas are eager and fast (around 1 second). Decisions to remove replicas are conservative (around 1 minute).

For example, if you want to deploy a model service and want to know many instances (pods/replicas) should be configured for target requests per second (RPS) and target response time. You can calculate the required replicas by using the following code:

```
from math import ceil
# target requests per second
targetRps = 20
# time to process the request (in seconds)
reqTime = 10
# Maximum requests per container
maxReqPerContainer = 1
# target_utilization. 70% in this example
targetUtilization = .7

concurrentRequests = targetRps * reqTime / targetUtilization

# Number of container replicas
replicas = ceil(concurrentRequests / maxReqPerContainer)
```

If you have RPS requirements higher than 10K, consider following options:

- Increase resource requests/limits for `azureml-fe` pods, by default it has 2 vCPU and 2G memory request/limit.
- Increase number of instances for `azureml-fe`, by default AzureML creates 3 `azureml-fe` instances per cluster.
- Reach out to Microsoft experts for help.

Understand connectivity requirements for AKS inferencing cluster

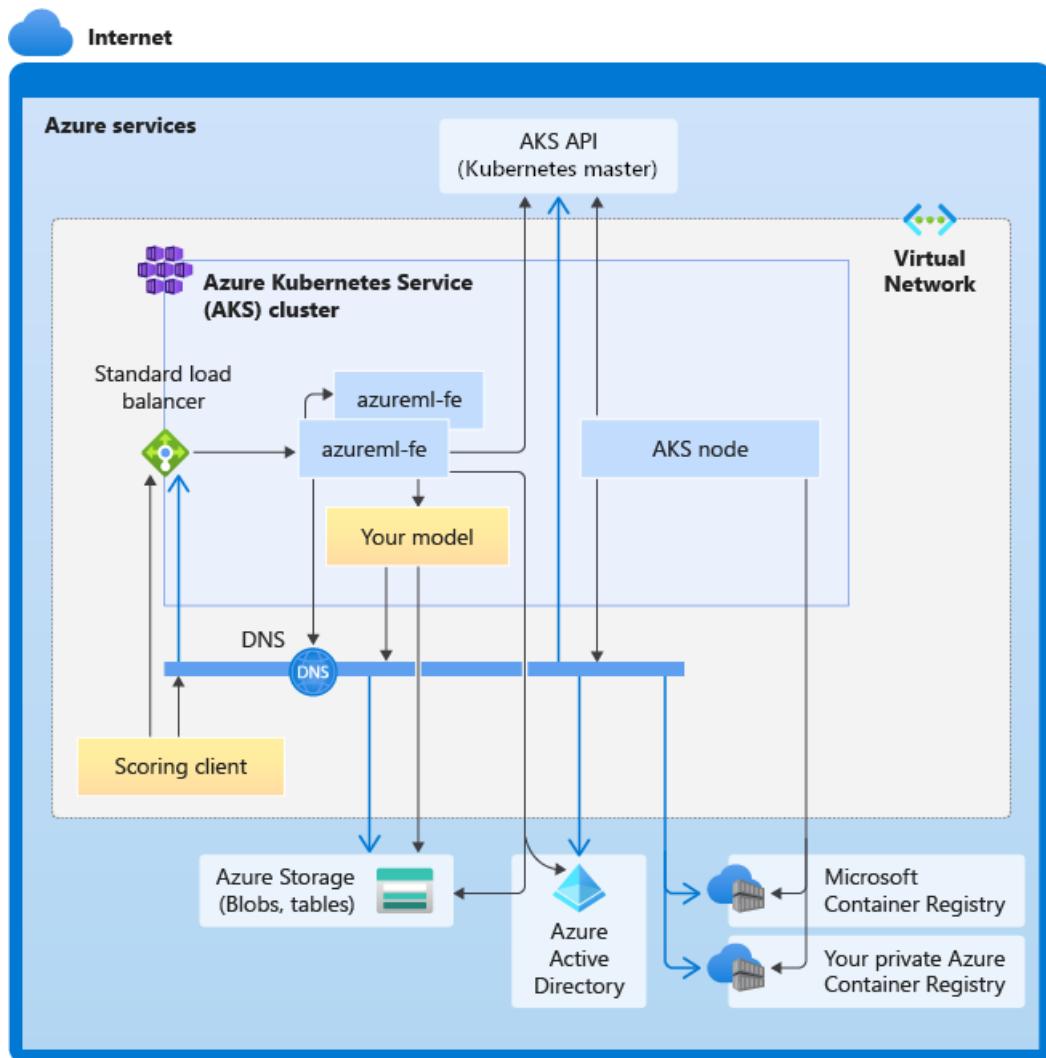
AKS cluster is deployed with one of the following two network models:

- Kubenet networking - The network resources are typically created and configured as the AKS cluster is deployed.
- Azure Container Networking Interface (CNI) networking - The AKS cluster is connected to an existing virtual network resource and configurations.

For Kubenet networking, the network is created and configured properly for Azure Machine Learning service. For the CNI networking, you need to understand the connectivity requirements and ensure DNS resolution and

outbound connectivity for AKS inferencing. For example, you may be using a firewall to block network traffic.

The following diagram shows the connectivity requirements for AKS inferencing. Black arrows represent actual communication, and blue arrows represent the domain names. You may need to add entries for these hosts to your firewall or to your custom DNS server.



For general AKS connectivity requirements, see [Control egress traffic for cluster nodes in Azure Kubernetes Service](#).

For accessing Azure ML services behind a firewall, see [How to access azureml behind firewall](#).

Overall DNS resolution requirements

DNS resolution within an existing VNet is under your control. For example, a firewall or custom DNS server. The following hosts must be reachable:

HOST NAME	USED BY
<cluster>.hcp.<region>.azmk8s.io	AKS API server
mcr.microsoft.com	Microsoft Container Registry (MCR)
<ACR name>.azurecr.io	Your Azure Container Registry (ACR)
<account>.table.core.windows.net	Azure Storage Account (table storage)
<account>.blob.core.windows.net	Azure Storage Account (blob storage)

HOST NAME	USED BY
api.azureml.ms	Azure Active Directory (Azure AD) authentication
ingest-vienna<region>.kusto.windows.net	Kusto endpoint for uploading telemetry

Connectivity requirements in chronological order: from cluster creation to model deployment

Right after azureml-fe is deployed, it will attempt to start and this requires to:

- Resolve DNS for AKS API server
- Query AKS API server to discover other instances of itself (it's a multi-pod service)
- Connect to other instances of itself

Once azureml-fe is started, it requires the following connectivity to function properly:

- Connect to Azure Storage to download dynamic configuration
- Resolve DNS for Azure AD authentication server api.azureml.ms and communicate with it when the deployed service uses Azure AD authentication.
- Query AKS API server to discover deployed models
- Communicate to deployed model PODs

At model deployment time, for a successful model deployment AKS node should be able to:

- Resolve DNS for customer's ACR
- Download images from customer's ACR
- Resolve DNS for Azure BLOBs where model is stored
- Download models from Azure BLOBs

After the model is deployed and service starts, azureml-fe will automatically discover it using AKS API, and will be ready to route request to it. It must be able to communicate to model PODs.

NOTE

If the deployed model requires any connectivity (e.g. querying external database or other REST service, downloading a BLOB etc), then both DNS resolution and outbound communication for these services should be enabled.

Next steps

- [Create and manage instance types](#)
- [Secure AKS inferencing environment](#)

Secure Azure Kubernetes Service inferencing environment

9/22/2022 • 3 minutes to read • [Edit Online](#)

If you have an Azure Kubernetes (AKS) cluster behind of VNet, you would need to secure Azure Machine Learning workspace resources and compute environment using the same VNet. In this article, you'll learn:

- What is a secure AKS inferencing environment
- How to configure a secure AKS inferencing environment

Limitations

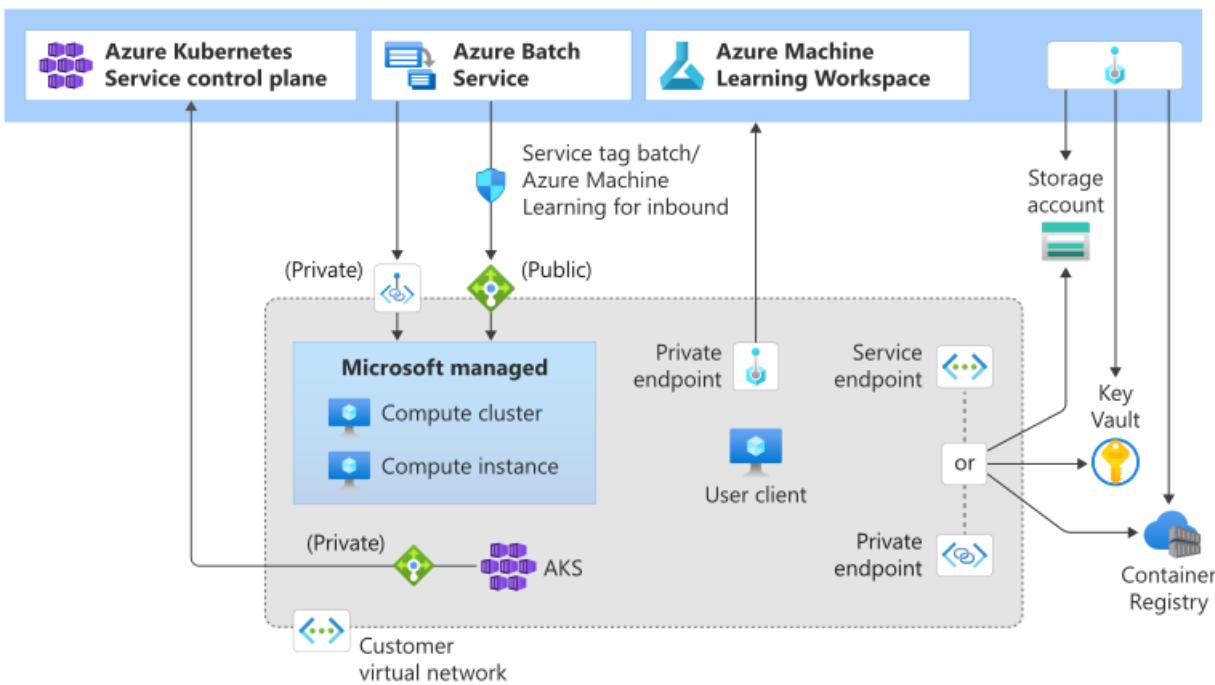
- If your AKS cluster is behind of a VNet, your workspace and its associated resources (storage, key vault, Azure Container Registry) must have private endpoints or service endpoints in the same VNet as AKS cluster's VNet. For more information on securing the workspace and associated resources, see [create a secure workspace](#).
- If your workspace has a **private endpoint**, the Azure Kubernetes Service cluster must be in the same Azure region as the workspace.
- Using a [public fully qualified domain name \(FQDN\) with a private AKS cluster](#) is **not supported** with Azure Machine learning.

What is a secure AKS inferencing environment

Azure Machine Learning AKS inferencing environment consists of workspace, your AKS cluster, and workspace associated resources - Azure Storage, Azure Key Vault, and Azure Container Services(ARC). The following table compares how services access different part of Azure Machine Learning network with or without a VNet.

SCENARIO	WORKSPACE	ASSOCIATED RESOURCES (STORAGE ACCOUNT, KEY VAULT, ACR)	AKS CLUSTER
No virtual network	Public IP	Public IP	Public IP
Public workspace, all other resources in a virtual network	Public IP	Public IP (service endpoint) - or - Private IP (private endpoint)	Private IP
Secure resources in a virtual network	Private IP (private endpoint)	Public IP (service endpoint) - or - Private IP (private endpoint)	Private IP

In a secure AKS inferencing environment, AKS cluster accesses different part of Azure Machine Learning services with private endpoint only (private IP). The following network diagram shows a secured Azure Machine Learning workspace with a private AKS cluster or default AKS cluster behind of VNet.



How to configure a secure AKS inferencing environment

To configure a secure AKS inferencing environment, you must have VNet information for AKS. [VNet](#) can be created independently or during AKS cluster deployment. There are two options for AKS cluster in a VNet:

- Deploy default AKS cluster to your VNet
- Or create private AKS cluster to your VNet

For default AKS cluster, you can find VNet information under the resource group of

`MC_[rg_name][aks_name][region]`.

After you have VNet information for AKS cluster and if you already have workspace available, use following steps to configure a secure AKS inferencing environment:

- Use your AKS cluster VNet information to add new private endpoints for the Azure Storage Account, Azure Key Vault, and Azure Container Registry used by your workspace. These private endpoints should exist in the same VNet as AKS cluster. For more information, see the [secure workspace with private endpoint](#) article.
- If you have other storage that is used by your workspace, add a new private endpoint for that storage. The private endpoint should exist in the AKS cluster VNet and have private DNS zone integration enabled.
- Add a new private endpoint to your workspace. This private endpoint should exist in AKS cluster VNet and have private DNS zone integration enabled.

If you have AKS cluster ready but don't have workspace created yet, you can use AKS cluster VNet when creating the workspace. Use the AKS cluster VNet information when following the [create secure workspace](#) tutorial. Once the workspace has been created, add a new private endpoint to your workspace as the last step. For all the above steps, it's important to ensure that all private endpoints should exist in the same AKS cluster VNet and have private DNS zone integration enabled.

Special notes for configuring a secure AKS inferencing environment:

- Use system-assigned managed identity when creating workspace, as storage account with private endpoint only allows access with system-assigned managed identity.
- When attaching AKS cluster to an HBI workspace, assign a system-assigned managed identity with both `Storage Blob Data Contributor` and `Storage Account Contributor` roles.
- If you're using default ACR created by workspace, ensure you have the **premium SKU** for ACR. Also enable the `Firewall exception` to allow trusted Microsoft services to access ACR.

- If your workspace is also behind of VNet, follow instruction [securely connect to your workspace](#) to access workspace.
- For storage account private endpoint, make sure to enable

Allow Azure services on the trusted services list to access this storage account .

Next steps

This article is part of a series on securing an Azure Machine Learning workflow. See the other articles in this series:

- [Virtual network overview](#)
- [Secure the training environment](#)
- [Secure online endpoints \(inference\)](#)
- [Enable studio functionality](#)
- [Use custom DNS](#)
- [Use a firewall](#)
- [Tutorial: Create a secure workspace](#)
- [Tutorial: Create a secure workspace using a template](#)
- [API platform network isolation](#)

Plan to manage costs for Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

This article describes how to plan and manage costs for Azure Machine Learning. First, you use the Azure pricing calculator to help plan for costs before you add any resources. Next, as you add the Azure resources, review the estimated costs.

After you've started using Azure Machine Learning resources, use the cost management features to set budgets and monitor costs. Also review the forecasted costs and identify spending trends to identify areas where you might want to act.

Understand that the costs for Azure Machine Learning are only a portion of the monthly costs in your Azure bill. If you are using other Azure services, you're billed for all the Azure services and resources used in your Azure subscription, including the third-party services. This article explains how to plan for and manage costs for Azure Machine Learning. After you're familiar with managing costs for Azure Machine Learning, apply similar methods to manage costs for all the Azure services used in your subscription.

For more information on optimizing costs, see [how to manage and optimize cost in Azure Machine Learning](#).

Prerequisites

Cost analysis in Cost Management supports most Azure account types, but not all of them. To view the full list of supported account types, see [Understand Cost Management data](#).

To view cost data, you need at least read access for an Azure account. For information about assigning access to Azure Cost Management data, see [Assign access to data](#).

Estimate costs before using Azure Machine Learning

- Use the [Azure pricing calculator](#) to estimate costs before you create the resources in an Azure Machine Learning workspace. On the left, select **AI + Machine Learning**, then select **Azure Machine Learning** to begin.

The following screenshot shows the cost estimation by using the calculator:

Your Estimate +

Your Estimate

Estimate total: \$167.17

Azure Machine Learning Model Deployment with AKS, 1 D3 v2 (4 ... \$167.17

Billing Option

Save up to 72% on pay-as-you-go prices with 1-year or 3-year Reserved Virtual Machine Instances. Reserved Instances are great for applications with steady-state usage and applications that require reserved capacity. [Learn more about Reserved VM Instances pricing.](#)

Pay as you go
 1 year reserved (~35% savings)
 3 year reserved (~58% savings)

1	x	730	=	\$167.17
Virtual		Hours		Per month

[Chat with Sales](#)

As you add new resources to your workspace, return to this calculator and add the same resource here to update your cost estimates.

For more information, see [Azure Machine Learning pricing](#).

Understand the full billing model for Azure Machine Learning

Azure Machine Learning runs on Azure infrastructure that accrues costs along with Azure Machine Learning when you deploy the new resource. It's important to understand that additional infrastructure might accrue cost. You need to manage that cost when you make changes to deployed resources.

Costs that typically accrue with Azure Machine Learning

When you create resources for an Azure Machine Learning workspace, resources for other Azure services are also created. They are:

- [Azure Container Registry](#) Basic account
- [Azure Block Blob Storage](#) (general purpose v1)
- [Key Vault](#)
- [Application Insights](#)

When you create a [compute instance](#), the VM stays on so it is available for your work. [Set up a schedule](#) to automatically start and stop the compute instance (preview) to save cost when you aren't planning to use it.

Costs might accrue before resource deletion

Before you delete an Azure Machine Learning workspace in the Azure portal or with Azure CLI, the following subresources are common costs that accumulate even when you are not actively working in the workspace. If you are planning on returning to your Azure Machine Learning workspace at a later time, these resources may

continue to accrue costs.

- VMs
- Load Balancer
- Virtual Network
- Bandwidth

Each VM is billed per hour it is running. Cost depends on VM specifications. VMs that are running but not actively working on a dataset will still be charged via the load balancer. For each compute instance, one load balancer will be billed per day. Every 50 nodes of a compute cluster will have one standard load balancer billed. Each load balancer is billed around \$0.33/day. To avoid load balancer costs on stopped compute instances and compute clusters, delete the compute resource. One virtual network will be billed per subscription and per region. Virtual networks cannot span regions or subscriptions. Setting up private endpoints in vNet setups may also incur charges. Bandwidth is charged by usage; the more data transferred, the more you are charged.

Costs might accrue after resource deletion

After you delete an Azure Machine Learning workspace in the Azure portal or with Azure CLI, the following resources continue to exist. They continue to accrue costs until you delete them.

- Azure Container Registry
- Azure Block Blob Storage
- Key Vault
- Application Insights

To delete the workspace along with these dependent resources, use the SDK:

APPLIES TO:  Python SDK `azure-ai-ml v2 (preview)`

```
from azure.ai.ml.entities import Workspace
ml_client.workspaces.begin_delete(name=ws.name, delete_dependent_resources=True)
```

If you create Azure Kubernetes Service (AKS) in your workspace, or if you attach any compute resources to your workspace you must delete them separately in [Azure portal](#).

Using Azure Prepayment credit with Azure Machine Learning

You can pay for Azure Machine Learning charges with your Azure Prepayment credit. However, you can't use Azure Prepayment credit to pay for charges for third party products and services including those from the Azure Marketplace.

Review estimated costs in the Azure portal

As you create compute resources for Azure Machine Learning, you see estimated costs.

To create a *compute instance *and view the estimated price:

1. Sign into the [Azure Machine Learning studio](#)
2. On the left side, select **Compute**.
3. On the top toolbar, select **+ New**.
4. Review the estimated price shown in for each available virtual machine size.
5. Finish creating the resource.

Microsoft Azure Machine Learning

Create compute instance

Select virtual machine
Select the virtual machine size you would like to use for your compute instance. Please note that a compute instance can not be shared. It can only be used by a single assigned user. By default, it will be assigned to the creator and you can change this to a different user in the advanced settings section.

Location centralus

Virtual machine type CPU

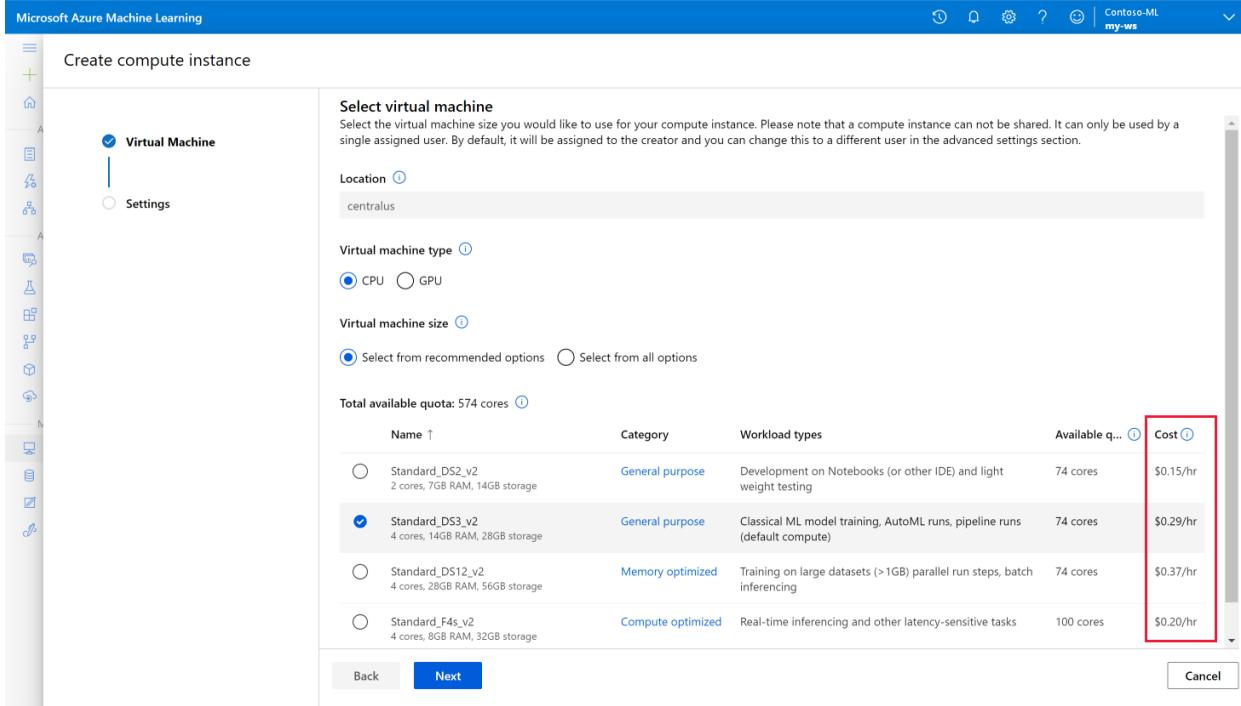
Virtual machine size

Select from recommended options Select from all options

Total available quota: 574 cores

Name ↑	Category	Workload types	Available q... <input type="radio"/>	Cost <input type="radio"/>
Standard_DS2_v2 2 cores, 7GB RAM, 14GB storage	General purpose	Development on Notebooks (or other IDE) and light weight testing	74 cores	\$0.15/hr
Standard_DS3_v2 4 cores, 14GB RAM, 28GB storage	General purpose	Classical ML model training, AutoML runs, pipeline runs (default compute)	74 cores	\$0.29/hr
Standard_DS12_v2 4 cores, 28GB RAM, 56GB storage	Memory optimized	Training on large datasets (>1GB) parallel run steps, batch training	74 cores	\$0.37/hr
Standard_F4s_v2 4 cores, 8GB RAM, 32GB storage	Compute optimized	Real-time inferencing and other latency-sensitive tasks	100 cores	\$0.20/hr

Back **Next** Cancel



If your Azure subscription has a spending limit, Azure prevents you from spending over your credit amount. As you create and use Azure resources, your credits are used. When you reach your credit limit, the resources that you deployed are disabled for the rest of that billing period. You can't change your credit limit, but you can remove it. For more information about spending limits, see [Azure spending limit](#).

Monitor costs

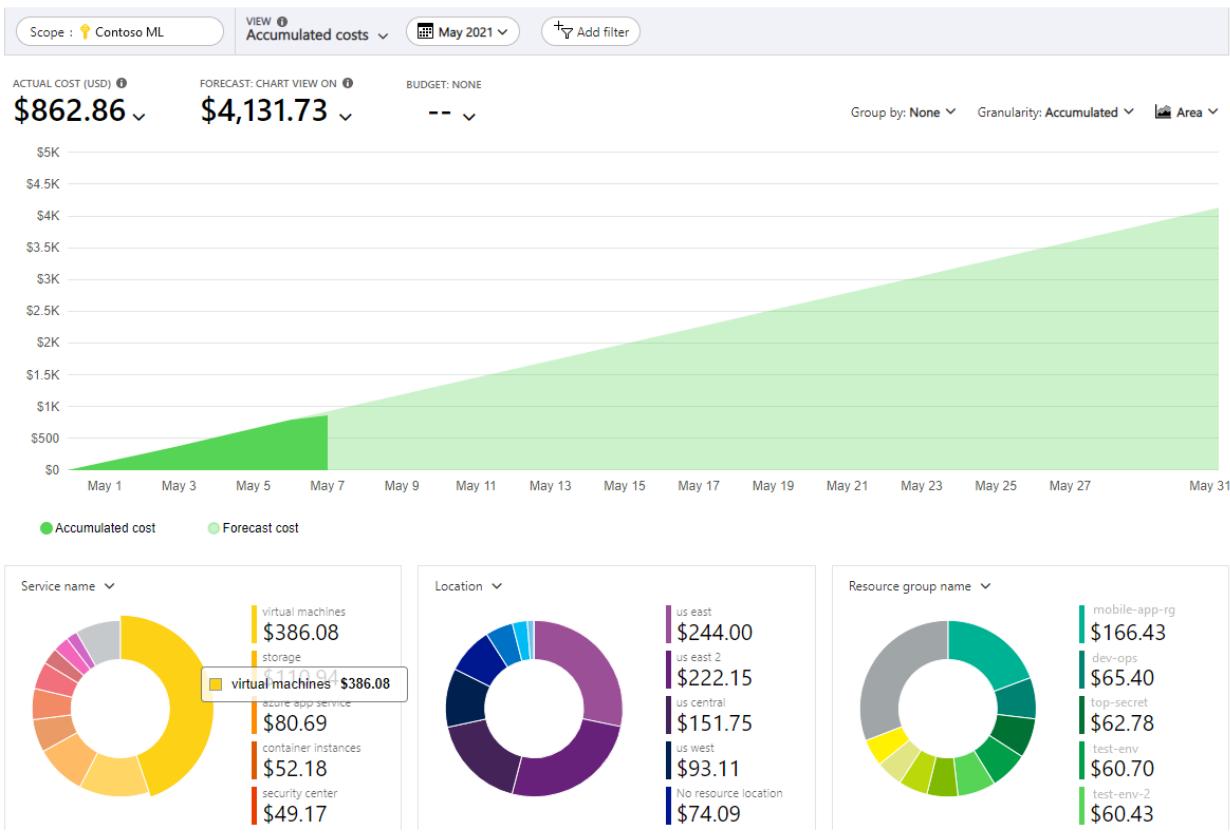
As you use Azure resources with Azure Machine Learning, you incur costs. Azure resource usage unit costs vary by time intervals (seconds, minutes, hours, and days) or by unit usage (bytes, megabytes, and so on.) As soon as Azure Machine Learning use starts, costs are incurred and you can see the costs in [cost analysis](#).

When you use cost analysis, you view Azure Machine Learning costs in graphs and tables for different time intervals. Some examples are by day, current and prior month, and year. You also view costs against budgets and forecasted costs. Switching to longer views over time can help you identify spending trends. And you see where overspending might have occurred. If you've created budgets, you can also easily see where they're exceeded.

To view Azure Machine Learning costs in cost analysis:

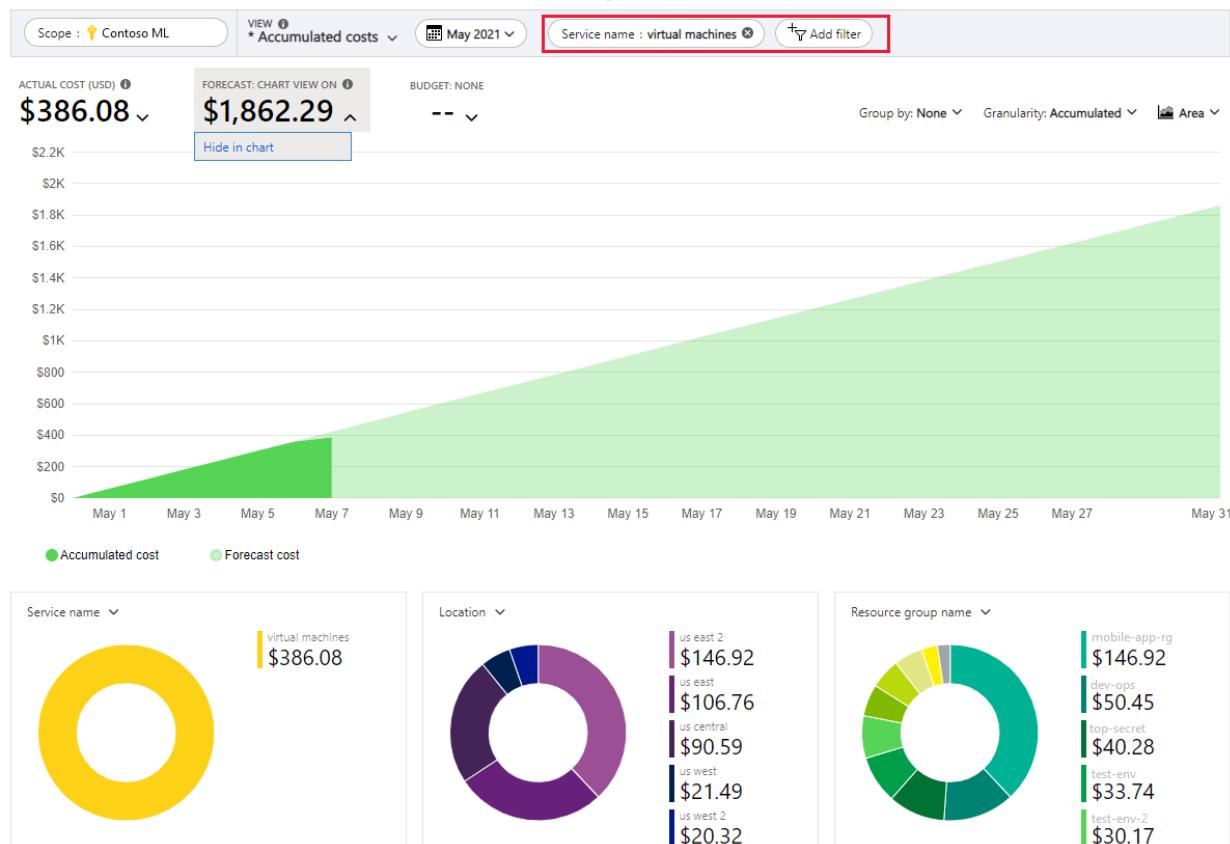
1. Sign in to the Azure portal.
2. Open the scope in the Azure portal and select **Cost analysis** in the menu. For example, go to **Subscriptions**, select a subscription from the list, and then select **Cost analysis** in the menu. Select **Scope** to switch to a different scope in cost analysis.
3. By default, cost for services are shown in the first donut chart. Select the area in the chart labeled Azure Machine Learning.

Actual monthly costs are shown when you initially open cost analysis. Here's an example showing all monthly usage costs.



To narrow costs for a single service, like Azure Machine Learning, select **Add filter** and then select **Service name**. Then, select **virtual machines**.

Here's an example showing costs for just Azure Machine Learning.



In the preceding example, you see the current cost for the service. Costs by Azure regions (locations) and Azure Machine Learning costs by resource group are also shown. From here, you can explore costs on your own.

Create budgets

You can create [budgets](#) to manage costs and create [alerts](#) that automatically notify stakeholders of spending anomalies and overspending risks. Alerts are based on spending compared to budget and cost thresholds. Budgets and alerts are created for Azure subscriptions and resource groups, so they're useful as part of an overall cost monitoring strategy.

Budgets can be created with filters for specific resources or services in Azure if you want more granularity present in your monitoring. Filters help ensure that you don't accidentally create new resources that cost you additional money. For more about the filter options when you create a budget, see [Group and filter options](#).

Export cost data

You can also [export your cost data](#) to a storage account. This is helpful when you need others to do additional data analysis for costs. For example, a finance teams can analyze the data using Excel or Power BI. You can export your costs on a daily, weekly, or monthly schedule and set a custom date range. Exporting cost data is the recommended way to retrieve cost datasets.

Other ways to manage and reduce costs for Azure Machine Learning

Use the following tips to help you manage and optimize your compute resource costs.

- Configure your training clusters for autoscaling
- Set quotas on your subscription and workspaces
- Set termination policies on your training job
- Use low-priority virtual machines (VM)
- Schedule compute instances to shut down and start up automatically
- Use an Azure Reserved VM Instance
- Train locally
- Parallelize training
- Set data retention and deletion policies
- Deploy resources to the same region
- Delete instances and clusters if you do not plan on using them in the near future.

For more information, see [manage and optimize costs in Azure Machine Learning](#).

Next steps

- [Manage and optimize costs in Azure Machine Learning](#).
- [Manage budgets, costs, and quota for Azure Machine Learning at organizational scale](#)
- Learn how to [optimize your cloud investment with Azure Cost Management](#).
- Learn more about managing costs with [cost analysis](#).
- Learn about how to [prevent unexpected costs](#).
- Take the [Cost Management](#) guided learning course.

Monitor Azure Machine Learning

9/22/2022 • 9 minutes to read • [Edit Online](#)

When you have critical applications and business processes relying on Azure resources, you want to monitor those resources for their availability, performance, and operation. This article describes the monitoring data generated by Azure Machine Learning and how to analyze and alert on this data with Azure Monitor.

TIP

The information in this document is primarily for **administrators**, as it describes monitoring for the Azure Machine Learning service and associated Azure services. If you are a **data scientist** or **developer**, and want to monitor information specific to your *model training runs*, see the following documents:

- [Start, monitor, and cancel training runs](#)
- [Log metrics for training runs](#)
- [Track experiments with MLflow](#)
- [Visualize runs with TensorBoard](#)

If you want to monitor information generated by models deployed to online endpoints, see [Monitor online endpoints](#).

What is Azure Monitor?

Azure Machine Learning creates monitoring data using [Azure Monitor](#), which is a full stack monitoring service in Azure. Azure Monitor provides a complete set of features to monitor your Azure resources. It can also monitor resources in other clouds and on-premises.

Start with the article [Monitoring Azure resources with Azure Monitor](#), which describes the following concepts:

- What is Azure Monitor?
- Costs associated with monitoring
- Monitoring data collected in Azure
- Configuring data collection
- Standard tools in Azure for analyzing and alerting on monitoring data

The following sections build on this article by describing the specific data gathered for Azure Machine Learning. These sections also provide examples for configuring data collection and analyzing this data with Azure tools.

TIP

To understand costs associated with Azure Monitor, see [Usage and estimated costs](#). To understand the time it takes for your data to appear in Azure Monitor, see [Log data ingestion time](#).

Monitoring data from Azure Machine Learning

Azure Machine Learning collects the same kinds of monitoring data as other Azure resources that are described in [Monitoring data from Azure resources](#).

See [Azure Machine Learning monitoring data reference](#) for a detailed reference of the logs and metrics created by Azure Machine Learning.

Collection and routing

Platform metrics and the Activity log are collected and stored automatically, but can be routed to other locations by using a diagnostic setting.

Resource Logs are not collected and stored until you create a diagnostic setting and route them to one or more locations. When you need to manage multiple Azure Machine Learning workspaces, you could route logs for all workspaces into the same logging destination and query all logs from a single place.

See [Create diagnostic setting to collect platform logs and metrics in Azure](#) for the detailed process for creating a diagnostic setting using the Azure portal, the Azure CLI, or PowerShell. When you create a diagnostic setting, you specify which categories of logs to collect. The categories for Azure Machine Learning are listed in [Azure Machine Learning monitoring data reference](#).

IMPORTANT

Enabling these settings requires additional Azure services (storage account, event hub, or Log Analytics), which may increase your cost. To calculate an estimated cost, visit the [Azure pricing calculator](#).

You can configure the following logs for Azure Machine Learning:

CATEGORY	DESCRIPTION
AmlComputeClusterEvent	Events from Azure Machine Learning compute clusters.
AmlComputeClusterNodeEvent (deprecated)	Events from nodes within an Azure Machine Learning compute cluster.
AmlComputeJobEvent	Events from jobs running on Azure Machine Learning compute.
AmlComputeCpuGpuUtilization	ML services compute CPU and GPU utilization logs.
AmlRunStatusChangedEvent	ML run status changes.
ModelsChangeEvent	Events when ML model is accessed created or deleted.
ModelsReadEvent	Events when ML model is read.
ModelsActionEvent	Events when ML model is accessed.
DeploymentReadEvent	Events when a model deployment is read.
DeploymentEventACI	Events when a model deployment happens on ACI (very chatty).
DeploymentEventAKS	Events when a model deployment happens on AKS (very chatty).
InferencingOperationAKS	Events for inference or related operation on AKS compute type.
InferencingOperationACI	Events for inference or related operation on ACI compute type.

CATEGORY	DESCRIPTION
EnvironmentChangeEvent	Events when ML environment configurations are created or deleted.
EnvironmentReadEvent	Events when ML environment configurations are read (very chatty).
DataLabelChangeEvent	Events when data label(s) or its projects is created or deleted.
DataLabelReadEvent	Events when data label(s) or its projects is read.
ComputeInstanceEvent	Events when ML Compute Instance is accessed (very chatty).
DataStoreChangeEvent	Events when ML datastore is created or deleted.
DataStoreReadEvent	Events when ML datastore is read.
DataSetChangeEvent	Events when ML datastore is created or deleted.
DataSetReadEvent	Events when ML datastore is read.
PipelineChangeEvent	Events when ML pipeline draft or endpoint or module are created or deleted.
PipelineReadEvent	Events when ML pipeline draft or endpoint or module are read.
RunEvent	Events when ML experiments are created or deleted.
RunReadEvent	Events when ML experiments are read.

NOTE

Effective February 2022, the `AmlComputeClusterNodeEvent` category will be deprecated. We recommend that you instead use the `AmlComputeClusterEvent` category.

NOTE

When you enable metrics in a diagnostic setting, dimension information is not currently included as part of the information sent to a storage account, event hub, or log analytics.

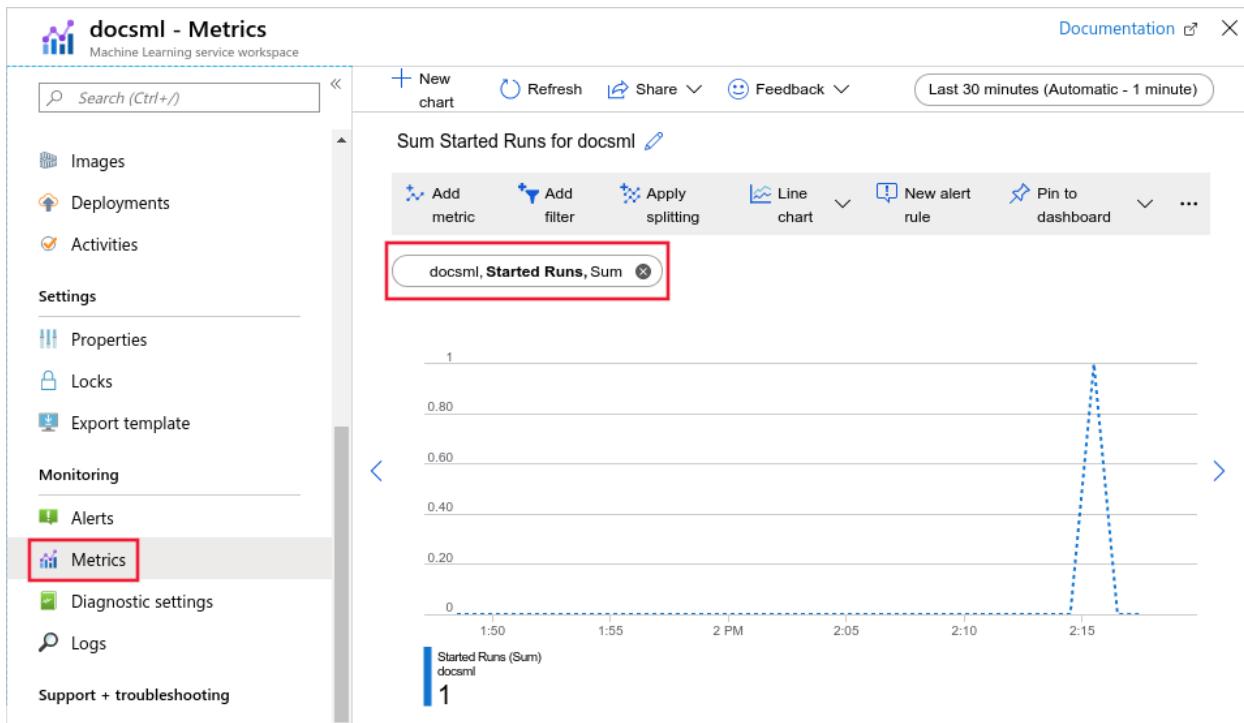
The metrics and logs you can collect are discussed in the following sections.

Analyzing metrics

You can analyze metrics for Azure Machine Learning, along with metrics from other Azure services, by opening **Metrics** from the **Azure Monitor** menu. See [Getting started with Azure Metrics Explorer](#) for details on using this tool.

For a list of the platform metrics collected, see [Monitoring Azure Machine Learning data reference metrics](#).

All metrics for Azure Machine Learning are in the namespace **Machine Learning Service Workspace**.



For reference, you can see a list of [all resource metrics supported in Azure Monitor](#).

TIP

Azure Monitor metrics data is available for 90 days. However, when creating charts only 30 days can be visualized. For example, if you want to visualize a 90 day period, you must break it into three charts of 30 days within the 90 day period.

Filtering and splitting

For metrics that support dimensions, you can apply filters using a dimension value. For example, filtering **Active Cores** for a **Cluster Name** of `cpu-cluster`.

You can also split a metric by dimension to visualize how different segments of the metric compare with each other. For example, splitting out the **Pipeline Step Type** to see a count of the types of steps used in the pipeline.

For more information of filtering and splitting, see [Advanced features of Azure Monitor](#).

Analyzing logs

Using Azure Monitor Log Analytics requires you to create a diagnostic configuration and enable [Send information to Log Analytics](#). For more information, see the [Collection and routing](#) section.

Data in Azure Monitor Logs is stored in tables, with each table having its own set of unique properties. Azure Machine Learning stores data in the following tables:

TABLE	DESCRIPTION
AmlComputeClusterEvent	Events from Azure Machine Learning compute clusters.
AmlComputeClusterNodeEvent (deprecated)	Events from nodes within an Azure Machine Learning compute cluster.
AmlComputeJobEvent	Events from jobs running on Azure Machine Learning compute.

TABLE	DESCRIPTION
AmlComputeInstanceEvent	Events when ML Compute Instance is accessed (read/write). Category includes:ComputeInstanceEvent (very chatty).
AmlDataLabelEvent	Events when data label(s) or its projects is accessed (read, created, or deleted). Category includes:DataLabelReadEvent,DataLabelChangeEvent.
AmlDataSetEvent	Events when a registered or unregistered ML dataset is accessed (read, created, or deleted). Category includes:DataSetReadEvent,DataSetChangeEvent.
AmlDataStoreEvent	Events when ML datastore is accessed (read, created, or deleted). Category includes:DataStoreReadEvent,DataStoreChangeEvent.
AmlDeploymentEvent	Events when a model deployment happens on ACI or AKS. Category includes:DeploymentReadEvent,DeploymentEventACI,DeploymentEventAKS.
AmlInferencingEvent	Events for inference or related operation on AKS or ACI compute type. Category includes:InferencingOperationACI (very chatty),InferencingOperationAKS (very chatty).
AmlModelsEvent	Events when ML model is accessed (read, created, or deleted). Includes events when packaging of models and assets happen into ready-to-build packages. Category includes:ModelsReadEvent,ModelsActionEvent .
AmlPipelineEvent	Events when ML pipeline draft or endpoint or module are accessed (read, created, or deleted).Category includes:PipelineReadEvent,PipelineChangeEvent.
AmlRunEvent	Events when ML experiments are accessed (read, created, or deleted). Category includes:RunReadEvent,RunEvent.
AmlEnvironmentEvent	Events when ML environment configurations (read, created, or deleted). Category includes:EnvironmentReadEvent (very chatty),EnvironmentChangeEvent.

NOTE

Effective February 2022, the AmlComputeClusterNodeEvent table will be deprecated. We recommend that you instead use the AmlComputeClusterEvent table.

IMPORTANT

When you select **Logs** from the Azure Machine Learning menu, Log Analytics is opened with the query scope set to the current workspace. This means that log queries will only include data from that resource. If you want to run a query that includes data from other databases or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

For a detailed reference of the logs and metrics, see [Azure Machine Learning monitoring data reference](#).

Sample Kusto queries

IMPORTANT

When you select **Logs** from the [service-name] menu, Log Analytics is opened with the query scope set to the current Azure Machine Learning workspace. This means that log queries will only include data from that resource. If you want to run a query that includes data from other workspaces or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

Following are queries that you can use to help you monitor your Azure Machine Learning resources:

- Get failed jobs in the last five days:

```
AmlComputeJobEvent  
| where TimeGenerated > ago(5d) and EventType == "JobFailed"  
| project TimeGenerated , ClusterId , EventType , ExecutionState , ToolType
```

- Get records for a specific job name:

```
AmlComputeJobEvent  
| where JobName == "automl_a9940991-dedb-4262-9763-2fd08b79d8fb_setup"  
| project TimeGenerated , ClusterId , EventType , ExecutionState , ToolType
```

- Get cluster events in the last five days for clusters where the VM size is Standard_D1_V2:

```
AmlComputeClusterEvent  
| where TimeGenerated > ago(4d) and VmSize == "STANDARD_D1_V2"  
| project ClusterName , InitialNodeCount , MaximumNodeCount , QuotaAllocated , QuotaUtilized
```

- Get the cluster node allocations in the last eight days::

```
AmlComputeClusterEvent  
| where TimeGenerated > ago(8d) and TargetNodeCount > CurrentNodeCount  
| project TimeGenerated, ClusterName, CurrentNodeCount, TargetNodeCount
```

When you connect multiple Azure Machine Learning workspaces to the same Log Analytics workspace, you can query across all resources.

- Get number of running nodes across workspaces and clusters in the last day:

```
AmlComputeClusterEvent  
| where TimeGenerated > ago(1d)  
| summarize avgRunningNodes=avg(TargetNodeCount), maxRunningNodes=max(TargetNodeCount)  
    by Workspace=tostring(split(_ResourceId, "/")[8]), ClusterName, ClusterType, VmSize,  
VmPriority
```

Create a workspace monitoring dashboard by using a template

A dashboard is a focused and organized view of your cloud resources in the Azure portal. For more information about creating dashboards, see [Create, view, and manage metric alerts using Azure Monitor](#).

To deploy a sample dashboard, you can use a publicly available [template](#). The sample dashboard is based on [Kusto queries](#), so you must enable [Log Analytics data collection](#) for your Azure Machine Learning workspace before you deploy the dashboard.

Alerts

You can access alerts for Azure Machine Learning by opening **Alerts** from the **Azure Monitor** menu. See [Create, view, and manage metric alerts using Azure Monitor](#) for details on creating alerts.

The following table lists common and recommended metric alert rules for Azure Machine Learning:

ALERT TYPE	CONDITION	DESCRIPTION
Model Deploy Failed	Aggregation type: Total, Operator: Greater than, Threshold value: 0	When one or more model deployments have failed
Quota Utilization Percentage	Aggregation type: Average, Operator: Greater than, Threshold value: 90	When the quota utilization percentage is greater than 90%
Unusable Nodes	Aggregation type: Total, Operator: Greater than, Threshold value: 0	When there are one or more unusable nodes

Next steps

- For a reference of the logs and metrics, see [Monitoring Azure Machine Learning data reference](#).
- For information on working with quotas related to Azure Machine Learning, see [Manage and request quotas for Azure resources](#).
- For details on monitoring Azure resources, see [Monitoring Azure resources with Azure Monitor](#).

Secure code best practices with Azure Machine Learning

9/22/2022 • 2 minutes to read • [Edit Online](#)

In Azure Machine Learning, you can upload files and content from any source into Azure. Content within Jupyter notebooks or scripts that you load can potentially read data from your sessions, access data within your organization in Azure, or run malicious processes on your behalf.

IMPORTANT

Only run notebooks or scripts from trusted sources. For example, where you or your security team have reviewed the notebook or script.

Potential threats

Development with Azure Machine Learning often involves web-based development environments (Notebooks & Azure ML studio). When using web-based development environments, the potential threats are:

- [Cross site scripting \(XSS\)](#)
 - **DOM injection:** This type of attack can modify the UI displayed in the browser. For example, by changing how the run button behaves in a Jupyter Notebook.
 - **Access token/cookies:** XSS attacks can also access local storage and browser cookies. Your Azure Active Directory (AAD) authentication token is stored in local storage. An XSS attack could use this token to make API calls on your behalf, and then send the data to an external system or API.
- [Cross site request forgery \(CSRF\)](#): This attack may replace the URL of an image or link with the URL of a malicious script or API. When the image is loaded, or link clicked, a call is made to the URL.

Azure ML studio notebooks

Azure Machine Learning studio provides a hosted notebook experience in your browser. Cells in a notebook can output HTML documents or fragments that contain malicious code. When the output is rendered, the code can be executed.

Possible threats:

- Cross site scripting (XSS)
- Cross site request forgery (CSRF)

Mitigations provided by Azure Machine Learning:

- **Code cell output** is sandboxed in an iframe. The iframe prevents the script from accessing the parent DOM, cookies, or session storage.
- **Markdown cell** contents are cleaned using the dompurify library. This blocks malicious scripts from executing with markdown cells are rendered.
- **Image URL** and **Markdown links** are sent to a Microsoft owned endpoint, which checks for malicious values. If a malicious value is detected, the endpoint rejects the request.

Recommended actions:

- Verify that you trust the contents of files before uploading to studio. When uploading, you must acknowledge that you're uploading trusted files.
- When selecting a link to open an external application, you'll be prompted to trust the application.

Azure ML compute instance

Azure Machine Learning compute instance hosts **Jupyter** and **Jupyter Lab**. When using either, cells in a notebook or code in can output HTML documents or fragments that contain malicious code. When the output is rendered, the code can be executed. The same threats also apply when using **RStudio** hosted on a compute instance.

Possible threats:

- Cross site scripting (XSS)
- Cross site request forgery (CSRF)

Mitigations provided by Azure Machine Learning:

- None. Jupyter and Jupyter Lab are open-source applications hosted on the Azure Machine Learning compute instance.

Recommended actions:

- Verify that you trust the contents of files before uploading to studio. When uploading, you must acknowledge that you're uploading trusted files.

Report security issues or concerns

Azure Machine Learning is eligible under the Microsoft Azure Bounty Program. For more information, visit<https://www.microsoft.com/msrc/bounty-microsoft-azure>.

Next steps

- [Enterprise security for Azure Machine Learning](#)

Audit and manage Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

When teams collaborate on Azure Machine Learning, they may face varying requirements to the configuration and organization of resources. Machine learning teams may look for flexibility in how to organize workspaces for collaboration, or size compute clusters to the requirements of their use cases. In these scenarios, it may lead to most productivity if the application team can manage their own infrastructure.

As a platform administrator, you can use policies to lay out guardrails for teams to manage their own resources. [Azure Policy](#) helps audit and govern resource state. In this article, you learn about available auditing controls and governance practices for Azure Machine Learning.

Policies for Azure Machine Learning

[Azure Policy](#) is a governance tool that allows you to ensure that Azure resources are compliant with your policies.

Azure Machine Learning provides a set of policies that you can use for common scenarios with Azure Machine Learning. You can assign these policy definitions to your existing subscription or use them as the basis to create your own custom definitions.

The table below includes a selection of policies you can assign with Azure Machine Learning. For a complete list of the built-in policies for Azure Machine Learning, see [Built-in policies for Azure Machine Learning](#).

POLICY	DESCRIPTION
Customer-managed key	Audit or enforce whether workspaces must use a customer-managed key.
Private link	Audit or enforce whether workspaces use a private endpoint to communicate with a virtual network.
Private endpoint	Configure the Azure Virtual Network subnet where the private endpoint should be created.
Private DNS zone	Configure the private DNS zone to use for the private link.
User-assigned managed identity	Audit or enforce whether workspaces use a user-assigned managed identity.
Disable public network access	Audit or enforce whether workspaces disable access from the public internet.
Disable local authentication	Audit or enforce whether Azure Machine Learning compute resources should have local authentication methods disabled.
Modify/disable local authentication	Configure compute resources to disable local authentication methods.
Compute cluster and instance is behind virtual network	Audit whether compute resources are behind a virtual network.

Policies can be set at different scopes, such as at the subscription or resource group level. For more information, see the [Azure Policy documentation](#).

Assigning built-in policies

To view the built-in policy definitions related to Azure Machine Learning, use the following steps:

1. Go to **Azure Policy** in the [Azure portal](#).
2. Select **Definitions**.
3. For **Type**, select *Built-in*, and for **Category**, select **Machine Learning**.

From here, you can select policy definitions to view them. While viewing a definition, you can use the **Assign** link to assign the policy to a specific scope, and configure the parameters for the policy. For more information, see [Assign a policy - portal](#).

You can also assign policies by using [Azure PowerShell](#), [Azure CLI](#), and [templates](#).

Conditional access policies

To control who can access your Azure Machine Learning workspace, use Azure Active Directory [Conditional Access](#).

Enable self-service using landing zones

Landing zones are an architectural pattern to set up Azure environments that accounts for scale, governance, security, and productivity. A data landing zone is an administrator-configured environment that an application team uses to host a data and analytics workload.

The purpose of the landing zone is to ensure when a team starts in the Azure environment, all infrastructure configuration work is done. For instance, security controls are set up in compliance with organizational standards and network connectivity is set up.

Using the landing zones pattern, machine learning teams can be enabled to self-service deploy and manage their own resources. By use of Azure policy, as an administrator you can audit and manage Azure resources for compliance and make sure workspaces are compliant to meet your requirements.

Azure Machine Learning integrates with [data landing zones](#) in the [Cloud Adoption Framework data management and analytics scenario](#). This reference implementation provides an optimized environment to migrate machine learning workloads onto and includes policies for Azure Machine Learning preconfigured.

Configure built-in policies

Workspace encryption with customer-managed key

Controls whether a workspace should be encrypted with a customer-managed key, or using a Microsoft-managed key to encrypt metrics and metadata. For more information on using customer-managed key, see the [Azure Cosmos DB](#) section of the data encryption article.

To configure this policy, set the effect parameter to **audit** or **deny**. If set to **audit**, you can create a workspace without a customer-managed key and a warning event is created in the activity log.

If the policy is set to **deny**, then you cannot create a workspace unless it specifies a customer-managed key.

Attempting to create a workspace without a customer-managed key results in an error similar to

`Resource 'clustername' was disallowed by policy` and creates an error in the activity log. The policy identifier is also returned as part of this error.

Workspace should use private link

Controls whether a workspace should use Azure Private Link to communicate with Azure Virtual Network. For more information on using private link, see [Configure private link for a workspace](#).

To configure this policy, set the effect parameter to **audit** or **deny**. If set to **audit**, you can create a workspace without using private link and a warning event is created in the activity log.

If the policy is set to **deny**, then you cannot create a workspace unless it uses a private link. Attempting to create a workspace without a private link results in an error. The error is also logged in the activity log. The policy identifier is returned as part of this error.

Workspace should use private endpoint

Configures a workspace to create a private endpoint within the specified subnet of an Azure Virtual Network.

To configure this policy, set the effect parameter to **DeployIfNotExists**. Set the **privateEndpointSubnetID** to the Azure Resource Manager ID of the subnet.

Workspace should use private DNS zones

Configures a workspace to use a private DNS zone, overriding the default DNS resolution for a private endpoint.

To configure this policy, set the effect parameter to **DeployIfNotExists**. Set the **privateDnsZoneId** to the Azure Resource Manager ID of the private DNS zone to use.

Workspace should use user-assigned managed identity

Controls whether a workspace is created using a system-assigned managed identity (default) or a user-assigned managed identity. The managed identity for the workspace is used to access associated resources such as Azure Storage, Azure Container Registry, Azure Key Vault, and Azure Application Insights. For more information, see [Use managed identities with Azure Machine Learning](#).

To configure this policy, set the effect parameter to **audit**, **deny**, or **disabled**. If set to **audit**, you can create a workspace without specifying a user-assigned managed identity. A system-assigned identity is used and a warning event is created in the activity log.

If the policy is set to **deny**, then you cannot create a workspace unless you provide a user-assigned identity during the creation process. Attempting to create a workspace without providing a user-assigned identity results in an error. The error is also logged to the activity log. The policy identifier is returned as part of this error.

Workspace should disable public network access

Controls whether a workspace should disable network access from the public internet.

To configure this policy, set the effect parameter to **audit**, **deny**, or **disabled**. If set to **audit**, you can create a workspace with public access and a warning event is created in the activity log.

If the policy is set to **deny**, then you cannot create a workspace that allows network access from the public internet.

Disable local authentication

Controls whether an Azure Machine Learning compute cluster or instance should disable local authentication (SSH).

To configure this policy, set the effect parameter to **audit**, **deny**, or **disabled**. If set to **audit**, you can create a compute with SSH enabled and a warning event is created in the activity log.

If the policy is set to **deny**, then you cannot create a compute unless SSH is disabled. Attempting to create a compute with SSH enabled results in an error. The error is also logged in the activity log. The policy identifier is returned as part of this error.

Modify/disable local authentication

Modifies any Azure Machine Learning compute cluster or instance creation request to disable local

authentication (SSH).

To configure this policy, set the effect parameter to **Modify** or **Disabled**. If set **Modify**, any creation of a compute cluster or instance within the scope where the policy applies will automatically have local authentication disabled.

Compute cluster and instance is behind virtual network

Controls auditing of compute cluster and instance resources behind a virtual network.

To configure this policy, set the effect parameter to **audit** or **disabled**. If set to **audit**, you can create a compute that is not configured behind a virtual network and a warning event is created in the activity log.

Next steps

- [Azure Policy documentation](#)
- [Built-in policies for Azure Machine Learning](#)
- [Working with security policies with Microsoft Defender for Cloud](#)
- The [Cloud Adoption Framework scenario for data management and analytics](#) outlines considerations in running data and analytics workloads in the cloud.
- [Cloud Adoption Framework data landing zones](#) provide a reference implementation for managing data and analytics workloads in Azure.
- [Learn how to use policy to integrate Azure Private Link with Azure Private DNS zones](#), to manage private link configuration for the workspace and dependent resources.

Search for Azure Machine Learning assets (preview)

9/22/2022 • 2 minutes to read • [Edit Online](#)

Use the search bar to find machine learning assets across all workspaces, resource groups, and subscriptions in your organization. Your search text will be used to find assets such as:

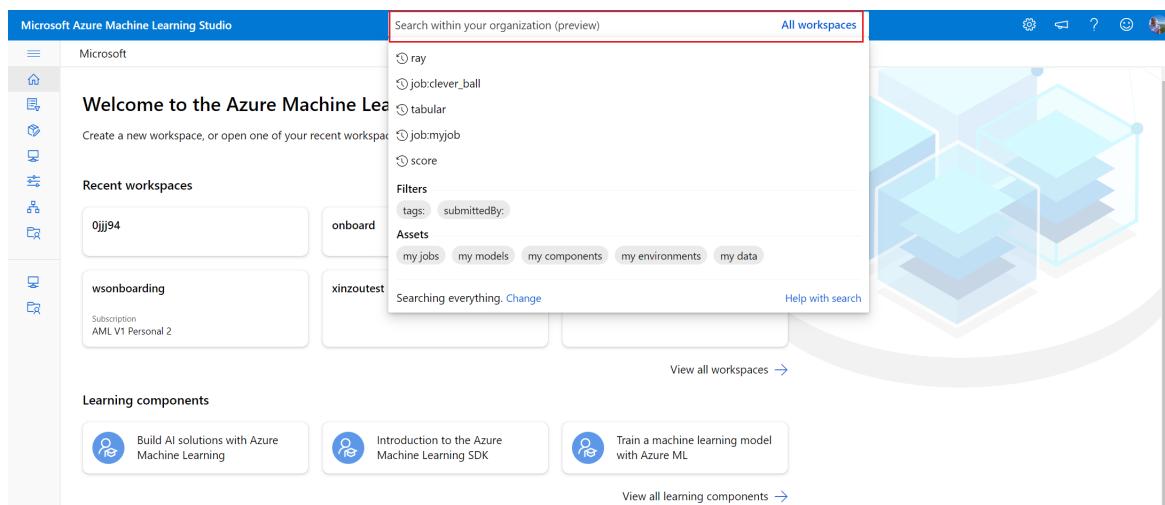
- Jobs
- Models
- Components
- Environments
- Data

IMPORTANT

The search functionality is currently in public preview. The preview version is provided without a service level agreement. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Free text search

1. Sign in to [Azure Machine Learning studio](#).
2. In the top studio titlebar, if a workspace is open, select **This workspace** or **All workspaces** to set the search context.



3. Type your text and hit enter to trigger a 'contains' search. A contains search scans across all metadata fields for the given asset and sorts results by relevancy score which is determined by weightings for different column properties.

Structured search

1. Sign in to [Azure Machine Learning studio](#).
2. In the top studio titlebar, select **All workspaces**.
3. Click inside the search field to display filters to create more specific search queries.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, a sidebar lists various workspace categories: Microsoft, New, Home (which is selected and highlighted in blue), Author, Notebooks, Automated ML, Designer, Assets, Data, Jobs, Components, Pipelines, Environments, Models, Endpoints, Manage, Compute, Datastores, Linked Services, and Data Labeling. The main content area is titled "Welcome to the Azure Machine Learning Studio". It features four cards: "Create new" (with a plus icon), "Notebooks" (with a calendar icon), "Automated ML" (with a gear icon), and "Designer" (with a cube icon). Below these are sections for "Recent resources" and a table of recent assets. The "Jobs" tab is selected in the "Recent resources" header. The table has columns: Display name, Experiment, Status, Logs, Submitted time, Submitted by, and Job type. The data includes several entries like "quirky_pizza_2z00lw4q", "diabetes model", and "Regression - Automob...".

The following filters are supported:

- Job
- Model
- Component
- Tags
- SubmittedBy
- Environment
- Data

If an asset filter (job, model, component, environment, data) is present, results are scoped to those tabs. Other filters apply to all assets unless an asset filter is also present in the query. Similarly, free text search can be provided alongside filters, but are scoped to the tabs chosen by asset filters, if present.

TIP

- Filters search for exact matches of text. Use free text queries for a contains search.
- Quotations are required around values that include spaces or other special characters.
- If duplicate filters are provided, only the first will be recognized in search results.
- Input text of any language is supported but filter strings must match the provided options (ex. submittedBy:).
- The tags filter can accept multiple key:value pairs separated by a comma (ex. tags:"key1:value1, key2:value2").

View search results

You can view your search results in the individual **Jobs**, **Models**, **Components**, **Environments**, and **Data** tabs. Select an asset to open its **Details** page in the context of the relevant workspace. Results from workspaces you don't have permissions to view aren't displayed.

Microsoft Azure Machine Learning Studio All workspaces Search within your tenant (preview) Vienna PM 2 UTVNext-WestUs2

Microsoft > UTVNext-WestUs2 > Search results

Search results

Showing search results for "demo" **across your organization.** [Change](#)

Jobs (1000+) Models (30) Components (700) Environments (9) Data assets (81)

Refresh Current view: Local Edit view Show only my jobs

Status Created by Created on All filters Clear all

Showing 1-25 of 1,094 jobs

Page size: 25

Display name	Experiment	Status	Created on
demo	demo	✖ Failed	Jul 15, 2021 1:28 PM
7e3df25c-48dc-46b5-8c44-18b33e80de...	demo	✓ Completed	Apr 15, 2020 10:54 PM
demo	demo	✖ Failed	Jul 19, 2021 12:09 PM
demo	demo	✖ Failed	Jul 19, 2021 1:43 PM
demo	FHL-Demo	✖ Failed	Jun 10, 2021 2:45 PM
790237a9-d8ab-471f-9748-4491a421b8...	demo	✓ Completed	Apr 15, 2020 10:42 PM
demo	Default	✖ Canceled	Mar 17, 2022 12:57 AM
demo	Default	✖ Failed	Mar 23, 2022 3:04 PM

< Prev Next >

If you've used this feature in a previous update, a search result error may occur. Reselect your preferred workspaces in the Directory + Subscription + Workspace tab.

IMPORTANT

Search results may be unexpected for multiword terms in other languages (ex. Chinese characters).

Customize search results

You can create, save and share different views for your search results.

1. On the search results page, select **Edit view**.

Search results

Showing search results for "demo" **across your organization.** [Change](#)

Jobs (1000+) Models (29) Components (698) Environments (9)

Refresh Current view: Local Edit view Show

- [Edit columns](#)
- [Reset](#)
- [Share](#)
- [New...](#)
- [Clone...](#)

Showing 1-25 of 1,009 jobs

Display name

demo

Use the menu to customize and create new views:

ITEM	DESCRIPTION
Edit columns	Add, delete, and re-order columns in the current view's search results table
Reset	Add all hidden columns back into the view
Share	Displays a URL you can copy to share this view
New...	Create a new view
Clone	Clone the current view as a new view

Since each tab displays different columns, you customize views separately for each tab.

Next steps

- [What is an Azure Machine Learning workspace?](#)
- [Data in Azure Machine Learning](#)

Set up a Python development environment for Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

Learn how to configure a Python development environment for Azure Machine Learning.

The following table shows each development environment covered in this article, along with pros and cons.

ENVIRONMENT	PROS	CONS
Local environment	Full control of your development environment and dependencies. Run with any build tool, environment, or IDE of your choice.	Takes longer to get started. Necessary SDK packages must be installed, and an environment must also be installed if you don't already have one.
The Data Science Virtual Machine (DSVM)	Similar to the cloud-based compute instance (Python and the SDK are pre-installed), but with additional popular data science and machine learning tools pre-installed. Easy to scale and combine with other custom tools and workflows.	A slower getting started experience compared to the cloud-based compute instance.
Azure Machine Learning compute instance	Easiest way to get started. The entire SDK is already installed in your workspace VM, and notebook tutorials are pre-cloned and ready to run.	Lack of control over your development environment and dependencies. Additional cost incurred for Linux VM (VM can be stopped when not in use to avoid charges). See pricing details .
Azure Databricks	Ideal for running large-scale intensive machine learning workflows on the scalable Apache Spark platform.	Overkill for experimental machine learning, or smaller-scale experiments and workflows. Additional cost incurred for Azure Databricks. See pricing details .

This article also provides additional usage tips for the following tools:

- Jupyter Notebooks: If you're already using Jupyter Notebooks, the SDK has some extras that you should install.
- Visual Studio Code: If you use Visual Studio Code, the [Azure Machine Learning extension](#) includes extensive language support for Python as well as features to make working with the Azure Machine Learning much more convenient and productive.

Prerequisites

- Azure Machine Learning workspace. If you don't have one, you can create an Azure Machine Learning workspace through the [Azure portal](#), [Azure CLI](#), and [Azure Resource Manager templates](#).

Local and DSVM only: Create a workspace configuration file

The workspace configuration file is a JSON file that tells the SDK how to communicate with your Azure Machine Learning workspace. The file is named *config.json*, and it has the following format:

```
{
    "subscription_id": "<subscription-id>",
    "resource_group": "<resource-group>",
    "workspace_name": "<workspace-name>"
}
```

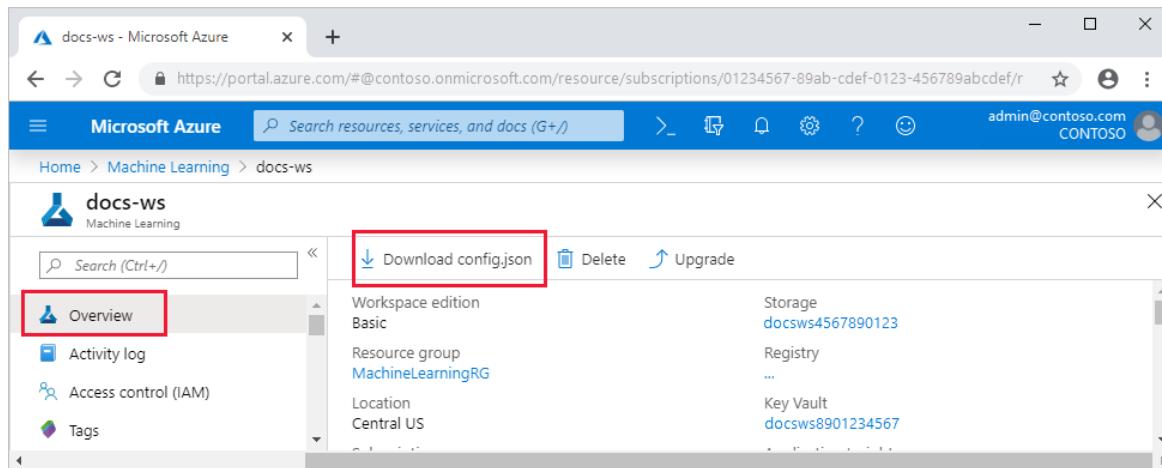
This JSON file must be in the directory structure that contains your Python scripts or Jupyter Notebooks. It can be in the same directory, a subdirectory named `.azureml/`, or in a parent directory.

To use this file from your code, use the `Workspace.from_config` method. This code loads the information from the file and connects to your workspace.

Create a workspace configuration file in one of the following methods:

- Azure portal

Download the file: In the [Azure portal](#), select **Download config.json** from the **Overview** section of your workspace.



- Azure Machine Learning Python SDK

Create a script to connect to your Azure Machine Learning workspace and use the `write_config` method to generate your file and save it as `.azureml/config.json`. Make sure to replace `subscription_id`, `resource_group`, and `workspace_name` with your own.

APPLIES TO: [Python SDK azureml v1](#)

```
from azureml.core import Workspace

subscription_id = '<subscription-id>'
resource_group = '<resource-group>'
workspace_name = '<workspace-name>'

try:
    ws = Workspace(subscription_id = subscription_id, resource_group = resource_group, workspace_name = workspace_name)
    ws.write_config()
    print('Library configuration succeeded')
except:
    print('Workspace not found')
```

Local computer or remote VM environment

You can set up an environment on a local computer or remote virtual machine, such as an Azure Machine Learning compute instance or Data Science VM.

To configure a local development environment or remote VM:

1. Create a Python virtual environment (virtualenv, conda).

NOTE

Although not required, it's recommended you use [Anaconda](#) or [Miniconda](#) to manage Python virtual environments and install packages.

IMPORTANT

If you're on Linux or macOS and use a shell other than bash (for example, zsh) you might receive errors when you run some commands. To work around this problem, use the `bash` command to start a new bash shell and run the commands there.

2. Activate your newly created Python virtual environment.
3. Install the [Azure Machine Learning Python SDK](#).
4. To configure your local environment to use your Azure Machine Learning workspace, [create a workspace configuration file](#) or use an existing one.

Now that you have your local environment set up, you're ready to start working with Azure Machine Learning. See the [Azure Machine Learning Python getting started guide](#) to get started.

Jupyter Notebooks

When running a local Jupyter Notebook server, it's recommended that you create an IPython kernel for your Python virtual environment. This helps ensure the expected kernel and package import behavior.

1. Enable environment-specific IPython kernels

```
conda install notebook ipykernel
```

2. Create a kernel for your Python virtual environment. Make sure to replace `<myenv>` with the name of your Python virtual environment.

```
ipython kernel install --user --name <myenv> --display-name "Python (<myenv>)"
```

3. Launch the Jupyter Notebook server

See the [Azure Machine Learning notebooks repository](#) to get started with Azure Machine Learning and Jupyter Notebooks. Also see the community-driven repository, [AzureML-Examples](#).

Visual Studio Code

To use Visual Studio Code for development:

1. Install [Visual Studio Code](#).
2. Install the [Azure Machine Learning Visual Studio Code extension \(preview\)](#).

Once you have the Visual Studio Code extension installed, use it to:

- [Manage your Azure Machine Learning resources](#)
- [Connect to an Azure Machine Learning compute instance](#)
- [Run and debug experiments](#)

- Deploy trained models.

Azure Machine Learning compute instance

The Azure Machine Learning [compute instance](#) is a secure, cloud-based Azure workstation that provides data scientists with a Jupyter Notebook server, JupyterLab, and a fully managed machine learning environment.

There is nothing to install or configure for a compute instance.

Create one anytime from within your Azure Machine Learning workspace. Provide just a name and specify an Azure VM type. Try it now with this [Tutorial: Setup environment and workspace](#).

To learn more about compute instances, including how to install packages, see [Create and manage an Azure Machine Learning compute instance](#).

TIP

To prevent incurring charges for an unused compute instance, [stop the compute instance](#).

In addition to a Jupyter Notebook server and JupyterLab, you can use compute instances in the [integrated notebook feature inside of Azure Machine Learning studio](#).

You can also use the Azure Machine Learning Visual Studio Code extension to [connect to a remote compute instance using VS Code](#).

Data Science Virtual Machine

The Data Science VM is a customized virtual machine (VM) image you can use as a development environment. It's designed for data science work that's pre-configured tools and software like:

- Packages such as TensorFlow, PyTorch, Scikit-learn, XGBoost, and the Azure Machine Learning SDK
- Popular data science tools such as Spark Standalone and Drill
- Azure tools such as the Azure CLI, AzCopy, and Storage Explorer
- Integrated development environments (IDEs) such as Visual Studio Code and PyCharm
- Jupyter Notebook Server

For a more comprehensive list of the tools, see the [Data Science VM tools guide](#).

IMPORTANT

If you plan to use the Data Science VM as a [compute target](#) for your training or inferencing jobs, only Ubuntu is supported.

To use the Data Science VM as a development environment:

1. Create a Data Science VM using one of the following methods:
 - Use the Azure portal to create an [Ubuntu](#) or [Windows](#) DSVM.
 - [Create a Data Science VM using ARM templates](#).
 - Use the Azure CLI

To create an Ubuntu Data Science VM, use the following command:

```
# create a Ubuntu Data Science VM in your resource group
# note you need to be at least a contributor to the resource group in order to execute this
# command successfully
# If you need to create a new resource group use: "az group create --name YOUR-RESOURCE-GROUP-
NAME --location YOUR-REGION (For example: westus2)"
az vm create --resource-group YOUR-RESOURCE-GROUP-NAME --name YOUR-VM-NAME --image microsoft-
dsvm:linux-data-science-vm-ubuntu:linuxdsmbuntu:latest --admin-username YOUR-USERNAME --
admin-password YOUR-PASSWORD --generate-ssh-keys --authentication-type password
```

To create a Windows DSVM, use the following command:

```
# create a Windows Server 2016 DSVM in your resource group
# note you need to be at least a contributor to the resource group in order to execute this
# command successfully
az vm create --resource-group YOUR-RESOURCE-GROUP-NAME --name YOUR-VM-NAME --image microsoft-
dsvm:dsvm-windows:server-2016:latest --admin-username YOUR-USERNAME --admin-password YOUR-
PASSWORD --authentication-type password
```

2. Activate the conda environment containing the Azure Machine Learning SDK.

- For Ubuntu Data Science VM:

```
conda activate py36
```

- For Windows Data Science VM:

```
conda activate AzureML
```

3. To configure the Data Science VM to use your Azure Machine Learning workspace, [create a workspace configuration file](#) or use an existing one.

Similar to local environments, you can use Visual Studio Code and the [Azure Machine Learning Visual Studio Code extension](#) to interact with Azure Machine Learning.

For more information, see [Data Science Virtual Machines](#).

Next steps

- [Train and deploy a model](#) on Azure Machine Learning with the MNIST dataset.
- See the [Azure Machine Learning SDK for Python reference](#).

Install and set up the CLI (v2)

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The `ml` extension (preview) to the [Azure CLI](#) is the enhanced interface for Azure Machine Learning. It enables you to train and deploy models from the command line, with features that accelerate scaling data science up and out while tracking the model lifecycle.

Prerequisites

- To use the CLI, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- To use the CLI commands in this document from your **local environment**, you need the [Azure CLI](#).

Installation

The new Machine Learning extension **requires Azure CLI version `>=2.15.0`**. Ensure this requirement is met:

```
az version
```

If it isn't, [upgrade your Azure CLI](#).

Check the Azure CLI extensions you've installed:

```
az extension list
```

Remove any existing installation of the `ml` extension and also the CLI v1 `azure-cli-ml` extension:

```
az extension remove -n azure-cli-ml
az extension remove -n ml
```

Now, install the `ml` extension:

```
az extension add -n ml -y
```

Run the help command to verify your installation and see available subcommands:

```
az ml -h
```

You can upgrade the extension to the latest version:

```
az extension update -n ml
```

Installation on Linux

If you're using Linux, the fastest way to install the necessary CLI version and the Machine Learning extension is:

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash  
az extension add -n ml -y
```

For more, see [Install the Azure CLI for Linux](#).

Set up

Login:

```
az login
```

If you have access to multiple Azure subscriptions, you can set your active subscription:

```
az account set -s "<YOUR_SUBSCRIPTION_NAME_OR_ID>"
```

Optionally, setup common variables in your shell for usage in subsequent commands:

```
GROUP="azureml-examples"  
  
LOCATION="eastus"  
  
WORKSPACE="main"
```

WARNING

This uses Bash syntax for setting variables -- adjust as needed for your shell. You can also replace the values in commands below inline rather than using variables.

If it doesn't already exist, you can create the Azure resource group:

```
az group create -n $GROUP -l $LOCATION
```

And create a machine learning workspace:

```
az ml workspace create -n $WORKSPACE -g $GROUP -l $LOCATION
```

Machine learning subcommands require the `--workspace/-w` and `--resource-group/-g` parameters. To avoid typing these repeatedly, configure defaults:

```
az configure --defaults group=$GROUP workspace=$WORKSPACE location=$LOCATION
```

TIP

Most code examples assume you have set a default workspace and resource group. You can override these on the command line.

You can show your current defaults using `--list-defaults/-l`:

```
az configure -l -o table
```

TIP

Combining with `--output/-o` allows for more readable output formats.

Secure communications

The `m1` CLI extension (sometimes called 'CLI v2') for Azure Machine Learning sends operational data (YAML parameters and metadata) over the public internet. All the `m1` CLI extension commands communicate with the Azure Resource Manager. This communication is secured using HTTPS/TLS 1.2.

Data in a data store that is secured in a virtual network is *not* sent over the public internet. For example, if your training data is located in the default storage account for the workspace, and the storage account is in a virtual network.

NOTE

With the previous extension (`azure-cli-m1`, sometimes called 'CLI v1'), only some of the commands communicate with the Azure Resource Manager. Specifically, commands that create, update, delete, list, or show Azure resources. Operations such as submitting a training job communicate directly with the Azure Machine Learning workspace. If your workspace is [secured with a private endpoint](#), that is enough to secure commands provided by the `azure-cli-m1` extension.

- [Public workspace](#)
- [Private workspace](#)

If your Azure Machine Learning workspace is public (that is, not behind a virtual network), then there is no additional configuration required. Communications are secured using HTTPS/TLS 1.2

Next steps

- [Train models using CLI \(v2\)](#)
- [Set up the Visual Studio Code Azure Machine Learning extension](#)
- [Train an image classification TensorFlow model using the Azure Machine Learning Visual Studio Code extension](#)
- [Explore Azure Machine Learning with examples](#)

Manage software environments in Azure Machine Learning studio

9/22/2022 • 2 minutes to read • [Edit Online](#)

In this article, learn how to create and manage Azure Machine Learning environments in the Azure Machine Learning studio. Use the environments to track and reproduce your projects' software dependencies as they evolve.

The examples in this article show how to:

- Browse curated environments.
- Create an environment and specify package dependencies.
- Edit an existing environment specification and its properties.
- Rebuild an environment and view image build logs.

For a high-level overview of how environments work in Azure Machine Learning, see [What are ML environments?](#) For information, see [How to set up a development environment for Azure Machine Learning](#).

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- An [Azure Machine Learning workspace](#).

Browse curated environments

Curated environments contain collections of Python packages and are available in your workspace by default. These environments are backed by cached Docker images which reduces the job preparation cost and support training and inferencing scenarios.

Click on an environment to see detailed information about its contents. For more information, see [Azure Machine Learning curated environments](#).

Create an environment

To create an environment:

1. Open your workspace in [Azure Machine Learning studio](#).
2. On the left side, select **Environments**.
3. Select the **Custom environments** tab.
4. Select the **Create** button.

Create an environment by specifying one of the following:

- Create a new docker [context](#)
- Start from an existing custom or curated environment
- Upload existing docker context
- Use existing docker image with conda

Create environment X

Use environments to track and reproduce your projects' software dependencies as they evolve.

Settings Customize Tags Review

Settings

Name *

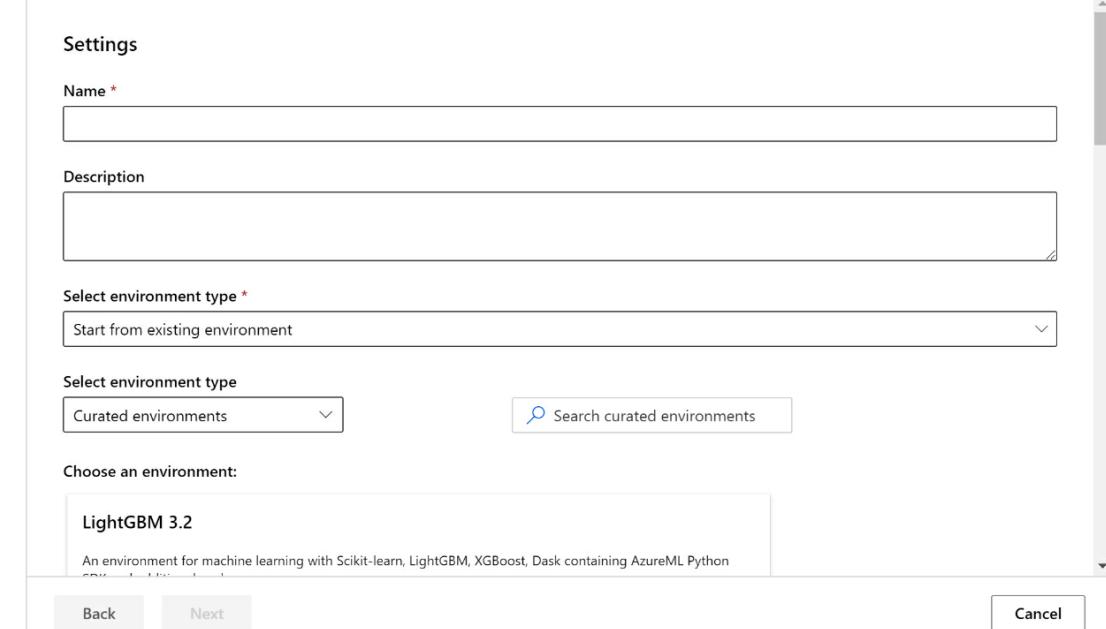
Description

Select environment type *

Select environment type

Choose an environment:

An environment for machine learning with Scikit-learn, LightGBM, XGBoost, Dask containing AzureML Python
Curated environments



You can customize the configuration file, add tags and descriptions, and review the properties before creating the entity.

If a new environment is given the same name as an existing environment in the workspace, a new version of the existing one will be created.

View and edit environment details

Once an environment has been created, view its details by clicking on the name. Use the dropdown menu to select different versions of the environment. Here you can view metadata and the contents of the environment through its various dependencies.

Click on the pencil icons to edit tags and descriptions as well as the configuration files under the **Context** tab.

Keep in mind that any changes to the Docker or Conda sections will create a new version of the environment.

example-environment Version: 1 (latest) 

Details Build log

Refresh Build

Details	Docker image
Build status Running	Parent image mcr.microsoft.com/azurermi/openmpi3.1.2-cuda10.2-cudnn8-ubuntu18.04:20210507.v1
Name example-environment	Conda
Version 1	<pre>1 channels: 2 - conda-forge 3 dependencies: 4 - python=3.6.1 5 - numpy 6 - pip 7 - pip: 8 - pandas 9 - matplotlib 10 name: azurermi_1a822b8d136308e0f37485ae45727166 11</pre>
Description	
small set of basic python packages to test an azurermi example	
Tags	
example-env	

View image build logs

Click on the **Build log** tab within the details page to view the image build logs of an environment version.

Rebuild an environment

In the details page, click on the **rebuild** button to rebuild the environment. Any unpinned package versions in your configuration files may be updated to the most recent version with this action.

How to create and manage files in your workspace

9/22/2022 • 2 minutes to read • [Edit Online](#)

Learn how to create and manage the files in your Azure Machine Learning workspace. These files are stored in the default workspace storage. Files and folders can be shared with anyone else with the read access to the workspace, and can be used from any compute instances in the workspace.

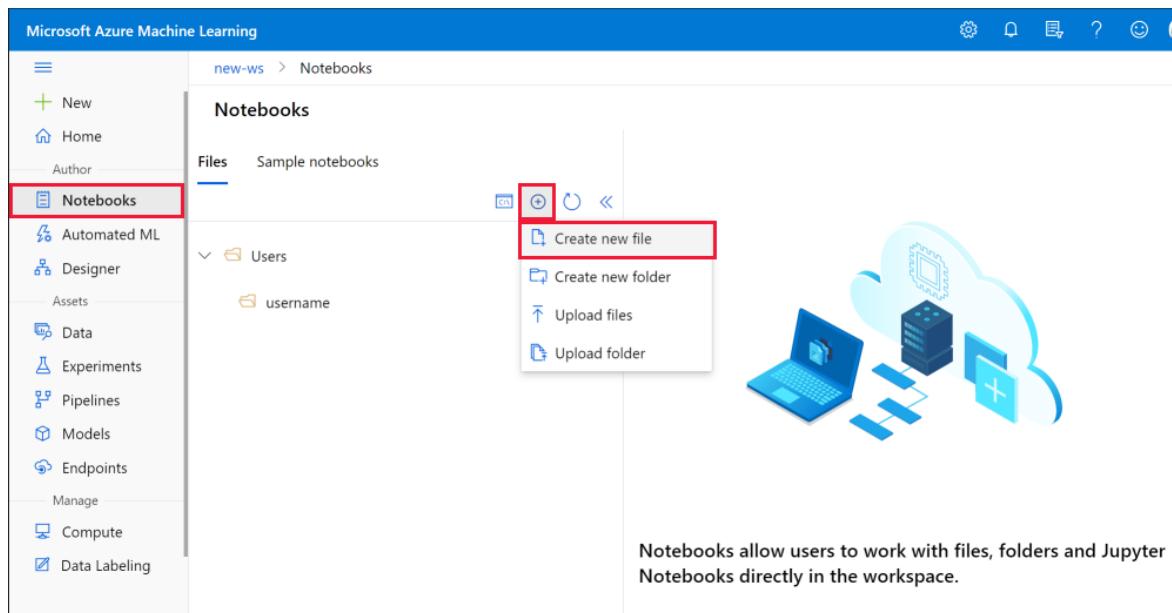
Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A Machine Learning workspace. [Create workspace resources](#).

Create files

To create a new file in your default folder (`Users > yourusername`):

1. Open your workspace in [Azure Machine Learning studio](#).
2. On the left side, select **Notebooks**.
3. Select the **+** image.
4. Select the **Create new file** image.



5. Name the file.

6. Select a file type.

7. Select **Create**.

Notebooks and most text file types display in the preview section. Most other file types don't have a preview.

To create a new file in a different folder:

1. Select the **...** at the end of the folder
2. Select **Create new file**.

IMPORTANT

Content in notebooks and scripts can potentially read data from your sessions and access data without your organization in Azure. Only load files from trusted sources. For more information, see [Secure code best practices](#).

Manage files with Git

Use a [compute instance terminal](#) to clone and manage Git repositories.

Clone samples

Your workspace contains a **Sample notebooks** folder with notebooks designed to help you explore the SDK and serve as examples for your own machine learning projects. Clone these notebooks into your own folder to run and edit them.

For an example, see [Tutorial: Create your first ML experiment](#).

Share files

Copy and paste the URL to share a file. Only other users of the workspace can access this URL. Learn more about [granting access to your workspace](#).

Delete a file

You *can't* delete the **Sample notebooks** files. These files are part of the studio and are updated each time a new SDK is published.

You *can* delete files found in your **Files** section in any of these ways:

- In the studio, select the ... at the end of a folder or file. Make sure to use a supported browser (Microsoft Edge, Chrome, or Firefox).
- [Use a terminal](#) from any compute instance in your workspace. The folder `~/cloudfiles` is mapped to storage on your workspace storage account.
- In either Jupyter or JupyterLab with their tools.

Run Jupyter notebooks in your workspace

9/22/2022 • 12 minutes to read • [Edit Online](#)

Learn how to run your Jupyter notebooks directly in your workspace in Azure Machine Learning studio. While you can launch [Jupyter](#) or [JupyterLab](#), you can also edit and run your notebooks without leaving the workspace.

For information on how to create and manage files, including notebooks, see [Create and manage files in your workspace](#).

IMPORTANT

Features marked as (preview) are provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A Machine Learning workspace. See [Create workspace resources](#).
- Your user identity must have access to your workspace's default storage account. Whether you can read, edit, or create notebooks depends on your [access level](#) to your workspace. For example, a Contributor can edit the notebook, while a Reader could only view it.

Access notebooks from your workspace

Use the **Notebooks** section of your workspace to edit and run Jupyter notebooks.

1. Sign into [Azure Machine Learning studio](#)
2. Select your workspace, if it isn't already open
3. On the left, select **Notebooks**

Edit a notebook

To edit a notebook, open any notebook located in the **User files** section of your workspace. Click on the cell you wish to edit. If you don't have any notebooks in this section, see [Create and manage files in your workspace](#).

You can edit the notebook without connecting to a compute instance. When you want to run the cells in the notebook, select or create a compute instance. If you select a stopped compute instance, it will automatically start when you run the first cell.

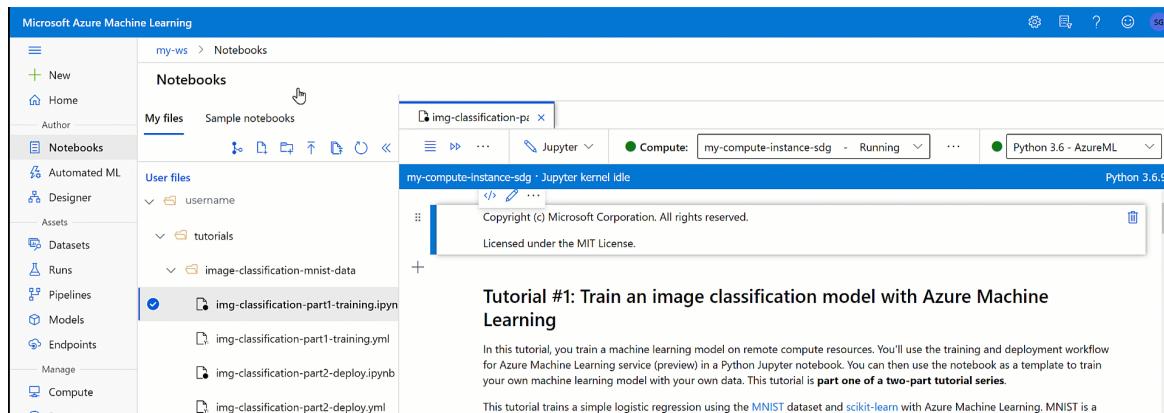
When a compute instance is running, you can also use code completion, powered by [Intellisense](#), in any Python notebook.

You can also launch Jupyter or JupyterLab from the notebook toolbar. Azure Machine Learning does not provide updates and fix bugs from Jupyter or JupyterLab as they are Open Source products outside of the boundary of Microsoft Support.

Focus mode

Use focus mode to expand your current view so you can focus on your active tabs. Focus mode hides the Notebooks file explorer.

1. In the terminal window toolbar, select **Focus mode** to turn on focus mode. Depending on your window width, the tool may be located under the ... menu item in your toolbar.
2. While in focus mode, return to the standard view by selecting **Standard view**.



Code completion (IntelliSense)

[IntelliSense](#) is a code-completion aid that includes many features: List Members, Parameter Info, Quick Info, and Complete Word. With only a few keystrokes, you can:

- Learn more about the code you're using
- Keep track of the parameters you're typing
- Add calls to properties and methods

Share a notebook

Your notebooks are stored in your workspace's storage account, and can be shared with others, depending on their [access level](#) to your workspace. They can open and edit the notebook as long as they have the appropriate access. For example, a Contributor can edit the notebook, while a Reader could only view it.

Other users of your workspace can find your notebook in the **Notebooks**, **User files** section of Azure ML studio. By default, your notebooks are in a folder with your username, and others can access them there.

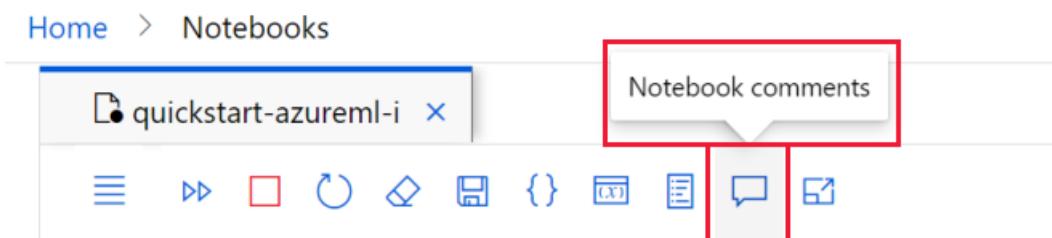
You can also copy the URL from your browser when you open a notebook, then send to others. As long as they have appropriate access to your workspace, they can open the notebook.

Since you don't share compute instances, other users who run your notebook will do so on their own compute instance.

Collaborate with notebook comments (preview)

Use a notebook comment to collaborate with others who have access to your notebook.

Toggle the comments pane on and off with the **Notebook comments** tool at the top of the notebook. If your screen isn't wide enough, find this tool by first selecting the ... at the end of the set of tools.



Whether the comments pane is visible or not, you can add a comment into any code cell:

1. Select some text in the code cell. You can only comment on text in a code cell.
2. Use the **New comment thread** tool to create your comment.

A screenshot of a Jupyter Notebook cell toolbar. The toolbar includes icons for copy, paste, cut, and other cell operations. A red box highlights the 'New comment thread' icon, which is a speech bubble icon. The cell itself contains Python code:

```

1 from utils import load_data
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import glob
-
```

3. If the comments pane was previously hidden, it will now open.
4. Type your comment and post it with the tool or use **Ctrl+Enter**.
5. Once a comment is posted, select ... in the top right to:
 - Edit the comment
 - Resolve the thread
 - Delete the thread

Text that has been commented will appear with a purple highlight in the code. When you select a comment in the comments pane, your notebook will scroll to the cell that contains the highlighted text.

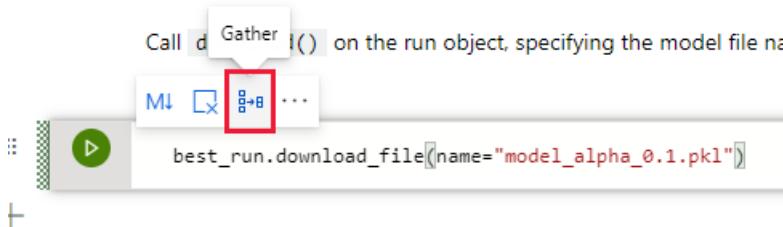
NOTE

Comments are saved into the code cell's metadata.

Clean your notebook (preview)

Over the course of creating a notebook, you typically end up with cells you used for data exploration or debugging. The *gather* feature will help you produce a clean notebook without these extraneous cells.

1. Run all of your notebook cells.
2. Select the cell containing the code you wish the new notebook to run. For example, the code that submits an experiment, or perhaps the code that registers a model.
3. Select the **Gather** icon that appears on the cell toolbar.



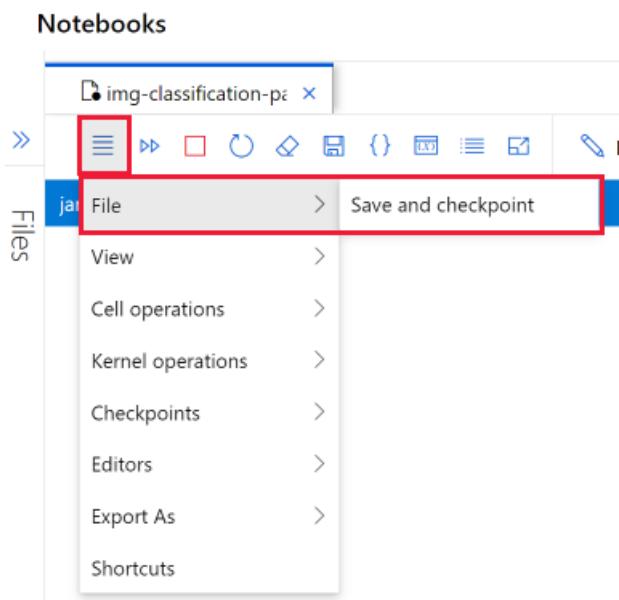
4. Enter the name for your new "gathered" notebook.

The new notebook contains only code cells, with all cells required to produce the same results as the cell you selected for gathering.

Save and checkpoint a notebook

Azure Machine Learning creates a checkpoint file when you create an *ipynb* file.

In the notebook toolbar, select the menu and then **File>Save and checkpoint** to manually save the notebook and it will add a checkpoint file associated with the notebook.



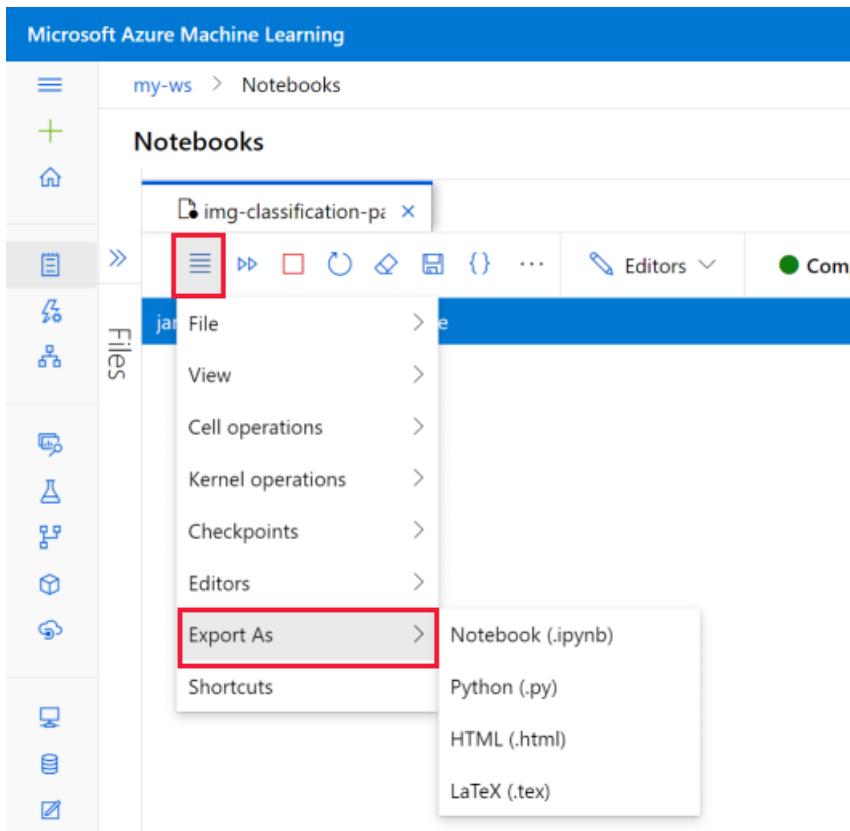
Every notebook is autosaved every 30 seconds. AutoSave updates only the initial *ipynb* file, not the checkpoint file.

Select **Checkpoints** in the notebook menu to create a named checkpoint and to revert the notebook to a saved checkpoint.

Export a notebook

In the notebook toolbar, select the menu and then **Export As** to export the notebook as any of the supported types:

- Notebook
- Python
- HTML
- LaTeX



The exported file is saved on your computer.

Run a notebook or Python script

To run a notebook or a Python script, you first connect to a running [compute instance](#).

- If you don't have a compute instance, use these steps to create one:
 1. In the notebook or script toolbar, to the right of the Compute dropdown, select **+ New Compute**. Depending on your screen size, this may be located under a ... menu.
 2. Name the Compute and choose a **Virtual Machine Size**.
 3. Select **Create**.
 4. The compute instance is connected to the file automatically. You can now run the notebook cells or the Python script using the tool to the left of the compute instance.
- If you have a stopped compute instance, select **Start compute** to the right of the Compute dropdown. Depending on your screen size, this may be located under a ... menu.

Once you are connected to a compute instance, use the toolbar to run all cells in the notebook, or Control + Enter to run a single selected cell.

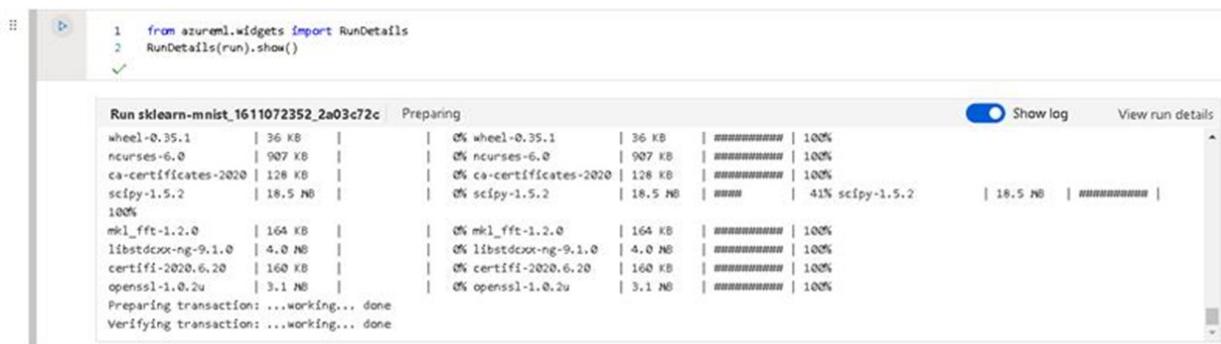
Only you can see and use the compute instances you create. Your **User files** are stored separately from the VM and are shared among all compute instances in the workspace.

View logs and output

Use [notebook widgets](#) to view the progress of the run and logs. A widget is asynchronous and provides updates until training finishes. Azure Machine Learning widgets are also supported in Jupyter and JupyterLab.

Jupyter widget

Watch the progress of the run with a Jupyter widget. Like the run submission, the widget is asynchronous and provides live updates every 10-15 seconds until the job completes.



The screenshot shows a Jupyter notebook cell with the following code:

```
1 from azureml.widgets import RunDetails
2 RunDetails(run).show()
```

The output area displays the progress of a package installation:

Run sklearn-mnist_1611072352_2a03c72c Preparing

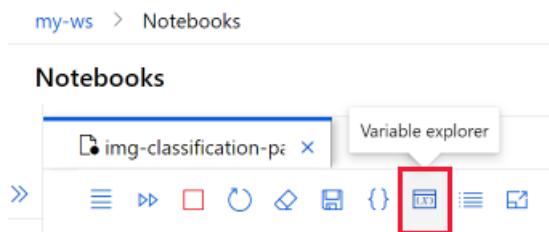
Package	Size	Progress (%)	Downloaded	Total	Status
wheel-0.35.1	36 KB	0%	wheel-0.35.1	36 KB	100%
ncurses-6.0	907 KB	0%	ncurses-6.0	907 KB	100%
ca-certificates-2020	128 KB	0%	ca-certificates-2020	128 KB	100%
scipy-1.5.2	18.5 MB	0%	scipy-1.5.2	18.5 MB	41% scipy-1.5.2
mkl_fft-1.2.0	164 KB	0%	mkl_fft-1.2.0	164 KB	100%
libstdcxx-ng-9.1.0	4.0 MB	0%	libstdcxx-ng-9.1.0	4.0 MB	100%
certifi-2020.6.20	160 KB	0%	certifi-2020.6.20	160 KB	100%
openssl-1.0.2u	3.1 MB	0%	openssl-1.0.2u	3.1 MB	100%

Preparing transaction: ...working... done
Verifying transaction: ...working... done

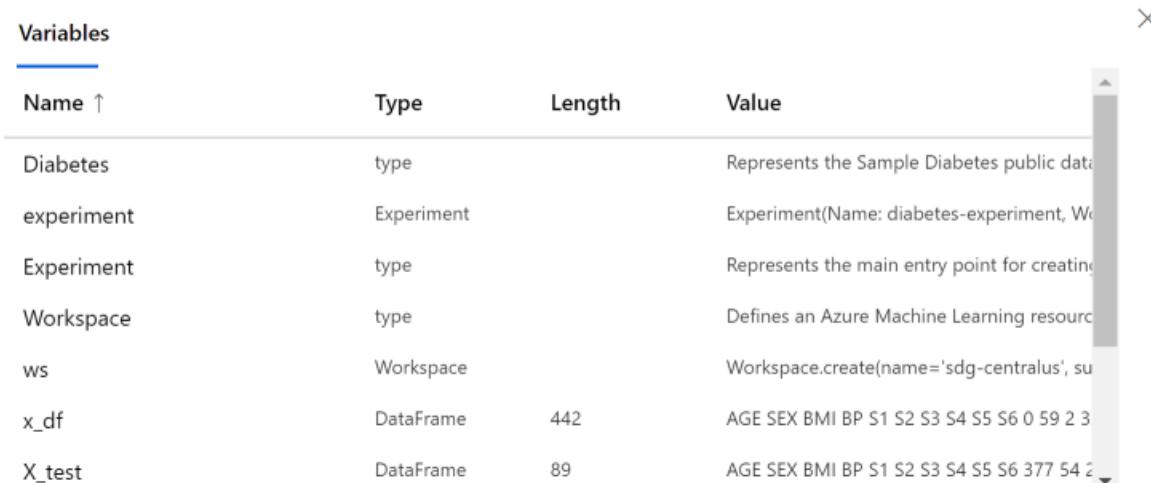
At the top right of the output area, there are "Show log" and "View run details" buttons.

Explore variables in the notebook

On the notebook toolbar, use the **Variable explorer** tool to show the name, type, length, and sample values for all variables that have been created in your notebook.



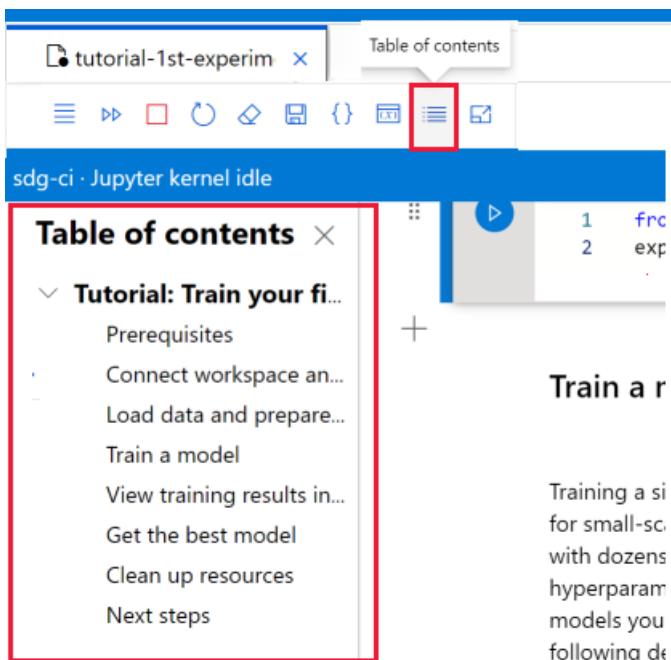
Select the tool to show the variable explorer window.



Name ↑	Type	Length	Value
Diabetes	type		Represents the Sample Diabetes public data
experiment	Experiment		Experiment(Name: diabetes-experiment, We
Experiment	type		Represents the main entry point for creating
Workspace	type		Defines an Azure Machine Learning resourc
ws	Workspace		Workspace.create(name='sdg-centralus', su
x_df	DataFrame	442	AGE SEX BMI BP S1 S2 S3 S4 S5 S6 0 59 2 3
X_test	DataFrame	89	AGE SEX BMI BP S1 S2 S3 S4 S5 S6 377 54 2

Navigate with a TOC

On the notebook toolbar, use the **Table of contents** tool to display or hide the table of contents. Start a markdown cell with a heading to add it to the table of contents. Click on an entry in the table to scroll to that cell in the notebook.



Change the notebook environment

The notebook toolbar allows you to change the environment on which your notebook runs.

These actions will not change the notebook state or the values of any variables in the notebook:

ACTION	RESULT
Stop the kernel	Stops any running cell. Running a cell will automatically restart the kernel.
Navigate to another workspace section	Running cells are stopped.

These actions will reset the notebook state and will reset all variables in the notebook.

ACTION	RESULT
Change the kernel	Notebook uses new kernel
Switch compute	Notebook automatically uses the new compute.
Reset compute	Starts again when you try to run a cell
Stop compute	No cells will run
Open notebook in Jupyter or JupyterLab	Notebook opened in a new tab.

Add new kernels

[Use the terminal](#) to create and add new kernels to your compute instance. The notebook will automatically find all Jupyter kernels installed on the connected compute instance.

Use the kernel dropdown on the right to change to any of the installed kernels.

Manage packages

Since your compute instance has multiple kernels, make sure use `%pip` or `%conda` [magic functions](#), which install packages into the currently-running kernel. Don't use `!pip` or `!conda`, which refers to all packages (including packages outside the currently-running kernel).

Status indicators

An indicator next to the **Compute** dropdown shows its status. The status is also shown in the dropdown itself.

COLOR	COMPUTE STATUS
Green	Compute running
Red	Compute failed
Black	Compute stopped
Light Blue	Compute creating, starting, restarting, setting Up
Gray	Compute deleting, stopping

An indicator next to the **Kernel** dropdown shows its status.

COLOR	KERNEL STATUS
Green	Kernel connected, idle, busy
Gray	Kernel not connected

Find compute details

Find details about your compute instances on the **Compute** page in [studio](#).

Useful keyboard shortcuts

Similar to Jupyter Notebooks, Azure Machine Learning Studio notebooks have a modal user interface. The keyboard does different things depending on which mode the notebook cell is in. Azure Machine Learning Studio notebooks support the following two modes for a given code cell: command mode and edit mode.

Command mode shortcuts

A cell is in command mode when there is no text cursor prompting you to type. When a cell is in Command mode, you can edit the notebook as a whole but not type into individual cells. Enter command mode by pressing `Esc` or using the mouse to select outside of a cell's editor area. The left border of the active cell is blue and solid, and its **Run** button is blue.



SHORTCUT	DESCRIPTION
Enter	Enter edit mode
Shift + Enter	Run cell, select below

SHORTCUT	DESCRIPTION
Control/Command + Enter	Run cell
Alt + Enter	Run cell, insert code cell below
Control/Command + Alt + Enter	Run cell, insert markdown cell below
Alt + R	Run all
Y	Convert cell to code
M	Convert cell to markdown
Up/K	Select cell above
Down/J	Select cell below
A	Insert code cell above
B	Insert code cell below
Control/Command + Shift + A	Insert markdown cell above
Control/Command + Shift + B	Insert markdown cell below
X	Cut selected cell
C	Copy selected cell
Shift + V	Paste selected cell above
V	Paste selected cell below
D D	Delete selected cell
O	Toggle output
Shift + O	Toggle output scrolling
I I	Interrupt kernel
0 0	Restart kernel
Shift + Space	Scroll up
Space	Scroll down
Tab	Change focus to next focusable item (when tab trap disabled)
Control/Command + S	Save notebook

SHORTCUT	DESCRIPTION
1	Change to h1
2	Change to h2
3	Change to h3
4	Change to h4
5	Change to h5
6	Change to h6

Edit mode shortcuts

Edit mode is indicated by a text cursor prompting you to type in the editor area. When a cell is in edit mode, you can type into the cell. Enter edit mode by pressing `Enter` or using the mouse to select on a cell's editor area. The left border of the active cell is green and hatched, and its Run button is green. You also see the cursor prompt in the cell in Edit mode.



Using the following keystroke shortcuts, you can more easily navigate and run code in Azure Machine Learning notebooks when in Edit mode.

SHORTCUT	DESCRIPTION
Escape	Enter command mode
Control/Command + Space	Activate IntelliSense
Shift + Enter	Run cell, select below
Control/Command + Enter	Run cell
Alt + Enter	Run cell, insert code cell below
Control/Command + Alt + Enter	Run cell, insert markdown cell below
Alt + R	Run all cells
Up	Move cursor up or previous cell
Down	Move cursor down or next cell
Control/Command + S	Save notebook
Control/Command + Up	Go to cell start
Control/Command + Down	Go to cell end

SHORTCUT	DESCRIPTION
Tab	Code completion or indent (if tab trap enabled)
Control/Command + M	Enable/disable tab trap
Control/Command +]	Indent
Control/Command + [Dedent
Control/Command + A	Select all
Control/Command + Z	Undo
Control/Command + Shift + Z	Redo
Control/Command + Y	Redo
Control/Command + Home	Go to cell start
Control/Command + End	Go to cell end
Control/Command + Left	Go one word left
Control/Command + Right	Go one word right
Control/Command + Backspace	Delete word before
Control/Command + Delete	Delete word after
Control/Command + /	Toggle comment on cell

Troubleshooting

- **Connecting to a notebook:** If you can't connect to a notebook, ensure that web socket communication is **not** disabled. For compute instance Jupyter functionality to work, web socket communication must be enabled. Ensure your [network allows websocket connections](#) to *.instances.azureml.net and *.instances.azureml.ms.
- **Private endpoint:** When a compute instance is deployed in a workspace with a private endpoint, it can be only be [accessed from within virtual network](#). If you are using custom DNS or hosts file, add an entry for < instance-name >.< region >.instances.azureml.ms with the private IP address of your workspace private endpoint. For more information see the [custom DNS](#) article.
- **Kernel crash:** If your kernel crashed and was restarted, you can run the following command to look at jupyter log and find out more details: `sudo journalctl -u jupyter`. If kernel issues persist, consider using a compute instance with more memory.
- **Kernel not found or Kernel operations were disabled:** When using the default Python 3.8 kernel on a compute instance, you may get an error such as "Kernel not found" or "Kernel operations were disabled". To fix, use one of the following methods:
 - Create a new compute instance. This will use a new image where this problem has been resolved.
 - Use the Py 3.6 kernel on the existing compute instance.

- From a terminal in the default py38 environment, run `pip install ipykernel==6.6.0` OR
`pip install ipykernel==6.0.3`
- **Expired token:** If you run into an expired token issue, sign out of your Azure ML studio, sign back in, and then restart the notebook kernel.
- **File upload limit:** When uploading a file through the notebook's file explorer, you are limited to files that are smaller than 5TB. If you need to upload a file larger than this, we recommend that you use one of the following methods:
 - Use the SDK to upload the data to a datastore. For more information, see the [Upload the data](#) section of the tutorial.
 - Use [Azure Data Factory](#) to create a data ingestion pipeline.

Next steps

- [Run your first experiment](#)
- [Backup your file storage with snapshots](#)
- [Working in secure environments](#)

Access a compute instance terminal in your workspace

9/22/2022 • 2 minutes to read • [Edit Online](#)

Access the terminal of a compute instance in your workspace to:

- Use files from Git and version files. These files are stored in your workspace file system, not restricted to a single compute instance.
- Install packages on the compute instance.
- Create extra kernels on the compute instance.

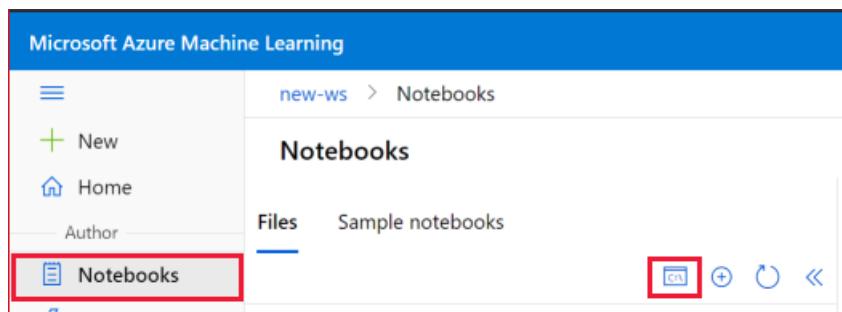
Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A Machine Learning workspace. See [Create workspace resources](#).

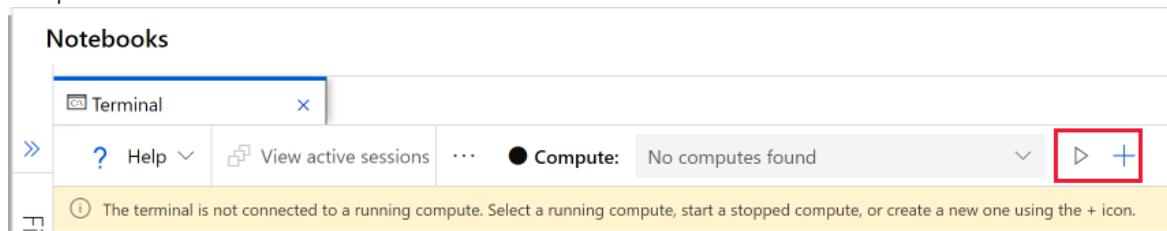
Access a terminal

To access the terminal:

1. Open your workspace in [Azure Machine Learning studio](#).
2. On the left side, select **Notebooks**.
3. Select the **Open terminal** image.



4. When a compute instance is running, the terminal window for that compute instance appears.
5. When no compute instance is running, use the **Compute** section on the right to start or create a compute instance.



In addition to the steps above, you can also access the terminal from:

- RStudio (See [Add RStudio](#)): Select the **Terminal** tab on top left.
- Jupyter Lab: Select the **Terminal** tile under the **Other** heading in the Launcher tab.
- Jupyter: Select **New>Terminal** on top right in the Files tab.

- SSH to the machine, if you enabled SSH access when the compute instance was created.

Copy and paste in the terminal

- Windows: `Ctrl-Insert` to copy and use `Ctrl-Shift-v` or `Shift-Insert` to paste.
- Mac OS: `Cmd-c` to copy and `Cmd-v` to paste.
- FireFox/IE may not support clipboard permissions properly.

Use files from Git and version files

Access all Git operations from the terminal. All Git files and folders will be stored in your workspace file system. This storage allows you to use these files from any compute instance in your workspace.

NOTE

Add your files and folders anywhere under the `~/cloudfiles/code/Users` folder so they will be visible in all your Jupyter environments.

Learn more about [cloning Git repositories into your workspace file system](#).

Install packages

Install packages from a terminal window. Install Python packages into the **Python 3.8 - AzureML** environment. Install R packages into the **R** environment.

Or you can install packages directly in Jupyter Notebook or RStudio:

- RStudio ([Add RStudio](#)): Use the **Packages** tab on the bottom right, or the **Console** tab on the top left.
- Python: Add install code and execute in a Jupyter Notebook cell.

NOTE

For package management within a notebook, use `%pip` or `%conda` magic functions to automatically install packages into the **currently-running kernel**, rather than `!pip` or `!conda` which refers to all packages (including packages outside the currently-running kernel)

Add new kernels

WARNING

While customizing the compute instance, make sure you do not delete the `azureml_py36` or `azureml_py38` conda environments. Also do not delete **Python 3.6 - AzureML** or **Python 3.8 - AzureML** kernels. These are needed for Jupyter/JupyterLab functionality.

To add a new Jupyter kernel to the compute instance:

1. Use the terminal window to create a new environment. For example, the code below creates `newenv`:

```
conda create --name newenv
```

2. Activate the environment. For example, after creating `newenv`:

```
conda activate newenv
```

3. Install pip and ipykernel package to the new environment and create a kernel for that conda env

```
conda install pip
conda install ipykernel
python -m ipykernel install --user --name newenv --display-name "Python (newenv)"
```

Any of the [available Jupyter Kernels](#) can be installed.

Manage terminal sessions

Select **View active sessions** in the terminal toolbar to see a list of all active terminal sessions. When there are no active sessions, this tab will be disabled.

WARNING

Make sure you close any unused sessions to preserve your compute instance's resources. Idle terminals may impact performance of compute instances.

Create & use software environments in Azure Machine Learning

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, learn how to create and manage Azure Machine Learning [environments](#). Use the environments to track and reproduce your projects' software dependencies as they evolve.

Software dependency management is a common task for developers. You want to ensure that builds are reproducible without extensive manual software configuration. The Azure Machine Learning `Environment` class accounts for local development solutions such as pip and Conda and distributed cloud development through Docker capabilities.

The examples in this article show how to:

- Create an environment and specify package dependencies.
- Retrieve and update environments.
- Use an environment for training.
- Use an environment for web service deployment.

For a high-level overview of how environments work in Azure Machine Learning, see [What are ML environments?](#) For information about managing environments in the Azure ML studio, see [Manage environments in the studio](#). For information about configuring development environments, see [Set up a Python development environment for Azure ML](#).

Prerequisites

- The [Azure Machine Learning SDK for Python](#) (>= 1.13.0)
- An [Azure Machine Learning workspace](#)

Create an environment

The following sections explore the multiple ways that you can create an environment for your experiments.

Instantiate an environment object

To manually create an environment, import the `Environment` class from the SDK. Then use the following code to instantiate an environment object.

```
from azureml.core.environment import Environment  
Environment(name="myenv")
```

Use a curated environment

Curated environments contain collections of Python packages and are available in your workspace by default. These environments are backed by cached Docker images which reduces the run preparation cost. You can select one of these popular curated environments to start with:

- The `AzureML-lightgbm-3.2-ubuntu18.04-py37-cpu` environment contains Scikit-learn, LightGBM, XGBoost, Dask as well as other AzureML Python SDK and additional packages.

- The `AzureML-sklearn-0.24-ubuntu18.04-py37-cpu` environment contains common data science packages. These packages include Scikit-Learn, Pandas, Matplotlib, and a larger set of `azureml-sdk` packages.

For a list of curated environments, see the [curated environments article](#).

Use the `Environment.get` method to select one of the curated environments:

```
from azureml.core import Workspace, Environment  
  
ws = Workspace.from_config()  
env = Environment.get(workspace=ws, name="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu")
```

You can list the curated environments and their packages by using the following code:

```
envs = Environment.list(workspace=ws)  
  
for env in envs:  
    if env.startswith("AzureML"):  
        print("Name", env)  
        print("packages", envs[env].python.conda_dependencies.serialize_to_string())
```

WARNING

Don't start your own environment name with the `AzureML` prefix. This prefix is reserved for curated environments.

To customize a curated environment, clone and rename the environment.

```
env = Environment.get(workspace=ws, name="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu")  
curated_clone = env.clone("customize_curated")
```

Use Conda dependencies or pip requirements files

You can create an environment from a Conda specification or a pip requirements file. Use the `from_conda_specification()` method or the `from_pip_requirements()` method. In the method argument, include your environment name and the file path of the file that you want.

```
# From a Conda specification file  
myenv = Environment.from_conda_specification(name = "myenv",  
                                              file_path = "path-to-conda-specification-file")  
  
# From a pip requirements file  
myenv = Environment.from_pip_requirements(name = "myenv",  
                                            file_path = "path-to-pip-requirements-file")
```

Enable Docker

Azure Machine Learning builds a Docker image and creates a Python environment within that container, given your specifications. The Docker images are cached and reused: the first run in a new environment typically takes longer as the image is build. For local runs, specify Docker within the [RunConfiguration](#).

By default, the newly built Docker image appears in the container registry that's associated with the workspace. The repository name has the form `azureml/azureml_<uuid>`. The unique identifier (`uuid`) part of the name corresponds to a hash that's computed from the environment configuration. This correspondence allows the service to determine whether an image for the given environment already exists for reuse.

Use a prebuilt Docker image

By default, the service automatically uses one of the Ubuntu Linux-based [base images](#), specifically the one defined by `azureml.core.environment.DEFAULT_CPU_IMAGE`. It then installs any specified Python packages defined by the provided Azure ML environment. Other Azure ML CPU and GPU base images are available in the container [repository](#). It is also possible to use a [custom Docker base image](#).

```
# Specify custom Docker base image and registry, if you don't want to use the defaults
myenv.docker.base_image="your_base-image"
myenv.docker.base_image_registry="your_registry_location"
```

IMPORTANT

Azure Machine Learning only supports Docker images that provide the following software:

- Ubuntu 18.04 or greater.
- Conda 4.7.# or greater.
- Python 3.6+.
- A POSIX compliant shell available at `/bin/sh` is required in any container image used for training.

Use your own Dockerfile

You can also specify a custom Dockerfile. It's simplest to start from one of Azure Machine Learning base images using Docker `FROM` command, and then add your own custom steps. Use this approach if you need to install non-Python packages as dependencies. Remember to set the base image to None.

Please note that Python is an implicit dependency in Azure Machine Learning so a custom dockerfile must have Python installed.

```
# Specify docker steps as a string.
dockerfile = r"""
FROM mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04
RUN echo "Hello from custom container!"
"""

# Set base image to None, because the image is defined by dockerfile.
myenv.docker.base_image = None
myenv.docker.base_dockerfile = dockerfile

# Alternatively, load the string from a file.
myenv.docker.base_image = None
myenv.docker.base_dockerfile = "./Dockerfile"
```

When using custom Docker images, it is recommended that you pin package versions in order to better ensure reproducibility.

Specify your own Python interpreter

In some situations, your custom base image may already contain a Python environment with packages that you want to use.

To use your own installed packages and disable Conda, set the parameter

`Environment.python.user_managed_dependencies = True`. Ensure that the base image contains a Python interpreter, and has the packages your training script needs.

For example, to run in a base Miniconda environment that has NumPy package installed, first specify a Dockerfile with a step to install the package. Then set the user-managed dependencies to `True`.

You can also specify a path to a specific Python interpreter within the image, by setting the

`Environment.python.interpreter_path` variable.

```
dockerfile = """
FROM mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210615.v1
RUN conda install numpy
"""

myenv.docker.base_image = None
myenv.docker.base_dockerfile = dockerfile
myenv.python.user_managed_dependencies=True
myenv.python.interpreter_path = "/opt/miniconda/bin/python"
```

WARNING

If you install some Python dependencies in your Docker image and forget to set `user_managed_dependencies=True`, those packages will not exist in the execution environment thus causing runtime failures. By default, Azure ML will build a Conda environment with dependencies you specified, and will execute the run in that environment instead of using any Python libraries that you installed on the base image.

Retrieve image details

For a registered environment, you can retrieve image details using the following code where `details` is an instance of `DockerImageDetails` (AzureML Python SDK >= 1.11) and provides all the information about the environment image such as the dockerfile, registry, and image name.

```
details = environment.get_image_details(workspace=ws)
```

To obtain the image details from an environment autosaved from the execution of a run, use the following code:

```
details = run.get_environment().get_image_details(workspace=ws)
```

Use existing environments

If you have an existing Conda environment on your local computer, then you can use the service to create an environment object. By using this strategy, you can reuse your local interactive environment on remote runs.

The following code creates an environment object from the existing Conda environment `mycondaenv`. It uses the `from_existing_conda_environment()` method.

```
myenv = Environment.from_existing_conda_environment(name="myenv",
                                                    conda_environment_name="mycondaenv")
```

An environment definition can be saved to a directory in an easily editable format with the `save_to_directory()` method. Once modified, a new environment can be instantiated by loading files from the directory.

```
# save the enviroment
myenv.save_to_directory(path="path-to-destination-directory", overwrite=False)
# modify the environment definition
newenv = Environment.load_from_directory(path="path-to-source-directory")
```

Implicitly use the default environment

If you don't specify an environment in your script run configuration before you submit the run, then a default environment is created for you.

```

from azureml.core import ScriptRunConfig, Experiment, Environment
# Create experiment
myexp = Experiment(workspace=ws, name = "environment-example")

# Attach training script and compute target to run config
src = ScriptRunConfig(source_directory=".", script="example.py", compute_target="local")

# Submit the run
run = myexp.submit(config=src)

# Show each step of run
run.wait_for_completion(show_output=True)

```

Add packages to an environment

Add packages to an environment by using Conda, pip, or private wheel files. Specify each package dependency by using the [CondaDependency](#) class. Add it to the environment's [PythonSection](#).

Conda and pip packages

If a package is available in a Conda package repository, then we recommend that you use the Conda installation rather than the pip installation. Conda packages typically come with prebuilt binaries that make installation more reliable.

The following example adds to the environment `myenv`. It adds version 1.17.0 of `numpy`. It also adds the `pillow` package. The example uses the [add_conda_package\(\)](#) method and the [add_pip_package\(\)](#) method, respectively.

```

from azureml.core.environment import Environment
from azureml.core.conda_dependencies import CondaDependencies

myenv = Environment(name="myenv")
conda_dep = CondaDependencies()

# Installs numpy version 1.17.0 conda package
conda_dep.add_conda_package("numpy==1.17.0")

# Installs pillow package
conda_dep.add_pip_package("pillow")

# Adds dependencies to PythonSection of myenv
myenv.python.conda_dependencies=conda_dep

```

IMPORTANT

If you use the same environment definition for another run, the Azure Machine Learning service reuses the cached image of your environment. If you create an environment with an unpinned package dependency, for example `numpy`, that environment will keep using the package version installed *at the time of environment creation*. Also, any future environment with matching definition will keep using the old version. For more information, see [Environment building, caching, and reuse](#).

Private Python packages

To use Python packages privately and securely without exposing them to the public internet, see the article [How to use private Python packages](#).

Manage environments

Manage environments so that you can update, track, and reuse them across compute targets and with other

users of the workspace.

Register environments

The environment is automatically registered with your workspace when you submit a run or deploy a web service. You can also manually register the environment by using the `register()` method. This operation makes the environment into an entity that's tracked and versioned in the cloud. The entity can be shared between workspace users.

The following code registers the `myenv` environment to the `ws` workspace.

```
myenv.register(workspace=ws)
```

When you use the environment for the first time in training or deployment, it's registered with the workspace. Then it's built and deployed on the compute target. The service caches the environments. Reusing a cached environment takes much less time than using a new service or one that has been updated.

Get existing environments

The `Environment` class offers methods that allow you to retrieve existing environments in your workspace. You can retrieve environments by name, as a list, or by a specific training run. This information is helpful for troubleshooting, auditing, and reproducibility.

View a list of environments

View the environments in your workspace by using the `Environment.list(workspace="workspace_name")` class. Then select an environment to reuse.

Get an environment by name

You can also get a specific environment by name and version. The following code uses the `get()` method to retrieve version `1` of the `myenv` environment on the `ws` workspace.

```
restored_environment = Environment.get(workspace=ws, name="myenv", version="1")
```

Train a run-specific environment

To get the environment that was used for a specific run after the training finishes, use the `get_environment()` method in the `Run` class.

```
from azureml.core import Run
Run.get_environment()
```

Update an existing environment

Say you change an existing environment, for example, by adding a Python package. This will take time to build as a new version of the environment is then created when you submit a run, deploy a model, or manually register the environment. The versioning allows you to view the environment's changes over time.

To update a Python package version in an existing environment, specify the version number for that package. If you don't use the exact version number, then Azure Machine Learning will reuse the existing environment with its original package versions.

Debug the image build

The following example uses the `build()` method to manually create an environment as a Docker image. It monitors the output logs from the image build by using `wait_for_completion()`. The built image then appears in the workspace's Azure Container Registry instance. This information is helpful for debugging.

```
from azureml.core import Image
build = env.build(workspace=ws)
build.wait_for_completion(show_output=True)
```

It is useful to first build images locally using the `build_local()` method. To build a docker image, set the optional parameter `useDocker=True`. To push the resulting image into the AzureML workspace container registry, set `pushImageToWorkspaceAcr=True`.

```
build = env.build_local(workspace=ws, useDocker=True, pushImageToWorkspaceAcr=True)
```

WARNING

Changing the order of dependencies or channels in an environment will result in a new environment and will require a new image build. In addition, calling the `build()` method for an existing image will update its dependencies if there are new versions.

Utilize adminless Azure Container Registry (ACR) with VNet

It is no longer required for users to have admin mode enabled on their workspace attached ACR in VNet scenarios. Ensure that the derived image build time on the compute is less than 1 hour to enable successful build. Once the image is pushed to the workspace ACR, this image can now only be accessed with a compute identity. For more information on set up, see [How to use managed identities with Azure Machine Learning](#).

Use environments for training

To submit a training run, you need to combine your environment, [compute target](#), and your training Python script into a run configuration. This configuration is a wrapper object that's used for submitting runs.

When you submit a training run, the building of a new environment can take several minutes. The duration depends on the size of the required dependencies. The environments are cached by the service. So as long as the environment definition remains unchanged, you incur the full setup time only once.

The following local script run example shows where you would use `ScriptRunConfig` as your wrapper object.

```
from azureml.core import ScriptRunConfig, Experiment
from azureml.core.environment import Environment

exp = Experiment(name="myexp", workspace = ws)
# Instantiate environment
myenv = Environment(name="myenv")

# Configure the ScriptRunConfig and specify the environment
src = ScriptRunConfig(source_directory=".", script="train.py", compute_target="local", environment=myenv)

# Submit run
run = exp.submit(src)
```

NOTE

To disable the run history or run snapshots, use the setting under `src.run_config.history`.

IMPORTANT

Use CPU SKUs for any image build on compute.

If you don't specify the environment in your run configuration, then the service creates a default environment when you submit your run.

Use environments for web service deployment

You can use environments when you deploy your model as a web service. This capability enables a reproducible, connected workflow. In this workflow, you can train, test, and deploy your model by using the same libraries in both your training compute and your inference compute.

If you are defining your own environment for web service deployment, you must list `azureml-defaults` with version $\geq 1.0.45$ as a pip dependency. This package contains the functionality that's needed to host the model as a web service.

To deploy a web service, combine the environment, inference compute, scoring script, and registered model in your deployment object, `deploy()`. For more information, see [How and where to deploy models](#).

In this example, assume that you've completed a training run. Now you want to deploy that model to Azure Container Instances. When you build the web service, the model and scoring files are mounted on the image, and the Azure Machine Learning inference stack is added to the image.

```
from azureml.core.model import InferenceConfig, Model
from azureml.core.webservice import AciWebservice, Webservice

# Register the model to deploy
model = run.register_model(model_name = "mymodel", model_path = "outputs/model.pkl")

# Combine scoring script & environment in Inference configuration
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)

# Set deployment configuration
deployment_config = AciWebservice.deploy_configuration(cpu_cores = 1, memory_gb = 1)

# Define the model, inference, & deployment configuration and web service name and location to deploy
service = Model.deploy(
    workspace = ws,
    name = "my_web_service",
    models = [model],
    inference_config = inference_config,
    deployment_config = deployment_config)
```

Notebooks

Code examples in this article are also included in the [using environments notebook](#).

To install a Conda environment as a kernel in a notebook, see [add a new Jupyter kernel](#).

[Deploy a model using a custom Docker base image](#) demonstrates how to deploy a model using a custom Docker base image.

This [example notebook](#) demonstrates how to deploy a Spark model as a web service.

Create and manage environments with the Azure CLI

For information on using the CLI v2, see [Manage environments with CLI v2](#).

Create and manage environments with Visual Studio Code

Using the Azure Machine Learning extension, you can create and manage environments in Visual Studio Code. For more information, see [manage Azure Machine Learning resources with the VS Code extension](#).

Next steps

- After you have a trained model, learn [how and where to deploy models](#).
- View the [Environment class SDK reference](#).

Use private Python packages with Azure Machine Learning

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azureml v1](#)

In this article, learn how to use private Python packages securely within Azure Machine Learning. Use cases for private Python packages include:

- You've developed a private package that you don't want to share publicly.
- You want to use a curated repository of packages stored within an enterprise firewall.

The recommended approach depends on whether you have few packages for a single Azure Machine Learning workspace, or an entire repository of packages for all workspaces within an organization.

The private packages are used through [Environment](#) class. Within an environment, you declare which Python packages to use, including private ones. To learn about environment in Azure Machine Learning in general, see [How to use environments](#).

Prerequisites

- The [Azure Machine Learning SDK for Python](#)
- An [Azure Machine Learning workspace](#)

Use small number of packages for development and testing

For a small number of private packages for a single workspace, use the static `Environment.add_private_pip_wheel()` method. This approach allows you to quickly add a private package to the workspace, and is well suited for development and testing purposes.

Point the file path argument to a local wheel file and run the `add_private_pip_wheel` command. The command returns a URL used to track the location of the package within your Workspace. Capture the storage URL and pass it the `add_pip_package()` method.

```
whl_url = Environment.add_private_pip_wheel(workspace=ws, file_path = "my-custom.whl")
myenv = Environment(name="myenv")
conda_dep = CondaDependencies()
conda_dep.add_pip_package(whl_url)
myenv.python.conda_dependencies=conda_dep
```

Internally, Azure Machine Learning service replaces the URL by secure SAS URL, so your wheel file is kept private and secure.

Use a repository of packages from Azure DevOps feed

If you're actively developing Python packages for your machine learning application, you can host them in an Azure DevOps repository as artifacts and publish them as a feed. This approach allows you to integrate the DevOps workflow for building packages with your Azure Machine Learning Workspace. To learn how to set up Python feeds using Azure DevOps, read [Get Started with Python Packages in Azure Artifacts](#)

This approach uses Personal Access Token to authenticate against the repository. The same approach is

applicable to other repositories with token based authentication, such as private GitHub repositories.

1. [Create a Personal Access Token \(PAT\)](#) for your Azure DevOps instance. Set the scope of the token to **Packaging > Read**.
2. Add the Azure DevOps URL and PAT as workspace properties, using the [Workspace.set_connection](#) method.

```
from azureml.core import Workspace

pat_token = input("Enter secret token")
ws = Workspace.from_config()
ws.set_connection(name="connection-1",
    category = "PythonFeed",
    target = "https://pkgs.dev.azure.com/<MY-ORG>",
    authType = "PAT",
    value = pat_token)
```

3. Create an Azure Machine Learning environment and add Python packages from the feed.

```
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies

env = Environment(name="my-env")
cd = CondaDependencies()
cd.add_pip_package("<my-package>")
cd.set_pip_option("--extra-index-url https://pkgs.dev.azure.com/<MY-ORG>/_packaging/<MY-
FEED>/pypi/simple")
env.python.conda_dependencies=cd
```

The environment is now ready to be used in training runs or web service endpoint deployments. When building the environment, Azure Machine Learning service uses the PAT to authenticate against the feed with the matching base URL.

Use a repository of packages from private storage

You can consume packages from an Azure storage account within your organization's firewall. The storage account can hold a curated set of packages or an internal mirror of publicly available packages.

To set up such private storage, see [Secure an Azure Machine Learning workspace and associated resources](#). You must also [place the Azure Container Registry \(ACR\) behind the VNet](#).

IMPORTANT

You must complete this step to be able to train or deploy models using the private package repository.

After completing these configurations, you can reference the packages in the Azure Machine Learning environment definition by their full URL in Azure blob storage.

Next steps

- Learn more about [enterprise security in Azure Machine Learning](#)

Where to save and write files for Azure Machine Learning experiments

9/22/2022 • 4 minutes to read • [Edit Online](#)

In this article, you learn where to save input files, and where to write output files from your experiments to prevent storage limit errors and experiment latency.

When launching training jobs on a [compute target](#), they are isolated from outside environments. The purpose of this design is to ensure reproducibility and portability of the experiment. If you run the same script twice, on the same or another compute target, you receive the same results. With this design, you can treat compute targets as stateless computation resources, each having no affinity to the jobs that are running after they are finished.

Where to save input files

Before you can initiate an experiment on a compute target or your local machine, you must ensure that the necessary files are available to that compute target, such as dependency files and data files your code needs to run.

Azure Machine Learning jobs training scripts by copying the entire source directory. If you have sensitive data that you don't want to upload, use a [ignore file](#) or don't include it in the source directory. Instead, access your data using a [datastore](#).

The storage limit for experiment snapshots is 300 MB and/or 2000 files.

For this reason, we recommend:

- **Storing your files in an Azure Machine Learning dataset.** This prevents experiment latency issues, and has the advantages of accessing data from a remote compute target, which means authentication and mounting are managed by Azure Machine Learning. Learn more about how to specify a dataset as your input data source in your training script with [Train with datasets](#).
- **If you only need a couple data files and dependency scripts and can't use a datastore,** place the files in the same folder directory as your training script. Specify this folder as your `source_directory` directly in your training script, or in the code that calls your training script.

Storage limits of experiment snapshots

For experiments, Azure Machine Learning automatically makes an experiment snapshot of your code based on the directory you suggest when you configure the job. This has a total limit of 300 MB and/or 2000 files. If you exceed this limit, you'll see the following error:

```
While attempting to take snapshot of .
Your total snapshot size exceeds the limit of 300.0 MB
```

To resolve this error, store your experiment files on a datastore. If you can't use a datastore, the below table offers possible alternate solutions.

EXPERIMENT DESCRIPTION

STORAGE LIMIT SOLUTION

EXPERIMENT DESCRIPTION	STORAGE LIMIT SOLUTION
Less than 2000 files & can't use a datastore	<p>Override snapshot size limit with</p> <pre>azureml._restclient.snapshots_client.SNAPSHOT_MAX_SIZE_BYTES = 'insert_desired_size'</pre> <p>This may take several minutes depending on the number and size of files.</p>
Must use specific script directory	<p>To prevent unnecessary files from being included in the snapshot, make an ignore file (<code>.gitignore</code> or <code>.amlignore</code>) in the directory. Add the files and directories to exclude to this file. For more information on the syntax to use inside this file, see syntax and patterns for <code>.gitignore</code>. The <code>.amlignore</code> file uses the same syntax. <i>If both files exist, the <code>.amlignore</code> file is used and the <code>.gitignore</code> file is unused.</i></p>
Pipeline	Use a different subdirectory for each step
Jupyter notebooks	Create a <code>.amlignore</code> file or move your notebook into a new, empty, subdirectory and run your code again.

Where to write files

Due to the isolation of training experiments, the changes to files that happen during jobs are not necessarily persisted outside of your environment. If your script modifies the files local to compute, the changes are not persisted for your next experiment job, and they're not propagated back to the client machine automatically. Therefore, the changes made during the first experiment job don't and shouldn't affect those in the second.

When writing changes, we recommend writing files to storage via an Azure Machine Learning dataset with an [OutputFileDatasetConfig object](#). See [how to create an OutputFileDatasetConfig](#).

Otherwise, write files to the `./outputs` and/or `./logs` folder.

IMPORTANT

Two folders, *outputs* and *logs*, receive special treatment by Azure Machine Learning. During training, when you write files to `./outputs` and `./logs` folders, the files will automatically upload to your job history, so that you have access to them once your job is finished.

- **For output such as status messages or scoring results**, write files to the `./outputs` folder, so they are persisted as artifacts in job history. Be mindful of the number and size of files written to this folder, as latency may occur when the contents are uploaded to job history. If latency is a concern, writing files to a datastore is recommended.
- **To save written file as logs in job history**, write files to `./logs` folder. The logs are uploaded in real time, so this method is suitable for streaming live updates from a remote job.

Next steps

- Learn more about [accessing data from storage](#).
- Learn more about [Create compute targets for model training and deployment](#)

Set up the Visual Studio Code Azure Machine Learning extension (preview)

9/22/2022 • 2 minutes to read • [Edit Online](#)

Learn how to set up the Azure Machine Learning Visual Studio Code extension for your machine learning workflows.

The Azure Machine Learning extension for VS Code provides a user interface to:

- Manage Azure Machine Learning resources (experiments, virtual machines, models, deployments, etc.)
- Develop locally using remote compute instances
- Train machine learning models
- Debug machine learning experiments locally
- Schema-based language support, autocompletion and diagnostics for specification file authoring

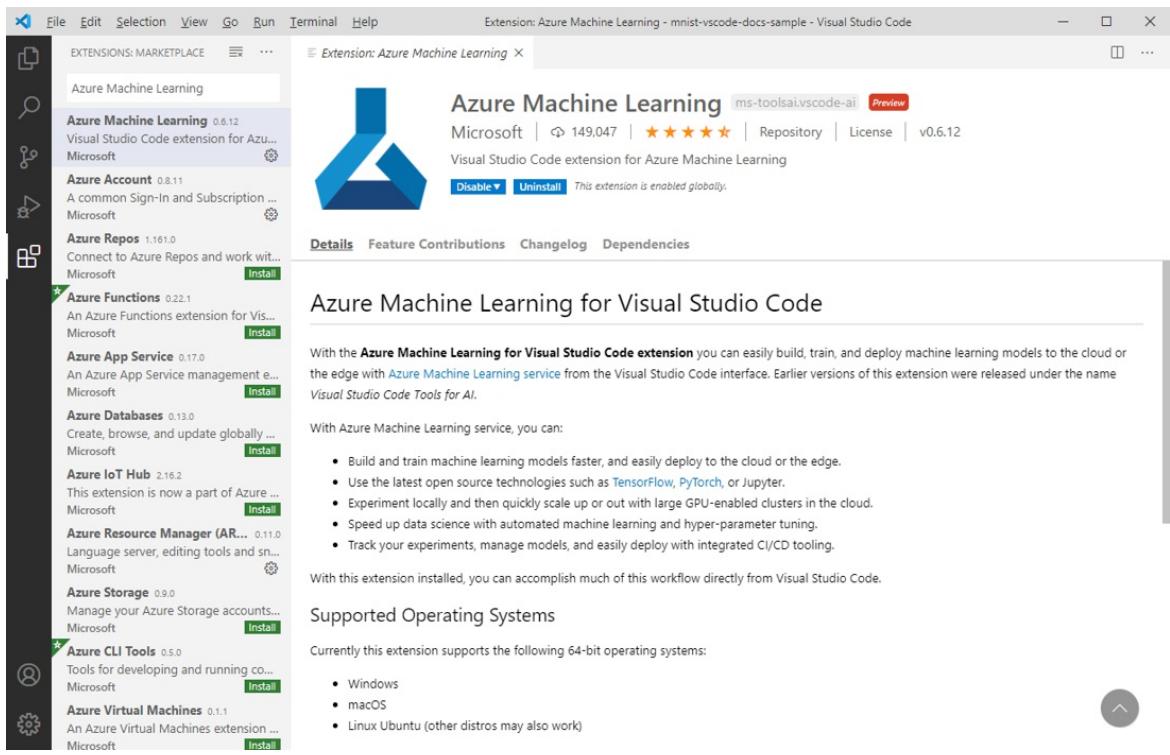
Prerequisites

- Azure subscription. If you don't have one, sign up to try the [free or paid version of Azure Machine Learning](#).
- Visual Studio Code. If you don't have it, [install it](#).
- [Python](#)
- (Optional) To create resources using the extension, you need to install the CLI (v2). For setup instructions, see [Install, set up, and use the CLI \(v2\)](#).
- Clone the community driven repository

```
git clone https://github.com/Azure/azureml-examples.git --depth 1
```

Install the extension

1. Open Visual Studio Code.
2. Select **Extensions** icon from the **Activity Bar** to open the Extensions view.
3. In the Extensions view search bar, type "Azure Machine Learning" and select the first extension.
4. Select **Install**.



NOTE

The Azure Machine Learning VS Code extension uses the CLI (v2) by default. To switch to the 1.0 CLI, set the `azureML.CLI Compatibility Mode` setting in Visual Studio Code to `1.0`. For more information on modifying your settings in Visual Studio, see the [user and workspace settings documentation](#).

Sign in to your Azure Account

In order to provision resources and job workloads on Azure, you have to sign in with your Azure account credentials. To assist with account management, Azure Machine Learning automatically installs the Azure Account extension. Visit the following site to [learn more about the Azure Account extension](#).

To sign into your Azure account, select the **Azure: Sign In** button in the bottom right corner on the Visual Studio Code status bar to start the sign in process.

Choose your default workspace

Choosing a default Azure Machine Learning workspace enables the following when authoring CLI (v2) YAML specification files:

- Schema validation
- Autocompletion
- Diagnostics

If you don't have a workspace, create one. For more information, see [manage Azure Machine Learning resources with the VS Code extension](#).

To choose your default workspace, select the **Set Azure ML Workspace** button on the Visual Studio Code status bar and follow the prompts to set your workspace.

Alternatively, use the `> Azure ML: Set Default Workspace` command in the command palette and follow the prompts to set your workspace.

Next Steps

- [Manage your Azure Machine Learning resources](#)
- [Develop on a remote compute instance locally](#)
- [Use a compute instances as a remote Jupyter server](#)
- [Train an image classification model using the Visual Studio Code extension](#)
- [Run and debug machine learning experiments locally](#)

Connect to an Azure Machine Learning compute instance in Visual Studio Code (preview)

9/22/2022 • 4 minutes to read • [Edit Online](#)

In this article, you'll learn how to connect to an Azure Machine Learning compute instance using Visual Studio Code.

An [Azure Machine Learning compute instance](#) is a fully managed cloud-based workstation for data scientists and provides management and enterprise readiness capabilities for IT administrators.

There are two ways you can connect to a compute instance from Visual Studio Code:

- Remote compute instance. This option provides you with a full-featured development environment for building your machine learning projects.
- Remote Jupyter Notebook server. This option allows you to set a compute instance as a remote Jupyter Notebook server.

Configure a remote compute instance

To configure a remote compute instance for development, you'll need a few prerequisites.

- Azure Machine Learning Visual Studio Code extension. For more information, see the [Azure Machine Learning Visual Studio Code Extension setup guide](#).
- Azure Machine Learning workspace. [Use the Azure Machine Learning Visual Studio Code extension to create a new workspace](#) if you don't already have one.
- Azure Machine Learning compute instance. [Use the Azure Machine Learning Visual Studio Code extension to create a new compute instance](#) if you don't have one.

IMPORTANT

To connect to a compute instance behind a firewall, see [use workspace behind a Firewall for Azure Machine Learning](#).

To connect to your remote compute instance:

- [Studio](#)
- [VS Code](#)

Navigate to [ml.azure.com](#)

IMPORTANT

In order to connect to your remote compute instance from Visual Studio Code, make sure that the account you're logged into in Azure Machine Learning studio is the same one you use in Visual Studio Code.

Compute

1. Select the **Compute** tab
2. In the *Application URI* column, select **VS Code** for the compute instance you want to connect to.

Name	State	Applications	Size	Created
test2	Running	JupyterLab Jupyter VS Code Terminal Notebook	STANDARD_DS12_V2	Jul 5,
test1	Stopped	JupyterLab Jupyter VS Code Terminal Notebook	STANDARD_DS12_V2	Jul 5,

Notebook

1. Select the **Notebook** tab
2. In the *Notebook* tab, select the file you want to edit.
3. Select Editors > **Edit in VS Code (preview)**.

A new window launches for your remote compute instance. When attempting to make a connection to a remote compute instance, the following tasks are taking place:

1. Authorization. Some checks are performed to make sure the user attempting to make a connection is authorized to use the compute instance.
2. VS Code Remote Server is installed on the compute instance.
3. A WebSocket connection is established for real-time interaction.

Once the connection is established, it's persisted. A token is issued at the start of the session which gets refreshed automatically to maintain the connection with your compute instance.

After you connect to your remote compute instance, use the editor to:

- Author and manage files on your remote compute instance or file share.
- Use the **VS Code integrated terminal** to run commands and applications on your remote compute instance.
- Debug your scripts and applications
- Use VS Code to manage your Git repositories

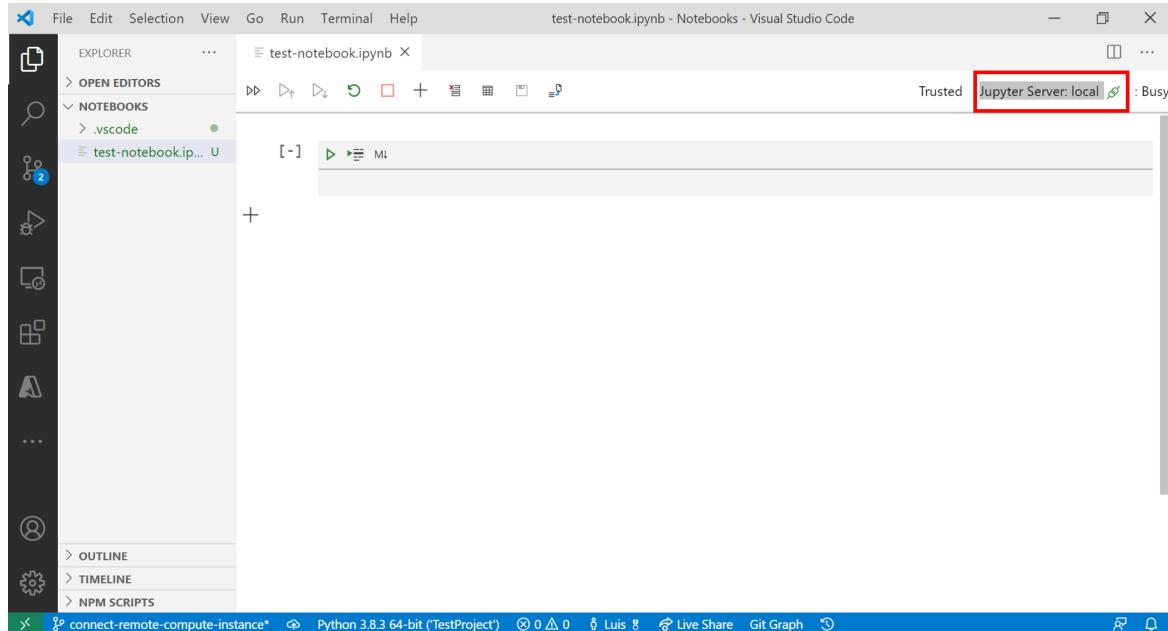
Configure compute instance as remote notebook server

In order to configure a compute instance as a remote Jupyter Notebook server you'll need a few prerequisites:

- Azure Machine Learning Visual Studio Code extension. For more information, see the [Azure Machine Learning Visual Studio Code Extension setup guide](#).
- Azure Machine Learning workspace. [Use the Azure Machine Learning Visual Studio Code extension to create a new workspace](#) if you don't already have one.

To connect to a compute instance:

1. Open a Jupyter Notebook in Visual Studio Code.
2. When the integrated notebook experience loads, select **Jupyter Server**.



Alternatively, you also use the command palette:

- a. Open the command palette by selecting **View > Command Palette** from the menu bar.
 - b. Enter into the text box **Azure ML: Connect to Compute instance Jupyter server**.
3. Choose **Azure ML Compute Instances** from the list of Jupyter server options.
 4. Select your subscription from the list of subscriptions. If you have previously configured your default Azure Machine Learning workspace, this step is skipped.
 5. Select your workspace.
 6. Select your compute instance from the list. If you don't have one, select **Create new Azure ML Compute Instance** and follow the prompts to create one.
 7. For the changes to take effect, you have to reload Visual Studio Code.
 8. Open a Jupyter Notebook and run a cell.

IMPORTANT

You **MUST** run a cell in order to establish the connection.

At this point, you can continue to run cells in your Jupyter Notebook.

TIP

You can also work with Python script files (.py) containing Jupyter-like code cells. For more information, see the [Visual Studio Code Python interactive documentation](#).

Next steps

Now that you've set up Visual Studio Code Remote, you can use a compute instance as remote compute from Visual Studio Code to [interactively debug your code](#).

[Tutorial: Train your first ML model](#) shows how to use a compute instance with an integrated notebook.

Git integration for Azure Machine Learning

9/22/2022 • 6 minutes to read • [Edit Online](#)

Git is a popular version control system that allows you to share and collaborate on your projects.

Azure Machine Learning fully supports Git repositories for tracking work - you can clone repositories directly onto your shared workspace file system, use Git on your local workstation, or use Git from a CI/CD pipeline.

When submitting a job to Azure Machine Learning, if source files are stored in a local git repository then information about the repo is tracked as part of the training process.

Since Azure Machine Learning tracks information from a local git repo, it isn't tied to any specific central repository. Your repository can be cloned from GitHub, GitLab, Bitbucket, Azure DevOps, or any other git-compatible service.

TIP

Use Visual Studio Code to interact with Git through a graphical user interface. To connect to an Azure Machine Learning remote compute instance using Visual Studio Code, see [Connect to an Azure Machine Learning compute instance in Visual Studio Code \(preview\)](#)

For more information on Visual Studio Code version control features, see [Using Version Control in VS Code](#) and [Working with GitHub in VS Code](#).

Clone Git repositories into your workspace file system

Azure Machine Learning provides a shared file system for all users in the workspace. To clone a Git repository into this file share, we recommend that you create a compute instance & [open a terminal](#). Once the terminal is opened, you have access to a full Git client and can clone and work with Git via the Git CLI experience.

We recommend that you clone the repository into your users directory so that others will not make collisions directly on your working branch.

TIP

There is a performance difference between cloning to the local file system of the compute instance or cloning to the mounted filesystem (mounted as the `~/cloudfiles/code` directory). In general, cloning to the local filesystem will have better performance than to the mounted filesystem. However, the local filesystem is lost if you delete and recreate the compute instance. The mounted filesystem is kept if you delete and recreate the compute instance.

You can clone any Git repository you can authenticate to (GitHub, Azure Repos, BitBucket, etc.)

For more information about cloning, see the guide on [how to use Git CLI](#).

Authenticate your Git Account with SSH

Generate a new SSH key

1. [Open the terminal window](#) in the Azure Machine Learning Notebook Tab.
2. Paste the text below, substituting in your email address.

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private rsa key pair.
```

3. When you're prompted to "Enter a file in which to save the key" press Enter. This accepts the default file location.
4. Verify that the default location is '/home/azureuser/.ssh' and press enter. Otherwise specify the location '/home/azureuser/.ssh'.

TIP

Make sure the SSH key is saved in '/home/azureuser/.ssh'. This file is saved on the compute instance is only accessible by the owner of the Compute Instance

```
> Enter a file in which to save the key (/home/azureuser/.ssh/id_rsa): [Press enter]
```

5. At the prompt, type a secure passphrase. We recommend you add a passphrase to your SSH key for added security

```
> Enter passphrase (empty for no passphrase): [Type a passphrase]  
> Enter same passphrase again: [Type passphrase again]
```

Add the public key to Git Account

1. In your terminal window, copy the contents of your public key file. If you renamed the key, replace id_rsa.pub with the public key file name.

```
cat ~/.ssh/id_rsa.pub
```

TIP

Copy and Paste in Terminal

- Windows: `ctrl-Insert` to copy and use `ctrl-Shift-v` or `Shift-Insert` to paste.
- Mac OS: `Cmd-c` to copy and `Cmd-v` to paste.
- FireFox/IE may not support clipboard permissions properly.

2. Select and copy the key output in the clipboard.

- [GitHub](#)
- [GitLab](#)
- [Azure DevOps](#) Start at Step 2.
- [BitBucket](#). Start at Step 4.

Clone the Git repository with SSH

1. Copy the SSH Git clone URL from the Git repo.

2. Paste the url into the `git clone` command below, to use your SSH Git repo URL. This will look something like:

```
git clone git@example.com:GitUser/azureml-example.git
Cloning into 'azureml-example'...
```

You will see a response like:

```
The authenticity of host 'example.com (192.30.255.112)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.255.112' (RSA) to the list of known hosts.
```

SSH may display the server's SSH fingerprint and ask you to verify it. You should verify that the displayed fingerprint matches one of the fingerprints in the SSH public keys page.

SSH displays this fingerprint when it connects to an unknown host to protect you from [man-in-the-middle attacks](#). Once you accept the host's fingerprint, SSH will not prompt you again unless the fingerprint changes.

3. When you are asked if you want to continue connecting, type `yes`. Git will clone the repo and set up the origin remote to connect with SSH for future Git commands.

Track code that comes from Git repositories

When you submit a training job from the Python SDK or Machine Learning CLI, the files needed to train the model are uploaded to your workspace. If the `git` command is available on your development environment, the upload process uses it to check if the files are stored in a git repository. If so, then information from your git repository is also uploaded as part of the training job. This information is stored in the following properties for the training job:

PROPERTY	GIT COMMAND USED TO GET THE VALUE	DESCRIPTION
<code>azureml.git.repository_uri</code>	<code>git ls-remote --get-url</code>	The URI that your repository was cloned from.
<code>mlflow.source.git.repoURL</code>	<code>git ls-remote --get-url</code>	The URI that your repository was cloned from.
<code>azureml.git.branch</code>	<code>git symbolic-ref --short HEAD</code>	The active branch when the job was submitted.
<code>mlflow.source.git.branch</code>	<code>git symbolic-ref --short HEAD</code>	The active branch when the job was submitted.
<code>azureml.git.commit</code>	<code>git rev-parse HEAD</code>	The commit hash of the code that was submitted for the job.
<code>mlflow.source.git.commit</code>	<code>git rev-parse HEAD</code>	The commit hash of the code that was submitted for the job.
<code>azureml.git.dirty</code>	<code>git status --porcelain .</code>	<code>True</code> , if the branch/commit is dirty; otherwise, <code>false</code> .

This information is sent for jobs that use an estimator, machine learning pipeline, or script run.

If your training files are not located in a git repository on your development environment, or the `git` command is not available, then no git-related information is tracked.

TIP

To check if the git command is available on your development environment, open a shell session, command prompt, PowerShell or other command line interface and type the following command:

```
git --version
```

If installed, and in the path, you receive a response similar to `git version 2.4.1`. For more information on installing git on your development environment, see the [Git website](#).

View the logged information

The git information is stored in the properties for a training job. You can view this information using the Azure portal or Python SDK.

Azure portal

1. From the [studio portal](#), select your workspace.
2. Select **Jobs**, and then select one of your experiments.
3. Select one of the jobs from the **Display name** column.
4. Select **Outputs + logs**, and then expand the **logs** and **azureml** entries. Select the link that begins with `####_azure`.

The logged information contains text similar to the following JSON:

```
"properties": {  
    "_azureml.ComputeTargetType": "batchai",  
    "ContentSnapshotId": "5ca66406-cbac-4d7d-bc95-f5a51dd3e57e",  
    "azureml.git.repository_uri": "git@github.com:azure/machinelearningnotebooks",  
    "mlflow.source.git.repoURL": "git@github.com:azure/machinelearningnotebooks",  
    "azureml.git.branch": "master",  
    "mlflow.source.git.branch": "master",  
    "azureml.git.commit": "4d2b93784676893f8e346d5f0b9fb894a9cf0742",  
    "mlflow.source.git.commit": "4d2b93784676893f8e346d5f0b9fb894a9cf0742",  
    "azureml.git.dirty": "True",  
    "AzureML.DerivedImageName": "azureml/azureml_9d3568242c6bfef9631879915768deaf",  
    "ProcessInfoFile": "azureml-logs/process_info.json",  
    "ProcessStatusFile": "azureml-logs/process_status.json"  
}
```

Next steps

- [Use compute targets for model training](#)

Manage Azure Machine Learning environments with the CLI (v2)

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Azure Machine Learning environments define the execution environments for your jobs or deployments and encapsulate the dependencies for your code. Azure ML uses the environment specification to create the Docker container that your training or scoring code runs in on the specified compute target. You can define an environment from a conda specification, Docker image, or Docker build context.

In this article, learn how to create and manage Azure ML environments using the CLI (v2).

Prerequisites

- To use the CLI, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- [Install and set up the Azure CLI extension for Machine Learning](#)

TIP

For a full-featured development environment, use Visual Studio Code and the [Azure Machine Learning extension](#) to [manage Azure Machine Learning resources](#) and [train machine learning models](#).

Clone examples repository

To run the training examples, first clone the examples repository and change into the `cli` directory:

```
git clone --depth 1 https://github.com/Azure/azureml-examples
cd azureml-examples/cli
```

Note that `--depth 1` clones only the latest commit to the repository which reduces time to complete the operation.

Curated environments

There are two types of environments in Azure ML: curated and custom environments. Curated environments are predefined environments containing popular ML frameworks and tooling. Custom environments are user-defined and can be created via `az ml environment create`.

Curated environments are provided by Azure ML and are available in your workspace by default. Azure ML routinely updates these environments with the latest framework version releases and maintains them for bug fixes and security patches. They are backed by cached Docker images, which reduces job preparation cost and model deployment time.

You can use these curated environments out of the box for training or deployment by referencing a specific environment using the `azureml:<curated-environment-name>:<version>` or `azureml:<curated-environment-name>@latest` syntax. You can also use them as reference for your own custom environments by modifying the Dockerfiles that back these curated environments.

You can see the set of available curated environments in the Azure ML studio UI, or by using the CLI (v2) via

```
az ml environments list.
```

Create an environment

You can define an environment from a conda specification, Docker image, or Docker build context. Configure the environment using a YAML specification file and create the environment using the following CLI command:

```
az ml environment create --file my_environment.yml
```

For the YAML reference documentation for Azure ML environments, see [CLI \(v2\) environment YAML schema](#).

Create an environment from a Docker image

To define an environment from a Docker image, provide the image URI of the image hosted in a registry such as Docker Hub or Azure Container Registry.

The following example is a YAML specification file for an environment defined from a Docker image. An image from the official PyTorch repository on Docker Hub is specified via the `image` property in the YAML file.

```
$schema: https://azureschemas.azureedge.net/latest/environment.schema.json
name: docker-image-example
image: pytorch/pytorch:latest
description: Environment created from a Docker image.
```

To create the environment:

```
az ml environment create --file assets/environment/docker-image.yml
```

TIP

Azure ML maintains a set of CPU and GPU Ubuntu Linux-based base images with common system dependencies. For example, the GPU images contain Miniconda, OpenMPI, CUDA, cuDNN, and NCCL. You can use these images for your environments, or use their corresponding Dockerfiles as reference when building your own custom images.

For the set of base images and their corresponding Dockerfiles, see the [AzureML-Containers repo](#).

Create an environment from a Docker build context

Instead of defining an environment from a prebuilt image, you can also define one from a Docker build context. To do so, specify the directory that will serve as the build context. This directory should contain a Dockerfile and any other files needed to build the image.

The following example is a YAML specification file for an environment defined from a build context. The local path to the build context folder is specified in the `build.path` field, and the relative path to the Dockerfile within that build context folder is specified in the `build.dockerfile_path` field. If `build.dockerfile_path` is omitted in the YAML file, Azure ML will look for a Dockerfile named `Dockerfile` at the root of the build context.

In this example, the build context contains a Dockerfile named `Dockerfile` and a `requirements.txt` file that is referenced within the Dockerfile for installing Python packages.

```
$schema: https://azureschemas.azureedge.net/latest/environment.schema.json
name: docker-context-example
build:
  path: docker-contexts/python-and-pip
```

To create the environment:

```
az ml environment create --file assets/environment/docker-context.yml
```

Azure ML will start building the image from the build context when the environment is created. You can monitor the status of the build and view the build logs in the studio UI.

Create an environment from a conda specification

You can define an environment using a standard conda YAML configuration file that includes the dependencies for the conda environment. See [Creating an environment manually](#) for information on this standard format.

You must also specify a base Docker image for this environment. Azure ML will build the conda environment on top of the Docker image provided. If you install some Python dependencies in your Docker image, those packages will not exist in the execution environment thus causing runtime failures. By default, Azure ML will build a Conda environment with dependencies you specified, and will execute the job in that environment instead of using any Python libraries that you installed on the base image.

The following example is a YAML specification file for an environment defined from a conda specification. Here the relative path to the conda file from the Azure ML environment YAML file is specified via the `conda_file` property. You can alternatively define the conda specification inline using the `conda_file` property, rather than defining it in a separate file.

```
$schema: https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: docker-image-plus-conda-example
image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04
conda_file: conda-yamls/pydata.yml
description: Environment created from a Docker image plus Conda environment.
```

To create the environment:

```
az ml environment create --file assets/environment/docker-image-plus-conda.yml
```

Azure ML will build the final Docker image from this environment specification when the environment is used in a job or deployment. You can also manually trigger a build of the environment in the studio UI.

Manage environments

The CLI (v2) provides a set of commands under `az ml environment` for managing the lifecycle of your Azure ML environment assets.

List

List all the environments in your workspace:

```
az ml environment list
```

List all the environment versions under a given name:

```
az ml environment list --name docker-image-example
```

Show

Get the details of a specific environment:

```
az ml environment list --name docker-image-example --version 1
```

Update

Update mutable properties of a specific environment:

```
az ml environment update --name docker-image-example --version 1 --set description="This is an updated description."
```

IMPORTANT

For environments, only `description` and `tags` can be updated. All other properties are immutable; if you need to change any of those properties you should create a new version of the environment.

Archive and restore

Archiving an environment will hide it by default from list queries (`az ml environment list`). You can still continue to reference and use an archived environment in your workflows. You can archive either an environment container or a specific environment version.

Archiving an environment container will archive all versions of the environment under that given name. If you create a new environment version under an archived environment container, that new version will automatically be set as archived as well.

Archive an environment container:

```
az ml environment archive --name docker-image-example
```

Archive a specific environment version:

```
az ml environment archive --name docker-image-example --version 1
```

You can restore an archived environment to no longer hide it from list queries.

If an entire environment container is archived, you can restore that archived container. You cannot restore only a specific environment version if the entire environment container is archived - you will need to restore the entire container.

Restore an environment container:

```
az ml environment restore --name docker-image-example
```

If only individual environment version(s) within an environment container are archived, you can restore those individual version(s).

Restore a specific environment version:

```
az ml environment restore --name docker-image-example --version 1
```

Use environments for training

To use an environment for a training job, specify the `environment` field of the job YAML configuration. You can

either reference an existing registered Azure ML environment via

```
environment: azureml:<environment-name>:<environment-version>
```

 or

```
environment: azureml:<environment-name>@latest
```

 (to reference the latest version of an environment), or define an environment specification inline. If defining an environment inline, do not specify the `name` and `version` fields, as these environments are treated as "unregistered" environments and are not tracked in your environment asset registry.

When you submit a training job, the building of a new environment can take several minutes. The duration depends on the size of the required dependencies. The environments are cached by the service. So as long as the environment definition remains unchanged, you incur the full setup time only once.

For more information on how to use environments in jobs, see [Train models with the CLI \(v2\)](#).

Use environments for model deployments

You can also use environments for your model deployments for both online and batch scoring. To do so, specify the `environment` field in the deployment YAML configuration.

For more information on how to use environments in deployments, see [Deploy and score a machine learning model by using a managed online endpoint](#).

Next steps

- [Train models \(create jobs\) with the CLI \(v2\)](#)
- [Deploy and score a machine learning model by using a managed online endpoint](#)
- [Environment YAML schema reference](#)

Troubleshooting environment image builds using troubleshooting log error messages

9/22/2022 • 14 minutes to read • [Edit Online](#)

In this article, learn how to troubleshoot common problems you may encounter with environment image builds.

Azure Machine Learning environments

Azure Machine Learning environments are an encapsulation of the environment where your machine learning training happens. They specify the base docker image, Python packages, and software settings around your training and scoring scripts. Environments are managed and versioned assets within your Machine Learning workspace that enable reproducible, auditable, and portable machine learning workflows across various compute targets.

Types of environments

Environments can broadly be divided into three categories: curated, user-managed, and system-managed.

Curated environments are pre-created environments that are managed by Azure Machine Learning (AzureML) and are available by default in every workspace provisioned.¹

Intended to be used as is, they contain collections of Python packages and settings to help you get started with various machine learning frameworks. These pre-created environments also allow for faster deployment time.

In user-managed environments, you're responsible for setting up your environment and installing every package that your training script needs on the compute target. Also be sure to include any dependencies needed for model deployment. These types of environments are represented by two subtypes, BYOC (bring your own container) – a Docker image user brings to AzureML and Docker build context based environment where AzureML materializes the image from the user provided content.

You use system-managed environments when you want conda to manage the Python environment for you. A new isolated conda environment is materialized from your conda specification on top of a base Docker image. By default, common properties are added to the derived image. Note that environment isolation implies that Python dependencies installed in the base image won't be available in the derived image.

Create and manage environments

You can create and manage environments from clients like AzureML Python SDK, AzureML CLI, AzureML Studio UI, VS code extension.

"Anonymous" environments are automatically registered in your workspace when you submit an experiment without registering or referencing an already existing environment. They won't be listed but may be retrieved by version or label.

AzureML builds environment definitions into Docker images. It also caches the environments in Azure Container Registry associated with your AzureML Workspace so they can be reused in subsequent training jobs and service endpoint deployments. Multiple environments with the same definition may result in the same image, so the cached image will be reused. Running a training script remotely requires the creation of a Docker image.

Reproducibility and vulnerabilities

Over time vulnerabilities are discovered and Docker images that correspond to AzureML environments may be flagged by the scanning tools. Updates for AzureML based images are released regularly, with a commitment of no unpatched vulnerabilities older than 30 days in the latest version of the image. It's your responsibility to evaluate the threat and address vulnerabilities in environments. Not all the vulnerabilities are exploitable, so you need to use your judgment when choosing between reproducibility and resolving vulnerabilities.

IMPORTANT

There's no guarantee that the same set of python dependencies will be materialized with an image rebuild or for a new environment with the same set of Python dependencies.

Environment definition problems

Environment name issues

"Curated prefix not allowed"

Terminology:

"Curated": environments Microsoft creates and maintains.

"Custom": environments you create and maintain.

- The name of your custom environment uses terms reserved only for curated environments
- Don't start your environment name with *Microsoft* or *AzureML*--these prefixes are reserved for curated environments
- To customize a curated environment, you must clone and rename the environment
- For more information about proper curated environment usage, see [create and manage reusable environments](#)

"Environment name is too long"

- Environment names can be up to 255 characters in length
- Consider renaming and shortening your environment name

Docker issues

To create a new environment, you must use one of the following approaches:

1. Base image
 - Provide base image name, repository from which to pull it, credentials if needed
 - Provide a conda specification
2. Base Dockerfile (V1 only, Deprecated)
 - Provide a Dockerfile
 - Provide a conda specification
3. Docker build context
 - Provide the location of the build context (URL)
 - The build context must contain at least a Dockerfile, but may contain other files as well

"Missing Docker definition"

- An environment has a `DockerSection` that must be populated with either a base image, base Dockerfile, or build context
- This section configures settings related to the final Docker image built to the specifications of the environment and whether to use Docker containers to build the environment
- See [DockerSection](#)

"Missing Docker build context location"

- If you're specifying a Docker build context as part of your environment build, you must provide the path of the build context directory
- See [BuildContext](#)

"Too many Docker options"

Only one of the following options can be specified:

V1

- `base_image`
- `base_dockerfile`
- `build_context`
- See [DockerSection](#)

V2

- `image`
- `build`
- See [azure.ai.ml.entities.Environment](#)

"Missing Docker option"

V1

- You must specify one of: base image, base Dockerfile, or build context

V2:

- You must specify one of: image or build context

"Container registry credentials missing either username or password"

- To access the base image in the container registry specified, you must provide both a username and password. One is missing.
- Note that providing credentials in this way is deprecated. For the current method of providing credentials, see the *secrets in base image registry* section.

"Multiple credentials for base image registry"

- When specifying credentials for a base image registry, you must specify only one set of credentials.
- The following authentication types are currently supported:
 - Basic (username/password)
 - Registry identity (clientid/resourceid)
- If you're using workspace connections to specify credentials, [delete one of the connections](#)
- If you've specified credentials directly in your environment definition, choose either username/password or registry identity to use, and set the other credentials you won't use to `null`
 - Specifying credentials in this way is deprecated. It's recommended that you use workspace connections. See *secrets in base image registry* below

"Secrets in base image registry"

- If you specify a base image in your `DockerSection`, you must specify the registry address from which the image will be pulled, and credentials to authenticate to the registry, if needed.
- Historically, credentials have been specified in the environment definition. However, this isn't secure and should be avoided.
- Users should set credentials using workspace connections. For instructions on how to do this, see [set_connection](#)

"Deprecated Docker attribute"

- The following `DockerSection` attributes are deprecated:

- `enabled`
- `arguments`
- `shared_volumes`
- `gpu_support`
 - Azure Machine Learning now automatically detects and uses NVIDIA Docker extension when available.
- `smh_size`
- Use [DockerConfiguration](#) instead
- See [DockerSection deprecated variables](#)

"Dockerfile length over limit"

- The specified Dockerfile can't exceed the maximum Dockerfile size of 100KB
- Consider shortening your Dockerfile to get it under this limit

Docker build context issues

"Missing Dockerfile path"

- In the Docker build context, a Dockerfile path must be specified
- This is the path to the Dockerfile relative to the root of Docker build context directory
- See [Build Context class](#)

"Not allowed to specify attribute with Docker build context"

- If a Docker build context is specified, then the following items can't also be specified in the environment definition:
 - Environment variables
 - Conda dependencies
 - R
 - Spark

"Location type not supported/Unknown location type"

- The following are accepted location types:
 - Git
 - Git URLs can be provided to AzureML, but images can't yet be built using them. Use a storage account until builds have Git support
 - [How to use git repository as build context](#)
 - Storage account

"Invalid location"

- The specified location of the Docker build context is invalid
- If the build context is stored in a git repository, the path of the build context must be specified as a git URL
- If the build context is stored in a storage account, the path of the build context must be specified as
 - `https://storage-account.blob.core.windows.net/container/path/`

Base image issues

"Base image is deprecated"

- The following base images are deprecated:
 - `azureml/base`
 - `azureml/base-gpu`
 - `azureml/base-lite`
 - `azureml/intelmpi2018.3-cuda10.0-cudnn7-ubuntu16.04`
 - `azureml/intelmpi2018.3-cuda9.0-cudnn7-ubuntu16.04`
 - `azureml/intelmpi2018.3-ubuntu16.04`

- `azureml/o16n-base/python-slim`
- `azureml/openmpi3.1.2-cuda10.0-cudnn7-ubuntu16.04`
- `azureml/openmpi3.1.2-ubuntu16.04`
- `azureml/openmpi3.1.2-cuda10.0-cudnn7-ubuntu18.04`
- `azureml/openmpi3.1.2-cuda10.1-cudnn7-ubuntu18.04`
- `azureml/openmpi3.1.2-cuda10.2-cudnn7-ubuntu18.04`
- `azureml/openmpi3.1.2-cuda10.2-cudnn8-ubuntu18.04`

- AzureML can't provide troubleshooting support for failed builds with deprecated images.
- Deprecated images are also at risk for vulnerabilities since they're no longer updated or maintained. It's best to use newer, non-deprecated versions.

"No tag or digest"

- For the environment to be reproducible, one of the following must be included on a provided base image:
 - Version tag
 - Digest
- See [image with immutable identifier](#)

Environment variable issues

"Misplaced runtime variables"

- An environment definition shouldn't contain runtime variables
- Use the `environment_variables` attribute on the [RunConfiguration object](#) instead

Python issues

"Python section missing"

V7

- An environment definition must have a Python section
- Conda dependencies are specified in this section, and Python (along with its version) should be one of them

```
"python": {
    "baseCondaEnvironment": null,
    "condaDependencies": {
        "channels": [
            "anaconda",
            "conda-forge"
        ],
        "dependencies": [
            "python=3.8"
        ],
    },
    "condaDependenciesFile": null,
    "interpreterPath": "python",
    "userManagedDependencies": false
}
```

- See [PythonSection class](#)

"Python version missing"

V7

- A Python version must be specified in the environment definition
- A Python version can be added by adding Python as a conda package, specifying the version (this is specific to SDK V1):

```
from azureml.core.environment import CondaDependencies

myenv = Environment(name="myenv")
conda_dep = CondaDependencies()
conda_dep.add_conda_package("python==3.8")
```

- See [Add conda package](#)

"Multiple Python versions"

- Only one Python version can be specified in the environment definition

"Python version not supported"

- The Python version provided in the environment definition isn't supported
- Consider using a newer version of Python
- See [Python versions](#) and [Python end-of-life dates](#)

"Python version not recommended"

- The Python version used in the environment definition is deprecated, and its use should be avoided
- Consider using a newer version of Python as the specified version will eventually unsupported
- See [Python versions](#) and [Python end-of-life dates](#)

"Failed to validate Python version"

- The provided Python version may have been formatted improperly or specified with incorrect syntax
- See [conda package pinning](#)

Conda issues

"Missing conda dependencies"

- The [environment definition](#) has a [PythonSection](#) that contains a `user_managed_dependencies` bool and a `conda_dependencies` object
- If `user_managed_dependencies` is set to `True`, you're responsible for ensuring that all the necessary packages are available in the Python environment in which you choose to run the script
- If `user_managed_dependencies` is set to `False` (the default), Azure ML will create a Python environment for you based on `conda_dependencies`. The environment is built once and is reused as long as the conda dependencies remain unchanged
- You'll receive a "*missing conda dependencies*" error when `user_managed_dependencies` is set to `False` and you haven't provided a conda specification.
- See [how to create a conda file manually](#)
- See [CondaDependencies class](#)
- See [how to set a conda specification on the environment definition](#)

"Invalid conda dependencies"

- Make sure the conda dependencies specified in your conda specification are formatted correctly
- See [how to create a conda file manually](#)

"Missing conda channels"

- If no conda channels are specified, conda will use defaults that might change
- For reproducibility of your environment, specify channels from which to pull dependencies
- See [how to manage conda channels](#) for more information

"Base conda environment not recommended"

- Partial environment updates can lead to dependency conflicts and/or unexpected runtime errors, so the use of base conda environments isn't recommended
- Instead, specify all packages needed for your environment in the `conda_dependencies` section of your environment definition

- See [from_conda_specification](#)
- See [CondaDependencies class](#)
- If you're using V2, add a conda specification to your [build context](#)

"Unpinned dependencies"

- For reproducibility, specify dependency versions for the packages in your conda specification
- If versions aren't specified, there's a chance that the conda or pip package resolver will choose a different version of a package on subsequent builds of an environment. This can lead to unexpected errors and incorrect behavior
- See [conda package pinning](#)

Pip issues

"Pip not specified"

- For reproducibility, pip should be specified as a dependency in your conda specification, and it should be pinned
- See [how to set a conda dependency](#)

"Pip not pinned"

- For reproducibility, specify the pip resolver version in your conda dependencies
- If the pip version isn't specified, there's a chance different versions of pip will be used on subsequent image builds on the environment
 - This could cause the build to fail if the different pip versions resolve your packages differently
 - To avoid this and to achieve reproducibility of your environment, specify the pip version
- See [conda package pinning](#)
- See [how to set pip as a dependency](#)

Deprecated environment property issues

"R section is deprecated"

- The Azure Machine Learning SDK for R will be deprecated by the end of 2021 to make way for an improved R training and deployment experience using Azure Machine Learning CLI 2.0
- See the [samples repository](#) to get started with the Public Preview edition of the 2.0 CLI

Image build problems

Miscellaneous issues

"Build log unavailable"

- Build logs are optional and not available for all environments since the image might already exist

"ACR unreachable"

- There was a failure communicating with the workspace's container registry
- If your scenario involves a VNet, you may need to build images using a compute cluster
- See [secure a workspace using virtual networks](#)

Docker pull issues

"Failed to pull Docker image"

- Possible issues:
 - The path name to the container registry might not be resolving correctly
 - For a registry `my-registry.io` and image `test/image` with tag `3.2`, a valid image path would be `my-registry.io/test/image:3.2`
 - See [registry path documentation](#)
 - If a container registry behind a virtual network is using a private endpoint in an [unsupported region](#), configure the container registry by using the service endpoint (public access) from the portal and

retry

- After you put the container registry behind a virtual network, run the [Azure Resource Manager template](#) so the workspace can communicate with the container registry instance
- The image you're trying to reference doesn't exist in the container registry you specified
 - Check that the correct tag is used and that `user_managed_dependencies` is set to `True`. Setting `user_managed_dependencies` to `True` disables conda and uses the user's installed packages.
- You haven't provided credentials for a private registry you're trying to pull the image from, or the provided credentials are incorrect
 - Set [workspace connections](#) for the container registry if needed

Conda issues during build

"Bad spec"

- Failed to create or update the conda environment due to an invalid package specification
 - See [package match specifications](#)
 - See [how to create a conda file manually](#)

"Communications error"

- Failed to communicate with a conda channel or package repository
- Retrying the image build may work if the issue is transient

"Compile error"

- Failed to build a package required for the conda environment
- Another version of the failing package may work. If it doesn't, review the image build log, hunt for a solution, and update the environment definition.

"Missing command"

- Failed to build a package required for the conda environment due to a missing command
- Identify the missing command from the image build log, determine how to add it to your image, and then update the environment definition.

"Conda timeout"

- Failed to create or update the conda environment because it took too long
- Consider removing unnecessary packages and pinning specific versions
- See [understanding and improving conda's performance](#)

"Out of memory"

- Failed to create or update the conda environment due to insufficient memory
- Consider removing unnecessary packages and pinning specific versions
- See [understanding and improving conda's performance](#)

"Package not found"

- One or more packages specified in your conda specification couldn't be found
- Ensure that all packages you've specified exist, and can be found using the channels you've specified in your conda specification
- If you don't specify conda channels, conda will use defaults that are subject to change
 - For reproducibility, specify channels from which to pull dependencies
- See [managing channels](#)

"Missing Python module"

- Check the Python modules specified in your environment definition and correct any misspellings or incorrect pinned versions.

"No matching distribution"

- Failed to find Python package matching a specified distribution

- Search for the distribution you're looking for and ensure it exists: [pypi](#)

"Cannot build mpi4py"

- Failed to build wheel for mpi4py
- Review and update your build environment or use a different installation method
- See [mpi4py installation](#)

"Interactive auth was attempted"

- Failed to create or update the conda environment because pip attempted interactive authentication
- Instead, provide authentication via [workspace connection](#)

"Forbidden blob"

- Failed to create or update the conda environment because a blob contained in the associated storage account was inaccessible
- Either open up permissions on the blob or add/replace the SAS token in the URL

"Horovod build"

- Failed to create or update the conda environment because horovod failed to build
- See [horovod installation](#)

"Conda command not found"

- Failed to create or update the conda environment because the conda command is missing
- For system-managed environments, conda should be in the path in order to create the user's environment from the provided conda specification

"Incompatible Python version"

- Failed to create or update the conda environment because a package specified in the conda environment isn't compatible with the specified python version
- Update the Python version or use a different version of the package

"Conda bare redirection"

- Failed to create or update the conda environment because a package was specified on the command line using ">" or "<" without using quotes. Consider adding quotes around the package specification

Pip issues during build

"Failed to install packages"

- Failed to install Python packages
- Review the image build log for more information on this error

"Cannot uninstall package"

- Pip failed to uninstall a Python package that was installed via the OS's package manager
- Consider creating a separate environment using conda instead

Create datastores

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO: Python SDK azure-ai-ml v2 (preview)

APPLIES TO: Azure CLI ml extension v2 (current)

In this article, learn how to connect to data storage services on Azure with Azure Machine Learning datastores.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- The [Azure Machine Learning SDK for Python](#).
- An Azure Machine Learning workspace.

NOTE

Azure Machine Learning datastores do **not** create the underlying storage accounts, rather they link an **existing** storage account for use in Azure Machine Learning. It is not a requirement to use Azure Machine Learning datastores - you can use storage URIs directly assuming you have access to the underlying data.

Create an Azure Blob datastore

- [CLI: Identity-based access](#)
- [CLI: Account key](#)
- [CLI: SAS](#)
- [Python SDK: Identity-based access](#)
- [Python SDK: Account key](#)
- [Python SDK: SAS](#)

Create the following YAML file (updating the values):

```
# my_blob_datastore.yml
$schema: https://azureschemas.azureedge.net/latest/azureBlob.schema.json
name: my_blob_ds # add name of your datastore here
type: azure_blob
description: here is a description # add a description of your datastore here
account_name: my_account_name # add storage account name here
container_name: my_container_name # add storage container name here
```

Create the Azure Machine Learning datastore in the CLI:

```
az ml datastore create --file my_blob_datastore.yml
```

Create an Azure Data Lake Gen2 datastore

- [CLI: Identity-based access](#)

- [CLI: Service principal](#)
- [Python SDK: Identity-based access](#)
- [Python SDK: Service principal](#)

Create the following YAML file (updating the values):

```
# my_adls_datastore.yml
$schema: https://azurermschemas.azureedge.net/latest/azureDataLakeGen2.schema.json
name: adls_gen2_credless_example
type: azure_data_lake_gen2
description: Credential-less datastore pointing to an Azure Data Lake Storage Gen2.
account_name: mytestdatalakegen2
filesystem: my-gen2-container
```

Create the Azure Machine Learning datastore in the CLI:

```
az ml datastore create --file my_adls_datastore.yml
```

Create an Azure Files datastore

- [CLI: Account key](#)
- [CLI: SAS](#)
- [Python SDK: Account key](#)
- [Python SDK: SAS](#)

Create the following YAML file (updating the values):

```
# my_files_datastore.yml
$schema: https://azurermschemas.azureedge.net/latest/azureFile.schema.json
name: file_example
type: azure_file
description: Datastore pointing to an Azure File Share.
account_name: mytestfilestore
file_share_name: my-share
credentials:
  account_key: XxXxXxXXXXXXXXXxXxxXxxXXXXXXXXXxXxxXXXXXXXXXxXxxXXXXXXXXXxXxxXXXXXXXXXxXxxXXXXXXXXXxXxxXXXXXXXXXxXxx
```

Create the Azure Machine Learning datastore in the CLI:

```
az ml datastore create --file my_files_datastore.yml
```

Create an Azure Data Lake Gen1 datastore

- [CLI: Identity-based access](#)
- [CLI: Service principal](#)
- [Python SDK: Identity-based access](#)
- [Python SDK: Service principal](#)

Create the following YAML file (updating the values):

```
# my_adls_datastore.yml
$schema: https://azuremlschemas.azureedge.net/latest/azureDataLakeGen1.schema.json
name: alds_gen1_credless_example
type: azure_data_lake_gen1
description: Credential-less datastore pointing to an Azure Data Lake Storage Gen1.
store_name: mytestdatalakegen1
```

Create the Azure Machine Learning datastore in the CLI:

```
az ml datastore create --file my_adls_datastore.yml
```

Next steps

- [Read data in a job](#)
- [Create data assets](#)
- [Data administration](#)

Create data assets

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

In this article, you learn how to create a data asset in Azure Machine Learning. By creating a data asset, you create a *reference* to the data source location, along with a copy of its metadata. Because the data remains in its existing location, you incur no extra storage cost, and don't risk the integrity of your data sources. You can create Data from datastores, Azure Storage, public URLs, and local files.

The benefits of creating data assets are:

- You can **share and reuse data** with other members of the team such that they do not need to remember file locations.
- You can **seamlessly access data** during model training (on any supported compute type) without worrying about connection strings or data paths.
- You can **version** the data.

Prerequisites

To create and work with data assets, you need:

- An Azure subscription. If you don't have one, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace. [Create workspace resources](#).
- The [Azure Machine Learning CLI/SDK installed](#) and MLTable package installed (`pip install mltable`).

Supported paths

When you create a data asset in Azure Machine Learning, you'll need to specify a `path` parameter that points to its location. Below is a table that shows the different data locations supported in Azure Machine Learning and examples for the `path` parameter:

LOCATION	EXAMPLES
A path on your local computer	<code>./home/username/data/my_data</code>
A path on a public http(s) server	<code>https://raw.githubusercontent.com/pandas-dev/pandas/main/doc/data/titanic.csv</code>
A path on Azure Storage	<code>https://<account_name>.blob.core.windows.net/<container_name>/path</code> <code>abfss://<file_system>@<account_name>.dfs.core.windows.net/<path></code>
A path on a datastore	<code>azureml://datastores/<data_store_name>/paths/<path></code>

NOTE

When you create a data asset from a local path, it will be automatically uploaded to the default Azure Machine Learning datastore in the cloud.

Create a `uri_folder` data asset

Below shows you how to create a *folder* as an asset:

- [Azure CLI](#)
- [Python SDK](#)

Create a `YAML` file (`<file-name>.yml`):

```
$schema: https://azuremlschemas.azureedge.net/latest/data.schema.json

# Supported paths include:
# local: ./<path>
# blob: https://<account_name>.blob.core.windows.net/<container_name>/<path>
# ADLS gen2: abfss://<file_system>@<account_name>.dfs.core.windows.net/<path>/
# Datastore: azureml://datastores/<data_store_name>/paths/<path>
type: uri_folder
name: <name_of_data>
description: <description goes here>
path: <path>
```

Next, create the data asset using the CLI:

```
az ml data create -f <file-name>.yml
```

Create a `uri_file` data asset

Below shows you how to create a *specific file* as a data asset:

- [Azure CLI](#)
- [Python SDK](#)

Sample `YAML` file `<file-name>.yml` for data in local path is as below:

```
$schema: https://azuremlschemas.azureedge.net/latest/data.schema.json

# Supported paths include:
# local: ./<path>/<file>
# blob: https://<account_name>.blob.core.windows.net/<container_name>/<path>/<file>
# ADLS gen2: abfss://<file_system>@<account_name>.dfs.core.windows.net/<path>/<file>
# Datastore: azureml://datastores/<data_store_name>/paths/<path>/<file>

type: uri_file
name: <name>
description: <description>
path: <uri>
```

```
> az ml data create -f <file-name>.yml
```

Create a `mltable` data asset

`mltable` is a way to abstract the schema definition for tabular data to make it easier to share data assets (an overview can be found in [MLTable](#)).

In this section, we show you how to create a data asset when the type is an `mltable`.

The `MLTable` file

The MLTable file is a file that provides the specification of the data's schema so that the `mltable` engine can materialize the data into an in-memory object (Pandas/Dask/Spark). An *example* MLTable file is provided below:

```
type: mltable

paths:
  - pattern: ./*.txt
transformations:
  - read_delimited:
    delimiter: ,
    encoding: ascii
    header: all_files_same_headers
```

IMPORTANT

We recommend co-locating the MLTable file with the underlying data in storage. For example:

```
└── my_data
    ├── MLTable
    └── file_1.txt
    .
    .
    |
    └── file_n.txt
```

Co-locating the MLTable with the data ensures a **self-contained artifact** where all that is needed is stored in that one folder (`my_data`); regardless of whether that folder is stored on your local drive or in your cloud store or on a public http server. You should **not** specify *absolute paths* in the MLTable file.

In your Python code, you materialize the MLTable artifact into a Pandas dataframe using:

```
import mltable

tbl = mltable.load(uri="./my_data")
df = tbl.to_pandas_dataframe()
```

The `uri` parameter in `mltable.load()` should be a valid path to a local or cloud **folder** which contains a valid MLTable file.

NOTE

You will need the `mltable` library installed in your Environment (`pip install mltable`).

Below shows you how to create an `mltable` data asset. The `path` can be any of the supported path formats outlined above.

- [Azure CLI](#)
- [Python SDK](#)

Create a `YAML` file (`<file-name>.yml`):

```
$schema: https://azuremlschemas.azureedge.net/latest/data.schema.json

# path must point to **folder** containing MLTable artifact (MLTable file + data
# Supported paths include:
# local: ./<path>
# blob: https://<account_name>.blob.core.windows.net/<container_name>/<path>
# ADLS gen2: abfss://<file_system>@<account_name>.dfs.core.windows.net/<path>/
# Datastore: azureml://datastores/<data_store_name>/paths/<path>

type: mltable
name: <name_of_data>
description: <description goes here>
path: <path>
```

NOTE

The path points to the **folder** containing the MLTable artifact.

Next, create the data asset using the CLI:

```
az ml data create -f <file-name>.yml
```

Next steps

- [Read data in a job](#)

Read and write data in a job

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

Learn how to read and write data for your jobs with the Azure Machine Learning Python SDK v2(preview) and the Azure Machine Learning CLI extension v2.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- The [Azure Machine Learning SDK for Python v2](#).
- An Azure Machine Learning workspace

Supported paths

When you provide a data input/output to a Job, you'll need to specify a `path` parameter that points to the data location. Below is a table that shows the different data locations supported in Azure Machine Learning and examples for the `path` parameter:

LOCATION	EXAMPLES
A path on your local computer	<code>./home/username/data/my_data</code>
A path on a public http(s) server	<code>https://raw.githubusercontent.com/pandas-dev/pandas/main/doc/data/titanic.csv</code>
A path on Azure Storage	<code>https://<account_name>.blob.core.windows.net/<container_name>/path</code> <code>abfss://<file_system>@<account_name>.dfs.core.windows.net/<path></code>
A path on a Datastore	<code>azureml://datastores/<data_store_name>/paths/<path></code>
A path to a Data Asset	<code>azureml:<my_data>:<version></code>

Supported modes

When you run a job with data inputs/outputs, you can specify the `mode` - for example, whether you would like the data to be read-only mounted or downloaded to the compute target. The table below shows the possible modes for different type/mode/input/output combinations:

TYPE	INPUT/O UTPUT	UPLOAD	DOWNLOAD	RO_MOUNT	RW_MOUNT	DIRECT	EVAL_DOWNLOAD	EVAL_MOUNT
<code>uri_folder</code>	Input		✓	✓		✓		
<code>uri_file</code>	Input		✓	✓		✓		
<code>mltable</code>	Input		✓	✓		✓	✓	✓

TYPE	INPUT/O UTPUT	UPLOAD	DOWNLOAD	RO_MOUNT	RW_MOUNT	DIRECT	EVAL_DOWNLOAD	EVAL_MOUNT
<code>uri_folder</code>	Output	✓			✓	✓		
<code>uri_file</code>	Output	✓			✓	✓		
<code>mltable</code>	Output	✓			✓	✓		

NOTE

`eval_download` and `eval_mount` are unique to `mltable`. Whilst `ro_mount` is the default mode for MLTable, there are scenarios where an MLTable can yield files that are not necessarily co-located with the MLTable file in storage. Alternatively, an `mltable` can subset or shuffle the data that resides in the storage. That view is only visible if the MLTable file is actually evaluated by the engine. These modes will provide that view of the files.

Read data in a job

- [Azure CLI](#)
- [Python SDK](#)

Create a job specification YAML file (`<file-name>.yml`). Specify in the `inputs` section of the job:

1. The `type`; whether the data you are pointing to is a specific file (`uri_file`) or a folder location (`uri_folder`) or an `mltable`.
2. The `path` of where your data is located; the path can be any of those outlined in the [Supported Paths](#) section.

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json

# Possible Paths for Data:
# Blob: https://<account_name>.blob.core.windows.net/<container_name>/<folder>/<file>
# Datastore: azureml://datastores/paths/<folder>/<file>
# Data Asset: azureml:<my_data>:<version>

command: |
    ls ${inputs.my_data}
code: <folder where code is located>
inputs:
    my_data:
        type: <type> # uri_file, uri_folder, mltable
        path: <path>
environment: azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azureml:cpu-cluster
```

Next, run in the CLI

```
az ml job create -f <file-name>.yml
```

Read V1 data assets

This section outlines how you can read V1 `FileDataset` and `TabularDataset` data entities in a V2 job.

Read a `FileDataset`

- [Azure CLI](#)
- [Python SDK](#)

Create a job specification YAML file (`<file-name>.yml`), with the type set to `mltable` and the mode set to `eval_mount`:

```
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json

command: |
    ls ${inputs.my_data}
code: <folder where code is located>
inputs:
    my_data:
        type: mltable
        mode: eval_mount
        path: azureml:<filedataset_name>@latest
environment: azureml:<environment_name>@latest
compute: azureml:cpu-cluster
```

Next, run in the CLI

```
az ml job create -f <file-name>.yml
```

Read a `TabularDataset`

- [Azure CLI](#)
- [Python SDK](#)

Create a job specification YAML file (`<file-name>.yml`), with the type set to `mltable` and the mode set to `direct`:

```
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json

command: |
    ls ${inputs.my_data}
code: <folder where code is located>
inputs:
    my_data:
        type: mltable
        mode: direct
        path: azureml:<tabulardataset_name>@latest
environment: azureml:<environment_name>@latest
compute: azureml:cpu-cluster
```

Next, run in the CLI

```
az ml job create -f <file-name>.yml
```

Write data in a job

In your job you can write data to your cloud-based storage using *outputs*. The [Supported modes](#) section showed that only job *outputs* can write data because the mode can be either `rw_mount` or `upload`.

- [Azure CLI](#)
- [Python SDK](#)

Create a job specification YAML file (`<file-name>.yml`), with the `outputs` section populated with the type and path of where you would like to write your data to:

```
$schema: https://azuremlschemas.azureedge.net/latest/CommandJob.schema.json

# Possible Paths for Data:
# Blob: https://<account_name>.blob.core.windows.net/<container_name>/<folder>/<file>
# Datastore: azureml://datastores/paths/<folder>/<file>
# Data Asset: azureml:<my_data>:<version>

code: src
command: >-
    python prep.py
    --raw_data ${inputs.raw_data}
    --prep_data ${outputs.prep_data}
inputs:
    raw_data:
        type: <type> # uri_file, uri_folder, mltable
        path: <path>
outputs:
    prep_data:
        type: <type> # uri_file, uri_folder, mltable
        path: <path>
environment: azureml:<environment_name>@latest
compute: azureml:cpu-cluster
```

Next create a job using the CLI:

```
az ml job create --file <file-name>.yml
```

Data in pipelines

If you're working with Azure Machine Learning pipelines, you can read data into and move data between pipeline components with the Azure Machine Learning CLI v2 extension or the Python SDK v2 (preview).

Azure Machine Learning CLI v2

The following YAML file demonstrates how to use the output data from one component as the input for another component of the pipeline using the Azure Machine Learning CLI v2 extension:

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```

$schema: https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline

display_name: 3b_pipeline_with_data
description: Pipeline with 3 component jobs with data dependencies

compute: azureml:cpu-cluster

outputs:
  final_pipeline_output:
    mode: rw_mount

jobs:
  component_a:
    type: command
    component: ./componentA.yml
    inputs:
      component_a_input:
        type: uri_folder
        path: ./data

    outputs:
      component_a_output:
        mode: rw_mount
  component_b:
    type: command
    component: ./componentB.yml
    inputs:
      component_b_input: ${parent.jobs.component_a.outputs.component_a_output}
    outputs:
      component_b_output:
        mode: rw_mount
  component_c:
    type: command
    component: ./componentC.yml
    inputs:
      component_c_input: ${parent.jobs.component_b.outputs.component_b_output}
    outputs:
      component_c_output: ${parent.outputs.final_pipeline_output}
      # mode: upload

```

Python SDK v2 (preview)

The following example defines a pipeline containing three nodes and moves data between each node.

- `prepare_data_node` that loads the image and labels from Fashion MNIST data set into `mnist_train.csv` and `mnist_test.csv`.
- `train_node` that trains a CNN model with Keras using the training data, `mnist_train.csv`.
- `score_node` that scores the model using test data, `mnist_test.csv`.

```
# define a pipeline containing 3 nodes: Prepare data node, train node, and score node
@pipeline(
    default_compute=cpu_compute_target,
)
def image_classification_keras_minist_convnet(pipeline_input_data):
    """E2E image classification pipeline with keras using python sdk."""
    prepare_data_node = prepare_data_component(input_data=pipeline_input_data)

    train_node = keras_train_component(
        input_data=prepare_data_node.outputs.training_data
    )
    train_node.compute = gpu_compute_target

    score_node = keras_score_component(
        input_data=prepare_data_node.outputs.test_data,
        input_model=train_node.outputs.output_model,
    )

# create a pipeline
pipeline_job = image_classification_keras_minist_convnet(pipeline_input_data=fashion_ds)
```

Next steps

- [Train models with the Python SDK v2 \(preview\)](#)
- [Tutorial: Create production ML pipelines with Python SDK v2 \(preview\)](#)
- Learn more about [Data in Azure Machine Learning](#)

Data administration

9/22/2022 • 5 minutes to read • [Edit Online](#)

Learn how to manage data access and how to authenticate in Azure Machine Learning

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

IMPORTANT

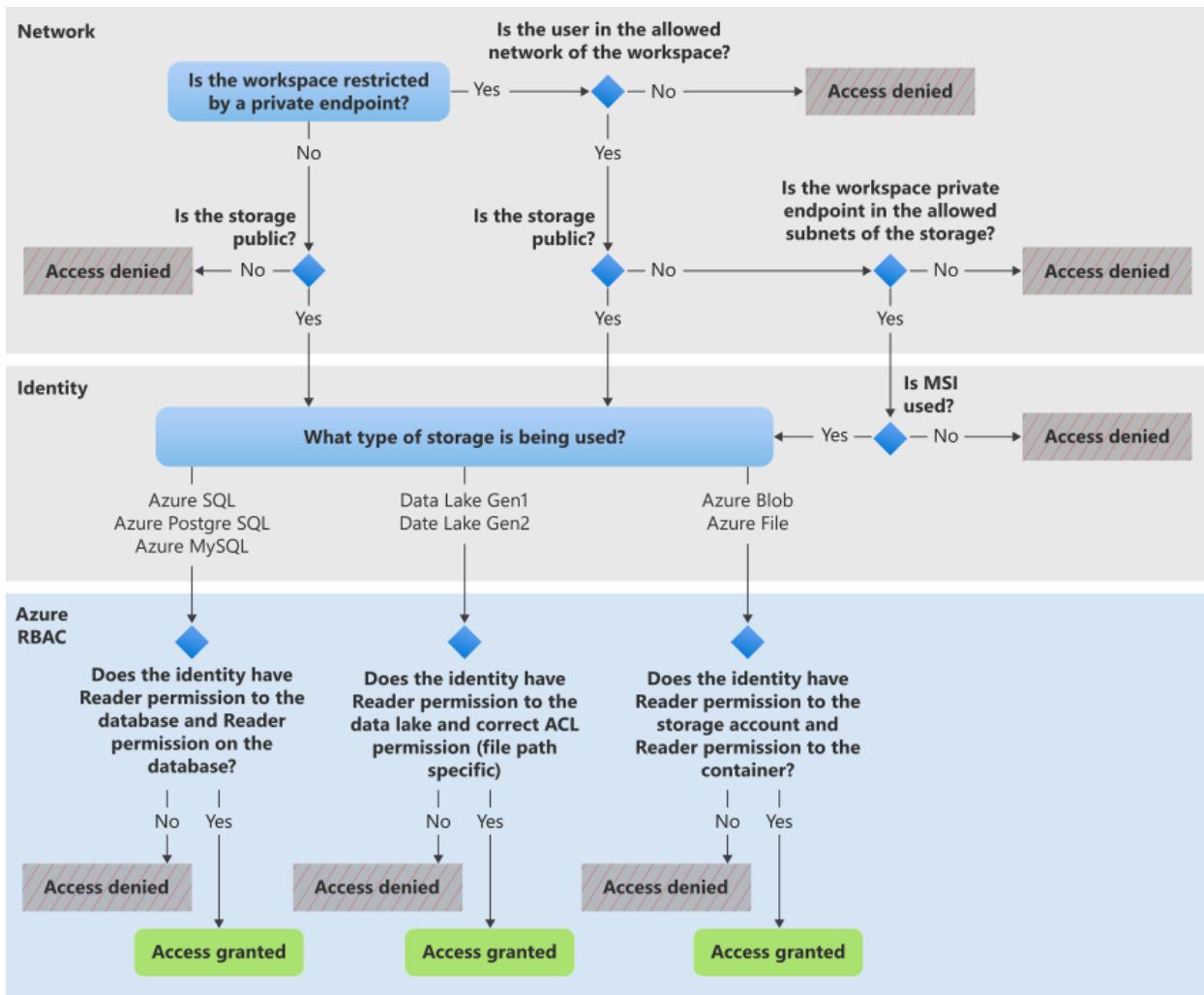
The information in this article is intended for Azure administrators who are creating the infrastructure required for an Azure Machine Learning solution.

In general, data access from studio involves the following checks:

- Who is accessing?
 - There are multiple different types of authentication depending on the storage type. For example, account key, token, service principal, managed identity, and user identity.
 - If authentication is made using a user identity, then it's important to know *which* user is trying to access storage. Learn more about [identity-based data access](#).
- Do they have permission?
 - Are the credentials correct? If so, does the service principal, managed identity, etc., have the necessary permissions on the storage? Permissions are granted using Azure role-based access controls (Azure RBAC).
 - [Reader](#) of the storage account reads metadata of the storage.
 - [Storage Blob Data Reader](#) reads data within a blob container.
 - [Contributor](#) allows write access to a storage account.
 - More roles may be required depending on the type of storage.
- Where is access from?
 - User: Is the client IP address in the VNet/subnet range?
 - Workspace: Is the workspace public or does it have a private endpoint in a VNet/subnet?
 - Storage: Does the storage allow public access, or does it restrict access through a service endpoint or a private endpoint?
- What operation is being performed?
 - Create, read, update, and delete (CRUD) operations on a data store/dataset are handled by Azure Machine Learning.
 - Data Access calls (such as preview or schema) go to the underlying storage and need extra permissions.
- Where is this operation being run; compute resources in your Azure subscription or resources hosted in a Microsoft subscription?
 - All calls to dataset and datastore services (except the "Generate Profile" option) use resources hosted in a **Microsoft subscription** to run the operations.
 - Jobs, including the "Generate Profile" option for datasets, run on a compute resource in **your subscription**, and access the data from there. So the compute identity needs permission to the storage rather than the identity of the user submitting the job.

The following diagram shows the general flow of a data access call. In this example, a user is trying to make a

data access call through a machine learning workspace, without using any compute resource.



Scenarios and identities

The following table lists what identities should be used for specific scenarios:

SCENARIO	USE WORKSPACE MANAGED SERVICE IDENTITY (MSI)	IDENTITY TO USE
Access from UI	Yes	Workspace MSI
Access from UI	No	User's Identity
Access from Job	Yes/No	Compute MSI
Access from Notebook	Yes/No	User's identity

Data access is complex and it's important to recognize that there are many pieces to it. For example, accessing data from Azure Machine Learning studio is different than using the SDK. When using the SDK on your local development environment, you're directly accessing data in the cloud. When using studio, you aren't always directly accessing the data store from your client. Studio relies on the workspace to access data on your behalf.

TIP

If you need to access data from outside Azure Machine Learning, such as using Azure Storage Explorer, *user* identity is probably what is used. Consult the documentation for the tool or service you are using for specific information. For more information on how Azure Machine Learning works with data, see [Identity-based data access to storage services on Azure](#).

Azure Storage Account

When using an Azure Storage Account from Azure Machine Learning studio, you must add the managed identity of the workspace to the following Azure RBAC roles for the storage account:

- [Blob Data Reader](#)
- If the storage account uses a private endpoint to connect to the VNet, you must grant the managed identity the [Reader](#) role for the storage account private endpoint.

For more information, see [Use Azure Machine Learning studio in an Azure Virtual Network](#).

See the following sections for information on limitations when using Azure Storage Account with your workspace in a VNet.

Secure communication with Azure Storage Account

To secure communication between Azure Machine Learning and Azure Storage Accounts, configure storage to [Grant access to trusted Azure services](#).

Azure Storage firewall

When an Azure Storage account is behind a virtual network, the storage firewall can normally be used to allow your client to directly connect over the internet. However, when using studio it isn't your client that connects to the storage account; it's the Azure Machine Learning service that makes the request. The IP address of the service isn't documented and changes frequently. **Enabling the storage firewall will not allow studio to access the storage account in a VNet configuration.**

Azure Storage endpoint type

When the workspace uses a private endpoint and the storage account is also in the VNet, there are extra validation requirements when using studio:

- If the storage account uses a [service endpoint](#), the workspace private endpoint and storage service endpoint must be in the same subnet of the VNet.
- If the storage account uses a [private endpoint](#), the workspace private endpoint and storage service endpoint must be in the same VNet. In this case, they can be in different subnets.

Azure Data Lake Storage Gen1

When using Azure Data Lake Storage Gen1 as a datastore, you can only use POSIX-style access control lists. You can assign the workspace's managed identity access to resources just like any other security principal. For more information, see [Access control in Azure Data Lake Storage Gen1](#).

Azure Data Lake Storage Gen2

When using Azure Data Lake Storage Gen2 as a datastore, you can use both Azure RBAC and POSIX-style access control lists (ACLs) to control data access inside of a virtual network.

To use Azure RBAC, follow the steps in the [Datastore: Azure Storage Account](#) section of the 'Use Azure Machine Learning studio in an Azure Virtual Network' article. Data Lake Storage Gen2 is based on Azure

Storage, so the same steps apply when using Azure RBAC.

To use ACLs, the managed identity of the workspace can be assigned access just like any other security principal. For more information, see [Access control lists on files and directories](#).

Next steps

For information on enabling studio in a network, see [Use Azure Machine Learning studio in an Azure Virtual Network](#).

Create an image labeling project and export labels

9/22/2022 • 15 minutes to read • [Edit Online](#)

Learn how to create and run data labeling projects to label images in Azure Machine Learning. Use machine-learning-assisted data labeling, or human-in-the-loop labeling, to aid with the task.

Set up labels for classification, object detection (bounding box), or instance segmentation (polygon).

You can also use the data labeling tool to [create a text labeling project](#).

Image labeling capabilities

Azure Machine Learning data labeling is a central place to create, manage, and monitor data labeling projects:

- Coordinate data, labels, and team members to efficiently manage labeling tasks.
- Tracks progress and maintains the queue of incomplete labeling tasks.
- Start and stop the project and control the labeling progress.
- Review the labeled data and export labeled as an Azure Machine Learning dataset.

IMPORTANT

Data images must be files available in an Azure blob datastore. (If you do not have an existing datastore, you may upload files during project creation.)

Image data can be files with any of these types: ".jpg", ".jpeg", ".png", ".jpe", ".jfif", ".bmp", ".tif", ".tiff", ".dcm", ".dicom". Each file is an item to be labeled.

Prerequisites

- The data that you want to label, either in local files or in Azure blob storage.
- The set of labels that you want to apply.
- The instructions for labeling.
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A Machine Learning workspace. See [Create an Azure Machine Learning workspace](#).

Create an image labeling project

Labeling projects are administered from Azure Machine Learning. You use the **Data Labeling** page to manage your projects.

If your data is already in Azure Blob storage, you should make it available as a datastore before you create the labeling project.

1. To create a project, select **Add project**. Give the project an appropriate name. The project name cannot be reused, even if the project is deleted in future.
2. Select **Image** to create an image labeling project.

- Choose **Image Classification Multi-class** for projects when you want to apply only a *single label* from a set of labels to an image.
- Choose **Image Classification Multi-label** for projects when you want to apply *one or more* labels from a set of labels to an image. For instance, a photo of a dog might be labeled with both *dog* and *daytime*.
- Choose **Object Identification (Bounding Box)** for projects when you want to assign a label and a bounding box to each object within an image.
- Choose **Instance Segmentation (Polygon)** for projects when you want to assign a label and draw a polygon around each object within an image.

3. Select **Next** when you're ready to continue.

Add workforce (optional)

Select **Use a vendor labeling company from Azure Marketplace** only if you've engaged a data labeling company from [Azure Marketplace](#). Then select the vendor. If your vendor doesn't appear in the list, unselect this option.

Make sure you first contact the vendor and sign a contract. For more information, see [Work with a data labeling vendor company \(preview\)](#).

Select **Next** to continue.

Specify the data to label

If you already created a dataset that contains your data, select it from the **Select an existing dataset** dropdown list. Or, select **Create a dataset** to use an existing Azure datastore or to upload local files.

NOTE

A project cannot contain more than 500,000 files. If your dataset has more, only the first 500,000 files will be loaded.

Create a dataset from an Azure datastore

In many cases, it's fine to just upload local files. But [Azure Storage Explorer](#) provides a faster and more robust way to transfer a large amount of data. We recommend Storage Explorer as the default way to move files.

To create a dataset from data that you've already stored in Azure Blob storage:

1. Select **Create a dataset > From datastore**.
2. Assign a **Name** to your dataset.
3. **Dataset type** is set to file, only file dataset types are supported for images.
4. Select the datastore.
5. If your data is in a subfolder within your blob storage, choose **Browse** to select the path.
 - Append "/**" to the path to include all the files in subfolders of the selected path.
 - Append "**/." to include all the data in the current container and its subfolders.
6. (Optional) Provide a description for your dataset.
7. Select **Next**.
8. Confirm the details. Select **Back** to modify the settings or **Create** to create the dataset.

Create a dataset from uploaded data

To directly upload your data:

1. Select **Create a dataset > From local files**.
2. Assign a **Name** to your dataset.
3. **Dataset type** is set to file, only file dataset types are supported for images.
4. (Optional) Provide a description for your dataset.
5. Select **Next**.
6. (Optional) Select or create a datastore. Or keep the default to upload to the default blob store ("workspaceblobstore") of your Machine Learning workspace.
7. Select **Browse** to select the local files or folder(s) to upload.
8. Select **Next**.
9. Confirm the details. Select **Back** to modify the settings or **Create** to create the dataset.

Configure incremental refresh

If you plan to add new files to your dataset, use incremental refresh to add these new files to your project.

When **incremental refresh at regular intervals** is enabled, the dataset is checked periodically for new files to be added to a project, based on the labeling completion rate. The check for new data stops when the project contains the maximum 500,000 files.

Select **Enable incremental refresh at regular intervals** when you want your project to continually monitor for new data in the datastore.

Unselect if you don't want new files in the datastore to automatically be added to your project.

To add more files to your project, use [Azure Storage Explorer](#) to upload to the appropriate folder in the blob storage.

After the project is created, use the **Details** tab to change **incremental refresh**, view the timestamp for the last refresh, and request an immediate refresh of data.

Specify label classes

On the **Label classes** page, specify the set of classes to categorize your data. Your labelers' accuracy and speed are affected by their ability to choose among the classes. For instance, instead of spelling out the full genus and

species for plants or animals, use a field code or abbreviate the genus.

Enter one label per row. Use the + button to add a new row. If you have more than three or four labels but fewer than 10, you may want to prefix the names with numbers ("1:", "2:") so the labelers can use the number keys to speed their work.

Describe the image labeling task

It's important to clearly explain the labeling task. On the **Labeling instructions** page, you can add a link to an external site for labeling instructions, or provide instructions in the edit box on the page. Keep the instructions task-oriented and appropriate to the audience. Consider these questions:

- What are the labels they'll see, and how will they choose among them? Is there a reference text to refer to?
- What should they do if no label seems appropriate?
- What should they do if multiple labels seem appropriate?
- What confidence threshold should they apply to a label? Do you want their "best guess" if they aren't certain?
- What should they do with partially occluded or overlapping objects of interest?
- What should they do if an object of interest is clipped by the edge of the image?
- What should they do after they submit a label if they think they made a mistake?
- What should they do if they discover image quality issues including poor lighting conditions, reflections, loss of focus, undesired background included, abnormal camera angles, and so on?
- What should they do if there are multiple reviewers who have different opinions on the labels?

For bounding boxes, important questions include:

- How is the bounding box defined for this task? Should it be entirely on the interior of the object, or should it be on the exterior? Should it be cropped as closely as possible, or is some clearance acceptable?
- What level of care and consistency do you expect the labelers to apply in defining bounding boxes?
- What is the visual definition of each label class? Is it possible to provide a list of normal, edge, and counter cases for each class?
- What should the labelers do if the object is tiny? Should it be labeled as an object or should it be ignored as background?
- How to label the object that is partially shown in the image?
- How to label the object that partially covered by other object?
- How to label the object if there is no clear boundary of the object?
- How to label the object which is not object class of interest but visually similar to an interested object type?

NOTE

Be sure to note that the labelers will be able to select the first 9 labels by using number keys 1-9.

Use ML-assisted data labeling

The **ML-assisted labeling** page lets you trigger automatic machine learning models to accelerate labeling tasks. Medical images (.dcm) are not included in assisted labeling.

At the beginning of your labeling project, the items are shuffled into a random order to reduce potential bias. However, any biases that are present in the dataset will be reflected in the trained model. For example, if 80% of your items are of a single class, then approximately 80% of the data used to train the model will be of that class.

Select *Enable ML assisted labeling* and specify a GPU to enable assisted labeling. If you don't have one in your workspace, a GPU cluster will be created for you and added to your workspace. The cluster is created with a minimum of 0 nodes, which means it doesn't cost anything when it's not in use.

ML-assisted labeling consists of two phases:

- Clustering
- Prelabeling

The exact number of labeled data necessary to start assisted labeling is not a fixed number. This can vary significantly from one labeling project to another. For some projects, it's sometimes possible to see prelabel or cluster tasks after 300 items have been manually labeled. ML Assisted Labeling uses a technique called *Transfer Learning*, which uses a pre-trained model to jump-start the training process. If your dataset's classes are similar to those in the pre-trained model, pre-labels may be available after only a few hundred manually labeled items. If your dataset is significantly different from the data used to pre-train the model, it may take much longer.

Since the final labels still rely on input from the labeler, this technology is sometimes called *human in the loop* labeling.

NOTE

ML assisted data labeling does not support default storage accounts secured behind a [virtual network](#). You must use a non-default storage account for ML assisted data labelling. The non-default storage account can be secured behind the virtual network.

Clustering

After a certain number of labels are submitted, the machine learning model for classification starts to group together similar items. These similar images are presented to the labelers on the same screen to speed up manual tagging. Clustering is especially useful when the labeler is viewing a grid of 4, 6, or 9 images.

Once a machine learning model has been trained on your manually labeled data, the model is truncated to its last fully-connected layer. Unlabeled images are then passed through the truncated model in a process commonly known as "embedding" or "featurization." This embeds each image in a high-dimensional space defined by this model layer. Images that are nearest neighbors in the space are used for clustering tasks.

The clustering phase does not appear for object detection models, or for text classification.

Prelabeling

After enough labels are submitted, a classification model is used to predict tags. Or an object detection model is used to predict bounding boxes. The labeler now sees pages that contain predicted labels already present on each item. For object detection, predicted boxes are also shown. The task is then to review these predictions and correct any mis-labeled images before submitting the page.

Once a machine learning model has been trained on your manually labeled data, the model is evaluated on a test set of manually labeled items to determine its accuracy at different confidence thresholds. This evaluation process is used to determine a confidence threshold above which the model is accurate enough to show pre-labels. The model is then evaluated against unlabeled data. Items with predictions more confident than this threshold are used for pre-labeling.

Initialize the image labeling project

After the labeling project is initialized, some aspects of the project are immutable. You can't change the task type or dataset. You *can* modify labels and the URL for the task description. Carefully review the settings before you create the project. After you submit the project, you're returned to the **Data Labeling** homepage, which will show the project as **Initializing**.

NOTE

This page may not automatically refresh. So, after a pause, manually refresh the page to see the project's status as **Created**.

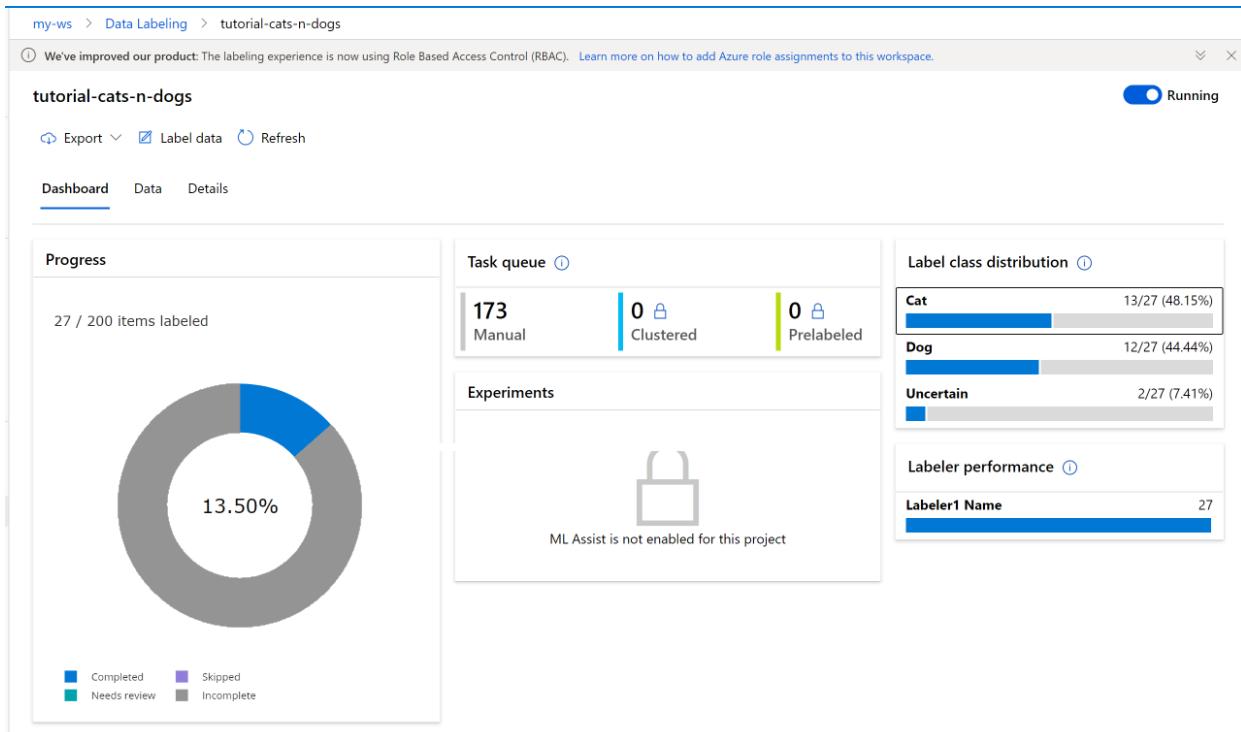
Run and monitor the project

After you initialize the project, Azure will begin running it. Select the project on the main **Data Labeling** page to see details of the project.

To pause or restart the project, toggle the **Running** status on the top right. You can only label data when the project is running.

Dashboard

The **Dashboard** tab shows the progress of the labeling task.



The progress chart shows how many items have been labeled, skipped, in need of review, or not yet done. Hover over the chart to see the number of item in each section.

The middle section shows the queue of tasks yet to be assigned. When ML assisted labeling is off, this section shows the number of manual tasks to be assigned. When ML assisted labeling is on, this will also show:

- Tasks containing clustered items in the queue
- Tasks containing prelabeled items in the queue

Additionally, when ML assisted labeling is enabled, a small progress bar shows when the next training run will occur. The Experiments sections give links for each of the machine learning runs.

- Training - trains a model to predict the labels
- Validation - determines whether this model's prediction will be used for pre-labeling the items
- Inference - prediction run for new items
- Featurization - clusters items (only for image classification projects)

On the right side is a distribution of the labels for those tasks that are complete. Remember that in some project types, an item can have multiple labels, in which case the total number of labels can be greater than the total

number items.

Data tab

On the **Data** tab, you can see your dataset and review labeled data. Scroll through the labeled data to see the labels. If you see incorrectly labeled data, select it and choose **Reject**, which will remove the labels and put the data back into the unlabeled queue.

Details tab

View and change details of your project. In this tab you can:

- View project details and input datasets
- Enable or disable incremental refresh at regular intervals or request an immediate refresh
- View details of the storage container used to store labeled outputs in your project
- Add labels to your project
- Edit instructions you give to your labels
- Edit details of ML assisted labeling, including enable/disable

Access for labelers

Anyone who has **Contributor** or **Owner** access to your workspace can label data in your project.

You can also add users and customize the permissions so that they can access labeling but not other parts of the workspace or your labeling project. For more information, see [Add users to your data labeling project](#).

Add new label class to a project

During the data labeling process, you may want to add more labels to classify your items. For example, you may want to add an "Unknown" or "Other" label to indicate confusion.

Use these steps to add one or more labels to a project:

1. Select the project on the main **Data Labeling** page.
2. At the top right of the page, toggle **Running** to **Paused** to stop labelers from their activity.
3. Select the **Details** tab.
4. In the list on the left, select **Label classes**.
5. At the top of the list, select **+ Add Labels**

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The top navigation bar includes 'Microsoft Azure Machine Learning Studio', 'sdg-ws', 'Data Labeling', and 'tutorial-cats-n-dogs2'. The top right features icons for notifications (2), settings, help, and user profile ('ML-docs sdg-ws'). The left sidebar has sections like 'New', 'Home', 'Author', 'Notebooks', 'Automated ML', 'Designer', 'Assets', 'Data', 'Jobs', 'Components', 'Pipelines', 'Environments', 'Models', 'Endpoints', 'Manage', 'Compute', 'Datastores', and 'Linked Services'. A red box highlights the 'Data Labeling' button at the bottom of the sidebar. The main content area has tabs 'Dashboard', 'Data', and 'Details' (which is selected and highlighted with a red box). On the left under 'Label classes', there's a 'Project details' section with a note about adding label classes while paused, and a 'Datasets' section with a '+ Add Labels' button and a search bar. The 'Label export (optional)' section lists 'Label Name' (Cat, Dog, Uncertain). The 'Label classes' section is highlighted with a red box and contains a table with columns 'Label Name' and 'Value'. The table rows are 'Cat', 'Dog', and 'Uncertain'. The 'General Settings' section is also visible.

6. In the form, add your new label. Then choose how to continue the project. Since you've changed the available labels, you choose how to treat the already labeled data:

- Start over, removing all existing labels. Choose this option if you want to start labeling from the beginning with the new full set of labels.
- Start over, keeping all existing labels. Choose this option to mark all data as unlabeled, but keep the existing labels as a default tag for images that were previously labeled.
- Continue, keeping all existing labels. Choose this option to keep all data already labeled as is, and start using the new label for data not yet labeled.

7. Modify your instructions page as necessary for the new label(s).

8. Once you've added all new labels, at the top right of the page toggle **Paused** to **Running** to restart the project.

Export the labels

Use the **Export** button on the **Project details** page of your labeling project. You can export the label data for Machine Learning experimentation at any time.

● Image labels can be exported as:

- [COCO format](#).The COCO file is created in the default blob store of the Azure Machine Learning workspace in a folder within *Labeling/export/coco*.
- An [Azure Machine Learning dataset with labels](#).

Access exported Azure Machine Learning datasets in the **Datasets** section of Machine Learning. The dataset details page also provides sample code to access your labels from Python.

The screenshot shows the Azure Machine Learning studio interface. On the left, there is a sidebar with various options like New, Home, Author, Notebooks, Automated ML, Designer, Assets, and Data (which is highlighted with a red box). The main content area displays the details of a dataset named 'tutorial-cats-n-dogs2_20220527_174752'. At the top, there are navigation links for Microsoft, sdg-ws, Data, and the specific dataset name. Below that is a 'Version 1 (latest)' dropdown and a star icon. The main content is divided into several sections: 'Attributes' (Properties: Tabular; Created by: Service Principal; Profile: No profile generated; Files in dataset: 1; Total size of files in dataset: 1.722 KiB; Current version: 1; Latest version: 1; Created time: May 27, 2022 1:47 PM; Modified time: May 27, 2022 1:47 PM), 'Tags' (labelingCreatedBy: Data Labeling; labelingProjectType: Image Classification Multi-class; SourceDatastoreName: workspaceblobstore; SourceRelativePath: UI/05-27-2022_052916_UTC/; labelingLabelName: ["Cat", "Dog", "Uncertain"]; HasConsensusLabel: False), and 'Description' (LabeledDs_tutorial-cats-n-dogs2 Of Type Image Classification Multi-class, Sourced From dfd6f0a6-137c-461b-84df-67450b4c696a). There is also a 'Data sources' section at the bottom.

Once you have exported your labeled data to an Azure Machine Learning dataset, you can use AutoML to build computer vision models trained on your labeled data. Learn more at [Set up AutoML to train computer vision models with Python \(preview\)](#)

Troubleshooting

Use these tips if you see any of these issues.

ISSUE	RESOLUTION
Only datasets created on blob datastores can be used.	Known limitation of the current release.
After creation, the project shows "Initializing" for a long time.	Manually refresh the page. Initialization should complete at roughly 20 datapoints per second. The lack of autorefresh is a known issue.
Newly labeled items not visible in data review.	To load all labeled items, choose the First button. The First button will take you back to the front of the list, but loads all labeled data.
Unable to assign set of tasks to a specific labeler.	Known limitation of the current release.

Object detection troubleshooting

ISSUE	RESOLUTION
Pressing Esc key while labeling for object detection creates a zero size label on the top-left corner. Submitting labels in this state fails.	Delete the label by clicking on the cross mark next to it.

Next steps

- [How to tag images](#)

Create a text labeling project and export labels

9/22/2022 • 14 minutes to read • [Edit Online](#)

Learn how to create and run data labeling projects to label text data in Azure Machine Learning. Specify either a single label or multiple labels to be applied to each text item.

You can also use the data labeling tool to [create an image labeling project](#).

Text labeling capabilities

Azure Machine Learning data labeling is a central place to create, manage, and monitor data labeling projects:

- Coordinate data, labels, and team members to efficiently manage labeling tasks.
- Tracks progress and maintains the queue of incomplete labeling tasks.
- Start and stop the project and control the labeling progress.
- Review the labeled data and export labeled as an Azure Machine Learning dataset.

IMPORTANT

Text data must be available in an Azure blob datastore. (If you do not have an existing datastore, you may upload files during project creation.)

Data formats available for text data:

- **.txt**: each file represents one item to be labeled.
- **.csv** or **.tsv**: each row represents one item presented to the labeler. You decide which columns the labeler can see in order to label the row.

Prerequisites

- The data that you want to label, either in local files or in Azure blob storage.
- The set of labels that you want to apply.
- The instructions for labeling.
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A Machine Learning workspace. See [Create an Azure Machine Learning workspace](#).

Create a text labeling project

Labeling projects are administered from Azure Machine Learning. You use the **Data Labeling** page to manage your projects.

If your data is already in Azure Blob storage, you should make it available as a datastore before you create the labeling project.

1. To create a project, select **Add project**. Give the project an appropriate name. The project name can't be reused, even if the project is deleted in future.
2. Select **Text** to create a text labeling project.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a sidebar with various navigation links such as New, Home, Notebooks, Automated ML, Designer, Assets, Data, Jobs, Components, Pipelines, Environments, Models, Endpoints, Compute, Datastores, and Linked Services. The main content area is titled 'Create project' under 'Data Labeling'. A sidebar on the right lists steps: Project details (selected), Add workforce (optional), Select or create data, Incremental refresh (optional), Label classes, Labeling instructions (optional), and ML assisted labeling (optional). The 'Project details' section contains fields for 'Project name' (with a note about a new feature for faster labeling), 'Media type' (set to 'Text'), and 'Labeling task type'. Three options are shown: 'Text Classification Multi-class' (selected), 'Text Classification Multi-label', and 'Text Named Entity Recognition (Preview)'. The 'Text Classification Multi-class' option has a blue border around its icon and text. At the bottom of the main panel are 'Back', 'Next', and 'Cancel' buttons.

- Choose **Text Classification Multi-class** for projects when you want to apply only a *single label* from a set of labels to each piece of text.
- Choose **Text Classification Multi-label** for projects when you want to apply *one or more labels* from a set of labels to each piece of text.
- Choose **Text Named Entity Recognition (Preview)** for projects when you want to apply labels to individual or multiple words of text in each entry.

IMPORTANT

Text Named Entity Recognition is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

3. Select **Next** when you're ready to continue.

Add workforce (optional)

Select **Use a vendor labeling company from Azure Marketplace** only if you've engaged a data labeling company from [Azure Marketplace](#). Then select the vendor. If your vendor doesn't appear in the list, unselect this option.

Make sure you first contact the vendor and sign a contract. For more information, see [Work with a data labeling vendor company \(preview\)](#).

Select **Next** to continue.

Select or create a dataset

If you already created a dataset that contains your data, select it from the **Select an existing dataset** drop-down list. Or, select **Create a dataset** to use an existing Azure datastore or to upload local files.

NOTE

A project cannot contain more than 500,000 files. If your dataset has more, only the first 500,000 files will be loaded.

Create a dataset from an Azure datastore

In many cases, it's fine to just upload local files. But [Azure Storage Explorer](#) provides a faster and more robust way to transfer a large amount of data. We recommend Storage Explorer as the default way to move files.

To create a dataset from data that you've already stored in Azure Blob storage:

1. Select **Create a dataset > From datastore**.
2. Assign a **Name** to your dataset.
3. Choose the **Dataset type**:
 - Select **Tabular** if you're using a .csv or .tsv file, where each row contains a response. Tabular isn't available for Text Named Entity Recognition projects.
 - Select **File** if you're using separate .txt files for each response.
4. (Optional) Provide a description for your dataset.
5. Select **Next**.
6. Select the datastore.
7. If your data is in a subfolder within your blob storage, choose **Browse** to select the path.
 - Append "/**" to the path to include all the files in subfolders of the selected path.
 - Append "**/." to include all the data in the current container and its subfolders.
8. Select **Next**.
9. Confirm the details. Select **Back** to modify the settings or **Create** to create the dataset.

Create a dataset from uploaded data

To directly upload your data:

1. Select **Create a dataset > From local files**.
2. Assign a **Name** to your dataset.
3. Choose the **Dataset type**:
 - Select **Tabular** if you're using a .csv or .tsv file, where each row is a response. Tabular isn't available for Text Named Entity Recognition projects.
 - Select **File** if you're using separate .txt files for each response.
4. (Optional) Provide a description of your dataset.
5. Select **Next**.
6. (Optional) Select or create a datastore. Or keep the default to upload to the default blob store ("workspaceblobstore") of your Machine Learning workspace.
7. Select **Upload** to select the local file(s) or folder(s) to upload.
8. Select **Next**.
9. If uploading .csv or .tsv files:
 - Confirm the settings and preview, select **Next**.
 - Include all columns of text you'd like the labeler to see when classifying that row. If you'll be using ML assisted labeling, adding numeric columns may degrade the ML assist model.
 - Select **Next**.
10. Confirm the details. Select **Back** to modify the settings or **Create** to create the dataset.

Configure incremental refresh

If you plan to add new files to your dataset, use incremental refresh to add these new files to your project.

When **incremental refresh at regular intervals** is enabled, the dataset is checked periodically for new files to be added to a project, based on the labeling completion rate. The check for new data stops when the project contains the maximum 500,000 files.

Select **Enable incremental refresh at regular intervals** when you want your project to continually monitor for new data in the datastore.

Unselect if you don't want new files in the datastore to automatically be added to your project.

To add more files to your project, use [Azure Storage Explorer](#) to upload to the appropriate folder in the blob storage.

After the project is created, use the **Details** tab to change **incremental refresh**, view the timestamp for the last refresh, and request an immediate refresh of data.

NOTE

Incremental refresh isn't available for projects that use tabular (.csv or .tsv) dataset input.

Specify label classes

On the **Label classes** page, specify the set of classes to categorize your data. Your labelers' accuracy and speed are affected by their ability to choose among the classes. For instance, instead of spelling out the full genus and species for plants or animals, use a field code or abbreviate the genus.

Enter one label per row. Use the **+** button to add a new row. If you have more than three or four labels but fewer than 10, you may want to prefix the names with numbers ("1:", "2:") so the labelers can use the number keys to speed their work.

Describe the text labeling task

It's important to clearly explain the labeling task. On the **Labeling instructions** page, you can add a link to an external site for labeling instructions, or provide instructions in the edit box on the page. Keep the instructions task-oriented and appropriate to the audience. Consider these questions:

- What are the labels they'll see, and how will they choose among them? Is there a reference text to refer to?
- What should they do if no label seems appropriate?
- What should they do if multiple labels seem appropriate?
- What confidence threshold should they apply to a label? Do you want their "best guess" if they aren't certain?
- What should they do with partially occluded or overlapping objects of interest?
- What should they do if an object of interest is clipped by the edge of the image?
- What should they do after they submit a label if they think they made a mistake?
- What should they do if they discover image quality issues including poor lighting conditions, reflections, loss of focus, undesired background included, abnormal camera angles, and so on?
- What should they do if there are multiple reviewers who have different opinions on the labels?

NOTE

Be sure to note that the labelers will be able to select the first 9 labels by using number keys 1-9.

Use ML-assisted data labeling

The **ML-assisted labeling** page lets you trigger automatic machine learning models to accelerate labeling

tasks. ML-assisted labeling is available for both file (.txt) and tabular (.csv) text data inputs. To use **ML-assisted labeling**:

- Select **Enable ML assisted labeling**.
- Select the **Dataset language** for the project. All languages supported by the [TextDNNLanguages Class](#) are present in this list.
- Specify a compute target to use. If you don't have one in your workspace, a compute cluster will be created for you and added to your workspace. The cluster is created with a minimum of 0 nodes, which means it doesn't cost anything when it's not in use.

How does ML-assisted labeling work?

At the beginning of your labeling project, the items are shuffled into a random order to reduce potential bias. However, any biases that are present in the dataset will be reflected in the trained model. For example, if 80% of your items are of a single class, then approximately 80% of the data used to train the model will be of that class.

For training the text DNN model used by ML-assist, the input text per training example will be limited to approximately the first 128 words in the document. For tabular input, all text columns are first concatenated before applying this limit. This is a practical limit imposed to allow for the model training to complete in a timely manner. The actual text in a document (for file input) or set of text columns (for tabular input) can exceed 128 words. The limit only pertains to what is internally leveraged by the model during the training process.

The exact number of labeled items necessary to start assisted labeling isn't a fixed number. This can vary significantly from one labeling project to another, depending on many factors, including the number of labels classes and label distribution.

Since the final labels still rely on input from the labeler, this technology is sometimes called *human in the loop* labeling.

NOTE

ML assisted data labeling does not support default storage accounts secured behind a [virtual network](#). You must use a non-default storage account for ML assisted data labelling. The non-default storage account can be secured behind the virtual network.

Pre-labeling

After enough labels are submitted for training, the trained model is used to predict tags. The labeler now sees pages that contain predicted labels already present on each item. The task is then to review these predictions and correct any mis-labeled items before submitting the page.

Once a machine learning model has been trained on your manually labeled data, the model is evaluated on a test set of manually labeled items to determine its accuracy at different confidence thresholds. This evaluation process is used to determine a confidence threshold above which the model is accurate enough to show pre-labels. The model is then evaluated against unlabeled data. Items with predictions more confident than this threshold are used for pre-labeling.

Initialize the text labeling project

After the labeling project is initialized, some aspects of the project are immutable. You can't change the task type or dataset. You *can* modify labels and the URL for the task description. Carefully review the settings before you create the project. After you submit the project, you're returned to the **Data Labeling** homepage, which will show the project as **Initializing**.

NOTE

This page may not automatically refresh. So, after a pause, manually refresh the page to see the project's status as **Created**.

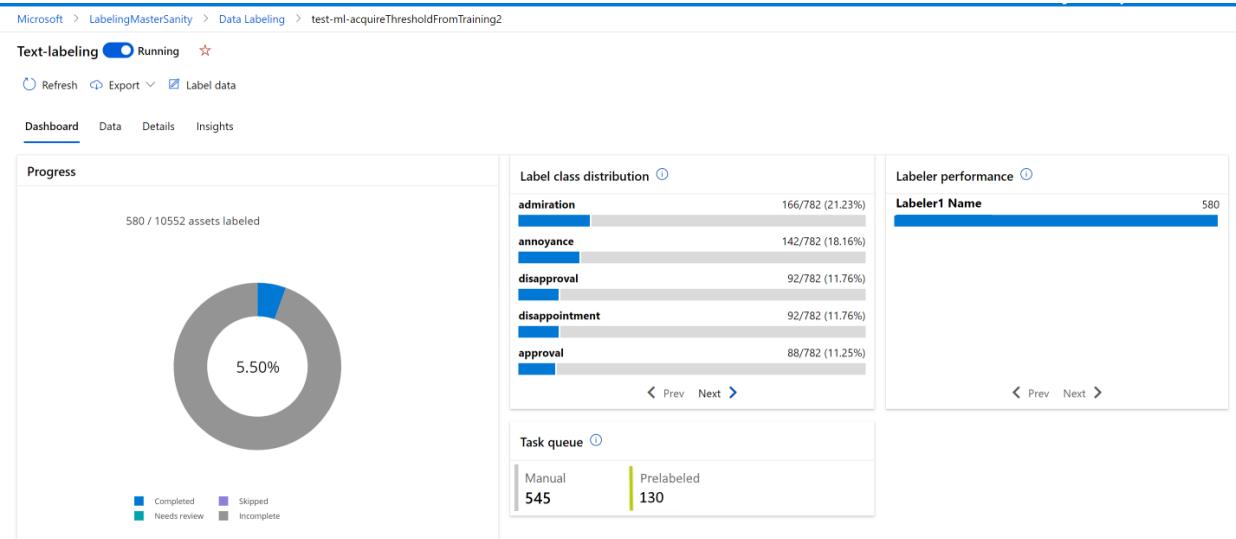
Run and monitor the project

After you initialize the project, Azure will begin running it. Select the project on the main **Data Labeling** page to see details of the project.

To pause or restart the project, toggle the **Running** status on the top right. You can only label data when the project is running.

Dashboard

The **Dashboard** tab shows the progress of the labeling task.



The progress chart shows how many items have been labeled, skipped, in need of review, or not yet done. Hover over the chart to see the number of items in each section.

The middle section shows the queue of tasks yet to be assigned. If ML-assisted labeling is on, you'll also see the number of pre-labeled items.

On the right side is a distribution of the labels for those tasks that are complete. Remember that in some project types, an item can have multiple labels, in which case the total number of labels can be greater than the total number items.

Data tab

On the **Data** tab, you can see your dataset and review labeled data. Scroll through the labeled data to see the labels. If you see incorrectly labeled data, select it and choose **Reject**, which will remove the labels and put the data back into the unlabeled queue.

Details tab

View and change details of your project. In this tab you can:

- View project details and input datasets
- Enable or disable **incremental refresh at regular intervals**, or request an immediate refresh.
- View details of the storage container used to store labeled outputs in your project
- Add labels to your project
- Edit instructions you give to your labels

Access for labelers

Anyone who has **Contributor** or **Owner** access to your workspace can label data in your project.

You can also add users and customize the permissions so that they can access labeling but not other parts of the workspace or your labeling project. For more information, see [Add users to your data labeling project](#).

Add new label class to a project

During the data labeling process, you may want to add more labels to classify your items. For example, you may want to add an "Unknown" or "Other" label to indicate confusion.

Use these steps to add one or more labels to a project:

1. Select the project on the main **Data Labeling** page.
2. At the top right of the page, toggle **Running** to **Paused** to stop labelers from their activity.
3. Select the **Details** tab.
4. In the list on the left, select **Label classes**.
5. At the top of the list, select **+ Add Labels**

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a sidebar with various project categories like Microsoft, New, Home, Author, Notebooks, Automated ML, Designer, Assets, Data, Jobs, Components, Pipelines, Environments, Models, Endpoints, Compute, Datastores, and Linked Services. The 'Data Labeling' option is highlighted with a red box. The main area shows a project named 'tutorial-cats-n-dogs2' which is currently 'Running'. The 'Details' tab is selected. A yellow callout box says 'Adding label classes: To add new label classes the project must be paused'. Below it, there's a 'Label Name' input field with 'Cat' and 'Dog' listed. A search bar is also present. The 'Label classes' section is highlighted with a red box. Other sections visible include Project details, Datasets, Incremental refresh (optional), Label export (optional), Instructions, ML assisted labeling (optional), and General Settings.

6. In the form, add your new label. Then choose how to continue the project. Since you've changed the available labels, you choose how to treat the already labeled data:
 - Start over, removing all existing labels. Choose this option if you want to start labeling from the beginning with the new full set of labels.
 - Start over, keeping all existing labels. Choose this option to mark all data as unlabeled, but keep the existing labels as a default tag for images that were previously labeled.
 - Continue, keeping all existing labels. Choose this option to keep all data already labeled as is, and start using the new label for data not yet labeled.
7. Modify your instructions page as necessary for the new label(s).
8. Once you've added all new labels, at the top right of the page toggle **Paused** to **Running** to restart the project.

Export the labels

Use the **Export** button on the **Project details** page of your labeling project. You can export the label data for

Machine Learning experimentation at any time.

For all project types other than **Text Named Entity Recognition**, you can export:

- A CSV file. The CSV file is created in the default blob store of the Azure Machine Learning workspace in a folder within *Labeling/export/csv*.
- An [Azure Machine Learning dataset with labels](#).

For **Text Named Entity Recognition** projects, you can export:

- An [Azure Machine Learning dataset with labels](#).
- A CoNLL file. For this export, you'll also have to assign a compute resource. The export process runs offline and generates the file as part of an experiment run. When the file is ready to download, you'll see a notification on the top right. Select this to open the notification, which includes the link to the file.



Access exported Azure Machine Learning datasets in the **Datasets** section of Machine Learning. The dataset details page also provides sample code to access your labels from Python.

A detailed screenshot of the Azure Machine Learning Datasets details page for a dataset named "tutorial-cats-n-dogs2_20220527_174752". The left sidebar is visible with the "Data" section highlighted. The main content area shows the dataset's attributes, tags, and descriptions. The "Attributes" section includes fields like "Properties: Tabular", "Created by: Service Principal", and "Files in dataset: 1". The "Tags" section lists labels such as "labelingCreatedBy", "labelingProjectType", and "SourceDatastoreName". The "Description" section contains a note about the dataset being a "LabeledDs_tutorial-cats-n-dogs2 Of Type Image Classification Multi-class, Sourced From dfd6f0a6-137c-461b-84df-67450b4c696a".

Troubleshooting

Use these tips if you see any of these issues.

ISSUE	RESOLUTION
Only datasets created on blob datastores can be used.	Known limitation of the current release.
After creation, the project shows "Initializing" for a long time.	Manually refresh the page. Initialization should complete at roughly 20 datapoints per second. The lack of autorefresh is a known issue.

ISSUE	RESOLUTION
Newly labeled items not visible in data review.	To load all labeled items, choose the First button. The First button will take you back to the front of the list, but loads all labeled data.
Unable to assign set of tasks to a specific labeler.	Known limitation of the current release.

Next steps

- [How to tag text](#)

Labeling images and text documents

9/22/2022 • 11 minutes to read • [Edit Online](#)

After your project administrator creates an [image data labeling project](#) or [text data labeling project](#) in Azure Machine Learning, you can use the labeling tool to rapidly prepare data for a Machine Learning project. This article describes:

- How to access your labeling projects
- The labeling tools
- How to use the tools for specific labeling tasks

Prerequisites

- A [Microsoft account](#) or an Azure Active Directory account for the organization and project.
- Contributor level access to the workspace that contains the labeling project.

Sign in to the studio

1. Sign in to [Azure Machine Learning studio](#).
2. Select the subscription and the workspace that contains the labeling project. Get this information from your project administrator.
3. Depending on your access level, you may see multiple sections on the left. If so, select **Data labeling** on the left-hand side to find the project.

Understand the labeling task

In the table of data labeling projects, select the **Label data** link for your project.

You see instructions that are specific to your project. They explain the type of data that you're facing, how you should make your decisions, and other relevant information. After you read this information, at the top of the page select **Tasks**. Or at the bottom of the page, select **Start labeling**.

Selecting a label

In all data labeling tasks, you choose an appropriate tag or tags from a set that's specified by the project administrator. You can select the first nine tags by using the number keys on your keyboard.

Assisted machine learning

Machine learning algorithms may be triggered during your labeling. If these algorithms are enabled in your project, you may see the following:

- Images
 - After some amount of data have been labeled, you may see **Tasks clustered** at the top of your screen next to the project name. This means that images are grouped together to present similar images on the same page. If so, switch to one of the multiple image views to take advantage of the grouping.
 - At a later point, you may see **Tasks prelabeled** next to the project name. Items will then appear

with a suggested label that comes from a machine learning classification model. No machine learning model has 100% accuracy. While we only use data for which the model is confident, these data might still be incorrectly prelabeled. When you see labels, correct any wrong labels before submitting the page.

- For object identification models, you may see bounding boxes and labels already present. Correct any that are incorrect before submitting the page.
 - For segmentation models, you may see polygons and labels already present. Correct any that are incorrect before submitting the page.
- Text

- At some point, you may see **Tasks prelabeled** next to the project name. Items will then appear with a suggested label that comes from a machine learning classification model. No machine learning model has 100% accuracy. While we only use data for which the model is confident, these data might still be incorrectly prelabeled. When you see labels, correct any wrong labels before submitting the page.

Especially early in a labeling project, the machine learning model may only be accurate enough to prelabel a small subset of images. Once these images are labeled, the labeling project will return to manual labeling to gather more data for the next round of model training. Over time, the model will become more confident about a higher proportion of images, resulting in more prelabel tasks later in the project.

When there are no more prelabeled tasks, you'll stop confirming or correcting labels and go back to manually tagging the items.

Image tasks

For image-classification tasks, you can choose to view multiple images simultaneously. Use the icons above the image area to select the layout.

To select all the displayed images simultaneously, use **Select all**. To select individual images, use the circular selection button in the upper-right corner of the image. You must select at least one image to apply a tag. If you select multiple images, any tag that you select will be applied to all the selected images.

Here we've chosen a two-by-two layout and are about to apply the tag "Mammal" to the images of the bear and orca. The image of the shark was already tagged as "Cartilaginous fish," and the iguana hasn't been tagged yet.

Preview Microsoft Azure | Machine Learning Data Labeling

All Projects > Animal Classes (Multiclass)

Animal Classes (Multiclass)

Instructions Tasks

Select all (2 selected)

Mammal Bird Bony fish Cartilaginous fish Reptile Anthozoa Other/Unknown

Cartilaginous fish X

Submit

This screenshot shows the Microsoft Azure Machine Learning Data Labeling interface for a multiclass classification task. The top navigation bar includes 'Preview', 'Microsoft Azure', and 'Machine Learning Data Labeling'. Below this, the project name 'Animal Classes (Multiclass)' is displayed. The 'Tasks' tab is selected. A message 'Select all (2 selected)' is shown above the image grid. To the right of the images is a 'Tags' sidebar with checkboxes for 'Mammal' (checked), 'Bird', 'Bony fish', 'Cartilaginous fish', 'Reptile', 'Anthozoa', and 'Other/Unknown'. The first image is a marine iguana, and the second is a brown bear. The third image is an orca, which is checked under 'Mammal'. The fourth image is a shark, which is checked under 'Cartilaginous fish'. A 'Submit' button is located at the bottom left.

IMPORTANT

Only switch layouts when you have a fresh page of unlabeled data. Switching layouts clears the page's in-progress tagging work.

Azure enables the **Submit** button when you've tagged all the images on the page. Select **Submit** to save your work.

After you submit tags for the data at hand, Azure refreshes the page with a new set of images from the work queue.

Medical image tasks

IMPORTANT

The capability to label DICOM or similar image types is not intended or made available for use as a medical device, clinical support, diagnostic tool, or other technology intended to be used in the diagnosis, cure, mitigation, treatment, or prevention of disease or other conditions, and no license or right is granted by Microsoft to use this capability for such purposes. This capability is not designed or intended to be implemented or deployed as a substitute for professional medical advice or healthcare opinion, diagnosis, treatment, or the clinical judgment of a healthcare professional, and should not be used as such. The customer is solely responsible for any use of Data Labeling for DICOM or similar image types.

Image projects support DICOM image format for X-ray file images.

Chest X-rays thorax diseases.

Disclaimer: The capability to label DICOM or similar image types is not intended or made available for use as a medical device, clinical support, diagnostic tool, or other purpose.

Instructions Tasks

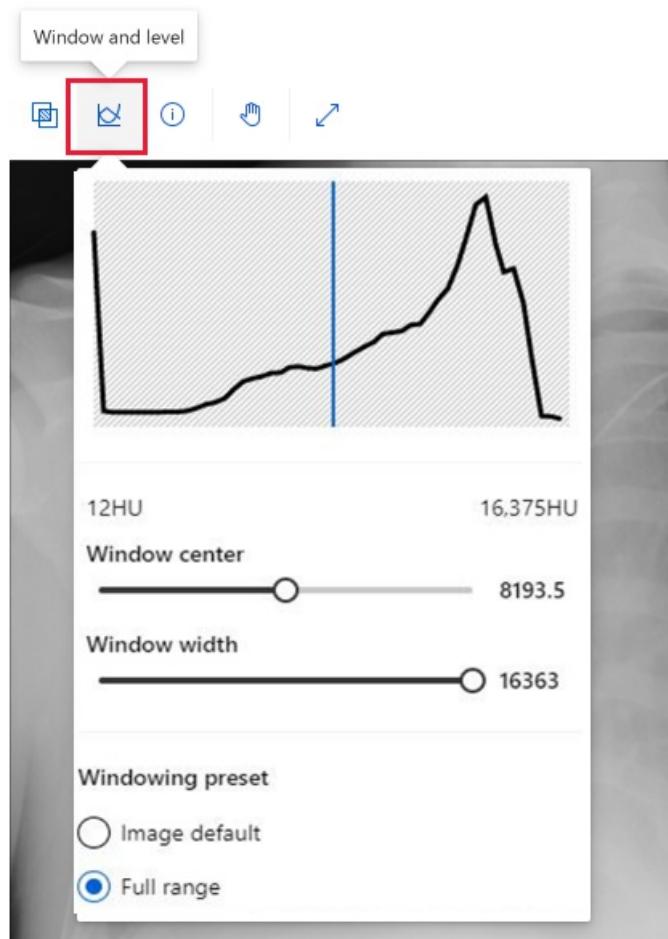
100%

Tags

Search tags

Tag	Count
1 Atelectasis	0
2 Cardiomegaly	0
3 Effusion	0
4 Infiltration	0
5 Mass	0
6 Nodule	0
7 Pneumonia	0
8 Pneumothorax	0

While you label the medical images with the same tools as any other images, there is an additional tool for DICOM images. Select the **Window and level** tool to change the intensity of the image. This tool is available only for DICOM images.



Tag images for multi-class classification

If your project is of type "Image Classification Multi-Class," you'll assign a single tag to the entire image. To review the directions at any time, go to the **Instructions** page and select **View detailed instructions**.

If you realize that you made a mistake after you assign a tag to an image, you can fix it. Select the "X" on the label that's displayed below the image to clear the tag. Or, select the image and choose another class. The newly selected value will replace the previously applied tag.

Tag images for multi-label classification

If you're working on a project of type "Image Classification Multi-Label," you'll apply one *or more* tags to an image. To see the project-specific directions, select **Instructions** and go to **View detailed instructions**.

Select the image that you want to label and then select the tag. The tag is applied to all the selected images, and then the images are deselected. To apply more tags, you must reselect the images. The following animation shows multi-label tagging:

1. **Select all** is used to apply the "Ocean" tag.
2. A single image is selected and tagged "Closeup."
3. Three images are selected and tagged "Wide angle."

All Projects > Photo labels (Multilabel)

Photo labels (Multilabel)

Instructions Tasks

Select all (0 selected)

Tags

- Land
- Ocean
- Closeup
- Wideangle

Submit

To correct a mistake, select the "X" to clear an individual tag or select the images and then select the tag, which clears the tag from all the selected images. This scenario is shown here. Selecting "Land" will clear that tag from the two selected images.

Photo labels (Multilabel)

Instructions

Tasks

 Select all (2 selected)

Tags

- Land
- Ocean
- Closeup
- Wideangle

**Submit**

Azure will only enable the **Submit** button after you've applied at least one tag to each image. Select **Submit** to save your work.

Tag images and specify bounding boxes for object detection

If your project is of type "Object Identification (Bounding Boxes)," you'll specify one or more bounding boxes in the image and apply a tag to each box. Images can have multiple bounding boxes, each with a single tag. Use [View detailed instructions](#) to determine if multiple bounding boxes are used in your project.

1. Select a tag for the bounding box that you plan to create.
2. Select the **Rectangular box** tool or select "R."
3. Select and drag diagonally across your target to create a rough bounding box. To adjust the bounding box, drag the edges or corners.

Photo Subject ID (Object Identification)

The screenshot shows a user interface for object identification. At the top, there are tabs for 'Instructions' and 'Tasks'. Below the tabs is a toolbar with various icons: a crop icon (highlighted with a red box and number 2), a copy icon, a paste icon, a lock icon, a template-based box icon, a brightness/contrast icon, a circular zoom icon, a hand icon, a search icon, and a magnifying glass icon. To the right of the toolbar is a 'Tags' section with a red circle containing the number 1. A search bar labeled 'Search tags' is next to it. A list of tags follows: (#1) Shark, (#2) Dog, (#3) Cat, (#4) Bird (which is selected and highlighted with a blue circle), and (#5) Other. The main area displays a photograph of a blue footed booby standing on a rock. A white dashed bounding box surrounds the entire bird. A red circle with the number 3 is located at the bottom right corner of the image area. At the bottom left is a 'Submit' button.

To delete a bounding box, select the X-shaped target that appears next to the bounding box after creation.

You can't change the tag of an existing bounding box. If you make a tag-assignment mistake, you have to delete the bounding box and create a new one with the correct tag.

By default, you can edit existing bounding boxes. The **Lock/unlock regions** tool or "L" toggles that behavior. If regions are locked, you can only change the shape or location of a new bounding box.

Use the **Regions manipulation** tool or "M" to adjust an existing bounding box. Drag the edges or corners to adjust the shape. Select in the interior to be able to drag the whole bounding box. If you can't edit a region, you've probably toggled the **Lock/unlock regions** tool.

Use the **Template-based box** tool or "T" to create multiple bounding boxes of the same size. If the image has no bounding boxes and you activate template-based boxes, the tool will produce 50-by-50-pixel boxes. If you create a bounding box and then activate template-based boxes, any new bounding boxes will be the size of the last box that you created. Template-based boxes can be resized after placement. Resizing a template-based box only resizes that particular box.

To delete *all* bounding boxes in the current image, select the **Delete all regions** tool .

After you create the bounding boxes for an image, select **Submit** to save your work, or your work in progress won't be saved.

Tag images and specify polygons for image segmentation

If your project is of type "Instance Segmentation (Polygon)," you'll specify one or more polygons in the image and apply a tag to each polygon. Images can have multiple bounding polygons, each with a single tag. Use **View**

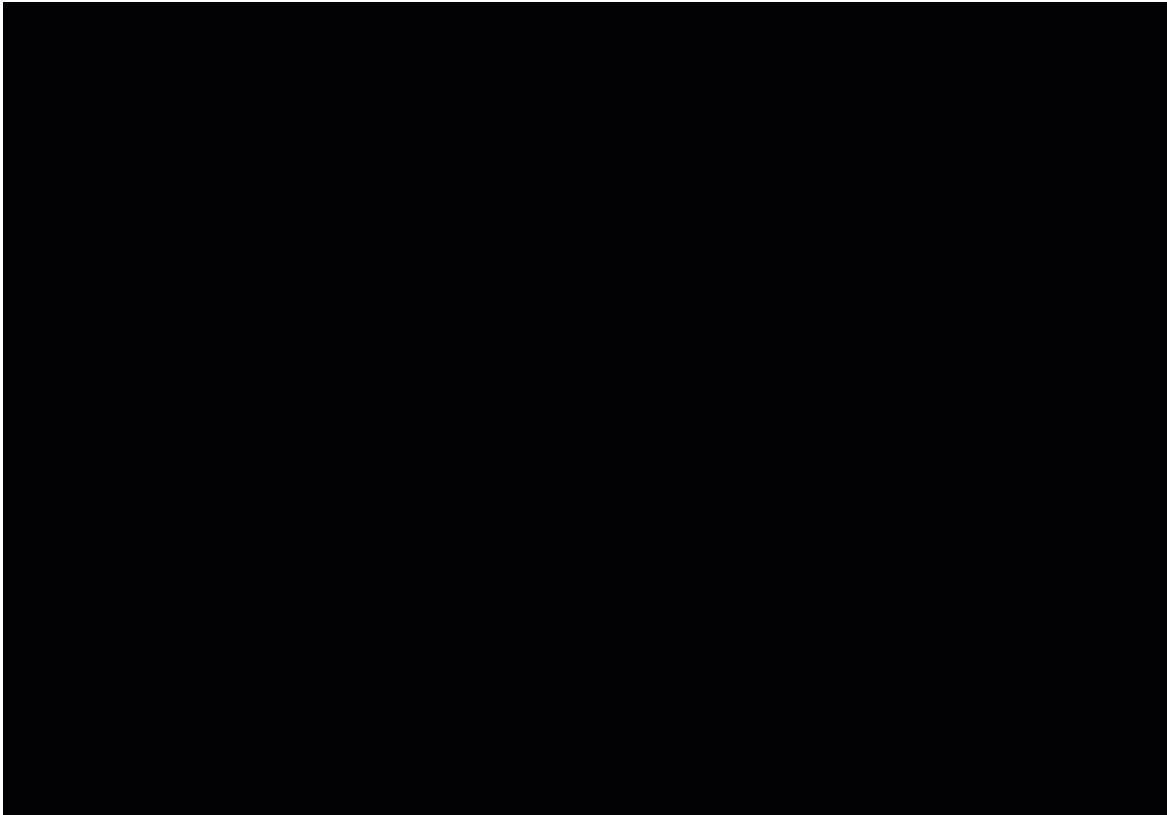
detailed instructions to determine if multiple bounding polygons are used in your project.

1. Select a tag for the polygon that you plan to create.



2. Select the **Draw polygon region** tool or select "P".

3. Select for each point in the polygon. When you've completed the shape, double-click to finish.



To delete a polygon, select the X-shaped target that appears next to the polygon after creation.

If you want to change the tag for a polygon, select the **Move region** tool, select the polygon, and select the correct tag.



You can edit existing polygons. The **Lock/unlock regions** tool or "L" toggles that behavior. If regions are locked, you can only change the shape or location of a new polygon.



Use the **Add or remove polygon points** tool or "U" to adjust an existing polygon. Select the polygon to add or remove a point. If you can't edit a region, you've probably toggled the **Lock/unlock regions** tool.



To delete *all* polygons in the current image, select the **Delete all regions** tool .

After you create the polygons for an image, select **Submit** to save your work, or your work in progress won't be saved.

Label text

When tagging text, use the toolbar to:

- Increase or decrease the text size
- Change the font
- Skip labeling this item and move to the next item

If you realize that you made a mistake after you assign a tag, you can fix it. Select the "X" on the label that's

displayed below the text to clear the tag.

There are three text project types:

PROJECT TYPE	DESCRIPTION
Classification Multi-Class	Assign a single tag to the entire text entry. You can only select one tag for each text item. Select a tag and then select Submit to move to the next entry.
Classification Multi-Label	Assign one <i>or more</i> tags to each text entry. You can select multiple tags for each text item. Select all the tags that apply and then select Submit to move to the next entry.
Named entity recognition (preview)	Tag different words or phrases in each text entry. See directions in the section below.

IMPORTANT

Named entity recognition is in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

To see the project-specific directions, select **Instructions** and go to [View detailed instructions](#).

Tag words and phrases (preview)

If your project is set up for named entity recognition, you tag different words or phrases in each text item. To label text:

1. Select the label or type the number corresponding to the appropriate label
2. Double-click on a word, or use your mouse to select multiple words.

The screenshot shows the 'Tasks' tab of the NER interface. On the left, there are buttons for 'Text size', 'Font', and 'Skip'. The main area contains a text entry: "Fabrikam, Inc organization x says it's holding merger talks with Contoso, Ltd organization x.". To the right is a 'Tags' panel with a search bar and an 'Expand all | Collapse all' button. The list of tags includes:

Label	Count
location	0
person	0
organization	2

Under the organization tag, there are two entries: "Fabrikam, Inc" and "Contoso, Ltd", each with a delete icon.

To change a label, you can:

- Delete the label and start again.
- Change the value for all or some of a specific label in your current item:
 - Select the label itself, which will select all instances of that label.
 - Select again on the instances of this label to unselect any instances you don't want to change.
 - Finally, select a new label to change all the labels that are still selected.

When you've tagged all the items in an entry, select **Submit** to move to the next entry.

Finish up

When you submit a page of tagged data, Azure assigns new unlabeled data to you from a work queue. If there's no more unlabeled data available, you'll get a message noting this along with a link to the portal home page.

When you're done labeling, select your name in the upper-right corner of the labeling portal and then select **sign-out**. If you don't sign out, eventually Azure will "time you out" and assign your data to another labeler.

Next steps

- Learn to [train image classification models in Azure](#)

Add users to your data labeling project

9/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows how to add users to your data labeling project so that they can label data, but not see the rest of your workspace. You can use these steps to add anyone to your project, whether or not they are from a [data labeling vendor company](#).

Prerequisites

- An Azure subscription. If you don't have an Azure subscription [create a free account](#) before you begin.
- An Azure Machine Learning workspace. See [Create workspace resources](#).

You'll need certain permission levels to follow the steps in this article. If you can't follow one of the steps, contact your administrator to get the appropriate permissions.

- To add a guest user, your organization's external collaboration settings must be configured to allow you to invite guests.
- To add a custom role, you must have `Microsoft.Authorization/roleAssignments/write` permissions for your subscription, such as [User Access Administrator](#) or [Owner](#).
- To add users to your workspace, you must be an [Owner](#) of the workspace.

Add custom role

To add a custom role, you must have `Microsoft.Authorization/roleAssignments/write` permissions for your subscription, such as [User Access Administrator](#).

1. Open your workspace in [Azure Machine Learning studio](#)
2. Open the menu on the top right and select **View all properties in Azure Portal**. You'll use Azure portal for all the rest of the steps in this article.
3. Select the **Resource group** link in the middle of the page.
4. On the left, select **Access control (IAM)**.
5. At the top, select **+ Add > Add custom role**.
6. For the **Custom role name**, type the name you want to use. For example, **Labeler**.
7. In the **Description** box, add a description. For example, **Labeler access for data labeling projects**.
8. Select **Start from JSON**.
9. At the bottom of the page, select **Next**.
10. Don't do anything for the **Permissions** tab, you'll add permissions in a later step. Select **Next**.
11. The **Assignable scopes** tab shows your subscription information. Select **Next**.
12. In the **JSON** tab, above the edit box, select **Edit**.
13. Select lines starting with "actions:" and "notActions:".

Here is your custom role in JSON format. [Learn more](#)

[Download](#)[Discard changes](#)[Save](#)

```
1  {
2    "properties": {
3      "roleName": "Labeler",
4      "description": "Labeler access for data labeling projects",
5      "assignableScopes": [
6        "/subscriptions/abcdef01-2345-6789-0abc-def012345678"
7      ],
8      "permissions": [
9        {
10         "actions": [],
11         "notActions": [],
12         "dataActions": [],
13         "notDataActions": []
14       }
15     ]
16   }
17 }
```

[Review + create](#)[Previous](#)[Next](#)

14. Replace these two lines with the `Actions` and `NotActions` from the appropriate role listed at [Manage access to an Azure Machine Learning workspace](#). Make sure to copy from `Actions` through the closing bracket, `]`.
15. Select **Save** at the top of the edit box to save your changes.

IMPORTANT

Don't select **Next** until you've saved your edits.

16. After you save your edits, select **Next**.
17. Select **Create** to create the custom role.
18. Select **OK**.

Add guest user

If your labelers are outside of your organization, you'll now add them so that they can access your workspace. If labelers are already inside your organization, skip this step.

To add a guest user, your organization's external collaboration settings must be configured to allow you to invite guests.

1. In [Azure portal](#), in the top-left corner, expand the menu and select **Azure Active Directory**.

The screenshot shows the left sidebar of the Azure portal. At the top is a red-bordered icon representing a menu or settings. Below it are several items with icons: a green plus sign for 'Create a resource', a blue house for 'Home', a teal dashboard icon for 'Dashboard', and a grey 'All services' icon. A horizontal line separates this from the 'FAVORITES' section, which also has a red star icon. The 'All services' section is expanded, showing a list of services with their respective icons: All resources (grid), Resource groups (cloud with gear), Quickstart Center (rocket), App Services (globe), Function App (lightning bolt), SQL databases (SQL server), Azure Cosmos DB (dynamodb), Virtual machines (monitor), Load balancers (diamond), Storage accounts (bar chart), Virtual networks (two monitors), Azure Active Directory (blue triangle), Monitor (clock), Advisor (cloud with gear), Security Center (key), Cost Management + Billing (coin), and Help + support (person). The 'Azure Active Directory' item is also highlighted with a red box.

- Create a resource
- Home
- Dashboard
- All services

★ **FAVORITES**

- All resources
- Resource groups
- Quickstart Center
- App Services
- Function App
- SQL databases
- Azure Cosmos DB
- Virtual machines
- Load balancers
- Storage accounts
- Virtual networks
- Azure Active Directory

2. On the left, select **Users**.
3. At the top, select **New user**.
4. Select **Invite external user**.
5. Fill in the name and email address for the user.
6. Add a message for the new user.
7. At the bottom of the page, select **Invite**.

Microsoft Azure (Preview) Search resources, services, and docs (G+)

Home > Microsoft > Users >

New user

Microsoft

Got feedback?

Bulk invite and create are now located under the 'Bulk operations' menu item on the 'All users' view. [View all users](#)

Select template

Create user
Create a new user in your organization.

Invite user
Invite a new guest user to collaborate with your organization. The user will be emailed an invitation they can accept in order to begin collaborating.

[Help me decide](#)

Identity

Email address *

chris@contoso.com

Personal message

Hi, Chris. Please accept this invitation so that you'll have access to our data labeling project!

Invite

Repeat for each of your labelers. Or use the link at the bottom of the **Invite user** box to invite multiple users in bulk.

TIP

Inform your labelers that they'll be receiving this email. They need to accept the invitation in order to gain access to your project.

Add users to your workspace

Now that you have your labelers added to the system, you're ready to add them to your workspace.

To add users to your workspace, you must be an owner of the workspace.

1. In [Azure portal](#), in the top search field, type **Machine Learning**.

2. Select **Machine Learning**.

The screenshot shows the Microsoft Azure portal interface. The search bar at the top has 'Machine Learning' typed into it. Below the search bar, the main content area is titled 'Machine Learning'. On the left sidebar, under 'Azure services', there is a 'Create a resource' button and a 'Virtual machines' section. Under 'Recent resources', there is a 'NAME' section. The main content area shows a list of services, with 'Machine Learning' highlighted by a red box. Other listed services include 'Machine Learning Studio (classic) workspaces', 'Machine Learning Studio (classic) web services', 'Machine Learning Studio (classic) web service plans', 'Virtual machines', 'CloudSimple Virtual Machines', 'SQL virtual machines', and 'Virtual machine scale sets'. To the right of the service list, there are sections for 'Marketplace', 'Documentation', and links to 'What is Azure Machine Learning | Microsoft' and 'What is automated ML / automl - Azure'.

3. Select the workspace that contains your data labeling project.

- On the left, select **Access control (IAM)**.
- At the top, select **+ Add > Add role assignment**.

The screenshot shows the 'Access control (IAM)' section of the Azure Machine Learning studio interface. On the left, there's a sidebar with various options like Overview, Activity log, Tags, Diagnose and solve problems, and Events. The 'Access control (IAM)' option is highlighted with a red box. At the top right, there's a search bar labeled 'Search (Ctrl+ /)', an 'Add' button (also highlighted with a red box), a 'Download role assignments' button, 'Edit columns' and 'Revert' buttons, and a 'My access' section with a 'View my access' button. A dropdown menu is open over the 'Add role assignment' button, showing 'Add role assignment' and 'Add co-administrator'.

- Select the **Labeler** or **Labeling Team Lead** role in the list. Use **Search** if necessary to find it.
- Select **Next**.
- In the middle of the page, next to **Members**, select the **+ Select members** link.
- Select each of the users you want to add. Use **Search** if necessary to find them.
- At the bottom of the page, select the **Select** button.
- Select **Next**.
- Verify that the **Role** is correct, and that your users appear in the **Members** list.
- Select **Review + assign**.

For your labelers

Your labelers are now all set up to begin labeling in your project. But they'll still need information from you to access the project.

If you haven't created your labeling project yet, do so before you contact your labelers.

- [Create an image labeling project](#).
- [Create a text labeling project \(preview\)](#)

Send the following to your labelers, after filling in your workspace and project names:

- Accept the invite from **Microsoft Invitations (invites@microsoft.com)**.
- Follow the steps on the web page after you accept. Don't worry if at the end you're on a page that says you don't have any apps.
- Open [Azure Machine Learning studio](#).
- Use the dropdown to select the workspace `<workspace-name>`.
- Select the **Label data** tool for `<project-name>`.

The screenshot shows the 'labeling-widgets' project in the Azure Machine Learning studio. At the top, there's a 'Project name' field containing 'labeling-widgets'. To the right of the project name is a star icon. Below the project name is a 'Label data' tool, which is highlighted with a red box. The 'Label data' tool has a checkmark icon inside it.

6. For more information about how to label data, see [Labeling images and text documents](#).

Next steps

- Learn more about [working with a data labeling vendor company](#)
- [Create an image labeling project and export labels](#)
- [Create a text labeling project and export labels \(preview\)](#)

Work with a data labeling vendor company

9/22/2022 • 3 minutes to read • [Edit Online](#)

Learn how to engage a data labeling vendor company to help you label your data. You can learn more about these companies and the labeling services they provide in their listing pages in [Azure Marketplace](#).

Workflow summary

Before you create your data labeling project:

1. Select a labeling service provider. To find a provider on Azure Marketplace:
 - a. Review the [listing details of these vendor labeling companies](#).
 - b. If the vendor labeling company meets your requirements, choose the **Contact Me** option in Azure Marketplace. Azure Marketplace will route your inquiry to the vendor labeling company. You may contact multiple vendor labeling companies before choosing the final company.
2. Contact and enter into a contract with the labeling service provider.

Once the contract with the vendor labeling company is in place:

1. Create the labeling project in [Azure Machine Learning studio](#). For more details on creating a project, see how to create an [image labeling project](#) or [text labeling project](#).
2. You're not limited to using a data labeling provider from Azure Marketplace. But if you do use a provider from Azure Marketplace:
 - a. Select **Use a vendor labeling company from Azure Marketplace** in the workforce step.
 - b. Select the appropriate data labeling company in the dropdown.

NOTE

The vendor labeling company name cannot be changed after you create the labeling project.

3. For any provider, whether found through Azure Marketplace or elsewhere, enable access (`labeler` role, `techlead` role) to the vendor labeling company using Azure Role Based Access (RBAC). These roles will allow the company to access resources to annotate your data.

Select a company

Microsoft has identified some labeling service providers with knowledge and experience who may be able to meet your needs. You can learn about the labeling service providers and choose a provider, taking into account the needs and requirements of your project(s) in their listing pages in [Azure Marketplace](#).

IMPORTANT

You can learn more about these companies and the labeling services they provide in their listing pages in Azure Marketplace. You are responsible for any decision to use a labeling company that offers services through Azure Marketplace, and you should independently assess whether a labeling company and its experience, services, staffing, terms, etc. will meet your project requirements. You may contact a labeling company that offers services through Azure Marketplace using the **Contact me** option in Azure Marketplace, and you can expect to hear from a contacted company within three business days. You will contract with and make payment to the labeling company directly.

Microsoft periodically reviews the list of potential labeling service providers in Azure Marketplace and may add or remove providers from the list at any time.

- If a provider is removed, it won't affect any existing projects, or that company's access to those projects.
- If you use a provider who is no longer listed in Azure Marketplace, don't select the **Use a vendor labeling company from Azure Marketplace** option in your new project.
- A removed provider will no longer have a listing in Azure Marketplace.
- A removed provider will no longer be able to be contacted through Azure Marketplace.

You can engage multiple vendor labeling companies for various labeling project needs. Each project will be linked to one vendor labeling company.

Below are vendor labeling companies who might help in getting your data labeled using Azure Machine Learning data labeling services. View the [listing of vendor companies](#).

- [iSoftStone](#)
- [Quadrant Resource](#)

Enter into a contract

After you have selected the labeling company you want to work with, you need to enter into a contract directly with the labeling company setting forth the terms of your engagement. Microsoft is not a party to this agreement and plays no role in determining or negotiating its terms. Amounts payable under this agreement will be paid directly to the labeling company.

If you enable ML Assisted labeling in a labeling project, Microsoft will charge you separately for the compute resources consumed in connection with this service. All other charges associated with your use of Azure Machine Learning (such as storage of data used in your Azure Machine Learning workspace) are governed by the terms of your agreement with Microsoft.

Enable access

In order for the vendor labeling company to have access into your projects, you'll next [add them as labelers to your project](#). If you are planning to use multiple vendor labeling companies for different labeling projects, we recommend you create separate workspaces for each company.

IMPORTANT

You, and not Microsoft, are responsible for all aspects of your engagement with a labeling company, including but not limited to issues relating to scope, quality, schedule, and pricing.

Next steps

- [Create an image labeling project and export labels](#)
- [Create a text labeling project and export labels \(preview\)](#)
- [Add users to your data labeling project](#)

Create a training job with the job creation UI (preview)

9/22/2022 • 5 minutes to read • [Edit Online](#)

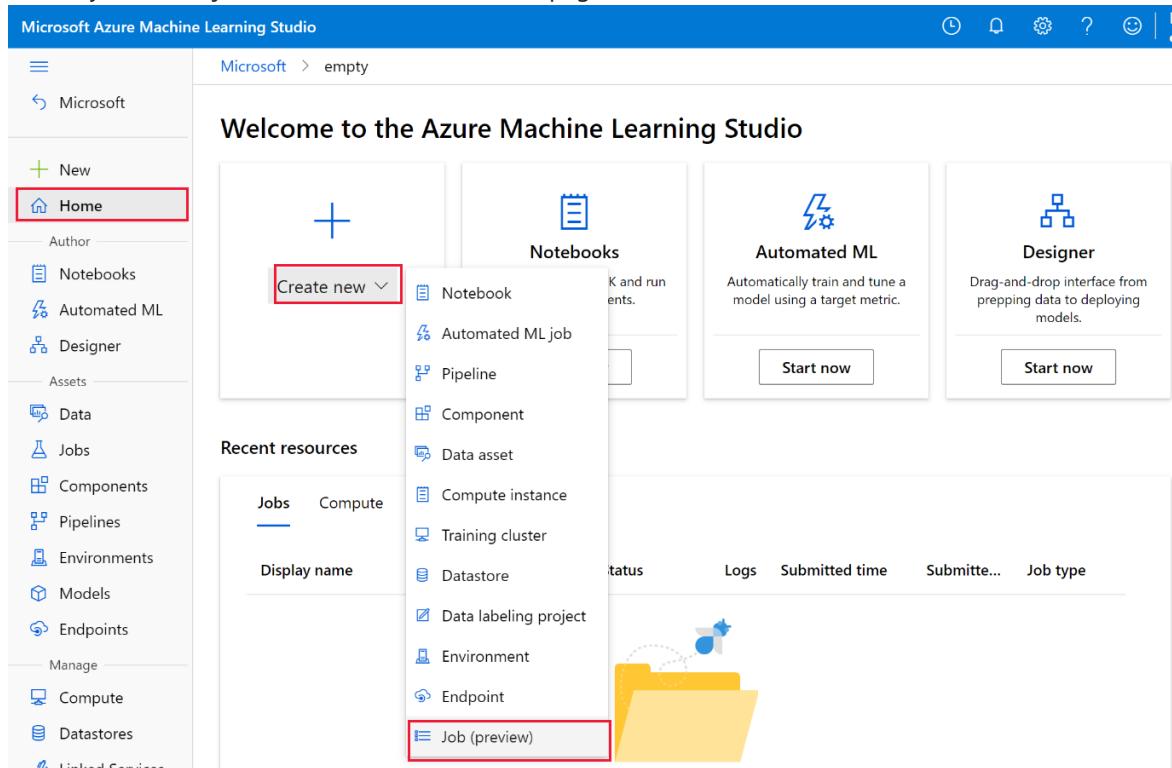
There are many ways to create a training job with Azure Machine Learning. You can use the CLI (see [Train models \(create jobs\) with the CLI \(v2\)](#)), the REST API (see [Train models with REST \(preview\)](#)), or you can use the UI to directly create a training job. In this article, you'll learn how to use your own data and code to train a machine learning model with the job creation UI in Azure Machine Learning studio.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- An Azure Machine Learning workspace. See [Create workspace resources](#).
- Understanding of what a job is in Azure Machine Learning. See [how to train models with the CLI \(v2\)](#).

Get started

1. Sign in to [Azure Machine Learning studio](#).
 2. Select your subscription and workspace.
- You may enter the job creation UI from the homepage. Click **Create new** and select **Job**.



- Or, you may enter the job creation from the left pane. Click **+ New** and select **Job**.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a navigation sidebar with a 'New' button highlighted by a red box. Below it are sections for Home, Author, Notebooks, Automated ML, Designer, Assets, Data, Jobs, Components, Pipelines, Environments, Models, and Endpoints. Under 'Endpoints', the 'Job (preview)' option is also highlighted by a red box. The main area is titled 'Welcome to the Azure Machine Learning Studio' and contains three cards: 'Notebooks' (code with Python SDK), 'Automated ML' (automatically train and tune a model), and 'Designer' (drag-and-drop interface). Below these cards is a table with columns for Models, Data, Experiment, Status, Logs, Submitted time, Submitted..., and Job type. A large yellow folder icon is centered at the bottom.

These options will all take you to the job creation panel, which has a wizard for configuring and creating a training job.

Select compute resources

The first step in the job creation UI is to select the compute target on which you'd like your job to run. The job creation UI supports several compute types:

COMPUTE TYPE	INTRODUCTION
Compute instance	What is an Azure Machine Learning compute instance?
Compute cluster	What is a compute cluster?
Attached Compute (Kubernetes cluster)	Configure and attach Kubernetes cluster anywhere (preview).

1. Select a compute type
2. Select an existing compute resource. The dropdown shows the node information and SKU type to help your choice.
3. For a compute cluster or a Kubernetes cluster, you may also specify how many nodes you want for the job in **Instance count**. The default number of instances is 1.
4. When you're satisfied with your choices, choose **Next**.

Create training job

A Job specifies a cloud resource which executes a given command in a specific virtual environment.

X

Compute

Environment

Job settings

Review

Compute

Select an existing compute target

Select compute type *

Azure ML compute cluster

Select compute cluster

cpu-cluster - Succeeded (Dedicated)

Instance count

1

Max number of instances: 6

If you don't input any number, we will set the instance count as 1 by default.

Back

Next

Cancel

If you're using Azure Machine Learning for the first time, you'll see an empty list and a link to create a new compute.

Create training job

A Job specifies a cloud resource which executes a given command in a specific virtual environment.

×

Compute

Select an existing compute target

Select compute type *

Azure ML compute instance

Select Azure ML compute instance

No computes found

You have no compute instances available for use.
[Create a new compute instance](#)

Back Next Cancel

For more information on creating the various types, see:

COMPUTE TYPE	HOW TO
Compute instance	Create and manage an Azure Machine Learning compute instance
Compute cluster	Create an Azure Machine Learning compute cluster
Attached Kubernetes cluster	Attach an Azure Arc-enabled Kubernetes cluster

Specify the necessary environment

After selecting a compute target, you need to specify the runtime environment for your job. The job creation UI supports three types of environment:

- Curated environments
- Custom environments
- Container registry image

Curated environments

Curated environments are Azure-defined collections of Python packages used in common ML workloads.

Curated environments are available in your workspace by default. These environments are backed by cached Docker images, which reduce the job preparation overhead. The cards displayed in the "Curated environments" page show details of each environment. To learn more, see [curated environments in Azure Machine Learning](#).

Create training job

A Job specifies a cloud resource which executes a given command in a specific virtual environment.

×

Compute

Environment

Job settings

Review

Environment

Select environment type

Curated environments

Search curated environments

Choose an environment:

AzureML-lightgbm-3.2-ubuntu18.04...

An environment for machine learning with Scikit-learn, LightGBM, XGBoost, Dask containing AzureML Python SDK and additional packages.

[Collapse](#)

AzureML-pytorch-1.7-ubuntu18.04-...

An environment for deep learning with PyTorch containing the Azure ML SDK and additional python packages.

AzureML-sklearn-0.24-ubuntu18.04...

An environment for tasks such as regression, clustering, and classification with Scikit-learn. Contains the Azure ML SDK and additional python packages.

[Collapse](#)

AzureML-tensorflow-2.4-ubuntu18....

An environment for deep learning with Tensorflow containing the Azure ML SDK and additional python packages.

Back

Next

Cancel

Custom environments

Custom environments are environments you've specified yourself. You can specify an environment or reuse an environment that you've already created. To learn more, see [Manage software environments in Azure Machine Learning studio \(preview\)](#).

Container registry image

If you don't want to use the Azure Machine Learning curated environments or specify your own custom environment, you can use a docker image from a public container registry such as [Docker Hub](#). If the image is in a private container, toggle **This is a private container registry**. For private registries, you will need to enter a valid username and password so Azure can get the image.

Create training job

A Job specifies a cloud resource which executes a given command in a specific virtual environment.

- Compute
- Environment
- Job settings
- Review

Environment

Select environment type

Container registry image

Container registry image path *

This is a private container registry

Username

Password

Back
Next
Cancel

Configure your job

After specifying the environment, you can configure your job with more settings.

FIELD	DESCRIPTION
Job name	The job name field is used to uniquely identify your job. It's also used as the display name for your job. Setting this field is optional; Azure will generate a GUID name for the job if you don't enter anything. Note: the job name must be unique.
Experiment name	This helps organize the job in Azure Machine Learning studio. Each job's run record will be organized under the corresponding experiment in the studio's "Experiment" tab. By default, Azure will put the job in the Default experiment.
Code	You can upload a code file or a folder from your machine, or upload a code file from the workspace's default blob storage. Azure will show the files to be uploaded after you make the selection.
Command	The command to execute. Command-line arguments can be explicitly written into the command or inferred from other sections, specifically inputs using curly braces notation, as discussed in the next section.

FIELD	DESCRIPTION
Inputs	Specify the input binding. We support three types of inputs: 1) Azure Machine Learning registered dataset; 2) workspace default blob storage; 3) upload local file. You can add multiple inputs.
Environment variables	Setting environment variables allows you to provide dynamic configuration of the job. You can add the variable and value here.
Tags	Add tags to your job to help with organization.

Specify code and inputs in the command box

Code

The command is run from the root directory of the uploaded code folder. After you select your code file or folder, you can see the files to be uploaded. Copy the relative path to the code containing your entry point and paste it into the box labeled **Enter the command to start the job**.

If the code is in the root directory, you can directly refer to it in the command. For instance, `python main.py`.

If the code isn't in the root directory, you should use the relative path. For example, the structure of the [word language model](#) is:

```
.
├── job.yml
├── data
└── src
    └── main.py
```

Here, the source code is in the `src` subdirectory. The command would be `python ./src/main.py` (plus other command-line arguments).

Code

Choose code location *

▼

Upload file

Upload folder

word-language-model (11 files, 12.59 MiB)

Files to be uploaded

README.md
src/data.py
src/generate.py
src/main.py
src/model.py
src/README.md

Enter the command to start the job *

python ./src/main.py --cuda --epoch 5 --model Transformer --lr 5 --data\${{inputs.wiki}} --save outputs/model.pt

The command will run from the root of the uploaded code folder. Add any parameters and input references as needed.

Inputs

When you use an input in the command, you need to specify the input name. To indicate an input variable, use the form `${{{inputs.input_name}}}`. For instance, `${{{inputs.wiki}}}`. You can then refer to it in the command, for instance, `--data ${{{inputs.wiki}}}`.

Enter the command to start the job *

```
python ./src/main.py --cuda --epoch 5 --model Transformer --lr 5 --data ${{inputs.wiki}} --save outputs/model.pt
```

The command will run from the root of the uploaded code folder. Add any parameters and input references as needed.

Inputs

\${{inputs.wiki}}  

Input name *

wiki

Input type *

Data

Data type *

File

Data source *

Azure ML registered data

Data name * 

wikitext-2

Review and Create

Once you've configured your job, choose **Next** to go to the **Review** page. To modify a setting, choose the pencil icon and make the change.

You may choose **view the YAML spec** to review and download the yaml file generated by this job configuration. This job yaml file can be used to submit the job from the CLI (v2). (See [Train models \(create jobs\) with the CLI \(v2\)](#).)

Create training job

A Job specifies a cloud resource which executes a given command in a specific virtual environment.

Review job specification
Review the job before submitting it. You can also [view the YAML spec](#) as well.

Compute

Environment

Job settings

Review

Compute	Settings
Compute type AmlCompute	Job name demo
Instance gpu-cluster	Experiment name Default
	Code word-language-model (11 files, 12.59 MiB)
	Command python ./src/main.py --cuda --epochs 5 --model Transformer --lr 5 --data ./data/wikitext-2 --save outputs/model.pt
Environment	Inputs
Name AzureML-Pytorch1.7-Cuda11-OpenMp4.1.0-py36	Interaction Endpoints
Version 1	Environment variables --
	Tags amlk8s status:Succeeded Queue Information:Job is in scheduling state, at 05/20/2021 23:44:18 +00:00 mlflow.source.type:JOB mlflow.source.name:null amlk8s job id:4cfad7076129962ee70c36839a1e3e15

[Back](#) [Create](#) [Cancel](#)

Create training job
A Job specifies a cloud resource which executes a given command in a specific virtual environment.

Compute

Environment

Job settings

Review

[← Back to job specification review](#)

YAML job specification
Review the YAML job specification.

[Download YAML](#)

```
1  name: demo
2  environment: >-
3  |   azureml:/subscriptions/b0d8b749-c65c-408d-b6e1-d5cc0261c258/resourceGroups/
4  |   amlk8s-agent-rg/providers/Microsoft.MachineLearningServices/workspaces/
5  |   amlk8s-eastus2euap/environments/AzureML-Pytorch1.7-Cuda11-OpenMp4.1.0-py36/
6  |   versions/1
7  command: >-
8  |   python ./src/main.py --cuda --epochs 5 --model Transformer --lr 5 --data
9  |   ./data/wikitext-2 --save outputs/model.pt
10 |   compute:
11 |     target: >-
12 |       azureml:/subscriptions/b0d8b749-c65c-408d-b6e1-d5cc0261c258/resourceGroups/
13 |       amlk8s-agent-rg/providers/Microsoft.MachineLearningServices/workspaces/
14 |       amlk8s-eastus2euap/computes/gpu-cluster
15 |     instance_count: 2
16 |     experiment_name: Default
17 |     code: >-
18 |       azureml:/subscriptions/b0d8b749-c65c-408d-b6e1-d5cc0261c258/resourceGroups/
19 |       amlk8s-agent-rg/providers/Microsoft.MachineLearningServices/workspaces/
20 |       amlk8s-eastus2euap/codes/06-23-2021_054120_UTC/versions/1
21 |     tags:
22 |       amlk8s status: Succeeded
23 |       Queue Information: 'Job is in scheduling state, at 05/20/2021 23:44:18 +00:00'
24 |       mlflow.source.type: JOB
25 |       mlflow.source.name: null
26 |       amlk8s job id: 4cfad7076129962ee70c36839a1e3e15
```

[Back](#) [Create](#) [Cancel](#)

To launch the job, choose **Create**. Once the job is created, Azure will show you the job details page, where you can monitor and manage your training job.

How to configure emails in the studio (preview)

To start receiving emails when your job, online endpoint, or batch endpoint is complete or if there's an issue (failed, canceled), follow the preceding instructions.

1. In Azure ML studio, go to settings by selecting the gear icon.
2. Select the **Email notifications** tab.
3. Toggle to enable or disable email notifications for a specific event.



Settings

X

Language

Email Notifications

These settings apply across all workspaces you have access to

Job succeeded

Send me an email when my job succeeded



Enabled

Issue with my job

Send me an email when my job failed or cancelled



Enabled

Issue with my endpoint deployment

Send me an email when my inferencing endpoint deployment failed or cancelled



Enabled

Endpoint deployment succeeded

Send me an email when my inferencing endpoint deployment succeeded



Enabled

Next steps

- Deploy and score a machine learning model with a managed online endpoint (preview).
- Train models (create jobs) with the CLI (v2)

Train models with the Azure ML Python SDK v2 (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you learn how to configure and submit Azure Machine Learning jobs to train your models. Snippets of code explain the key parts of configuration and submission of a training job. Then use one of the [example notebooks](#) to find the full end-to-end working examples.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today
- The Azure Machine Learning [SDK v2 for Python](#)
- An Azure Machine Learning workspace

Clone examples repository

To run the training examples, first clone the examples repository and change into the `sdk` directory:

```
git clone --depth 1 https://github.com/Azure/azureml-examples  
cd azureml-examples/sdk
```

TIP

Use `--depth 1` to clone only the latest commit to the repository, which reduces time to complete the operation.

Start on your local machine

Start by running a script, which trains a model using `lightgbm`. The script file is available [here](#). The script needs three inputs

- *input data*: You'll use data from a web location for your run - [web location](#). In this example, we're using a file in a remote location for brevity, but you can use a local file as well.
- *learning-rate*: You'll use a learning rate of `0.9`
- *boosting*: You'll use the Gradient Boosting `gdbt`

Run this script file as follows

```
cd jobs/single-step/lightgbm/iris

python src/main.py --iris-csv https://azurermlexamples.blob.core.windows.net/datasets/iris.csv --learning-rate 0.9 --boosting gbdt
```

The output expected is as follows:

```
2022/04/21 15:02:44 INFO mlflow.tracking.fluent: Autologging successfully enabled for lightgbm.
2022/04/21 15:02:44 INFO mlflow.tracking.fluent: Autologging successfully enabled for sklearn.
2022/04/21 15:02:45 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID
'a1d5f652796e4d88961176166de52253', which will track hyperparameters, performance metrics, model artifacts,
and lineage information for the current lightgbm workflow
lightgbm\engine.py:177: UserWarning: Found `num_iterations` in params. Will use it instead of argument
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000164 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Move to the cloud

Now that the local run works, move this run to an Azure Machine Learning workspace. To run this on Azure ML, you need:

- A workspace to run
- A compute on which to run it
- An environment on the compute to ensure you have the required packages to run your script

Let us tackle these steps below

1. Connect to the workspace

To connect to the workspace, you need identifier parameters - a subscription, resource group and workspace name. You'll use these details in the `MLClient` from `azure.ai.ml` to get a handle to the required Azure Machine Learning workspace. To authenticate, you use the [default Azure authentication](#). Check this [example](#) for more details on how to configure credentials and connect to a workspace.

```
#import required libraries
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

#Enter details of your AzureML workspace
subscription_id = '<SUBSCRIPTION_ID>'
resource_group = '<RESOURCE_GROUP>'
workspace = '<AZUREML_WORKSPACE_NAME>'

#connect to the workspace
ml_client = MLClient(DefaultAzureCredential(), subscription_id, resource_group, workspace)
```

2. Create compute

You'll create a compute called `cpu-cluster` for your job, with this code:

```

from azure.ai.ml.entities import AmlCompute

# specify aml compute name.
cpu_compute_target = "cpu-cluster"

try:
    ml_client.compute.get(cpu_compute_target)
except Exception:
    print("Creating a new cpu compute target...")
    compute = AmlCompute(
        name=cpu_compute_target, size="STANDARD_D2_V2", min_instances=0, max_instances=4
    )
    ml_client.compute.begin_create_or_update(compute)

```

3. Environment to run the script

To run your script on `cpu-cluster`, you need an environment, which has the required packages and dependencies to run your script. There are a few options available for environments:

- Use a curated environment in your workspace - Azure ML offers several curated [environments](#), which cater to various needs.
- Use a custom environment - Azure ML allows you to create your own environment using
 - A docker image
 - A base docker image with a conda YAML to customize further
 - A docker build context

Check this [example](#) on how to create custom environments.

You'll use a curated environment provided by Azure ML for `lightgbm` called `AzureML-lightgbm-3.2-ubuntu18.04-py37-cpu`

4. Submit a job to run the script

To run this script, you'll use a `command`. The command will be run by submitting it as a `job` to Azure ML.

```

from azure.ai.ml import command, Input

# define the command
command_job = command(
    code="./src",
    command="python main.py --iris-csv ${{inputs.iris_csv}} --learning-rate ${{inputs.learning_rate}} --
boosting ${{inputs.boosting}}",
    environment="AzureML-lightgbm-3.2-ubuntu18.04-py37-cpu@latest",
    inputs={
        "iris_csv": Input(
            type="uri_file",
            path="https://azuremlexamples.blob.core.windows.net/datasets/iris.csv",
        ),
        "learning_rate": 0.9,
        "boosting": "gbdt",
    },
    compute="cpu-cluster",
)

```

```

# submit the command
returned_job = ml_client.jobs.create_or_update(command_job)
# get a URL for the status of the job
returned_job.services["Studio"].endpoint

```

In the above, you configured:

- `code` - path where the code to run the command is located
- `command` - command that needs to be run
- `inputs` - dictionary of inputs using name value pairs to the command. The key is a name for the input within the context of the job and the value is the input value. Inputs are referenced in the `command` using the `${inputs.<input_name>}` expression. To use files or folders as inputs, you can use the `Input` class.

For more details, refer to the [reference documentation](#).

Improve the model using hyperparameter sweep

Now that you have run a job on Azure, let us make it better using Hyperparameter tuning. Also called hyperparameter optimization, this is the process of finding the configuration of hyperparameters that results in the best performance. Azure Machine Learning provides a `sweep` function on the `command` to do hyperparameter tuning.

To perform a sweep, there needs to be input(s) against which the sweep needs to be performed. These inputs can have a discrete or continuous value. The `sweep` function will run the `command` multiple times using different combination of input values specified. Each input is a dictionary of name value pairs. The key is the name of the hyperparameter and the value is the parameter expression.

Let us improve our model by sweeping on `learning_rate` and `boosting` inputs to the script. In the previous step, you used a specific value for these parameters, but now you'll use a range or choice of values.

```
# we will reuse the command_job created before. we call it as a function so that we can apply inputs
# we do not apply the 'iris_csv' input again -- we will just use what was already defined earlier
command_job_for_sweep = command_job(
    learning_rate=Uniform(min_value=0.01, max_value=0.9),
    boosting=Choice(values=["gbdt", "dart"]),
)
```

Now that you've defined the parameters, run the sweep

```
# apply the sweep parameter to obtain the sweep_job
sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm="random",
    primary_metric="test-multi_logloss",
    goal="Minimize",
)

# define the limits for this sweep
sweep_job.set_limits(max_total_trials=20, max_concurrent_trials=10, timeout=7200)
```

```
# submit the sweep
returned_sweep_job = ml_client.create_or_update(sweep_job)
# get a URL for the status of the job
returned_sweep_job.services["Studio"].endpoint
```

As seen above, the `sweep` function allows user to configure the following key aspects:

- `sampling_algorithm` - The hyperparameter sampling algorithm to use over the search_space. Allowed values are `random`, `grid` and `bayesian`.
- `objective` - the objective of the sweep
 - `primary_metric` - The name of the primary metric reported by each trial job. The metric must be logged in the user's training script using `mlflow.log_metric()` with the same corresponding metric

name.

- `goal` - The optimization goal of the objective.primary_metric. The allowed values are `maximize` and `minimize`.
- `compute` - Name of the compute target to execute the job on.
- `limits` - Limits for the sweep job

Once this job completes, you can look at the metrics and the job details in the [Azure ML Portal](#). The job details page will identify the best performing child run.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a sidebar with various options like New, Home, Notebooks, Automated ML, Designer, Data, Jobs (which is selected), Components, Pipelines, Environments, Models, Endpoints, Compute, Datastores, Linked Services, and Data Labeling. The main area shows a breadcrumb path: Microsoft > my-ws > Jobs > iris > teal_apple_2x94cy3vcf. Below the path are buttons for Refresh, Create model, Cancel, and Delete. There are tabs for Overview, Trials, Metrics, Outputs + logs, and Code. The Overview tab is selected. In the Properties section, it shows Status as Completed, Created on Jun 10, 2022 11:03 AM, Start time Jun 10, 2022 11:03 AM, Duration 9m 8.957s, Compute duration 9m 8.957s, Compute target sgilley-cluster, Name teal_apple_2x94cy3vcf, Created by Sheri Gilley, Job type Sweep, and Experiment. In the Parameter sampling section, it shows Sampling policy name RANDOM and Parameter space {"learning_rate": "uniform", [0.01, 0.9]}, {"boosting": ["choice", ["gbdt", "dart"]]}]. In the Early termination policy section, it shows Early termination policy DEFAULT and Properties {}. In the Primary metric section, it shows Primary metric name test-multi_logloss, Primary metric goal minimize, and Best child run teal_apple_2x94cy3vcf_5, which is highlighted with a red border.

Distributed training

Azure Machine Learning supports PyTorch, TensorFlow, and MPI-based distributed training. Let us look at how to configure a command for distribution for the `command_job` you created earlier

```
# Distribute using PyTorch
from azure.ai.ml import PyTorchDistribution
command_job.distribution = PyTorchDistribution(process_count_per_instance=4)

# Distribute using TensorFlow
from azure.ai.ml import TensorFlowDistribution
command_job.distribution = TensorFlowDistribution(parameter_server_count=1, worker_count=2)

# Distribute using MPI
from azure.ai.ml import MpiDistribution
job.distribution = MpiDistribution(process_count_per_instance=3)
```

Next steps

Try these next steps to learn how to use the Azure Machine Learning SDK (v2) for Python:

- Use pipelines with the Azure ML Python SDK (v2)

Hyperparameter tuning a model (v2)

9/22/2022 • 14 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current)

APPLIES TO: Python SDK azure-ai-ml v2 (preview)

Automate efficient hyperparameter tuning using Azure Machine Learning SDK v2 and CLI v2 by way of the SweepJob type.

1. Define the parameter search space for your trial
2. Specify the sampling algorithm for your sweep job
3. Specify the objective to optimize
4. Specify early termination policy for low-performing jobs
5. Define limits for the sweep job
6. Launch an experiment with the defined configuration
7. Visualize the training jobs
8. Select the best configuration for your model

What is hyperparameter tuning?

Hyperparameters are adjustable parameters that let you control the model training process. For example, with neural networks, you decide the number of hidden layers and the number of nodes in each layer. Model performance depends heavily on hyperparameters.

Hyperparameter tuning, also called **hyperparameter optimization**, is the process of finding the configuration of hyperparameters that results in the best performance. The process is typically computationally expensive and manual.

Azure Machine Learning lets you automate hyperparameter tuning and run experiments in parallel to efficiently optimize hyperparameters.

Define the search space

Tune hyperparameters by exploring the range of values defined for each hyperparameter.

Hyperparameters can be discrete or continuous, and has a distribution of values described by a [parameter expression](#).

Discrete hyperparameters

Discrete hyperparameters are specified as a `choice` among discrete values. `Choice` can be:

- one or more comma-separated values
- a `range` object
- any arbitrary `list` object

```

from azure.ai.ml.sweep import Choice

command_job_for_sweep = command_job(
    batch_size=Choice(values=[16, 32, 64, 128]),
    number_of_hidden_layers=Choice(values=range(1,5)),
)

```

In this case, `batch_size` one of the values [16, 32, 64, 128] and `number_of_hidden_layers` takes one of the values [1, 2, 3, 4].

The following advanced discrete hyperparameters can also be specified using a distribution:

- `QUniform(min_value, max_value, q)` - Returns a value like $\text{round}(\text{Uniform}(\text{min_value}, \text{max_value}) / q) * q$
- `QLogUniform(min_value, max_value, q)` - Returns a value like $\text{round}(\exp(\text{Uniform}(\text{min_value}, \text{max_value})) / q) * q$
- `QNormal(mu, sigma, q)` - Returns a value like $\text{round}(\text{Normal}(\mu, \sigma) / q) * q$
- `QLogNormal(mu, sigma, q)` - Returns a value like $\text{round}(\exp(\text{Normal}(\mu, \sigma)) / q) * q$

Continuous hyperparameters

The Continuous hyperparameters are specified as a distribution over a continuous range of values:

- `Uniform(min_value, max_value)` - Returns a value uniformly distributed between `min_value` and `max_value`
- `LogUniform(min_value, max_value)` - Returns a value drawn according to $\exp(\text{Uniform}(\text{min_value}, \text{max_value}))$ so that the logarithm of the return value is uniformly distributed
- `Normal(mu, sigma)` - Returns a real value that's normally distributed with mean `mu` and standard deviation `sigma`
- `LogNormal(mu, sigma)` - Returns a value drawn according to $\exp(\text{Normal}(\mu, \sigma))$ so that the logarithm of the return value is normally distributed

An example of a parameter space definition:

```

from azure.ai.ml.sweep import Normal, Uniform

command_job_for_sweep = command_job(
    learning_rate=Normal(mu=10, sigma=3),
    keep_probability=Uniform(min_value=0.05, max_value=0.1),
)

```

This code defines a search space with two parameters - `learning_rate` and `keep_probability`. `learning_rate` has a normal distribution with mean value 10 and a standard deviation of 3. `keep_probability` has a uniform distribution with a minimum value of 0.05 and a maximum value of 0.1.

For the CLI, you can use the [sweep job YAML schema](#), to define the search space in your YAML:

```

search_space:
  conv_size:
    type: choice
    values: [2, 5, 7]
  dropout_rate:
    type: uniform
    min_value: 0.1
    max_value: 0.2

```

Sampling the hyperparameter space

Specify the parameter sampling method to use over the hyperparameter space. Azure Machine Learning supports the following methods:

- Random sampling
- Grid sampling
- Bayesian sampling

Random sampling

Random sampling supports discrete and continuous hyperparameters. It supports early termination of low-performance jobs. Some users do an initial search with random sampling and then refine the search space to improve results.

In random sampling, hyperparameter values are randomly selected from the defined search space. After creating your command job, you can use the sweep parameter to define the sampling algorithm.

```
from azure.ai.ml.sweep import Normal, Uniform, RandomParameterSampling

command_job_for_sweep = command_job(
    learning_rate=Normal(mu=10, sigma=3),
    keep_probability=Uniform(min_value=0.05, max_value=0.1),
    batch_size=Choice(values=[16, 32, 64, 128]),
)

sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm = "random",
    ...
)
```

Sobol

Sobol is a type of random sampling supported by sweep job types. You can use sobol to reproduce your results using seed and cover the search space distribution more evenly.

To use sobol, use the RandomParameterSampling class to add the seed and rule as shown in the example below.

```
from azure.ai.ml.sweep import RandomParameterSampling

sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm = RandomParameterSampling(seed=123, rule="sobol"),
    ...
)
```

Grid sampling

Grid sampling supports discrete hyperparameters. Use grid sampling if you can budget to exhaustively search over the search space. Supports early termination of low-performance jobs.

Grid sampling does a simple grid search over all possible values. Grid sampling can only be used with choice hyperparameters. For example, the following space has six samples:

```

from azure.ai.ml.sweep import Choice

command_job_for_sweep = command_job(
    batch_size=Choice(values=[16, 32]),
    number_of_hidden_layers=Choice(values=[1,2,3]),
)

sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm = "grid",
    ...
)

```

Bayesian sampling

Bayesian sampling is based on the Bayesian optimization algorithm. It picks samples based on how previous samples did, so that new samples improve the primary metric.

Bayesian sampling is recommended if you have enough budget to explore the hyperparameter space. For best results, we recommend a maximum number of jobs greater than or equal to 20 times the number of hyperparameters being tuned.

The number of concurrent jobs has an impact on the effectiveness of the tuning process. A smaller number of concurrent jobs may lead to better sampling convergence, since the smaller degree of parallelism increases the number of jobs that benefit from previously completed jobs.

Bayesian sampling only supports `choice`, `uniform`, and `quniform` distributions over the search space.

```

from azure.ai.ml.sweep import Uniform, Choice

command_job_for_sweep = command_job(
    learning_rate=Uniform(min_value=0.05, max_value=0.1),
    batch_size=Choice(values=[16, 32, 64, 128]),
)

sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm = "bayesian",
    ...
)

```

Specify the objective of the sweep

Define the objective of your sweep job by specifying the primary metric and goal you want hyperparameter tuning to optimize. Each training job is evaluated for the primary metric. The early termination policy uses the primary metric to identify low-performance jobs.

- `primary_metric`: The name of the primary metric needs to exactly match the name of the metric logged by the training script
- `goal`: It can be either `Maximize` or `Minimize` and determines whether the primary metric will be maximized or minimized when evaluating the jobs.

```

from azure.ai.ml.sweep import Uniform, Choice

command_job_for_sweep = command_job(
    learning_rate=Uniform(min_value=0.05, max_value=0.1),
    batch_size=Choice(values=[16, 32, 64, 128]),
)

sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm = "bayesian",
    primary_metric="accuracy",
    goal="Maximize",
)

```

This sample maximizes "accuracy".

Log metrics for hyperparameter tuning

The training script for your model **must** log the primary metric during model training using the same corresponding metric name so that the SweepJob can access it for hyperparameter tuning.

Log the primary metric in your training script with the following sample snippet:

```

import mlflow
mlflow.log_metric("accuracy", float(val_accuracy))

```

The training script calculates the `val_accuracy` and logs it as the primary metric "accuracy". Each time the metric is logged, it's received by the hyperparameter tuning service. It's up to you to determine the frequency of reporting.

For more information on logging values for training jobs, see [Enable logging in Azure ML training jobs](#).

Specify early termination policy

Automatically end poorly performing jobs with an early termination policy. Early termination improves computational efficiency.

You can configure the following parameters that control when a policy is applied:

- `evaluation_interval` : the frequency of applying the policy. Each time the training script logs the primary metric counts as one interval. An `evaluation_interval` of 1 will apply the policy every time the training script reports the primary metric. An `evaluation_interval` of 2 will apply the policy every other time. If not specified, `evaluation_interval` is set to 0 by default.
- `delay_evaluation` : delays the first policy evaluation for a specified number of intervals. This is an optional parameter that avoids premature termination of training jobs by allowing all configurations to run for a minimum number of intervals. If specified, the policy applies every multiple of `evaluation_interval` that is greater than or equal to `delay_evaluation`. If not specified, `delay_evaluation` is set to 0 by default.

Azure Machine Learning supports the following early termination policies:

- [Bandit policy](#)
- [Median stopping policy](#)
- [Truncation selection policy](#)
- [No termination policy](#)

Bandit policy

[Bandit policy](#) is based on slack factor/slack amount and evaluation interval. Bandit policy ends a job when the

primary metric isn't within the specified slack factor/slack amount of the most successful job.

Specify the following configuration parameters:

- `slack_factor` or `slack_amount`: the slack allowed with respect to the best performing training job.
`slack_factor` specifies the allowable slack as a ratio. `slack_amount` specifies the allowable slack as an absolute amount, instead of a ratio.

For example, consider a Bandit policy applied at interval 10. Assume that the best performing job at interval 10 reported a primary metric is 0.8 with a goal to maximize the primary metric. If the policy specifies a `slack_factor` of 0.2, any training jobs whose best metric at interval 10 is less than 0.66 ($0.8/(1 + \text{slack_factor})$) will be terminated.

- `evaluation_interval` : (optional) the frequency for applying the policy
- `delay_evaluation` : (optional) delays the first policy evaluation for a specified number of intervals

```
from azure.ai.ml.sweep import BanditPolicy
sweep_job.early_termination = BanditPolicy(slack_factor = 0.1, delay_evaluation = 5, evaluation_interval = 1)
```

In this example, the early termination policy is applied at every interval when metrics are reported, starting at evaluation interval 5. Any jobs whose best metric is less than $(1/(1+0.1))$ or 91% of the best performing jobs will be terminated.

Median stopping policy

[Median stopping](#) is an early termination policy based on running averages of primary metrics reported by the jobs. This policy computes running averages across all training jobs and stops jobs whose primary metric value is worse than the median of the averages.

This policy takes the following configuration parameters:

- `evaluation_interval` : the frequency for applying the policy (optional parameter).
- `delay_evaluation` : delays the first policy evaluation for a specified number of intervals (optional parameter).

```
from azure.ai.ml.sweep import MedianStoppingPolicy
sweep_job.early_termination = MedianStoppingPolicy(delay_evaluation = 5, evaluation_interval = 1)
```

In this example, the early termination policy is applied at every interval starting at evaluation interval 5. A job is stopped at interval 5 if its best primary metric is worse than the median of the running averages over intervals 1:5 across all training jobs.

Truncation selection policy

[Truncation selection](#) cancels a percentage of lowest performing jobs at each evaluation interval. jobs are compared using the primary metric.

This policy takes the following configuration parameters:

- `truncation_percentage` : the percentage of lowest performing jobs to terminate at each evaluation interval. An integer value between 1 and 99.
- `evaluation_interval` : (optional) the frequency for applying the policy
- `delay_evaluation` : (optional) delays the first policy evaluation for a specified number of intervals
- `exclude_finished_jobs` : specifies whether to exclude finished jobs when applying the policy

```
from azure.ai.ml.sweep import TruncationSelectionPolicy
sweep_job.early_termination = TruncationSelectionPolicy(evaluation_interval=1, truncation_percentage=20,
delay_evaluation=5, exclude_finished_jobs=true)
```

In this example, the early termination policy is applied at every interval starting at evaluation interval 5. A job terminates at interval 5 if its performance at interval 5 is in the lowest 20% of performance of all jobs at interval 5 and will exclude finished jobs when applying the policy.

No termination policy (default)

If no policy is specified, the hyperparameter tuning service will let all training jobs execute to completion.

```
sweep_job.early_termination = None
```

Picking an early termination policy

- For a conservative policy that provides savings without terminating promising jobs, consider a Median Stopping Policy with `evaluation_interval` 1 and `delay_evaluation` 5. These are conservative settings that can provide approximately 25%-35% savings with no loss on primary metric (based on our evaluation data).
- For more aggressive savings, use Bandit Policy with a smaller allowable slack or Truncation Selection Policy with a larger truncation percentage.

Set limits for your sweep job

Control your resource budget by setting limits for your sweep job.

- `max_total_trials` : Maximum number of trial jobs. Must be an integer between 1 and 1000.
- `max_concurrent_trials` : (optional) Maximum number of trial jobs that can run concurrently. If not specified, all jobs launch in parallel. If specified, must be an integer between 1 and 100.
- `timeout` : Maximum time in seconds the entire sweep job is allowed to run. Once this limit is reached the system will cancel the sweep job, including all its trials.
- `trial_timeout` : Maximum time in seconds each trial job is allowed to run. Once this limit is reached the system will cancel the trial.

NOTE

If both `max_total_trials` and `max_concurrent_trials` are specified, the hyperparameter tuning experiment terminates when the first of these two thresholds is reached.

NOTE

The number of concurrent trial jobs is gated on the resources available in the specified compute target. Ensure that the compute target has the available resources for the desired concurrency.

```
sweep_job.set_limits(max_total_trials=20, max_concurrent_trials=4, timeout=1200)
```

This code configures the hyperparameter tuning experiment to use a maximum of 20 total trial jobs, running four trial jobs at a time with a timeout of 1200 seconds for the entire sweep job.

Configure hyperparameter tuning experiment

To configure your hyperparameter tuning experiment, provide the following:

- The defined hyperparameter search space
- Your sampling algorithm
- Your early termination policy
- Your objective
- Resource limits
- CommandJob or CommandComponent
- SweepJob

SweepJob can run a hyperparameter sweep on the Command or Command Component.

NOTE

The compute target used in `sweep_job` must have enough resources to satisfy your concurrency level. For more information on compute targets, see [Compute targets](#).

Configure your hyperparameter tuning experiment:

```

from azure.ai.ml import MLClient
from azure.ai.ml import command, Input
from azure.ai.ml.sweep import Choice, Uniform, MedianStoppingPolicy
from azure.identity import DefaultAzureCredential

# Create your base command job
command_job = command(
    code="./src",
    command="python main.py --iris-csv ${{inputs.iris_csv}} --learning-rate ${{inputs.learning_rate}} --
boosting ${{inputs.boosting}}",
    environment="AzureML-lightgbm-3.2-ubuntu18.04-py37-cpu@latest",
    inputs={
        "iris_csv": Input(
            type="uri_file",
            path="https://azurermlexamples.blob.core.windows.net/datasets/iris.csv",
        ),
        "learning_rate": 0.9,
        "boosting": "gbdt",
    },
    compute="cpu-cluster",
)

# Override your inputs with parameter expressions
command_job_for_sweep = command_job(
    learning_rate=Uniform(min_value=0.01, max_value=0.9),
    boosting=Choice(values=["gbdt", "dart"]),
)

# Call sweep() on your command job to sweep over your parameter expressions
sweep_job = command_job_for_sweep.sweep(
    compute="cpu-cluster",
    sampling_algorithm="random",
    primary_metric="test-multi_logloss",
    goal="Minimize",
)

# Specify your experiment details
sweep_job.display_name = "lightgbm-iris-sweep-example"
sweep_job.experiment_name = "lightgbm-iris-sweep-example"
sweep_job.description = "Run a hyperparameter sweep job for LightGBM on Iris dataset."

# Define the limits for this sweep
sweep_job.set_limits(max_total_trials=20, max_concurrent_trials=10, timeout=7200)

# Set early stopping on this one
sweep_job.early_termination = MedianStoppingPolicy(
    delay_evaluation=5, evaluation_interval=2
)

```

The `command_job` is called as a function so we can apply the parameter expressions to the sweep inputs. The `sweep` function is then configured with `trial`, `sampling-algorithm`, `objective`, `limits`, and `compute`. The above code snippet is taken from the sample notebook [Run hyperparameter sweep on a Command or CommandComponent](#). In this sample, the `learning_rate` and `boosting` parameters will be tuned. Early stopping of jobs will be determined by a `MedianStoppingPolicy`, which stops a job whose primary metric value is worse than the median of the averages across all training jobs.(see [MedianStoppingPolicy class reference](#)).

To see how the parameter values are received, parsed, and passed to the training script to be tuned, refer to this [code sample](#)

IMPORTANT

Every hyperparameter sweep job restarts the training from scratch, including rebuilding the model and *all the data loaders*. You can minimize this cost by using an Azure Machine Learning pipeline or manual process to do as much data preparation as possible prior to your training jobs.

Submit hyperparameter tuning experiment

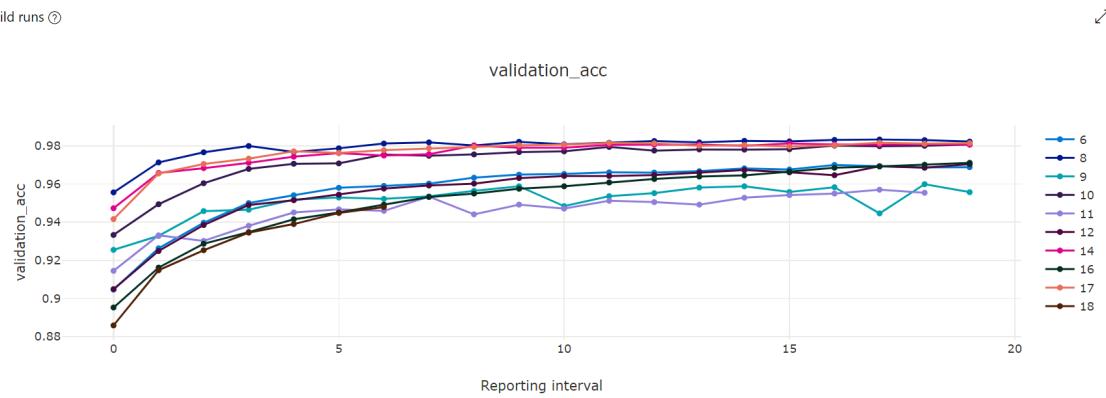
After you define your hyperparameter tuning configuration, [submit the experiment](#):

```
# submit the sweep
returned_sweep_job = ml_client.create_or_update(sweep_job)
# get a URL for the status of the job
returned_sweep_job.services["Studio"].endpoint
```

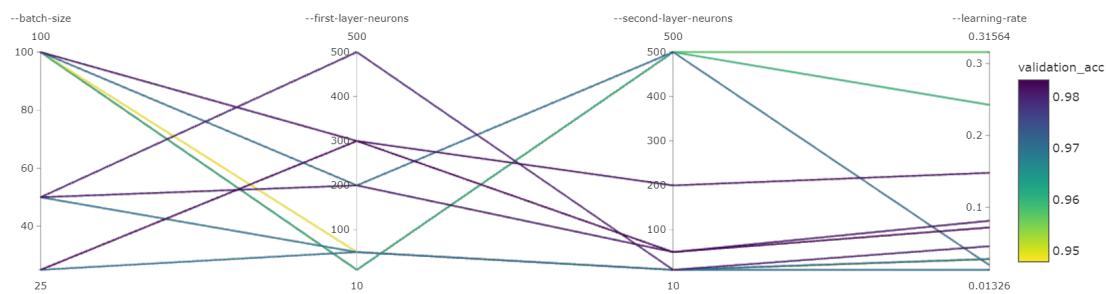
Visualize hyperparameter tuning jobs

You can visualize all of your hyperparameter tuning jobs in the [Azure Machine Learning studio](#). For more information on how to view an experiment in the portal, see [View job records in the studio](#).

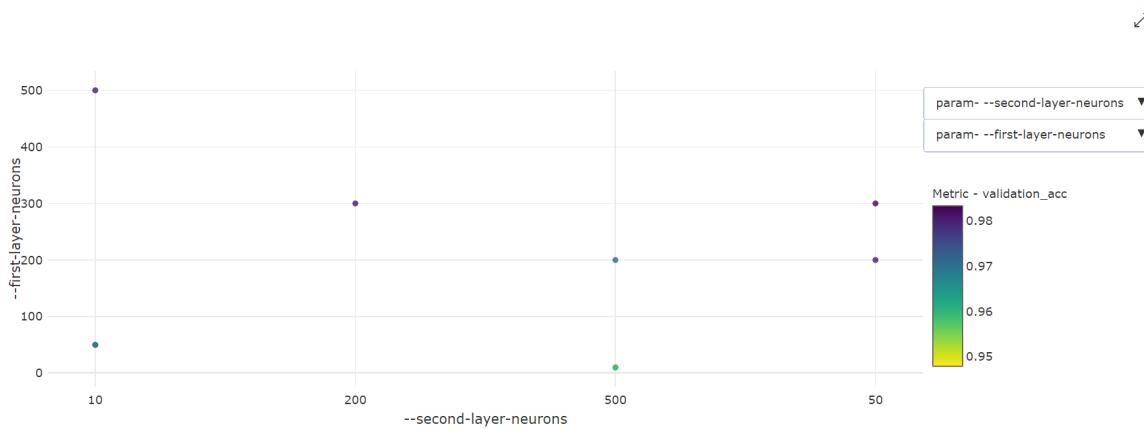
- **Metrics chart:** This visualization tracks the metrics logged for each hyperdrive child job over the duration of hyperparameter tuning. Each line represents a child job, and each point measures the primary metric value at that iteration of runtime.



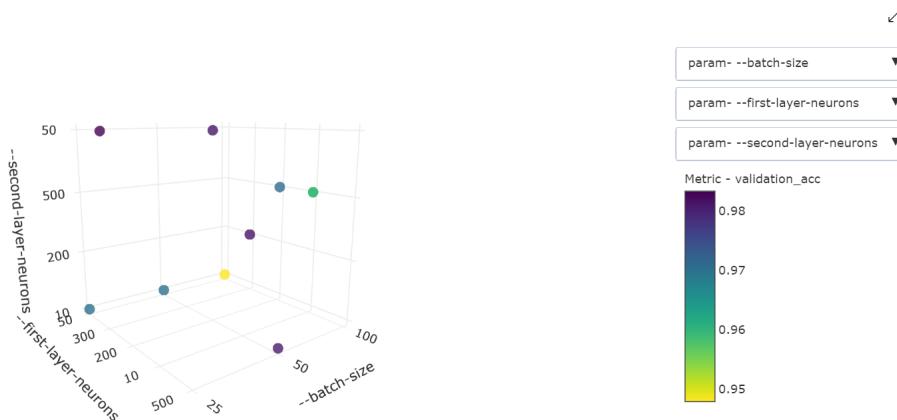
- **Parallel Coordinates Chart:** This visualization shows the correlation between primary metric performance and individual hyperparameter values. The chart is interactive via movement of axes (click and drag by the axis label), and by highlighting values across a single axis (click and drag vertically along a single axis to highlight a range of desired values). The parallel coordinates chart includes an axis on the rightmost portion of the chart that plots the best metric value corresponding to the hyperparameters set for that job instance. This axis is provided in order to project the chart gradient legend onto the data in a more readable fashion.



- **2-Dimensional Scatter Chart:** This visualization shows the correlation between any two individual hyperparameters along with their associated primary metric value.



- **3-Dimensional Scatter Chart:** This visualization is the same as 2D but allows for three hyperparameter dimensions of correlation with the primary metric value. You can also click and drag to reorient the chart to view different correlations in 3D space.



Find the best trial job

Once all of the hyperparameter tuning jobs have completed, retrieve your best trial outputs:

```
# Download best trial model output
ml_client.jobs.download(returned_sweep_job.name, output_name="model")
```

You can use the CLI to download all default and named outputs of the best trial job and logs of the sweep job.

```
az ml job download --name <sweep-job> --all
```

Optionally, to solely download the best trial output

```
az ml job download --name <sweep-job> --output-name model
```

References

- [Hyperparameter tuning example](#)
- [CLI \(v2\) sweep job YAML schema here](#)

Next steps

- [Track an experiment](#)
- [Deploy a trained model](#)

Distributed GPU training guide

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azureml v1](#)

Learn more about how to use distributed GPU training code in Azure Machine Learning (ML). This article will not teach you about distributed training. It will help you run your existing distributed training code on Azure Machine Learning. It offers tips and examples for you to follow for each framework:

- Message Passing Interface (MPI)
 - Horovod
 - DeepSpeed
 - Environment variables from Open MPI
- PyTorch
 - Process group initialization
 - Launch options
 - DistributedDataParallel (per-process-launch)
 - Using `torch.distributed.launch` (per-node-launch)
 - PyTorch Lightning
 - Hugging Face Transformers
- TensorFlow
 - Environment variables for TensorFlow (TF_CONFIG)
- Accelerate GPU training with InfiniBand

Prerequisites

Review these [basic concepts of distributed GPU training](#) such as *data parallelism*, *distributed data parallelism*, and *model parallelism*.

TIP

If you don't know which type of parallelism to use, more than 90% of the time you should use **Distributed Data Parallelism**.

MPI

Azure ML offers an [MPI job](#) to launch a given number of processes in each node. You can adopt this approach to run distributed training using either per-process-launcher or per-node-launcher, depending on whether `process_count_per_node` is set to 1 (the default) for per-node-launcher, or equal to the number of devices/GPUs for per-process-launcher. Azure ML constructs the full MPI launch command (`mpirun`) behind the scenes. You can't provide your own full head-node-launcher commands like `mpirun` or `DeepSpeed launcher`.

TIP

The base Docker image used by an Azure Machine Learning MPI job needs to have an MPI library installed. Open MPI is included in all the [AzureML GPU base images](#). When you use a custom Docker image, you are responsible for making sure the image includes an MPI library. Open MPI is recommended, but you can also use a different MPI implementation such as Intel MPI. Azure ML also provides [curated environments](#) for popular frameworks.

To run distributed training using MPI, follow these steps:

1. Use an Azure ML environment with the preferred deep learning framework and MPI. AzureML provides [curated environment](#) for popular frameworks.
2. Define `MpiConfiguration` with `process_count_per_node` and `node_count`. `process_count_per_node` should be equal to the number of GPUs per node for per-process-launch, or set to 1 (the default) for per-node-launch if the user script will be responsible for launching the processes per node.
3. Pass the `MpiConfiguration` object to the `distributed_job_config` parameter of `ScriptRunConfig`.

```
from azureml.core import Workspace, ScriptRunConfig, Environment, Experiment
from azureml.core.runconfig import MpiConfiguration

curated_env_name = 'AzureML-PyTorch-1.6-GPU'
pytorch_env = Environment.get(workspace=ws, name=curated_env_name)
distr_config = MpiConfiguration(process_count_per_node=4, node_count=2)

run_config = ScriptRunConfig(
    source_directory= './src',
    script='train.py',
    compute_target=compute_target,
    environment=pytorch_env,
    distributed_job_config=distr_config,
)

# submit the run configuration to start the job
run = Experiment(ws, "experiment_name").submit(run_config)
```

Horovod

Use the MPI job configuration when you use [Horovod](#) for distributed training with the deep learning framework.

Make sure your code follows these tips:

- The training code is instrumented correctly with Horovod before adding the Azure ML parts
- Your Azure ML environment contains Horovod and MPI. The PyTorch and TensorFlow curated GPU environments come pre-configured with Horovod and its dependencies.
- Create an `MpiConfiguration` with your desired distribution.

Horovod example

- [azureml-examples: TensorFlow distributed training using Horovod](#)

DeepSpeed

Don't use DeepSpeed's custom launcher to run distributed training with the [DeepSpeed](#) library on Azure ML. Instead, configure an MPI job to launch the training job [with MPI](#).

Make sure your code follows these tips:

- Your Azure ML environment contains DeepSpeed and its dependencies, Open MPI, and mpi4py.
- Create an `MpiConfiguration` with your distribution.

DeepSeed example

- [azureml-examples: Distributed training with DeepSpeed on CIFAR-10](#)

Environment variables from Open MPI

When running MPI jobs with Open MPI images, the following environment variables for each process launched:

1. `OMPI_COMM_WORLD_RANK` - the rank of the process
2. `OMPI_COMM_WORLD_SIZE` - the world size
3. `AZ_BATCH_MASTER_NODE` - primary address with port, `MASTER_ADDR:MASTER_PORT`
4. `OMPI_COMM_WORLD_LOCAL_RANK` - the local rank of the process on the node
5. `OMPI_COMM_WORLD_LOCAL_SIZE` - number of processes on the node

TIP

Despite the name, environment variable `OMPI_COMM_WORLD_NODE_RANK` does not correspond to the `NODE_RANK`. To use per-node-launcher, set `process_count_per_node=1` and use `OMPI_COMM_WORLD_RANK` as the `NODE_RANK`.

PyTorch

Azure ML supports running distributed jobs using PyTorch's native distributed training capabilities (`torch.distributed`).

TIP

For data parallelism, the [official PyTorch guidance](#) is to use `DistributedDataParallel` (DDP) over `DataParallel` for both single-node and multi-node distributed training. PyTorch also [recommends using `DistributedDataParallel` over the multiprocessing package](#). Azure Machine Learning documentation and examples will therefore focus on `DistributedDataParallel` training.

Process group initialization

The backbone of any distributed training is based on a group of processes that know each other and can communicate with each other using a backend. For PyTorch, the process group is created by calling `torch.distributed.init_process_group` in **all distributed processes** to collectively form a process group.

```
torch.distributed.init_process_group(backend='nccl', init_method='env://', ...)
```

The most common communication backends used are `mpi`, `nccl`, and `gloo`. For GPU-based training `nccl` is recommended for best performance and should be used whenever possible.

`init_method` tells how each process can discover each other, how they initialize and verify the process group using the communication backend. By default if `init_method` is not specified PyTorch will use the environment variable initialization method (`env://`). `init_method` is the recommended initialization method to use in your training code to run distributed PyTorch on Azure ML. PyTorch will look for the following environment variables for initialization:

- `MASTER_ADDR` - IP address of the machine that will host the process with rank 0.
- `MASTER_PORT` - A free port on the machine that will host the process with rank 0.
- `WORLD_SIZE` - The total number of processes. Should be equal to the total number of devices (GPU) used for distributed training.
- `RANK` - The (global) rank of the current process. The possible values are 0 to (world size - 1).

For more information on process group initialization, see the [PyTorch documentation](#).

Beyond these, many applications will also need the following environment variables:

- `LOCAL_RANK` - The local (relative) rank of the process within the node. The possible values are 0 to (# of processes on the node - 1). This information is useful because many operations such as data preparation only should be performed once per node --- usually on `local_rank` = 0.
- `NODE_RANK` - The rank of the node for multi-node training. The possible values are 0 to (total # of nodes - 1).

PyTorch launch options

The Azure ML PyTorch job supports two types of options for launching distributed training:

- **Per-process-launcher:** The system will launch all distributed processes for you, with all the relevant information (such as environment variables) to set up the process group.
- **Per-node-launcher:** You provide Azure ML with the utility launcher that will get run on each node. The utility launcher will handle launching each of the processes on a given node. Locally within each node, `RANK` and `LOCAL_RANK` are set up by the launcher. The `torch.distributed.launch` utility and PyTorch Lightning both belong in this category.

There are no fundamental differences between these launch options. The choice is largely up to your preference or the conventions of the frameworks/libraries built on top of vanilla PyTorch (such as Lightning or Hugging Face).

The following sections go into more detail on how to configure Azure ML PyTorch jobs for each of the launch options.

DistributedDataParallel (per-process-launch)

You don't need to use a launcher utility like `torch.distributed.launch`. To run a distributed PyTorch job:

1. Specify the training script and arguments
2. Create a `PyTorchConfiguration` and specify the `process_count` and `node_count`. The `process_count` corresponds to the total number of processes you want to run for your job. `process_count` should typically equal `# GPUs per node x # nodes`. If `process_count` isn't specified, Azure ML will by default launch one process per node.

Azure ML will set the `MASTER_ADDR`, `MASTER_PORT`, `WORLD_SIZE`, and `NODE_RANK` environment variables on each node, and set the process-level `RANK` and `LOCAL_RANK` environment variables.

To use this option for multi-process-per-node training, use Azure ML Python SDK `>= 1.22.0`. `Process_count` was introduced in 1.22.0.

```
from azureml.core import ScriptRunConfig, Environment, Experiment
from azureml.core.runconfig import PyTorchConfiguration

curated_env_name = 'AzureML-PyTorch-1.6-GPU'
pytorch_env = Environment.get(workspace=ws, name=curated_env_name)
distr_config = PyTorchConfiguration(process_count=8, node_count=2)

run_config = ScriptRunConfig(
    source_directory='./src',
    script='train.py',
    arguments=['--epochs', 50],
    compute_target=compute_target,
    environment=pytorch_env,
    distributed_job_config=distr_config,
)

run = Experiment(ws, 'experiment_name').submit(run_config)
```

TIP

If your training script passes information like local rank or rank as script arguments, you can reference the environment variable(s) in the arguments:

```
arguments=['--epochs', 50, '--local_rank', $LOCAL_RANK]
```

Pytorch per-process-launch example

- [azureml-examples: Distributed training with PyTorch on CIFAR-10](#)

Using `torch.distributed.launch` (per-node-launch)

PyTorch provides a launch utility in `torch.distributed.launch` that you can use to launch multiple processes per node. The `torch.distributed.launch` module spawns multiple training processes on each of the nodes.

The following steps demonstrate how to configure a PyTorch job with a per-node-launcher on Azure ML. The job achieves the equivalent of running the following command:

```
python -m torch.distributed.launch --nproc_per_node <num processes per node> \
--nnodes <num nodes> --node_rank $NODE_RANK --master_addr $MASTER_ADDR \
--master_port $MASTER_PORT --use_env \
<your training script> <your script arguments>
```

1. Provide the `torch.distributed.launch` command to the `command` parameter of the `ScriptRunConfig` constructor. Azure ML runs this command on each node of your training cluster. `--nproc_per_node` should be less than or equal to the number of GPUs available on each node. `MASTER_ADDR`, `MASTER_PORT`, and `NODE_RANK` are all set by Azure ML, so you can just reference the environment variables in the command. Azure ML sets `MASTER_PORT` to `6105`, but you can pass a different value to the `--master_port` argument of `torch.distributed.launch` command if you wish. (The launch utility will reset the environment variables.)
2. Create a `PyTorchConfiguration` and specify the `node_count`.

```
from azureml.core import ScriptRunConfig, Environment, Experiment
from azureml.core.runconfig import PyTorchConfiguration

curated_env_name = 'AzureML-PyTorch-1.6-GPU'
pytorch_env = Environment.get(workspace=ws, name=curated_env_name)
distr_config = PyTorchConfiguration(node_count=2)
launch_cmd = "python -m torch.distributed.launch --nproc_per_node 4 --nnodes 2 --node_rank $NODE_RANK --master_addr $MASTER_ADDR --master_port $MASTER_PORT --use_env train.py --epochs 50".split()

run_config = ScriptRunConfig(
    source_directory='./src',
    command=launch_cmd,
    compute_target=compute_target,
    environment=pytorch_env,
    distributed_job_config=distr_config,
)

run = Experiment(ws, 'experiment_name').submit(run_config)
```

TIP

Single-node multi-GPU training: If you are using the launch utility to run single-node multi-GPU PyTorch training, you do not need to specify the `distributed_job_config` parameter of `ScriptRunConfig`.

```
launch_cmd = "python -m torch.distributed.launch --nproc_per_node 4 --use_env train.py --epochs 50".split()

run_config = ScriptRunConfig(
    source_directory='./src',
    command=launch_cmd,
    compute_target=compute_target,
    environment=pytorch_env,
)
```

PyTorch per-node-launch example

- [azureml-examples: Distributed training with PyTorch on CIFAR-10](#)

PyTorch Lightning

[PyTorch Lightning](#) is a lightweight open-source library that provides a high-level interface for PyTorch. Lightning abstracts away many of the lower-level distributed training configurations required for vanilla PyTorch.

Lightning allows you to run your training scripts in single GPU, single-node multi-GPU, and multi-node multi-GPU settings. Behind the scene, it launches multiple processes for you similar to `torch.distributed.launch`.

For single-node training (including single-node multi-GPU), you can run your code on Azure ML without needing to specify a `distributed_job_config`. To run an experiment using multiple nodes with multiple GPUs, there are 2 options:

- Using PyTorch configuration (recommended): Define `PyTorchConfiguration` and specify `communication_backend="Nccl"`, `node_count`, and `process_count` (note that this is the total number of processes, ie, `num_nodes * process_count_per_node`). In Lightning Trainer module, specify both `num_nodes` and `gpus` to be consistent with `PyTorchConfiguration`. For example, `num_nodes = node_count` and `gpus = process_count_per_node`.
- Using MPI Configuration:
 - Define `MpiConfiguration` and specify both `node_count` and `process_count_per_node`. In Lightning Trainer, specify both `num_nodes` and `gpus` to be respectively the same as `node_count` and `process_count_per_node` from `MpiConfiguration`.
 - For multi-node training with MPI, Lightning requires the following environment variables to be set on each node of your training cluster:
 - `MASTER_ADDR`
 - `MASTER_PORT`
 - `NODE_RANK`
 - `LOCAL_RANK`

Manually set these environment variables that Lightning requires in the main training scripts:

```

import os
from argparse import ArgumentParser

def set_environment_variables_for_mpi(num_nodes, gpus_per_node, master_port=54965):
    if num_nodes > 1:
        os.environ["MASTER_ADDR"], os.environ["MASTER_PORT"] =
os.environ["AZ_BATCH_MASTER_NODE"].split(":")
    else:
        os.environ["MASTER_ADDR"] = os.environ["AZ_BATCHAI_MPI_MASTER_NODE"]
        os.environ["MASTER_PORT"] = str(master_port)

    try:
        os.environ["NODE_RANK"] = str(int(os.environ.get("OMPI_COMM_WORLD_RANK")) // gpus_per_node)
        # additional variables
        os.environ["MASTER_ADDRESS"] = os.environ["MASTER_ADDR"]
        os.environ["LOCAL_RANK"] = os.environ["OMPI_COMM_WORLD_LOCAL_RANK"]
        os.environ["WORLD_SIZE"] = os.environ["OMPI_COMM_WORLD_SIZE"]
    except:
        # fails when used with pytorch configuration instead of mpi
        pass

if __name__ == "__main__":
    parser = ArgumentParser()
    parser.add_argument("--num_nodes", type=int, required=True)
    parser.add_argument("--gpus_per_node", type=int, required=True)
    args = parser.parse_args()
    set_environment_variables_for_mpi(args.num_nodes, args.gpus_per_node)

    trainer = Trainer(
        num_nodes=args.num_nodes,
        gpus=args.gpus_per_node
    )

```

Lightning handles computing the world size from the Trainer flags `--gpus` and `--num_nodes`.

```

from azureml.core import ScriptRunConfig, Experiment
from azureml.core.runconfig import MpiConfiguration

nnodes = 2
gpus_per_node = 4
args = ['--max_epochs', 50, '--gpus_per_node', gpus_per_node, '--accelerator', 'ddp', '--num_nodes',
nnodes]
distr_config = MpiConfiguration(node_count=nnodes, process_count_per_node=gpus_per_node)

run_config = ScriptRunConfig(
    source_directory='./src',
    script='train.py',
    arguments=args,
    compute_target=compute_target,
    environment=pytorch_env,
    distributed_job_config=distr_config,
)

run = Experiment(ws, 'experiment_name').submit(run_config)

```

Hugging Face Transformers

Hugging Face provides many [examples](#) for using its Transformers library with `torch.distributed.launch` to run distributed training. To run these examples and your own custom training scripts using the Transformers Trainer API, follow the [Using `torch.distributed.launch`](#) section.

Sample job configuration code to fine-tune the BERT large model on the text classification MNLI task using the `run_glue.py` script on one node with 8 GPUs:

```

from azureml.core import ScriptRunConfig
from azureml.core.runconfig import PyTorchConfiguration

distr_config = PyTorchConfiguration() # node_count defaults to 1
launch_cmd = "python -m torch.distributed.launch --nproc_per_node 8 text-classification/run_glue.py --
model_name_or_path bert-large-uncased-whole-word-masking --task_name mnli --do_train --do_eval --
max_seq_length 128 --per_device_train_batch_size 8 --learning_rate 2e-5 --num_train_epochs 3.0 --output_dir
/tmp/mnli_output".split()

run_config = ScriptRunConfig(
    source_directory='./src',
    command=launch_cmd,
    compute_target=compute_target,
    environment=pytorch_env,
    distributed_job_config=distr_config,
)

```

You can also use the [per-process-launch](#) option to run distributed training without using `torch.distributed.launch`. One thing to keep in mind if using this method is that the transformers [TrainingArguments](#) expect the local rank to be passed in as an argument (`--local_rank`). `torch.distributed.launch` takes care of this when `--use_env=False`, but if you are using per-process-launch you'll need to explicitly pass the local rank in as an argument to the training script `--local_rank=$LOCAL_RANK` as Azure ML only sets the `LOCAL_RANK` environment variable.

TensorFlow

If you're using [native distributed TensorFlow](#) in your training code, such as TensorFlow 2.x's `tf.distribute.Strategy` API, you can launch the distributed job via Azure ML using the `TensorflowConfiguration`.

To do so, specify a `TensorflowConfiguration` object to the `distributed_job_config` parameter of the `ScriptRunConfig` constructor. If you're using `tf.distribute.experimental.MultiWorkerMirroredStrategy`, specify the `worker_count` in the `TensorflowConfiguration` corresponding to the number of nodes for your training job.

```

from azureml.core import ScriptRunConfig, Environment, Experiment
from azureml.core.runconfig import TensorflowConfiguration

curated_env_name = 'AzureML-TensorFlow-2.3-GPU'
tf_env = Environment.get(workspace=ws, name=curated_env_name)
distr_config = TensorflowConfiguration(worker_count=2, parameter_server_count=0)

run_config = ScriptRunConfig(
    source_directory='./src',
    script='train.py',
    compute_target=compute_target,
    environment=tf_env,
    distributed_job_config=distr_config,
)

# submit the run configuration to start the job
run = Experiment(ws, "experiment_name").submit(run_config)

```

If your training script uses the parameter server strategy for distributed training, such as for legacy TensorFlow 1.x, you'll also need to specify the number of parameter servers to use in the job, for example,

```
tf_config = TensorflowConfiguration(worker_count=2, parameter_server_count=1).
```

TF_CONFIG

In TensorFlow, the `TF_CONFIG` environment variable is required for training on multiple machines. For TensorFlow jobs, Azure ML will configure and set the `TF_CONFIG` variable appropriately for each worker before

executing your training script.

You can access TF_CONFIG from your training script if you need to: `os.environ['TF_CONFIG']`.

Example TF_CONFIG set on a chief worker node:

```
TF_CONFIG='{
  "cluster": {
    "worker": ["host0:2222", "host1:2222"]
  },
  "task": {"type": "worker", "index": 0},
  "environment": "cloud"
}'
```

TensorFlow example

- [azureml-examples: Distributed TensorFlow training with MultiWorkerMirroredStrategy](#)

Accelerating distributed GPU training with InfiniBand

As the number of VMs training a model increases, the time required to train that model should decrease. The decrease in time, ideally, should be linearly proportional to the number of training VMs. For instance, if training a model on one VM takes 100 seconds, then training the same model on two VMs should ideally take 50 seconds. Training the model on four VMs should take 25 seconds, and so on.

InfiniBand can be an important factor in attaining this linear scaling. InfiniBand enables low-latency, GPU-to-GPU communication across nodes in a cluster. InfiniBand requires specialized hardware to operate. Certain Azure VM series, specifically the NC, ND, and H-series, now have RDMA-capable VMs with SR-IOV and InfiniBand support. These VMs communicate over the low latency and high-bandwidth InfiniBand network, which is much more performant than Ethernet-based connectivity. SR-IOV for InfiniBand enables near bare-metal performance for any MPI library (MPI is used by many distributed training frameworks and tooling, including NVIDIA's NCCL software.) These SKUs are intended to meet the needs of computationally intensive, GPU-accelerated machine learning workloads. For more information, see [Accelerating Distributed Training in Azure Machine Learning with SR-IOV](#).

Typically, VM SKUs with an 'r' in their name contain the required InfiniBand hardware, and those without an 'r' typically do not. ('r' is a reference to RDMA, which stands for "remote direct memory access.") For instance, the VM SKU `Standard_NC24rs_v3` is InfiniBand-enabled, but the SKU `Standard_NC24s_v3` is not. Aside from the InfiniBand capabilities, the specs between these two SKUs are largely the same – both have 24 cores, 448 GB RAM, 4 GPUs of the same SKU, etc. [Learn more about RDMA- and InfiniBand-enabled machine SKUs](#).

WARNING

The older-generation machine SKU `Standard_NC24r` is RDMA-enabled, but it does not contain SR-IOV hardware required for InfiniBand.

If you create an `AmlCompute` cluster of one of these RDMA-capable, InfiniBand-enabled sizes, the OS image will come with the Mellanox OFED driver required to enable InfiniBand preinstalled and preconfigured.

Next steps

- [Deploy machine learning models to Azure](#)
- [Deploy and score a machine learning model by using a managed online endpoint \(preview\)](#)
- [Reference architecture for distributed deep learning training in Azure](#)

Train scikit-learn models at scale with Azure Machine Learning

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, learn how to run your scikit-learn training scripts with Azure Machine Learning.

The example scripts in this article are used to classify iris flower images to build a machine learning model based on scikit-learn's [iris dataset](#).

Whether you're training a machine learning scikit-learn model from the ground-up or you're bringing an existing model into the cloud, you can use Azure Machine Learning to scale out open-source training jobs using elastic cloud compute resources. You can build, deploy, version, and monitor production-grade models with Azure Machine Learning.

Prerequisites

You can run this code in either an Azure Machine Learning compute instance, or your own Jupyter Notebook:

- Azure Machine Learning compute instance
 - Complete the [Quickstart: Get started with Azure Machine Learning](#) to create a compute instance. Every compute instance includes a dedicated notebook server pre-loaded with the SDK and the notebooks sample repository.
 - Select the notebook tab in the Azure Machine Learning studio. In the samples training folder, find a completed and expanded notebook by navigating to this directory: **how-to-use-azureml > ml-frameworks > scikit-learn > train-hyperparameter-tune-deploy-with-sklearn** folder.
 - You can use the pre-populated code in the sample training folder to complete this tutorial.
- Create a Jupyter Notebook server and run the code in the following sections.
 - [Install the Azure Machine Learning SDK \(>= 1.13.0\)](#).
 - [Create a workspace configuration file](#).

Set up the experiment

This section sets up the training experiment by loading the required Python packages, initializing a workspace, defining the training environment, and preparing the training script.

Initialize a workspace

The [Azure Machine Learning workspace](#) is the top-level resource for the service. It provides you with a centralized place to work with all the artifacts you create. In the Python SDK, you can access the workspace artifacts by creating a `workspace` object.

Create a workspace object from the `config.json` file created in the [prerequisites section](#).

```
from azureml.core import Workspace  
  
ws = Workspace.from_config()
```

Prepare scripts

In this tutorial, the [training script `train_iris.py`](#) is already provided for you. In practice, you should be able to take any custom training script as is and run it with Azure ML without having to modify your code.

Notes:

- The provided training script shows how to log some metrics to your Azure ML run using the `Run` object within the script.
- The provided training script uses example data from the `iris = datasets.load_iris()` function. To use and access your own data, see [how to train with datasets](#) to make data available during training.

Define your environment

To define the Azure ML [Environment](#) that encapsulates your training script's dependencies, you can either define a custom environment or use an Azure ML curated environment.

Use a curated environment

Optionally, Azure ML provides prebuilt, [curated environments](#) if you don't want to define your own environment.

If you want to use a curated environment, you can run the following command instead:

```
from azureml.core import Environment  
  
sklearn_env = Environment.get(workspace=ws, name='AzureML-Tutorial')
```

Create a custom environment

You can also create your own custom environment. Define your conda dependencies in a YAML file; in this example the file is named `conda_dependencies.yml`.

```
dependencies:  
  - python=3.6.2  
  - scikit-learn  
  - numpy  
  - pip:  
    - azureml-defaults
```

Create an Azure ML environment from this Conda environment specification. The environment will be packaged into a Docker container at runtime.

```
from azureml.core import Environment  
  
sklearn_env = Environment.from_conda_specification(name='sklearn-env', file_path='conda_dependencies.yml')
```

For more information on creating and using environments, see [Create and use software environments in Azure Machine Learning](#).

Configure and submit your training run

Create a ScriptRunConfig

Create a `ScriptRunConfig` object to specify the configuration details of your training job, including your training script, environment to use, and the compute target to run on. Any arguments to your training script will be passed via command line if specified in the `arguments` parameter.

The following code will configure a `ScriptRunConfig` object for submitting your job for execution on your local machine.

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory='.',
                      script='train_iris.py',
                      arguments=['--kernel', 'linear', '--penalty', 1.0],
                      environment=sklearn_env)
```

If you want to instead run your job on a remote cluster, you can specify the desired compute target to the `compute_target` parameter of `ScriptRunConfig`.

```
from azureml.core import ScriptRunConfig

compute_target = ws.compute_targets['<my-cluster-name>']
src = ScriptRunConfig(source_directory='.',
                      script='train_iris.py',
                      arguments=['--kernel', 'linear', '--penalty', 1.0],
                      compute_target=compute_target,
                      environment=sklearn_env)
```

Submit your run

```
from azureml.core import Experiment

run = Experiment(ws,'Tutorial-TrainIRIS').submit(src)
run.wait_for_completion(show_output=True)
```

WARNING

Azure Machine Learning runs training scripts by copying the entire source directory. If you have sensitive data that you don't want to upload, use a [.ignore file](#) or don't include it in the source directory . Instead, access your data using an [Azure ML dataset](#).

What happens during run execution

As the run is executed, it goes through the following stages:

- **Preparing:** A docker image is created according to the environment defined. The image is uploaded to the workspace's container registry and cached for later runs. Logs are also streamed to the run history and can be viewed to monitor progress. If a curated environment is specified instead, the cached image backing that curated environment will be used.
- **Scaling:** The cluster attempts to scale up if the Batch AI cluster requires more nodes to execute the run than are currently available.
- **Running:** All scripts in the script folder are uploaded to the compute target, data stores are mounted or copied, and the `script` is executed. Outputs from stdout and the `./logs` folder are streamed to the run history and can be used to monitor the run.
- **Post-Processing:** The `./outputs` folder of the run is copied over to the run history.

Save and register the model

Once you've trained the model, you can save and register it to your workspace. Model registration lets you store and version your models in your workspace to simplify [model management and deployment](#).

Add the following code to your training script, `train_iris.py`, to save the model.

```
import joblib

joblib.dump(svm_model_linear, 'model.joblib')
```

Register the model to your workspace with the following code. By specifying the parameters `model_framework`, `model_framework_version`, and `resource_configuration`, no-code model deployment becomes available. No-code model deployment allows you to directly deploy your model as a web service from the registered model, and the `ResourceConfiguration` object defines the compute resource for the web service.

```
from azureml.core import Model
from azureml.core.resource_configuration import ResourceConfiguration

model = run.register_model(model_name='sklearn-iris',
                           model_path='outputs/model.joblib',
                           model_framework=Model.Framework.SCIKITLEARN,
                           model_framework_version='0.19.1',
                           resource_configuration=ResourceConfiguration(cpu=1, memory_in_gb=0.5))
```

Deployment

The model you just registered can be deployed the exact same way as any other registered model in Azure ML. The deployment how-to contains a section on registering models, but you can skip directly to [creating a compute target](#) for deployment, since you already have a registered model.

(Preview) No-code model deployment

Instead of the traditional deployment route, you can also use the no-code deployment feature (preview) for scikit-learn. No-code model deployment is supported for all built-in scikit-learn model types. By registering your model as shown above with the `model_framework`, `model_framework_version`, and `resource_configuration` parameters, you can simply use the `deploy()` static function to deploy your model.

```
web_service = Model.deploy(ws, "scikit-learn-service", [model])
```

NOTE: These dependencies are included in the pre-built scikit-learn inference container.

```
- azureml-defaults
- inference-schema[numpy-support]
- scikit-learn
- numpy
```

The full [how-to](#) covers deployment in Azure Machine Learning in greater depth.

Next steps

In this article, you trained and registered a scikit-learn model, and learned about deployment options. See these other articles to learn more about Azure Machine Learning.

- [Track run metrics during training](#)
- [Tune hyperparameters](#)

Train TensorFlow models at scale with Azure Machine Learning

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, learn how to run your [TensorFlow](#) training scripts at scale using Azure Machine Learning.

This example trains and registers a TensorFlow model to classify handwritten digits using a deep neural network (DNN).

Whether you're developing a TensorFlow model from the ground-up or you're bringing an [existing model](#) into the cloud, you can use Azure Machine Learning to scale out open-source training jobs to build, deploy, version, and monitor production-grade models.

Prerequisites

Run this code on either of these environments:

- Azure Machine Learning compute instance - no downloads or installation necessary
 - Complete the [Quickstart: Get started with Azure Machine Learning](#) to create a dedicated notebook server pre-loaded with the SDK and the sample repository.
 - In the samples deep learning folder on the notebook server, find a completed and expanded notebook by navigating to this directory: `how-to-use-azureml > ml-frameworks > tensorflow > train-hyperparameter-tune-deploy-with-tensorflow` folder.
- Your own Jupyter Notebook server
 - [Install the Azure Machine Learning SDK \(>= 1.15.0\)](#).
 - [Create a workspace configuration file](#).
 - [Download the sample script files](#) `tf_mnist.py` and `utils.py`

You can also find a completed [Jupyter Notebook version](#) of this guide on the GitHub samples page. The notebook includes expanded sections covering intelligent hyperparameter tuning, model deployment, and notebook widgets.

Before you can run the code in this article to create a GPU cluster, you'll need to [request a quota increase](#) for your workspace.

Set up the experiment

This section sets up the training experiment by loading the required Python packages, initializing a workspace, creating the compute target, and defining the training environment.

Import packages

First, import the necessary Python libraries.

```
import os
import urllib
import shutil
import azureml

from azureml.core import Experiment
from azureml.core import Workspace, Run
from azureml.core import Environment

from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException
```

Initialize a workspace

The [Azure Machine Learning workspace](#) is the top-level resource for the service. It provides you with a centralized place to work with all the artifacts you create. In the Python SDK, you can access the workspace artifacts by creating a `workspace` object.

Create a workspace object from the `config.json` file created in the [prerequisites section](#).

```
ws = Workspace.from_config()
```

Create a file dataset

A `FileDataset` object references one or multiple files in your workspace datastore or public urls. The files can be of any format, and the class provides you with the ability to download or mount the files to your compute. By creating a `FileDataset`, you create a reference to the data source location. If you applied any transformations to the data set, they'll be stored in the data set as well. The data remains in its existing location, so no extra storage cost is incurred. For more information the `Dataset` package, see the [How to create register datasets article](#).

```
from azureml.core.dataset import Dataset

web_paths = [
    'http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz',
    'http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz',
    'http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz',
    'http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz'
]
dataset = Dataset.File.from_files(path = web_paths)
```

Use the `register()` method to register the data set to your workspace so they can be shared with others, reused across various experiments, and referred to by name in your training script.

```
dataset = dataset.register(workspace=ws,
                           name='mnist-dataset',
                           description='training and test dataset',
                           create_new_version=True)

# list the files referenced by dataset
dataset.to_path()
```

Create a compute target

Create a compute target for your TensorFlow job to run on. In this example, create a GPU-enabled Azure Machine Learning compute cluster.

IMPORTANT

Before you can create a GPU cluster, you'll need to [request a quota increase](#) for your workspace.

```
cluster_name = "gpu-cluster"

try:
    compute_target = ComputeTarget(workspace=ws, name=cluster_name)
    print('Found existing compute target')
except ComputeTargetException:
    print('Creating a new compute target...')
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_NC6',
                                                           max_nodes=4)

    compute_target = ComputeTarget.create(ws, cluster_name, compute_config)

    compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)
```

NOTE

You may choose to use [low-priority VMs](#) to run some or all of your workloads. See how to [create a low-priority VM](#).

For more information on compute targets, see the [what is a compute target](#) article.

Define your environment

To define the Azure ML [Environment](#) that encapsulates your training script's dependencies, you can either define a custom environment or use an Azure ML curated environment.

Use a curated environment

Azure ML provides prebuilt, curated environments if you don't want to define your own environment. Azure ML has several CPU and GPU curated environments for TensorFlow corresponding to different versions of TensorFlow. For more info, see [Azure ML Curated Environments](#).

If you want to use a curated environment, you can run the following command instead:

```
curated_env_name = 'AzureML-TensorFlow-2.2-GPU'
tf_env = Environment.get(workspace=ws, name=curated_env_name)
```

To see the packages included in the curated environment, you can write out the conda dependencies to disk:

```
tf_env.save_to_directory(path=curated_env_name)
```

Make sure the curated environment includes all the dependencies required by your training script. If not, you'll have to modify the environment to include the missing dependencies. If the environment is modified, you'll have to give it a new name, as the 'AzureML' prefix is reserved for curated environments. If you modified the conda dependencies YAML file, you can create a new environment from it with a new name, for example:

```
tf_env = Environment.from_conda_specification(name='tensorflow-2.2-gpu',
                                               file_path='./conda_dependencies.yml')
```

If you had instead modified the curated environment object directly, you can clone that environment with a new name:

```
tf_env = tf_env.clone(new_name='tensorflow-2.2-gpu')
```

Create a custom environment

You can also create your own Azure ML environment that encapsulates your training script's dependencies.

First, define your conda dependencies in a YAML file; in this example the file is named `conda_dependencies.yml`.

```
channels:
- conda-forge
dependencies:
- python=3.6.2
- pip:
  - azureml-defaults
  - tensorflow-gpu==2.2.0
```

Create an Azure ML environment from this conda environment specification. The environment will be packaged into a Docker container at runtime.

By default if no base image is specified, Azure ML will use a CPU image

`azureml.core.environment.DEFAULT_CPU_IMAGE` as the base image. Since this example runs training on a GPU cluster, you'll need to specify a GPU base image that has the necessary GPU drivers and dependencies. Azure ML maintains a set of base images published on Microsoft Container Registry (MCR) that you can use, see the [Azure/AzureML-Containers GitHub repo](#) for more information.

```
tf_env = Environment.from_conda_specification(name='tensorflow-2.2-gpu',
file_path='./conda_dependencies.yml')

# Specify a GPU base image
tf_env.docker.enabled = True
tf_env.docker.base_image = 'mcr.microsoft.com/azureml/openmpi3.1.2-cuda10.1-cudnn7-ubuntu18.04'
```

TIP

Optionally, you can just capture all your dependencies directly in a custom Docker image or Dockerfile, and create your environment from that. For more information, see [Train with custom image](#).

For more information on creating and using environments, see [Create and use software environments in Azure Machine Learning](#).

Configure and submit your training run

Create a ScriptRunConfig

Create a `ScriptRunConfig` object to specify the configuration details of your training job, including your training script, environment to use, and the compute target to run on. Any arguments to your training script will be passed via command line if specified in the `arguments` parameter.

```
from azureml.core import ScriptRunConfig

args = ['--data-folder', dataset.as_mount(),
       '--batch-size', 64,
       '--first-layer-neurons', 256,
       '--second-layer-neurons', 128,
       '--learning-rate', 0.01]

src = ScriptRunConfig(source_directory=script_folder,
                      script='tf_mnist.py',
                      arguments=args,
                      compute_target=compute_target,
                      environment=tf_env)
```

WARNING

Azure Machine Learning runs training scripts by copying the entire source directory. If you have sensitive data that you don't want to upload, use a `.ignore` file or don't include it in the source directory. Instead, access your data using an Azure ML [dataset](#).

For more information on configuring jobs with `ScriptRunConfig`, see [Configure and submit training runs](#).

WARNING

If you were previously using the TensorFlow estimator to configure your TensorFlow training jobs, please note that Estimators have been deprecated as of the 1.19.0 SDK release. With Azure ML SDK >= 1.15.0, `ScriptRunConfig` is the recommended way to configure training jobs, including those using deep learning frameworks. For common migration questions, see the [Estimator to ScriptRunConfig migration guide](#).

Submit a run

The [Run object](#) provides the interface to the run history while the job is running and after it has completed.

```
run = Experiment(workspace=ws, name='Tutorial-TF-Mnist').submit(src)
run.wait_for_completion(show_output=True)
```

What happens during run execution

As the run is executed, it goes through the following stages:

- **Preparing:** A docker image is created according to the environment defined. The image is uploaded to the workspace's container registry and cached for later runs. Logs are also streamed to the run history and can be viewed to monitor progress. If a curated environment is specified instead, the cached image backing that curated environment will be used.
- **Scaling:** The cluster attempts to scale up if the Batch AI cluster requires more nodes to execute the run than are currently available.
- **Running:** All scripts in the script folder are uploaded to the compute target, data stores are mounted or copied, and the `script` is executed. Outputs from stdout and the `./logs` folder are streamed to the run history and can be used to monitor the run.
- **Post-Processing:** The `./outputs` folder of the run is copied over to the run history.

Register or download a model

Once you've trained the model, you can register it to your workspace. Model registration lets you store and

version your models in your workspace to simplify [model management and deployment](#).

Optional: by specifying the parameters `model_framework`, `model_framework_version`, and `resource_configuration`, no-code model deployment becomes available. This allows you to directly deploy your model as a web service from the registered model, and the `ResourceConfiguration` object defines the compute resource for the web service.

```
from azureml.core import Model
from azureml.core.resource_configuration import ResourceConfiguration

model = run.register_model(model_name='tf-mnist',
                           model_path='outputs/model',
                           model_framework=Model.Framework.TENSORFLOW,
                           model_framework_version='2.0',
                           resource_configuration=ResourceConfiguration(cpu=1, memory_in_gb=0.5))
```

You can also download a local copy of the model by using the Run object. In the training script `tf_mnist.py`, a TensorFlow saver object persists the model to a local folder (local to the compute target). You can use the Run object to download a copy.

```
# Create a model folder in the current directory
os.makedirs('./model', exist_ok=True)
run.download_files(prefix='outputs/model', output_directory='./model', append_prefix=False)
```

Distributed training

Azure Machine Learning also supports multi-node distributed TensorFlow jobs so that you can scale your training workloads. You can easily run distributed TensorFlow jobs and Azure ML will manage the orchestration for you.

Azure ML supports running distributed TensorFlow jobs with both Horovod and TensorFlow's built-in distributed training API.

For more information about distributed training, see the [Distributed GPU training guide](#).

Deploy a TensorFlow model

The deployment how-to contains a section on registering models, but you can skip directly to [creating a compute target](#) for deployment, since you already have a registered model.

(Preview) No-code model deployment

Instead of the traditional deployment route, you can also use the no-code deployment feature (preview) for TensorFlow. By registering your model as shown above with the `model_framework`, `model_framework_version`, and `resource_configuration` parameters, you can use the `deploy()` static function to deploy your model.

```
service = Model.deploy(ws, "tensorflow-web-service", [model])
```

The full [how-to](#) covers deployment in Azure Machine Learning in greater depth.

Next steps

In this article, you trained and registered a TensorFlow model, and learned about options for deployment. See these other articles to learn more about Azure Machine Learning.

- [Track run metrics during training](#)

- Tune hyperparameters
- Reference architecture for distributed deep learning training in Azure

Train Keras models at scale with Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, learn how to run your Keras training scripts with Azure Machine Learning.

The example code in this article shows you how to train and register a Keras classification model built using the TensorFlow backend with Azure Machine Learning. It uses the popular [MNIST dataset](#) to classify handwritten digits using a deep neural network (DNN) built using the [Keras Python library](#) running on top of [TensorFlow](#).

Keras is a high-level neural network API capable of running top of other popular DNN frameworks to simplify development. With Azure Machine Learning, you can rapidly scale out training jobs using elastic cloud compute resources. You can also track your training runs, version models, deploy models, and much more.

Whether you're developing a Keras model from the ground-up or you're bringing an existing model into the cloud, Azure Machine Learning can help you build production-ready models.

NOTE

If you are using the Keras API `tf.keras` built into TensorFlow and not the standalone Keras package, refer instead to [Train TensorFlow models](#).

Prerequisites

Run this code on either of these environments:

- Azure Machine Learning compute instance - no downloads or installation necessary
 - Complete the [Quickstart: Get started with Azure Machine Learning](#) to create a dedicated notebook server pre-loaded with the SDK and the sample repository.
 - In the samples folder on the notebook server, find a completed and expanded notebook by navigating to this directory: `how-to-use-azureml > ml-frameworks > keras > train-hyperparameter-tune-deploy-with-keras` folder.
- Your own Jupyter Notebook server
 - [Install the Azure Machine Learning SDK \(>= 1.15.0\)](#).
 - [Create a workspace configuration file](#).
 - [Download the sample script files](#) `keras_mnist.py` and `utils.py`

You can also find a completed [Jupyter Notebook version](#) of this guide on the GitHub samples page. The notebook includes expanded sections covering intelligent hyperparameter tuning, model deployment, and notebook widgets.

Before you can run the code in this article to create a GPU cluster, you'll need to [request a quota increase](#) for your workspace.

Set up the experiment

This section sets up the training experiment by loading the required Python packages, initializing a workspace,

creating the FileDataset for the input training data, creating the compute target, and defining the training environment.

Import packages

First, import the necessary Python libraries.

```
import os
import azureml
from azureml.core import Experiment
from azureml.core import Environment
from azureml.core import Workspace, Run
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException
```

Initialize a workspace

The [Azure Machine Learning workspace](#) is the top-level resource for the service. It provides you with a centralized place to work with all the artifacts you create. In the Python SDK, you can access the workspace artifacts by creating a `workspace` object.

Create a workspace object from the `config.json` file created in the [prerequisites section](#).

```
ws = Workspace.from_config()
```

Create a file dataset

A `FileDataset` object references one or multiple files in your workspace datastore or public urls. The files can be of any format, and the class provides you with the ability to download or mount the files to your compute. By creating a `FileDataset`, you create a reference to the data source location. If you applied any transformations to the data set, they will be stored in the data set as well. The data remains in its existing location, so no extra storage cost is incurred. See the [how-to guide](#) on the `Dataset` package for more information.

```
from azureml.core.dataset import Dataset

web_paths = [
    'http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz',
    'http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz',
    'http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz',
    'http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz'
]
dataset = Dataset.File.from_files(path=web_paths)
```

You can use the `register()` method to register the data set to your workspace so they can be shared with others, reused across various experiments, and referred to by name in your training script.

```
dataset = dataset.register(workspace=ws,
                           name='mnist-dataset',
                           description='training and test dataset',
                           create_new_version=True)
```

Create a compute target

Create a compute target for your training job to run on. In this example, create a GPU-enabled Azure Machine Learning compute cluster.

IMPORTANT

Before you can create a GPU cluster, you'll need to [request a quota increase](#) for your workspace.

```
cluster_name = "gpu-cluster"

try:
    compute_target = ComputeTarget(workspace=ws, name=cluster_name)
    print('Found existing compute target')
except ComputeTargetException:
    print('Creating a new compute target...')
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_NC6',
                                                           max_nodes=4)

    compute_target = ComputeTarget.create(ws, cluster_name, compute_config)

    compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)
```

NOTE

You may choose to use [low-priority VMs](#) to run some or all of your workloads. See how to [create a low-priority VM](#).

For more information on compute targets, see the [what is a compute target](#) article.

Define your environment

Define the Azure ML [Environment](#) that encapsulates your training script's dependencies.

First, define your conda dependencies in a YAML file; in this example the file is named `conda_dependencies.yml`.

```
channels:
- conda-forge
dependencies:
- python=3.6.2
- pip:
  - azureml-defaults
  - tensorflow-gpu==2.0.0
  - keras<=2.3.1
  - matplotlib
```

Create an Azure ML environment from this conda environment specification. The environment will be packaged into a Docker container at runtime.

By default if no base image is specified, Azure ML will use a CPU image

`azureml.core.environment.DEFAULT_CPU_IMAGE` as the base image. Since this example runs training on a GPU cluster, you will need to specify a GPU base image that has the necessary GPU drivers and dependencies. Azure ML maintains a set of base images published on Microsoft Container Registry (MCR) that you can use, see the [Azure/AzureML-Containers](#) GitHub repo for more information.

```
keras_env = Environment.from_conda_specification(name='keras-env', file_path='conda_dependencies.yml')

# Specify a GPU base image
keras_env.docker.enabled = True
keras_env.docker.base_image = 'mcr.microsoft.com/azureml/openmpi3.1.2-cuda10.0-cudnn7-ubuntu18.04'
```

For more information on creating and using environments, see [Create and use software environments in Azure Machine Learning](#).

Configure and submit your training run

Create a ScriptRunConfig

First get the data from the workspace datastore using the `Dataset` class.

```
dataset = Dataset.get_by_name(ws, 'mnist-dataset')

# list the files referenced by mnist-dataset
dataset.to_path()
```

Create a `ScriptRunConfig` object to specify the configuration details of your training job, including your training script, environment to use, and the compute target to run on.

Any arguments to your training script will be passed via command line if specified in the `arguments` parameter. The `DatasetConsumptionConfig` for our `FileDataset` is passed as an argument to the training script, for the `--data-folder` argument. Azure ML will resolve this `DatasetConsumptionConfig` to the mount-point of the backing datastore, which can then be accessed from the training script.

```
from azureml.core import ScriptRunConfig

args = ['--data-folder', dataset.as_mount(),
        '--batch-size', 50,
        '--first-layer-neurons', 300,
        '--second-layer-neurons', 100,
        '--learning-rate', 0.001]

src = ScriptRunConfig(source_directory=script_folder,
                      script='keras_mnist.py',
                      arguments=args,
                      compute_target=compute_target,
                      environment=keras_env)
```

For more information on configuring jobs with `ScriptRunConfig`, see [Configure and submit training runs](#).

WARNING

If you were previously using the TensorFlow estimator to configure your Keras training jobs, please note that Estimators have been deprecated as of the 1.19.0 SDK release. With Azure ML SDK >= 1.15.0, `ScriptRunConfig` is the recommended way to configure training jobs, including those using deep learning frameworks. For common migration questions, see the [Estimator to ScriptRunConfig migration guide](#).

Submit your run

The `Run` object provides the interface to the run history while the job is running and after it has completed.

```
run = Experiment(workspace=ws, name='Tutorial-Keras-Minst').submit(src)
run.wait_for_completion(show_output=True)
```

What happens during run execution

As the run is executed, it goes through the following stages:

- **Preparing:** A docker image is created according to the environment defined. The image is uploaded to the workspace's container registry and cached for later runs. Logs are also streamed to the run history and can be viewed to monitor progress. If a curated environment is specified instead, the cached image backing that curated environment will be used.

- **Scaling:** The cluster attempts to scale up if the Batch AI cluster requires more nodes to execute the run than are currently available.
- **Running:** All scripts in the script folder are uploaded to the compute target, data stores are mounted or copied, and the `script` is executed. Outputs from stdout and the `./logs` folder are streamed to the run history and can be used to monitor the run.
- **Post-Processing:** The `./outputs` folder of the run is copied over to the run history.

Register the model

Once you've trained the model, you can register it to your workspace. Model registration lets you store and version your models in your workspace to simplify [model management and deployment](#).

```
model = run.register_model(model_name='keras-mnist', model_path='outputs/model')
```

TIP

The deployment how-to contains a section on registering models, but you can skip directly to [creating a compute target](#) for deployment, since you already have a registered model.

You can also download a local copy of the model. This can be useful for doing additional model validation work locally. In the training script, `keras_mnist.py`, a TensorFlow saver object persists the model to a local folder (local to the compute target). You can use the Run object to download a copy from the run history.

```
# Create a model folder in the current directory
os.makedirs('./model', exist_ok=True)

for f in run.get_file_names():
    if f.startswith('outputs/model'):
        output_file_path = os.path.join('./model', f.split('/')[-1])
        print('Downloading from {} to {}'.format(f, output_file_path))
        run.download_file(name=f, output_file_path=output_file_path)
```

Next steps

In this article, you trained and registered a Keras model on Azure Machine Learning. To learn how to deploy a model, continue on to our model deployment article.

- [How and where to deploy models](#)
- [Track run metrics during training](#)
- [Tune hyperparameters](#)
- [Reference architecture for distributed deep learning training in Azure](#)

Train PyTorch models at scale with Azure Machine Learning

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, learn how to run your [PyTorch](#) training scripts at enterprise scale using Azure Machine Learning.

The example scripts in this article are used to classify chicken and turkey images to build a deep learning neural network (DNN) based on [PyTorch's transfer learning tutorial](#). Transfer learning is a technique that applies knowledge gained from solving one problem to a different but related problem. Transfer learning shortens the training process by requiring less data, time, and compute resources than training from scratch. To learn more about transfer learning, see the [deep learning vs machine learning](#) article.

Whether you're training a deep learning PyTorch model from the ground-up or you're bringing an existing model into the cloud, you can use Azure Machine Learning to scale out open-source training jobs using elastic cloud compute resources. You can build, deploy, version, and monitor production-grade models with Azure Machine Learning.

Prerequisites

Run this code on either of these environments:

- Azure Machine Learning compute instance - no downloads or installation necessary
 - Complete the [Quickstart: Get started with Azure Machine Learning](#) to create a dedicated notebook server pre-loaded with the SDK and the sample repository.
 - In the samples deep learning folder on the notebook server, find a completed and expanded notebook by navigating to this directory: `how-to-use-azureml > ml-frameworks > pytorch > train-hyperparameter-tune-deploy-with-pytorch` folder.
- Your own Jupyter Notebook server
 - [Install the Azure Machine Learning SDK \(>= 1.15.0\)](#).
 - [Create a workspace configuration file](#).
 - [Download the sample script files](#) `pytorch_train.py`

You can also find a completed [Jupyter Notebook version](#) of this guide on the GitHub samples page. The notebook includes expanded sections covering intelligent hyperparameter tuning, model deployment, and notebook widgets.

Before you can run the code in this article to create a GPU cluster, you'll need to [request a quota increase](#) for your workspace.

Set up the experiment

This section sets up the training experiment by loading the required Python packages, initializing a workspace, creating the compute target, and defining the training environment.

Import packages

First, import the necessary Python libraries.

```
import os
import shutil

from azureml.core.workspace import Workspace
from azureml.core import Experiment
from azureml.core import Environment

from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException
```

Initialize a workspace

The [Azure Machine Learning workspace](#) is the top-level resource for the service. It provides you with a centralized place to work with all the artifacts you create. In the Python SDK, you can access the workspace artifacts by creating a `workspace` object.

Create a workspace object from the `config.json` file created in the [prerequisites section](#).

```
ws = Workspace.from_config()
```

Get the data

The dataset consists of about 120 training images each for turkeys and chickens, with 100 validation images for each class. We'll download and extract the dataset as part of our training script `pytorch_train.py`. The images are a subset of the [Open Images v5 Dataset](#). For more steps on creating a JSONL to train with your own data, see this [Jupyter notebook](#).

Prepare training script

In this tutorial, the training script, `pytorch_train.py`, is already provided. In practice, you can take any custom training script, as is, and run it with Azure Machine Learning.

Create a folder for your training script(s).

```
project_folder = './pytorch-birds'
os.makedirs(project_folder, exist_ok=True)
shutil.copy('pytorch_train.py', project_folder)
```

Create a compute target

Create a compute target for your PyTorch job to run on. In this example, create a GPU-enabled Azure Machine Learning compute cluster.

IMPORTANT

Before you can create a GPU cluster, you'll need to [request a quota increase](#) for your workspace.

```

# Choose a name for your CPU cluster
cluster_name = "gpu-cluster"

# Verify that cluster does not exist already
try:
    compute_target = ComputeTarget(workspace=ws, name=cluster_name)
    print('Found existing compute target')
except ComputeTargetException:
    print('Creating a new compute target...')
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_NC6',
                                                          max_nodes=4)

    # Create the cluster with the specified name and configuration
    compute_target = ComputeTarget.create(ws, cluster_name, compute_config)

    # Wait for the cluster to complete, show the output log
    compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)

```

If you instead want to create a CPU cluster, provide a different VM size to the `vm_size` parameter, such as `STANDARD_D2_V2`.

NOTE

You may choose to use [low-priority VMs](#) to run some or all of your workloads. See how to [create a low-priority VM](#).

For more information on compute targets, see the [what is a compute target](#) article.

Define your environment

To define the [Azure ML Environment](#) that encapsulates your training script's dependencies, you can either define a custom environment or use an Azure ML curated environment.

Use a curated environment

Azure ML provides prebuilt, [curated environments](#) if you don't want to define your own environment. There are several CPU and GPU curated environments for PyTorch corresponding to different versions of PyTorch.

If you want to use a curated environment, you can run the following command instead:

```

curated_env_name = 'AzureML-PyTorch-1.6-GPU'
pytorch_env = Environment.get(workspace=ws, name=curated_env_name)

```

To see the packages included in the curated environment, you can write out the conda dependencies to disk:

```

pytorch_env.save_to_directory(path=curated_env_name)

```

Make sure the curated environment includes all the dependencies required by your training script. If not, you'll have to modify the environment to include the missing dependencies. If the environment is modified, you'll have to give it a new name, as the 'AzureML' prefix is reserved for curated environments. If you modified the conda dependencies YAML file, you can create a new environment from it with a new name, for example:

```

pytorch_env = Environment.from_conda_specification(name='pytorch-1.6-gpu',
                                                    file_path='./conda_dependencies.yml')

```

If you had instead modified the curated environment object directly, you can clone that environment with a new name:

```
pytorch_env = pytorch_env.clone(new_name='pytorch-1.6-gpu')
```

Create a custom environment

You can also create your own Azure ML environment that encapsulates your training script's dependencies.

First, define your conda dependencies in a YAML file; in this example the file is named `conda_dependencies.yml`.

```
channels:
- conda-forge
dependencies:
- python=3.6.2
- pip=21.3.1
- pip:
  - azureml-defaults
  - torch==1.6.0
  - torchvision==0.7.0
  - future==0.17.1
- pillow
```

Create an Azure ML environment from this conda environment specification. The environment will be packaged into a Docker container at runtime.

By default if no base image is specified, Azure ML will use a CPU image

`azureml.core.environment.DEFAULT_CPU_IMAGE` as the base image. Since this example runs training on a GPU cluster, you'll need to specify a GPU base image that has the necessary GPU drivers and dependencies. Azure ML maintains a set of base images published on Microsoft Container Registry (MCR) that you can use. For more information, see [AzureML-Containers GitHub repo](#).

```
pytorch_env = Environment.from_conda_specification(name='pytorch-1.6-gpu',
file_path='./conda_dependencies.yml')

# Specify a GPU base image
pytorch_env.docker.enabled = True
pytorch_env.docker.base_image = 'mcr.microsoft.com/azureml/openmpi3.1.2-cuda10.1-cudnn7-ubuntu18.04'
```

TIP

Optionally, you can just capture all your dependencies directly in a custom Docker image or Dockerfile, and create your environment from that. For more information, see [Train with custom image](#).

For more information on creating and using environments, see [Create and use software environments in Azure Machine Learning](#).

Configure and submit your training run

Create a ScriptRunConfig

Create a `ScriptRunConfig` object to specify the configuration details of your training job, including your training script, environment to use, and the compute target to run on. Any arguments to your training script will be passed via command line if specified in the `arguments` parameter. The following code will configure a single-node PyTorch job.

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory=project_folder,
                      script='pytorch_train.py',
                      arguments=['--num_epochs', 30, '--output_dir', './outputs'],
                      compute_target=compute_target,
                      environment=pytorch_env)
```

WARNING

Azure Machine Learning runs training scripts by copying the entire source directory. If you have sensitive data that you don't want to upload, use a [.ignore file](#) or don't include it in the source directory . Instead, access your data using an Azure ML [dataset](#).

For more information on configuring jobs with ScriptRunConfig, see [Configure and submit training runs](#).

WARNING

If you were previously using the PyTorch estimator to configure your PyTorch training jobs, please note that Estimators have been deprecated as of the 1.19.0 SDK release. With Azure ML SDK >= 1.15.0, ScriptRunConfig is the recommended way to configure training jobs, including those using deep learning frameworks. For common migration questions, see the [Estimator to ScriptRunConfig migration guide](#).

Submit your run

The [Run object](#) provides the interface to the run history while the job is running and after it has completed.

```
run = Experiment(ws, name='Tutorial-pytorch-birds').submit(src)
run.wait_for_completion(show_output=True)
```

What happens during run execution

As the run is executed, it goes through the following stages:

- **Preparing:** A docker image is created according to the environment defined. The image is uploaded to the workspace's container registry and cached for later runs. Logs are also streamed to the run history and can be viewed to monitor progress. If a curated environment is specified instead, the cached image backing that curated environment will be used.
- **Scaling:** The cluster attempts to scale up if the Batch AI cluster requires more nodes to execute the run than are currently available.
- **Running:** All scripts in the script folder are uploaded to the compute target, data stores are mounted or copied, and the `script` is executed. Outputs from stdout and the `./logs` folder are streamed to the run history and can be used to monitor the run.
- **Post-Processing:** The `./outputs` folder of the run is copied over to the run history.

Register or download a model

Once you've trained the model, you can register it to your workspace. Model registration lets you store and version your models in your workspace to simplify [model management and deployment](#).

```
model = run.register_model(model_name='pytorch-birds', model_path='outputs/model.pt')
```

TIP

The deployment how-to contains a section on registering models, but you can skip directly to [creating a compute target](#) for deployment, since you already have a registered model.

You can also download a local copy of the model by using the Run object. In the training script

`pytorch_train.py`, a PyTorch save object persists the model to a local folder (local to the compute target). You can use the Run object to download a copy.

```
# Create a model folder in the current directory
os.makedirs('./model', exist_ok=True)

# Download the model from run history
run.download_file(name='outputs/model.pt', output_file_path='./model/model.pt'),
```

Distributed training

Azure Machine Learning also supports multi-node distributed PyTorch jobs so that you can scale your training workloads. You can easily run distributed PyTorch jobs and Azure ML will manage the orchestration for you.

Azure ML supports running distributed PyTorch jobs with both Horovod and PyTorch's built-in `DistributedDataParallel` module.

For more information about distributed training, see the [Distributed GPU training guide](#).

Export to ONNX

To optimize inference with the [ONNX Runtime](#), convert your trained PyTorch model to the ONNX format. Inference, or model scoring, is the phase where the deployed model is used for prediction, most commonly on production data. For an example, see the [Exporting model from PyTorch to ONNX tutorial](#).

Next steps

In this article, you trained and registered a deep learning, neural network using PyTorch on Azure Machine Learning. To learn how to deploy a model, continue on to our model deployment article.

- [How and where to deploy models](#)
- [Track run metrics during training](#)
- [Tune hyperparameters](#)
- [Reference architecture for distributed deep learning training in Azure](#)

Train a model by using a custom Docker image

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azureml v1](#)

In this article, learn how to use a custom Docker image when you're training models with Azure Machine Learning. You'll use the example scripts in this article to classify pet images by creating a convolutional neural network.

Azure Machine Learning provides a default Docker base image. You can also use Azure Machine Learning environments to specify a different base image, such as one of the maintained [Azure Machine Learning base images](#) or your own [custom image](#). Custom base images allow you to closely manage your dependencies and maintain tighter control over component versions when running training jobs.

Prerequisites

Run the code on either of these environments:

- Azure Machine Learning compute instance (no downloads or installation necessary):
 - Complete the [Quickstart: Get started with Azure Machine Learning](#) tutorial to create a dedicated notebook server preloaded with the SDK and the sample repository.
- Your own Jupyter Notebook server:
 - Create a [workspace configuration file](#).
 - Install the [Azure Machine Learning SDK](#).
 - Create an [Azure container registry](#) or other Docker registry that's available on the internet.

Set up a training experiment

In this section, you set up your training experiment by initializing a workspace, defining your environment, and configuring a compute target.

Initialize a workspace

The [Azure Machine Learning workspace](#) is the top-level resource for the service. It gives you a centralized place to work with all the artifacts that you create. In the Python SDK, you can access the workspace artifacts by creating a `Workspace` object.

Create a `Workspace` object from the config.json file that you created as a [prerequisite](#).

```
from azureml.core import Workspace  
  
ws = Workspace.from_config()
```

Define your environment

Create an `Environment` object.

```
from azureml.core import Environment  
  
fastai_env = Environment("fastai2")
```

The specified base image in the following code supports the fast.ai library, which allows for distributed deep-

learning capabilities. For more information, see the [fast.ai Docker Hub repository](#).

When you're using your custom Docker image, you might already have your Python environment properly set up. In that case, set the `user_managed_dependencies` flag to `True` to use your custom image's built-in Python environment. By default, Azure Machine Learning builds a Conda environment with dependencies that you specified. The service runs the script in that environment instead of using any Python libraries that you installed on the base image.

```
fastai_env.docker.base_image = "fastdotai/fastai2:latest"
fastai_env.python.user_managed_dependencies = True
```

Use a private container registry (optional)

To use an image from a private container registry that isn't in your workspace, use `docker.base_image_registry` to specify the address of the repository and a username and password:

```
# Set the container registry information.
fastai_env.docker.base_image_registry.address = "myregistry.azurecr.io"
fastai_env.docker.base_image_registry.username = "username"
fastai_env.docker.base_image_registry.password = "password"
```

Use a custom Dockerfile (optional)

It's also possible to use a custom Dockerfile. Use this approach if you need to install non-Python packages as dependencies. Remember to set the base image to `None`.

```
# Specify Docker steps as a string.
dockerfile = r"""
FROM mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210615.v1
RUN echo "Hello from custom container!"
"""

# Set the base image to None, because the image is defined by Dockerfile.
fastai_env.docker.base_image = None
fastai_env.docker.base_dockerfile = dockerfile

# Alternatively, load the string from a file.
fastai_env.docker.base_image = None
fastai_env.docker.base_dockerfile = "./Dockerfile"
```

IMPORTANT

Azure Machine Learning only supports Docker images that provide the following software:

- Ubuntu 18.04 or greater.
- Conda 4.7.# or greater.
- Python 3.6+.
- A POSIX compliant shell available at /bin/sh is required in any container image used for training.

For more information about creating and managing Azure Machine Learning environments, see [Create and use software environments](#).

Create or attach a compute target

You need to create a [compute target](#) for training your model. In this tutorial, you create `AmlCompute` as your training compute resource.

Creation of `AmlCompute` takes a few minutes. If the `AmlCompute` resource is already in your workspace, this code skips the creation process.

As with other Azure services, there are limits on certain resources (for example, `AmlCompute`) associated with the Azure Machine Learning service. For more information, see [Default limits and how to request a higher quota](#).

```
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

# Choose a name for your cluster.
cluster_name = "gpu-cluster"

try:
    compute_target = ComputeTarget(workspace=ws, name=cluster_name)
    print('Found existing compute target.')
except ComputeTargetException:
    print('Creating a new compute target...')
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_NC6',
                                                           max_nodes=4)

    # Create the cluster.
    compute_target = ComputeTarget.create(ws, cluster_name, compute_config)

    compute_target.wait_for_completion(show_output=True)

# Use get_status() to get a detailed status for the current AmlCompute.
print(compute_target.get_status().serialize())
```

IMPORTANT

Use CPU SKUs for any image build on compute.

Configure your training job

For this tutorial, use the training script `train.py` on [GitHub](#). In practice, you can take any custom training script and run it, as is, with Azure Machine Learning.

Create a `ScriptRunConfig` resource to configure your job for running on the desired [compute target](#).

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory='fastai-example',
                      script='train.py',
                      compute_target=compute_target,
                      environment=fastai_env)
```

Submit your training job

When you submit a training run by using a `ScriptRunConfig` object, the `submit` method returns an object of type `ScriptRun`. The returned `ScriptRun` object gives you programmatic access to information about the training run.

```
from azureml.core import Experiment

run = Experiment(ws,'Tutorial-fastai').submit(src)
run.wait_for_completion(show_output=True)
```

WARNING

Azure Machine Learning runs training scripts by copying the entire source directory. If you have sensitive data that you don't want to upload, use an [.ignore file](#) or don't include it in the source directory. Instead, access your data by using a [datastore](#).

Next steps

In this article, you trained a model by using a custom Docker image. See these other articles to learn more about Azure Machine Learning:

- [Track run metrics](#) during training.
- [Deploy a model](#) by using a custom Docker image.

Use authentication credential secrets in Azure Machine Learning jobs

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

Authentication information such as your user name and password are secrets. For example, if you connect to an external database in order to query training data, you would need to pass your username and password to the remote job context. Coding such values into training scripts in clear text is insecure as it would potentially expose the secret.

The Azure Key Vault allows you to securely store and retrieve secrets. In this article, learn how you can retrieve secrets stored in a key vault from a training job running on a compute cluster.

IMPORTANT

The Azure Machine Learning Python SDK v2 and Azure CLI extension v2 for machine learning do not provide the capability to set or get secrets. Instead, the information in this article uses the [Azure Key Vault Secrets client library for Python](#).

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

TIP

Many of the prerequisites in this section require **Contributor**, **Owner**, or equivalent access to your Azure subscription, or the Azure Resource Group that contains the resources. You may need to contact your Azure administrator and have them perform these actions.

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Quickstart: Create workspace resources](#) article to create one.
- An Azure Key Vault. If you used the [Quickstart: Create workspace resources](#) article to create your workspace, a key vault was created for you. You can also create a separate key vault instance using the information in the [Quickstart: Create a key vault](#) article.

TIP

You do not have to use same key vault as the workspace.

- An Azure Machine Learning compute cluster configured to use a [managed identity](#). The cluster can be configured for either a system-assigned or user-assigned managed identity.
- Grant the managed identity for the compute cluster access to the secrets stored in key vault. The method used to grant access depends on how your key vault is configured:

- [Azure role-based access control \(Azure RBAC\)](#): When configured for Azure RBAC, add the managed identity to the **Key Vault Secrets User** role on your key vault.
- [Azure Key Vault access policy](#): When configured to use access policies, add a new policy that grants the **get** operation for secrets and assign it to the managed identity.
- A stored secret value in the key vault. This value can then be retrieved using a key. For more information, see [Quickstart: Set and retrieve a secret from Azure Key Vault](#).

TIP

The quickstart link is to the steps for using the Azure Key Vault Python SDK. In the table of contents in the left navigation area are links to other ways to set a key.

Getting secrets

1. Add the `azure-keyvault-secrets` and `azure-identity` packages to the [Azure Machine Learning environment](#) used when training the model. For example, by adding them to the conda file used to build the environment.

The environment is used to build the Docker image that the training job runs in on the compute cluster.

2. From your training code, use the [Azure Identity SDK](#) and [Key Vault client library](#) to get the managed identity credentials and authenticate to key vault:

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.secret import SecretClient

credential = DefaultAzureCredential()

secret_client = SecretClient(vault_url="https://my-key-vault.vault.azure.net/",
                             credential=credential)
```

3. After authenticating, use the Key Vault client library to retrieve a secret by providing the associated key:

```
secret = secret_client.get_secret("secret-name")
print(secret.value)
```

Next steps

For an example of submitting a training job using the Azure Machine Learning Python SDK v2 (preview), see [Train models with the Python SDK v2](#).

Train models with the CLI (v2)

9/22/2022 • 21 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The Azure Machine Learning CLI (v2) is an Azure CLI extension enabling you to accelerate the model training process while scaling up and out on Azure compute, with the model lifecycle tracked and auditable.

Training a machine learning model is typically an iterative process. Modern tooling makes it easier than ever to train larger models on more data faster. Previously tedious manual processes like hyperparameter tuning and even algorithm selection are often automated. With the Azure Machine Learning CLI (v2), you can track your jobs (and models) in a [workspace](#) with hyperparameter sweeps, scale-up on high-performance Azure compute, and scale-out utilizing distributed training.

Prerequisites

- To use the CLI (v2), you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- [Install and set up CLI \(v2\)](#).

TIP

For a full-featured development environment with schema validation and autocompletion for job YAMLS, use Visual Studio Code and the [Azure Machine Learning extension](#).

Clone examples repository

To run the training examples, first clone the examples repository and change into the `cli` directory:

```
git clone --depth 1 https://github.com/Azure/azureml-examples  
cd azureml-examples/cli
```

Using `--depth 1` clones only the latest commit to the repository, which reduces time to complete the operation.

Create compute

You can create an Azure Machine Learning compute cluster from the command line. For instance, the following commands will create one cluster named `cpu-cluster` and one named `gpu-cluster`.

```
az ml compute create -n cpu-cluster --type amlcompute --min-instances 0 --max-instances 8  
az ml compute create -n gpu-cluster --type amlcompute --min-instances 0 --max-instances 4 --size  
Standard_NC12
```

You are not charged for compute at this point as `cpu-cluster` and `gpu-cluster` will remain at zero nodes until a job is submitted. Learn more about how to [manage and optimize cost for AmlCompute](#).

The following example jobs in this article use one of `cpu-cluster` or `gpu-cluster`. Adjust these names in the example jobs throughout this article as needed to the name of your cluster(s). Use `az ml compute create -h` for more details on compute create options.

Hello world

For the Azure Machine Learning CLI (v2), jobs are authored in YAML format. A job aggregates:

- What to run
- How to run it
- Where to run it

The "hello world" job has all three:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world"
environment:
  image: library/python:latest
compute: azureml:cpu-cluster
```

WARNING

Python must be installed in the environment used for jobs. Run `apt-get update -y && apt-get install python3 -y` in your Dockerfile to install if needed, or derive from a base image with Python installed already.

TIP

The `$schema:` throughout examples allows for schema validation and autocompletion if authoring YAML files in [VSCode](#) with the Azure Machine Learning extension.

Which you can run:

```
az ml job create -f jobs/basics/hello-world.yml --web
```

TIP

The `--web` parameter will attempt to open your job in the Azure Machine Learning studio using your default web browser. The `--stream` parameter can be used to stream logs to the console and block further commands.

Overriding values on create or update

YAML job specification values can be overridden using `--set` when creating or updating a job. For instance:

```
az ml job create -f jobs/basics/hello-world.yml \
--set environment.image="python:3.8" \
--web
```

Job names

Most `az ml job` commands other than `create` and `list` require `--name/-n`, which is a job's name or "Run ID" in the studio. You typically should not directly set a job's `name` property during creation as it must be unique per workspace. Azure Machine Learning generates a random GUID for the job name if it is not set that can be obtained from the output of job creation in the CLI or by copying the "Run ID" property in the studio and MLflow APIs.

To automate jobs in scripts and CI/CD flows, you can capture a job's name when it is created by querying and stripping the output by adding `--query name -o tsv`. The specifics will vary by shell, but for Bash:

```
run_id=$(az ml job create -f jobs/basics/hello-world.yml --query name -o tsv)
```

Then use `$run_id` in subsequent commands like `update`, `show`, or `stream`:

```
az ml job show -n $run_id --web
```

Organize jobs

To organize jobs, you can set a display name, experiment name, description, and tags. Descriptions support markdown syntax in the studio. These properties are mutable after a job is created. A full example:

```
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world"
environment:
  image: library/python:latest
compute: azureml:cpu-cluster
tags:
  hello: world
display_name: hello-world-example
experiment_name: hello-world-example
description: |
  # Azure Machine Learning "hello world" job

  This is a "hello world" job running in the cloud via Azure Machine Learning!

## Description

Markdown is supported in the studio for job descriptions! You can edit the description there or via CLI.
```

You can run this job, where these properties will be immediately visible in the studio:

```
az ml job create -f jobs/basics/hello-world-org.yml --web
```

Using `--set` you can update the mutable values after the job is created:

```
az ml job update -n $run_id --set \
  display_name="updated display name" \
  experiment_name="updated experiment name" \
  description="updated description" \
  tags.hello="updated tag"
```

Environment variables

You can set environment variables for use in your job:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo $hello_env_var
environment:
  image: library/python:latest
compute: azurerm:cpu-cluster
environment_variables:
  hello_env_var: "hello world"
```

You can run this job:

```
az ml job create -f jobs/basics/hello-world-env-var.yml --web
```

WARNING

You should use `inputs` for parameterizing arguments in the `command`. See [inputs and outputs](#).

Track models and source code

Production machine learning models need to be auditable (if not reproducible). It is crucial to keep track of the source code for a given model. Azure Machine Learning takes a snapshot of your source code and keeps it with the job. Additionally, the source repository and commit are tracked if you are running jobs from a Git repository.

TIP

If you're following along and running from the examples repository, you can see the source repository and commit in the studio on any of the jobs run so far.

You can specify the `code` field in a job with the value as the path to a source code directory. A snapshot of the directory is taken and uploaded with the job. The contents of the directory are directly available from the working directory of the job.

WARNING

The source code should not include large data inputs for model training. Instead, [use data inputs](#). You can use a `.gitignore` file in the source code directory to exclude files from the snapshot. The limits for snapshot size are 300 MB or 2000 files.

Let's look at a job that specifies code:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: python hello-mlflow.py
code: src
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster
```

The Python script is in the local source code directory. The command then invokes `python` to run the script. The same pattern can be applied for other programming languages.

WARNING

The "hello" family of jobs shown in this article are for demonstration purposes and do not necessarily follow recommended best practices. Using `&&` or similar to run many commands in a sequence is not recommended -- instead, consider writing the commands to a script file in the source code directory and invoking the script in your `command`. Installing dependencies in the `command`, as shown above via `pip install`, is not recommended -- instead, all job dependencies should be specified as part of your environment. See [how to manage environments with the CLI \(v2\)](#) for details.

Model tracking with MLflow

While iterating on models, data scientists need to be able to keep track of model parameters and training metrics. Azure Machine Learning integrates with MLflow tracking to enable the logging of models, artifacts, metrics, and parameters to a job. To use MLflow in your Python scripts add `import mlflow` and call `mlflow.log_*` or `mlflow.autolog()` APIs in your training code.

WARNING

The `mlflow` and `azureml-mlflow` packages must be installed in your Python environment for MLflow tracking features.

TIP

The `mlflow.autolog()` call is supported for many popular frameworks and takes care of the majority of logging for you.

Let's take a look at Python script invoked in the job above that uses `mlflow` to log a parameter, a metric, and an artifact:

```
# imports
import os
import mlflow

from random import random

# define functions
def main():
    mlflow.log_param("hello_param", "world")
    mlflow.log_metric("hello_metric", random())
    os.system(f"echo 'hello world' > helloworld.txt")
    mlflow.log_artifact("helloworld.txt")

# run functions
if __name__ == "__main__":
    # run main function
    main()
```

You can run this job in the cloud via Azure Machine Learning, where it is tracked and auditable:

```
az ml job create -f jobs/basics/hello-mlflow.yml --web
```

Query metrics with MLflow

After running jobs, you might want to query the jobs' run results and their logged metrics. Python is better suited for this task than a CLI. You can query runs and their metrics via `mlflow` and load into familiar objects like Pandas dataframes for analysis.

First, retrieve the MLflow tracking URI for your Azure Machine Learning workspace:

```
az ml workspace show --query mlflow_tracking_uri -o tsv
```

Use the output of this command in `mlflow.set_tracking_uri(<YOUR_TRACKING_URI>)` from a Python environment with MLflow imported. MLflow calls will now correspond to jobs in your Azure Machine Learning workspace.

Inputs and outputs

Jobs typically have inputs and outputs. Inputs can be model parameters, which might be swept over for hyperparameter optimization, or cloud data inputs that are mounted or downloaded to the compute target. Outputs (ignoring metrics) are artifacts that can be written or copied to the default outputs or a named data output.

Literal inputs

Literal inputs are directly resolved in the command. You can modify our "hello world" job to use literal inputs:

```
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json
command: |
    echo ${{inputs.hello_string}}
    echo ${{inputs.hello_number}}
environment:
    image: library/python:latest
inputs:
    hello_string: "hello world"
    hello_number: 42
compute: azureml:cpu-cluster
```

You can run this job:

```
az ml job create -f jobs/basics/hello-world-input.yml --web
```

You can use `--set` to override inputs:

```
az ml job create -f jobs/basics/hello-world-input.yml --set \
    inputs.hello_string="hello there" \
    inputs.hello_number=24 \
    --web
```

Literal inputs to jobs can be [converted to search space inputs](#) for hyperparameter sweeps on model training.

Search space inputs

For a sweep job, you can specify a search space for literal inputs to be chosen from. For the full range of options for search space inputs, see the [sweep job YAML syntax reference](#).

Let's demonstrate the concept with a simple Python script that takes in arguments and logs a random metric:

```

# imports
import os
import mlflow
import argparse

from random import random

# define functions
def main(args):
    # print inputs
    print(f"A: {args.A}")
    print(f"B: {args.B}")
    print(f"C: {args.C}")

    # log inputs as parameters
    mlflow.log_param("A", args.A)
    mlflow.log_param("B", args.B)
    mlflow.log_param("C", args.C)

    # log a random metric
    mlflow.log_metric("random_metric", random())


def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--A", type=float, default=0.5)
    parser.add_argument("--B", type=str, default="hello world")
    parser.add_argument("--C", type=float, default=1.0)

    # parse args
    args = parser.parse_args()

    # return args
    return args


# run script
if __name__ == "__main__":
    # parse args
    args = parse_args()

    # run main function
    main(args)

```

And create a corresponding sweep job:

```
$schema: https://azurermschemas.azureedge.net/latest/sweepJob.schema.json
type: sweep
trial:
  command: >-
    python hello-sweep.py
    --A ${{inputs.A}}
    --B ${{search_space.B}}
    --C ${{search_space.C}}
  code: src
  environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
inputs:
  A: 0.5
compute: azurerm:cpu-cluster
sampling_algorithm: random
search_space:
  B:
    type: choice
    values: ["hello", "world", "hello_world"]
  C:
    type: uniform
    min_value: 0.1
    max_value: 1.0
objective:
  goal: minimize
  primary_metric: random_metric
limits:
  max_total_trials: 4
  max_concurrent_trials: 2
  timeout: 3600
display_name: hello-sweep-example
experiment_name: hello-sweep-example
description: Hello sweep job example.
```

And run it:

```
az ml job create -f jobs/basics/hello-sweep.yml --web
```

Data inputs

Data inputs are resolved to a path on the job compute's local filesystem. Let's demonstrate with the classic Iris dataset, which is hosted publicly in a blob container at

```
https://azurermexamples.blob.core.windows.net/datasets/iris.csv.
```

You can author a Python script that takes the path to the Iris CSV file as an argument, reads it into a dataframe, prints the first 5 lines, and saves it to the `outputs` directory.

```

# imports
import os
import argparse

import pandas as pd

# define functions
def main(args):
    # read in data
    df = pd.read_csv(args.iris_csv)

    # print first 5 lines
    print(df.head())

    # ensure outputs directory exists
    os.makedirs("outputs", exist_ok=True)

    # save data to outputs
    df.to_csv("outputs/iris.csv", index=False)

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--iris-csv", type=str)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # parse args
    args = parse_args()

    # run main function
    main(args)

```

Azure storage URI inputs can be specified, which will mount or download data to the local filesystem. You can specify a single file:

```

$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json
command: |
    echo "--iris-csv: ${{inputs.iris_csv}}"
    python hello-iris.py --iris-csv ${{inputs.iris_csv}}
code: src
inputs:
    iris_csv:
        type: uri_file
        path: https://azuremlexamples.blob.core.windows.net/datasets/iris.csv
environment: azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azureml:cpu-cluster

```

And run:

```
az ml job create -f jobs/basics/hello-iris-file.yml --web
```

Or specify an entire folder:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  ls ${inputs.data_dir}
  echo "--iris-csv: ${inputs.data_dir}/iris.csv"
  python hello-iris.py --iris-csv ${inputs.data_dir}/iris.csv
code: src
inputs:
  data_dir:
    type: uri_folder
    path: wasbs://datasets@azurermexamples.blob.core.windows.net/
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster
```

And run:

```
az ml job create -f jobs/basics/hello-iris-folder.yml --web
```

Make sure you accurately specify the input `type` field to either `type: uri_file` or `type: uri_folder` corresponding to whether the data points to a single file or a folder. The default if the `type` field is omitted is `uri_folder`.

Private data

For private data in Azure Blob Storage or Azure Data Lake Storage connected to Azure Machine Learning through a datastore, you can use Azure Machine Learning URIs of the format

`azurerm://datastores/<DATASTORE_NAME>/paths/<PATH_TO_DATA>` for input data. For instance, if you upload the Iris CSV to a directory named `/example-data/` in the Blob container corresponding to the datastore named `workspaceblobstore` you can modify a previous job to use the file in the datastore:

WARNING

Running these jobs will fail for you if you have not copied the Iris CSV to the same location in `workspaceblobstore`.

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  echo "--iris-csv: ${inputs.iris_csv}"
  python hello-iris.py --iris-csv ${inputs.iris_csv}
code: src
inputs:
  iris_csv:
    type: uri_file
    path: azurerm://datastores/workspaceblobstore/paths/example-data/iris.csv
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster
```

Or the entire directory:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  ls ${{inputs.data_dir}}
  echo "--iris-csv: ${{inputs.data_dir}}/iris.csv"
  python hello-iris.py --iris-csv ${{inputs.data_dir}}/iris.csv
code: src
inputs:
  data_dir:
    type: uri_folder
    path: azurerm://datastores/workspaceblobstore/paths/example-data/
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster
```

Default outputs

The `./outputs` and `./logs` directories receive special treatment by Azure Machine Learning. If you write any files to these directories during your job, these files will get uploaded to the job so that you can still access them once the job is complete. The `./outputs` folder is uploaded at the end of the job, while the files written to `./logs` are uploaded in real time. Use the latter if you want to stream logs during the job, such as TensorBoard logs.

In addition, any files logged from MLflow via autologging or `mlflow.log_*` for artifact logging will get automatically persisted as well. Collectively with the aforementioned `./outputs` and `./logs` directories, this set of files and directories will be persisted to a directory that corresponds to that job's default artifact location.

You can modify the "hello world" job to output to a file in the default outputs directory instead of printing to `stdout`:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world" > ./outputs/helloworld.txt
environment:
  image: library/python:latest
compute: azurerm:cpu-cluster
```

You can run this job:

```
az ml job create -f jobs/basics/hello-world-output.yml --web
```

And download the logs, where `helloworld.txt` will be present in the `<RUN_ID>/outputs/` directory:

```
az ml job download -n $run_id
```

Data outputs

You can specify named data outputs. This will create a directory in the default datastore which will be read/write mounted by default.

You can modify the earlier "hello world" job to write to a named data output:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world" > ${outputs.hello_output}/helloworld.txt
outputs:
  hello_output:
environment:
  image: python
compute: azurerm:cpu-cluster
```

Hello pipelines

Pipeline jobs can run multiple jobs in parallel or in sequence. If there are input/output dependencies between steps in a pipeline, the dependent step will run after the other completes.

You can split a "hello world" job into two jobs:

```
$schema: https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline
jobs:
  hello_job:
    command: echo "hello"
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    compute: azureml:cpu-cluster
  world_job:
    command: echo "world"
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    compute: azureml:cpu-cluster
```

And run it:

```
az ml job create -f jobs/basics/hello-pipeline.yml --web
```

The "hello" and "world" jobs respectively will run in parallel if the compute target has the available resources to do so.

To pass data between steps in a pipeline, define a data output in the "hello" job and a corresponding input in the "world" job, which refers to the prior's output:

```
$schema: https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline_io
jobs:
  hello_job:
    command: echo "hello" && echo "world" > ${{outputs.world_output}}/world.txt
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    compute: azureml:cpu-cluster
    outputs:
      world_output:
  world_job:
    command: cat ${{inputs.world_input}}/world.txt
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:23
    compute: azureml:cpu-cluster
    inputs:
      world_input: ${{parent.jobs.hello_job.outputs.world_output}}
```

And run it:

```
az ml job create -f jobs/basics/hello-pipeline-io.yml --web
```

This time, the "world" job will run after the "hello" job completes.

To avoid duplicating common settings across jobs in a pipeline, you can set them outside the jobs:

```

$schema: https://azurermschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline_settings

settings:
  default_datastore: azureml:workspaceblobstore
  default_compute: azureml:cpu-cluster
jobs:
  hello_job:
    command: echo 202204190 & echo "hello"
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:23
  world_job:
    command: echo 202204190 & echo "hello"
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:23

```

You can run this:

```
az ml job create -f jobs/basics/hello-pipeline-settings.yml --web
```

The corresponding setting on an individual job will override the common settings for a pipeline job. The concepts so far can be combined into a three-step pipeline job with jobs "A", "B", and "C". The "C" job has a data dependency on the "B" job, while the "A" job can run independently. The "A" job will also use an individually set environment and bind one of its inputs to a top-level pipeline job input:

```

$schema: https://azurermschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline_abc
compute: azureml:cpu-cluster

inputs:
  hello_string_top_level_input: "hello world"
jobs:
  a:
    command: echo hello ${{inputs.hello_string}}
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    inputs:
      hello_string: ${{parent.inputs.hello_string_top_level_input}}
  b:
    command: echo "world" >> ${{outputs.world_output}}/world.txt
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    outputs:
      world_output:
  c:
    command: echo ${{inputs.world_input}}/world.txt
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    inputs:
      world_input: ${{parent.jobs.b.outputs.world_output}}

```

You can run this:

```
az ml job create -f jobs/basics/hello-pipeline-abc.yml --web
```

Train a model

In Azure Machine Learning you basically have two possible ways to train a model:

1. Leverage automated ML to train models with your data and get the best model for you. This approach maximizes productivity by automating the iterative process of tuning hyperparameters and trying out

different algorithms.

2. Train a model with your own custom training script. This approach offers the most control and allows you to customize your training.

Train a model with automated ML

Automated ML is the easiest way to train a model because you don't need to know how training algorithms work exactly but you just need to provide your training/validation/test datasets and some basic configuration parameters such as 'ML Task', 'target column', 'primary metric', 'timeout' etc, and the service will train multiple models and try out various algorithms and hyperparameter combinations for you.

When you train with automated ML via the CLI (v2), you just need to create a .YAML file with an AutoML configuration and provide it to the CLI for training job creation and submission.

The following example shows an AutoML configuration file for training a classification model where,

- The primary metric is `accuracy`
- The training has a time out of 180 minutes
- The data for training is in the folder `./training-mltable-folder`. Automated ML jobs only accept data in the form of an `MLTable`.

```
$schema: https://azuremlsdk2.blob.core.windows.net/preview/0.0.1/autoMLJob.schema.json
type: automl

experiment_name: dpv2-cli-automl-classifier-experiment
# name: dpv2-cli-classifier-train-job-basic-01
description: A Classification job using bank marketing

compute: azureml:cpu-cluster

task: classification
primary_metric: accuracy

target_column_name: "y"
training_data:
    path: "./training-mltable-folder"
    type: mltable

limits:
    timeout_minutes: 180
    max_trials: 40
    enable_early_termination: true

featurization:
    mode: auto
```

That mentioned `MLTable` definition is what points to the training data file, in this case a local `.csv` file that will be uploaded automatically:

```
paths:
  - file: ./bank_marketing_train_data.csv
transformations:
  - read_delimited:
      delimiter: ','
      encoding: 'ascii'
```

Finally, you can run it (create the AutoML job) with this CLI command:

```
/> az ml job create --file ./hello-automl-job-basic.yml
```

Or like the following if providing workspace IDs explicitly instead of using the by default workspace:

```
/> az ml job create --file ./hello-automl-job-basic.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

To investigate additional AutoML model training examples using other ML-tasks such as regression, time-series forecasting, image classification, object detection, NLP text-classification, etc., see the complete list of [AutoML CLI examples](#).

Train a model with a custom script

When training by using your own custom script, the first thing you need is that python script (.py), so let's add some `sklearn` code into a Python script with MLflow tracking to train a model on the Iris CSV:

```
# imports
import os
import mlflow
import argparse

import pandas as pd

from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# define functions
def main(args):
    # enable auto logging
    mlflow.autolog()

    # setup parameters
    params = {
        "C": args.C,
        "kernel": args.kernel,
        "degree": args.degree,
        "gamma": args.gamma,
        "coef0": args.coef0,
        "shrinking": args.shrinking,
        "probability": args.probability,
        "tol": args.tol,
        "cache_size": args.cache_size,
        "class_weight": args.class_weight,
        "verbose": args.verbose,
        "max_iter": args.max_iter,
        "decision_function_shape": args.decision_function_shape,
        "break_ties": args.break_ties,
        "random_state": args.random_state,
    }

    # read in data
    df = pd.read_csv(args.iris_csv)

    # process data
    X_train, X_test, y_train, y_test = process_data(df, args.random_state)

    # train model
    model = train_model(params, X_train, X_test, y_train, y_test)

def process_data(df, random_state):
    # split dataframe into X and y
    X = df.drop(["species"], axis=1)
    y = df["species"]

    # train/test split
    X_train, X_test, y_train, y_test = train_test_split(
```

```

        X, y, test_size=0.2, random_state=random_state
    )

    # return split data
    return X_train, X_test, y_train, y_test

def train_model(params, X_train, X_test, y_train, y_test):
    # train model
    model = SVC(**params)
    model = model.fit(X_train, y_train)

    # return model
    return model

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--iris-csv", type=str)
    parser.add_argument("--C", type=float, default=1.0)
    parser.add_argument("--kernel", type=str, default="rbf")
    parser.add_argument("--degree", type=int, default=3)
    parser.add_argument("--gamma", type=str, default="scale")
    parser.add_argument("--coef0", type=float, default=0)
    parser.add_argument("--shrinking", type=bool, default=False)
    parser.add_argument("--probability", type=bool, default=False)
    parser.add_argument("--tol", type=float, default=1e-3)
    parser.add_argument("--cache_size", type=float, default=1024)
    parser.add_argument("--class_weight", type=dict, default=None)
    parser.add_argument("--verbose", type=bool, default=False)
    parser.add_argument("--max_iter", type=int, default=-1)
    parser.add_argument("--decision_function_shape", type=str, default="ovr")
    parser.add_argument("--break_ties", type=bool, default=False)
    parser.add_argument("--random_state", type=int, default=42)

    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # parse args
    args = parse_args()

    # run main function
    main(args)

```

The scikit-learn framework is supported by MLflow for autologging, so a single `mlflow.autolog()` call in the script will log all model parameters, training metrics, model artifacts, and some extra artifacts (in this case a confusion matrix image).

To run this in the cloud, specify as a job:

```
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json
code: src
command: >-
    python main.py
    --iris-csv ${{inputs.iris_csv}}
    --C ${{inputs.C}}
    --kernel ${{inputs.kernel}}
    --coef0 ${{inputs.coef0}}
inputs:
    iris_csv:
        type: uri_file
        path: wasbs://datasets@azuremlexamples.blob.core.windows.net/iris.csv
    C: 0.8
    kernel: "rbf"
    coef0: 0.1
environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
compute: azureml:cpu-cluster
display_name: sklearn-iris-example
experiment_name: sklearn-iris-example
description: Train a scikit-learn SVM on the Iris dataset.
```

And run it:

```
az ml job create -f jobs/single-step/scikit-learn/iris/job.yml --web
```

To register a model, you can upload the model files from the run to the model registry:

```
az ml model create -n sklearn-iris-example -v 1 -p runs:$run_id/model --type mlflow_model
```

For the full set of configurable options for running command jobs, see the [command job YAML schema reference](#).

Sweep hyperparameters

You can modify the previous job to sweep over hyperparameters:

```

$schema: https://azurermschemas.azureedge.net/latest/sweepJob.schema.json
type: sweep
trial:
  code: src
  command: >-
    python main.py
    --iris-csv ${{inputs.iris_csv}}
    --C ${{search_space.C}}
    --kernel ${{search_space.kernel}}
    --coef0 ${{search_space.coef0}}
environment: azurerm:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
inputs:
  iris_csv:
    type: uri_file
    path: wasbs://datasets@azurermexamples.blob.core.windows.net/iris.csv
compute: azurerm:cpu-cluster
sampling_algorithm: random
search_space:
  C:
    type: uniform
    min_value: 0.5
    max_value: 0.9
  kernel:
    type: choice
    values: ["rbf", "linear", "poly"]
  coef0:
    type: uniform
    min_value: 0.1
    max_value: 1
objective:
  goal: minimize
  primary_metric: training_f1_score
limits:
  max_total_trials: 20
  max_concurrent_trials: 10
  timeout: 7200
display_name: sklearn-iris-sweep-example
experiment_name: sklearn-iris-sweep-example
description: Sweep hyperparameters for training a scikit-learn SVM on the Iris dataset.

```

And run it:

```
az ml job create -f jobs/single-step/scikit-learn/iris/job-sweep.yml --web
```

TIP

Check the "Child runs" tab in the studio to monitor progress and view parameter charts..

For the full set of configurable options for sweep jobs, see the [sweep job YAML schema reference](#).

Distributed training

Azure Machine Learning supports PyTorch, TensorFlow, and MPI-based distributed training. See the [distributed section of the command job YAML syntax reference](#) for details.

As an example, you can train a convolutional neural network (CNN) on the CIFAR-10 dataset using distributed PyTorch. The full script is [available in the examples repository](#).

The CIFAR-10 dataset in `torchvision` expects as input a directory that contains the `cifar-10-batches-py` directory. You can download the zipped source and extract into a local directory:

```
mkdir data  
  
wget "https://azuremlexamples.blob.core.windows.net/datasets/cifar-10-python.tar.gz"  
  
tar -xvzf cifar-10-python.tar.gz -C data
```

Then create an Azure Machine Learning data asset from the local directory, which will be uploaded to the default datastore:

```
az ml data create --name cifar-10-example --version 1 --set path=data
```

Optionally, remove the local file and directory:

```
rm cifar-10-python.tar.gz  
rm -r data
```

Registered data assets can be used as inputs to job using the `path` field for a job input. The format is `azureml:<data_name>:<data_version>`, so for the CIFAR-10 dataset just created, it is `azureml:cifar-10-example:1`. You can optionally use the `azureml:<data_name>@latest` syntax instead if you want to reference the latest version of the data asset. Azure ML will resolve that reference to the explicit version.

With the data asset in place, you can author a distributed PyTorch job to train our model:

```
$schema: https://azureschemas.azureedge.net/latest/commandJob.schema.json  
code: src  
command: >-  
  python train.py  
  --epochs ${{inputs.epochs}}  
  --learning-rate ${{inputs.learning_rate}}  
  --data-dir ${{inputs.cifar}}  
inputs:  
  epochs: 1  
  learning_rate: 0.2  
  cifar:  
    type: uri_folder  
    path: azureml:cifar-10-example:1  
environment: azureml:AzureML-pytorch-1.9-ubuntu18.04-py37-cuda11-gpu@latest  
compute: azureml:gpu-cluster  
distribution:  
  type: pytorch  
  process_count_per_instance: 1  
resources:  
  instance_count: 2  
display_name: pytorch-cifar-distributed-example  
experiment_name: pytorch-cifar-distributed-example  
description: Train a basic convolutional neural network (CNN) with PyTorch on the CIFAR-10 dataset, distributed via PyTorch.
```

And run it:

```
az ml job create -f jobs/single-step/pytorch/cifar-distributed/job.yml --web
```

Build a training pipeline

The CIFAR-10 example above translates well to a pipeline job. The previous job can be split into three jobs for orchestration in a pipeline:

- "get-data" to run a Bash script to download and extract `cifar-10-batches-py`
- "train-model" to take the data and train a model with distributed PyTorch
- "eval-model" to take the data and the trained model and evaluate accuracy

Both "train-model" and "eval-model" will have a dependency on the "get-data" job's output. Additionally, "eval-model" will have a dependency on the "train-model" job's output. Thus the three jobs will run sequentially.

Pipelines can also be written using reusable components. For more, see [Create and run components-based machine learning pipelines with the Azure Machine Learning CLI \(Preview\)](#).

Next steps

- [Deploy and score a machine learning model with a managed online endpoint \(preview\)](#)

Train models with REST (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

Learn how to use the Azure Machine Learning REST API to create and manage training jobs (preview).

The REST API uses standard HTTP verbs to create, retrieve, update, and delete resources. The REST API works with any language or tool that can make HTTP requests. REST's straightforward structure makes it a good choice in scripting environments and for MLOps automation.

In this article, you learn how to use the new REST APIs to:

- Create machine learning assets
- Create a basic training job
- Create a hyperparameter tuning sweep job

Prerequisites

- An **Azure subscription** for which you have administrative rights. If you don't have such a subscription, try the [free or paid personal subscription](#).
- An [Azure Machine Learning workspace](#).
- A service principal in your workspace. Administrative REST requests use [service principal authentication](#).
- A service principal authentication token. Follow the steps in [Retrieve a service principal authentication token](#) to retrieve this token.
- The `curl` utility. The `curl` program is available in the [Windows Subsystem for Linux](#) or any UNIX distribution. In PowerShell, `curl` is an alias for `Invoke-WebRequest` and `curl -d "key=val" -X POST uri` becomes `Invoke-WebRequest -Body "key=val" -Method POST -Uri uri`.

Azure Machine Learning jobs

A job is a resource that specifies all aspects of a computation job. It aggregates three things:

- What to run?
- How to run it?
- Where to run it?

There are many ways to submit an Azure Machine Learning job including the SDK, Azure CLI, and visually with the studio. The following example submits a LightGBM training job with the REST API.

Create machine learning assets

First, set up your Azure Machine Learning assets to configure your job.

In the following REST API calls, we use `$SUBSCRIPTION_ID`, `$RESOURCE_GROUP`, `$LOCATION`, and `$WORKSPACE` as placeholders. Replace the placeholders with your own values.

Administrative REST requests a [service principal authentication token](#). Replace `$TOKEN` with your own value. You can retrieve this token with the following command:

```
TOKEN=$(az account get-access-token --query accessToken -o tsv)
```

The service provider uses the `api-version` argument to ensure compatibility. The `api-version` argument varies from service to service. Set the API version as a variable to accommodate future versions:

```
API_VERSION="2022-05-01"
```

Compute

Running machine learning jobs requires compute resources. You can list your workspace's compute resources:

```
curl  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/computes?api-version=$API_VERSION" \  
--header "Authorization: Bearer $TOKEN"
```

For this example, we use an existing compute cluster named `cpu-cluster`. We set the compute name as a variable for encapsulation:

```
COMPUTE_NAME="cpu-cluster"
```

TIP

You can [create or overwrite a named compute resource with a PUT request](#).

Environment

The LightGBM example needs to run in a LightGBM environment. Create the environment with a PUT request. Use a docker image from Microsoft Container Registry.

You can configure the docker image with `Docker` and add conda dependencies with `condaFile`:

```
ENV_VERSION=$RANDOM  
curl --location --request PUT  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/environments/lightgbm-environment/versions/$ENV_VERSION?  
api-version=$API_VERSION" \  
--header "Authorization: Bearer $TOKEN" \  
--header "Content-Type: application/json" \  
--data-raw "{  
    \"properties\":{  
        \"condaFile\": \"$CONDA_FILE\",  
        \"image\": \"mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04\"  
    }  
}"
```

Datastore

The training job needs to run on data, so you need to specify a datastore. In this example, you get the default datastore and Azure Storage account for your workspace. Query your workspace with a GET request to return a JSON file with the information.

You can use the tool `jq` to parse the JSON result and get the required values. You can also use the Azure portal to find the same information.

```

response=$(curl --location --request GET
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/datastores?api-version=$API_VERSION&isDefault=true" \
--header "Authorization: Bearer $TOKEN")

AZURE_STORAGE_ACCOUNT=$(echo $response | jq '.value[0].properties.contents.accountName')
AZUREML_DEFAULT_DATASTORE=$(echo $response | jq '.value[0].name')
AZUREML_DEFAULT_CONTAINER=$(echo $response | jq '.value[0].properties.contents.containerName')
AZURE_STORAGE_KEY=$(az storage account keys list --account-name $AZURE_STORAGE_ACCOUNT | jq '.[0].value')

```

Data

Now that you have the datastore, you can create a dataset. For this example, use the common dataset `iris.csv`.

```

DATA_VERSION=$RANDOM
curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/data/iris-data/versions/$DATA_VERSION?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\": {
    \"description\": \"Iris dataset\",
    \"dataType\": \"UriFile\",
    \"dataUri\": \"https://azuremlexamples.blob.core.windows.net/datasets/iris.csv\"
  }
}"

```

Code

Now that you have the dataset and datastore, you can upload the training script that will run on the job. Use the Azure Storage CLI to upload a blob into your default container. You can also use other methods to upload, such as the Azure portal or Azure Storage Explorer.

```

az storage blob upload-batch -d $AZUREML_DEFAULT_CONTAINER/src \
-s jobs/train/lightgbm/iris/src --account-name $AZURE_STORAGE_ACCOUNT --account-key $AZURE_STORAGE_KEY

```

Once you upload your code, you can specify your code with a PUT request and reference the url through `codeUri`.

```

curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/codes/train-lightgbm/versions/1?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\": {
    \"description\": \"Train code\",
    \"codeUri\": \"https://trainws1352661735.blob.core.windows.net/training-scripts/main.py\"
  }
}"

```

Submit a training job

Now that your assets are in place, you can run the LightGBM job, which outputs a trained model and metadata. You need the following information to configure the training job:

- **run_id**: [Optional] The name of the job, which must be unique across all jobs. Unless a name is specified either in the YAML file via the `name` field or the command line via `--name/-n`, a GUID/UUID is automatically generated and used for the name.
- **jobType**: The job type. For a basic training job, use `Command`.
- **codeId**: The ARMid reference of the name and version of your training script.
- **command**: The command to execute. Input data can be written into the command and can be referred to with data binding.
- **environmentId**: The ARMid reference of the name and version of your environment.
- **inputDataBindings**: Data binding can help you reference input data. Create an environment variable and the name of the binding will be added to `AZURE_ML_INPUT_`, which you can refer to in `command`. You can directly reference a public blob url file as a `UriFile` through the `uri` parameter.
- **experimentName**: [Optional] Tags the job to help you organize jobs in Azure Machine Learning studio. Each job's run record is organized under the corresponding experiment in the studio "Experiment" tab. If omitted, tags default to the name of the working directory when the job is created.
- **computeId**: The `computeId` specifies the compute target name through an ARMid.

Use the following commands to submit the training job:

```
run_id=$(uuidgen)
curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/jobs/$run_id?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    "properties": {
        "jobType": "Command",
        "codeId": "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/codes/train-lightgbm/versions/1\",
        "command": "python main.py --iris-csv \$AZURE_ML_INPUT_iris\",
        "environmentId": "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/environments/lightgbm-environment/versions/$ENV_VERSION\",
        "inputDataBindings": {
            "iris": {
                "jobInputType": "UriFile",
                "uri": "https://azuremlexamples.blob.core.windows.net/datasets/iris.csv"
            }
        },
        "experimentName": "lightgbm-iris",
        "computeId": "/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/computes/$COMPUTE_NAME"
    }
}"
```

Submit a hyperparameter sweep job

Azure Machine Learning also lets you efficiently tune training hyperparameters. You can create a hyperparameter tuning suite, with the REST APIs. For more information on Azure Machine Learning's hyperparameter tuning options, see [Hyperparameter tuning a model](#). Specify the hyperparameter tuning parameters to configure the sweep:

- **jobType**: The job type. For a sweep job, it will be `Sweep`.
- **algorithm**: The sampling algorithm class - class "random" is often a good place to start. See the sweep job [schema](#) for the enumeration of options.

- **trial**: The command job configuration for each trial to be run.
- **objective**: The `primaryMetric` is the optimization metric, which must match the name of a metric logged from the training code. The `goal` specifies the direction (minimize or maximize). See the [schema](#) for the full enumeration of options.
- **searchSpace**: A generic object of hyperparameters to sweep over. The key is a name for the hyperparameter, for example, `learning_rate`. The value is the hyperparameter distribution. See the [schema](#) for the enumeration of options.
- **Limits**: `JobLimitsType` of type `sweep` is an object definition of the sweep job limits parameters. `maxTotalTrials` [Optional] is the maximum number of individual trials to run. `maxConcurrentTrials` is the maximum number of trials to run concurrently on your compute cluster.

To create a sweep job with the same LightGBM example, use the following commands:

```
run_id=$(uuidgen)
curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/jobs/$run_id?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    \"properties\": {
        \"samplingAlgorithm\": {
            \"samplingAlgorithmType\": \"Random\",
        },
        \"jobType\": \"Sweep\",
        \"trial\":{
            \"codeId\": \"$/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/codes/train-lightgbm/versions/1\",
            \"command\": \"python main.py --iris-csv \$AZURE_ML_INPUT_iris\",
            \"environmentId\": \"$/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/environments/lightgbm-environment/versions/$ENV_VERSION\",
        },
        \"experimentName\": \"lightgbm-iris-sweep\",
        \"computeId\": \"target\",
    },
    \"objective\": {
        \"primaryMetric\": \"test-multi_logloss\",
        \"goal\": \"minimize\"
    },
    \"searchSpace\": {
        \"learning_rate\": [\"uniform\", [0.01, 0.9]],
        \"boosting\": [\"choice\", [[\"gbdt\", \"dart\"]]]
    },
    \"limits\": {
        \"jobLimitsType\": \"sweep\",
        \"maxTotalTrials\": 20,
        \"maxConcurrentTrials\": 10,
    }
}
}"
```

Next steps

Now that you have a trained model, learn [how to deploy your model](#).

Monitor and analyze jobs in studio

9/22/2022 • 4 minutes to read • [Edit Online](#)

You can use [Azure Machine Learning studio](#) to monitor, organize, and track your jobs for training and experimentation. Your ML job history is an important part of an explainable and repeatable ML development process.

This article shows how to do the following tasks:

- Add job display name.
- Create a custom view.
- Add a job description.
- Tag and find jobs.
- Run search over your job history.
- Cancel or fail jobs.
- Monitor the job status by email notification.
- Monitor your job resources (preview)

TIP

- If you're looking for information on using the Azure Machine Learning SDK v1 or CLI v1, see [How to track, monitor, and analyze jobs \(v1\)](#).
- If you're looking for information on monitoring training jobs from the CLI or SDK v2, see [Track experiments with MLflow and CLI v2](#).
- If you're looking for information on monitoring the Azure Machine Learning service and associated Azure services, see [How to monitor Azure Machine Learning](#).

If you're looking for information on monitoring models deployed to online endpoints, see [Monitor online endpoints](#).

Prerequisites

You'll need the following items:

- To use Azure Machine Learning, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- You must have an Azure Machine Learning workspace. A workspace is created in [Install, set up, and use the CLI \(v2\)](#).

Job display name

The job display name is an optional and customizable name that you can provide for your job. To edit the job display name:

1. Navigate to the **Jobs** list.
2. Select the job to edit.

Microsoft Azure Machine Learning Studio

Microsoft > sdg-ws > Jobs

Jobs

All experiments All jobs

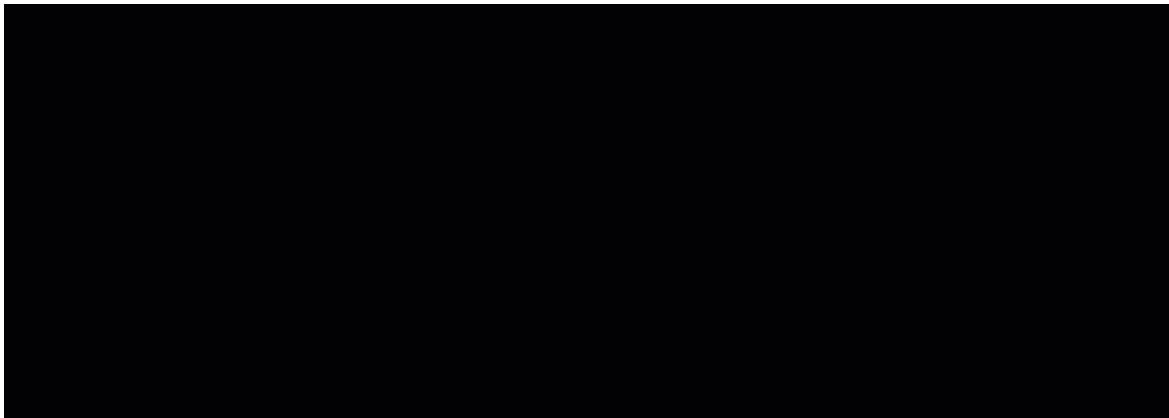
Refresh Archive experiment Edit columns Reset view

Search View archived experiments

Showing 1-7 experiments

Experiment	Latest job	Last submitted
DesignerRegression	Regression - Automobile Price Prediction (Comp...)	Jun 9, 2022 11:45 AM
LabelingExperiment	purple_insect_ryfqcmt6	Jun 3, 2022 9:54 AM
e2e_registered_components	credit_defaults_pipeline	May 20, 2022 3:41 PM
lightgbm-iris-local-example	ivory_mango_b1v3fszy	May 20, 2022 3:32 PM

3. Select the **Edit** button to edit the job display name.



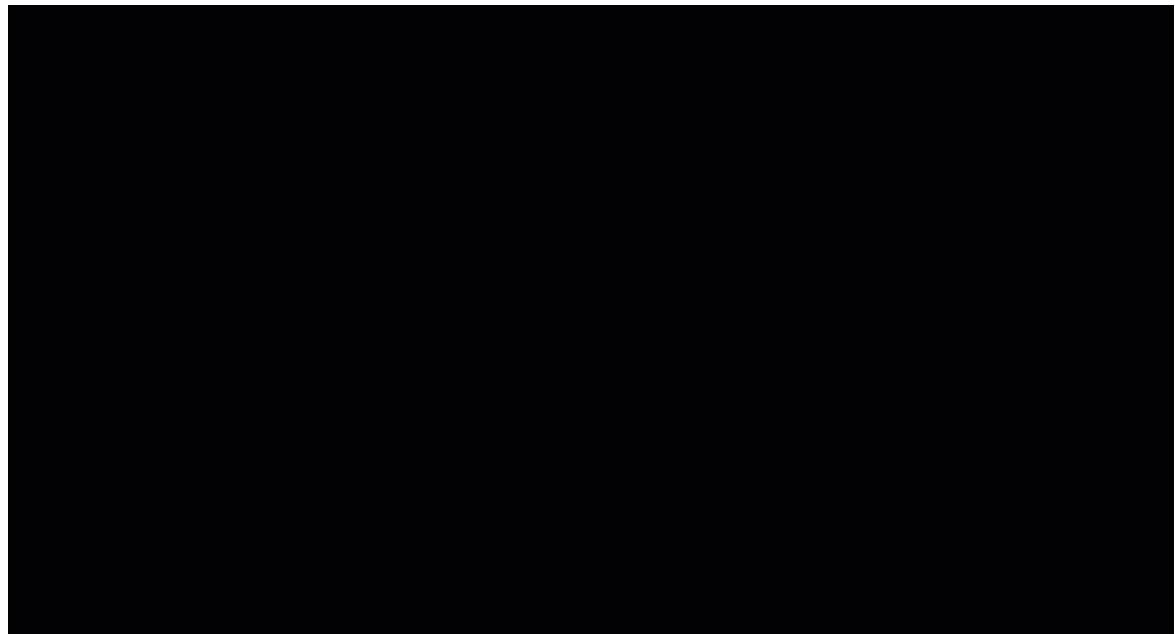
Custom View

To view your jobs in the studio:

1. Navigate to the **Jobs** tab.
2. Select either **All experiments** to view all the jobs in an experiment or select **All jobs** to view all the jobs submitted in the Workspace.

In the **All jobs**' page, you can filter the jobs list by tags, experiments, compute target and more to better organize and scope your work.

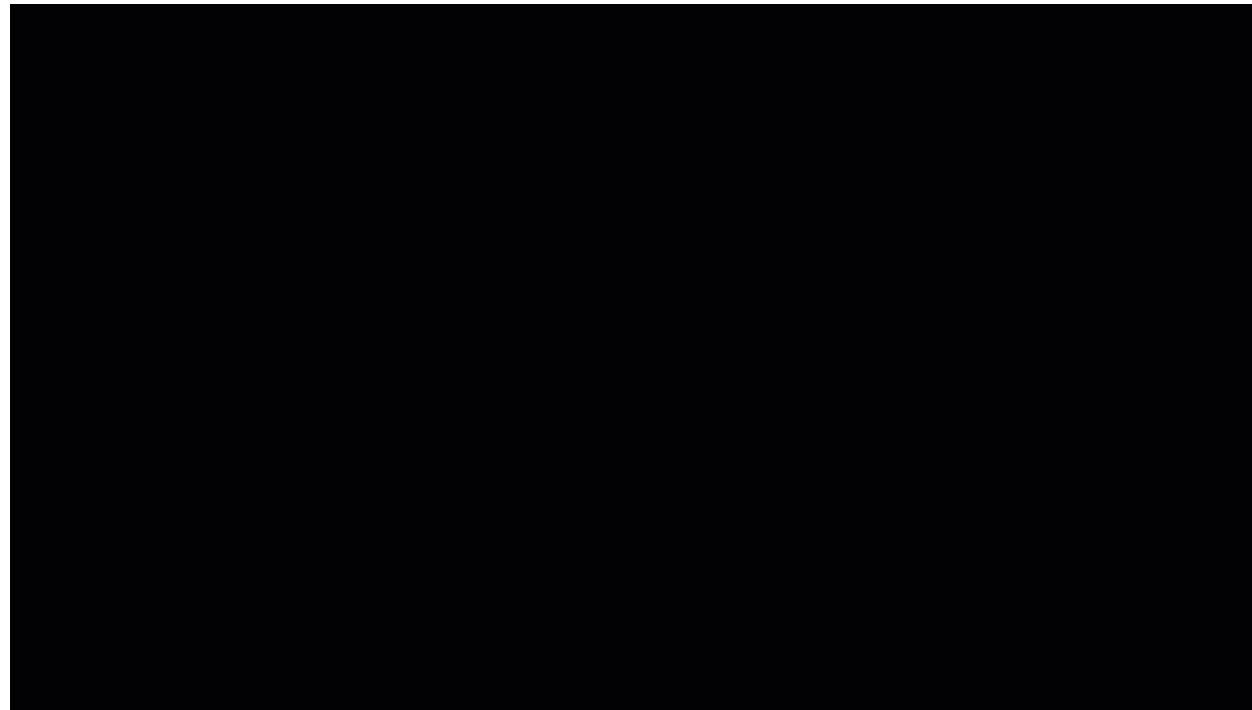
1. Make customizations to the page by selecting jobs to compare, adding charts or applying filters. These changes can be saved as a **Custom View** so you can easily return to your work. Users with workspace permissions can edit, or view the custom view. Also, share the custom view with team members for enhanced collaboration by selecting **Share view**.
2. To view the job logs, select a specific job and in the **Outputs + logs** tab, you can find diagnostic and error logs for your job.



Job description

A job description can be added to a job to provide more context and information to the job. You can also search on these descriptions from the jobs list and add the job description as a column in the jobs list.

Navigate to the **Job Details** page for your job and select the edit or pencil icon to add, edit, or delete descriptions for your job. To persist the changes to the jobs list, save the changes to your existing Custom View or a new Custom View. Markdown format is supported for job descriptions, which allows images to be embedded and deep linking as shown below.



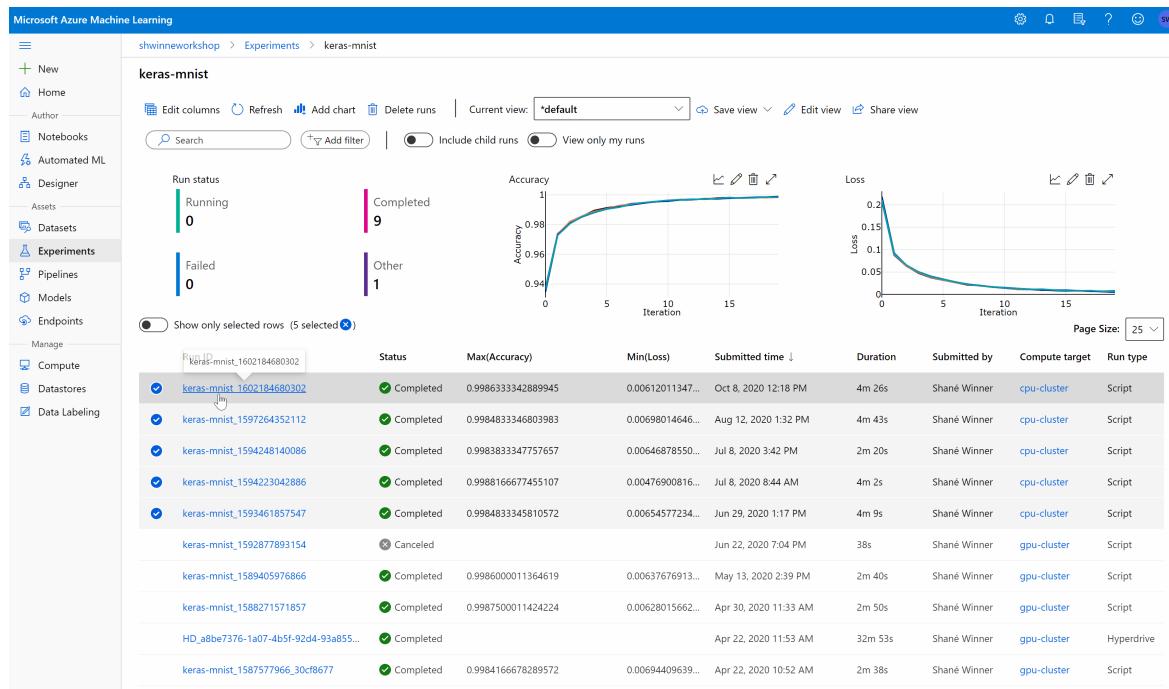
Tag and find jobs

In Azure Machine Learning, you can use properties and tags to help organize and query your jobs for important information.

- Edit tags

You can add, edit, or delete job tags from the studio. Navigate to the **Job Details** page for your job and

select the edit, or pencil icon to add, edit, or delete tags for your jobs. You can also search and filter on these tags from the jobs list page.



- Query properties and tags

You can query jobs within an experiment to return a list of jobs that match specific properties and tags.

To search for specific jobs, navigate to the **All jobs** list. From there you have two options:

1. Use the **Add filter** button and select filter on tags to filter your jobs by tag that was assigned to the job(s).

OR

2. Use the search bar to quickly find jobs by searching on the job metadata like the job status, descriptions, experiment names, and submitter name.

Cancel or fail jobs

If you notice a mistake or if your job is taking too long to finish, you can cancel the job.

To cancel a job in the studio, using the following steps:

1. Go to the running pipeline in either the **Jobs** or **Pipelines** section.
2. Select the pipeline job number you want to cancel.
3. In the toolbar, select **Cancel**.

Monitor the job status by email notification

1. In the [Azure portal](#), in the left navigation bar, select the **Monitor** tab.
2. Select **Diagnostic settings** and then select **+ Add diagnostic setting**.

Dashboard >

Diagnostic settings

Machine learning

Search (Ctrl+ /) Refresh Feedback

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Events

Settings

- Private endpoint connections
- Properties
- Locks

Monitoring

- Alerts
- Metrics
- Diagnostic settings** (selected)
- Logs

Automation

- Tasks (preview)
- Export template

Support + troubleshooting

- Usage + quotas
- New support request

Diagnostic settings

Name	Storage account
diag	-

+ Add diagnostic setting

Click 'Add Diagnostic setting' above to configure the collection of the following data:

- AmlComputeClusterEvent
- AmlComputeClusterNodeEvent
- AmlComputeJobEvent
- AmlComputeCpuGpuUtilization
- AmlRunStatusChangedEvent
- AllMetrics

3. In the Diagnostic Setting,

- under the **Category details**, select the **AmlRunStatusChangedEvent**.
- In the **Destination details**, select the **Send to Log Analytics workspace** and specify the **Subscription** and **Log Analytics workspace**.

NOTE

The **Azure Log Analytics Workspace** is a different type of Azure Resource than the **Azure Machine Learning service Workspace**. If there are no options in that list, you can [create a Log Analytics Workspace](#).

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name *

Category details

log

- AmlComputeClusterEvent
- AmlComputeClusterNodeEvent
- AmlComputeJobEvent
- AmlComputeCpuGpuUtilization
- AmlRunStatusChangedEvent

metric

- AllMetrics

Destination details

Send to Log Analytics workspace

Subscription

Log Analytics workspace

Archive to a storage account

Stream to an event hub

4. In the Logs tab, add a New alert rule.

selecting another time range'."/>

Machine learning

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Events

Settings

Properties

Locks

Monitoring

Alerts

Metrics

Diagnostic settings

Logs

New Query 1*

Select scope

Run Time range : Last 24 hours

1 | AmlRunStatusChangedEvent
2 | where Status == "Failed" or Status == "Canceled"
3 | limit 1

Results Chart

Completed

No results found from the last 24 hours
Try [selecting another time range](#)

5. See [how to create and manage log alerts using Azure Monitor](#).

Monitor your job resources (preview)

Navigate to your job in the studio and select the Monitoring tab. This view provides insights on your job's resources on a 30 day rolling basis.



NOTE

This view supports only compute that is managed by AzureML. Jobs with a runtime of less than 5 minutes will not have enough data to populate this view.

Next steps

- To learn how to log metrics for your experiments, see [Log metrics during training jobs](#).
- To learn how to monitor resources and logs from Azure Machine Learning, see [Monitoring Azure Machine Learning](#).

Log metrics, parameters and files with MLflow

9/22/2022 • 10 minutes to read • [Edit Online](#)

Azure Machine Learning supports logging and tracking experiments using [MLflow Tracking](#). You can log models, metrics, parameters, and artifacts with MLflow as it supports local mode to cloud portability.

IMPORTANT

Unlike the Azure Machine Learning SDK v1, there is no logging functionality in the Azure Machine Learning SDK for Python (v2). See this guidance to learn how to log with MLflow. If you were using Azure Machine Learning SDK v1 before, we recommend you to start leveraging MLflow for tracking experiments. See [Migrate logging from SDK v1 to MLflow](#) for specific guidance.

Logs can help you diagnose errors and warnings, or track performance metrics like parameters and model performance. In this article, you learn how to enable logging in the following scenarios:

- Log metrics, parameters and models when submitting jobs.
- Tracking runs when training interactively.
- Viewing diagnostic information about training.

TIP

This article shows you how to monitor the model training process. If you're interested in monitoring resource usage and events from Azure Machine learning, such as quotas, completed training jobs, or completed model deployments, see [Monitoring Azure Machine Learning](#).

TIP

For information on logging metrics in Azure Machine Learning designer, see [How to log metrics in the designer](#).

Prerequisites

- You must have an Azure Machine Learning workspace. [Create one if you don't have any](#).
- You must have `mlflow`, and `azureml-mlflow` packages installed. If you don't, use the following command to install them in your development environment:

```
pip install mlflow azureml-mlflow
```

- If you are doing remote tracking (tracking experiments running outside Azure Machine Learning), configure MLflow to track experiments using Azure Machine Learning. See [Setup your tracking environment](#) for more details.

Getting started

To log metrics, parameters, artifacts and models in your experiments in Azure Machine Learning using MLflow, just import MLflow in your training script:

```
import mlflow
```

Configuring experiments

MLflow organizes the information in experiments and runs (in Azure Machine Learning, runs are called Jobs). There are some differences in how to configure them depending on how you are running your code:

- [Training interactively](#)
- [Training with jobs](#)

When training interactively, such as in a Jupyter Notebook, use the following pattern:

1. Create or set the active experiment.
2. Start the job.
3. Use logging methods to log metrics and other information.
4. End the job.

For example, the following code snippet demonstrates configuring the experiment, and then logging during a job:

```
import mlflow
mlflow.set_experiment("mlflow-experiment")

# Start the run, log metrics, end the run
mlflow_run = mlflow.start_run()
mlflow.log_metric('mymetric', 1)
mlflow.end_run()
```

TIP

Technically you don't have to call `start_run()` as a new run is created if one doesn't exist and you call a logging API. In that case, you can use `mlflow.active_run()` to retrieve the run. However, the `mlflow.ActiveRun` object returned by `mlflow.active_run()` won't contain items like parameters, metrics, etc. For more information, see [mlflow.active_run\(\)](#).

You can also use the context manager paradigm:

```
import mlflow
mlflow.set_experiment("mlflow-experiment")

# Start the run, log metrics, end the run
with mlflow.start_run() as run:
    # Run started when context manager is entered, and ended when context manager exits
    mlflow.log_metric('mymetric', 1)
    mlflow.log_metric('anothermetric',1)
    pass
```

When you start a new run with `mlflow.start_run`, it may be useful to indicate the parameter `run_name` which will then translate to the name of the run in Azure Machine Learning user interface and help you identify the run quicker:

```
with mlflow.start_run(run_name="iris-classifier-random-forest") as run:
    mlflow.log_metric('mymetric', 1)
    mlflow.log_metric('anothermetric',1)
```

For more information on MLflow logging APIs, see the [MLflow reference](#).

Logging parameters

MLflow supports the logging parameters used by your experiments. Parameters can be of any type, and can be logged using the following syntax:

```
mlflow.log_param("num_epochs", 20)
```

MLflow also offers a convenient way to log multiple parameters by indicating all of them using a dictionary. Several frameworks can also pass parameters to models using dictionaries and hence this is a convenient way to log them in the experiment.

```
params = {  
    "num_epochs": 20,  
    "dropout_rate": .6,  
    "objective": "binary_crossentropy"  
}  
  
mlflow.log_params(params)
```

NOTE

Azure ML SDK v1 logging can't log parameters. We recommend the use of MLflow for tracking experiments as it offers a superior set of features.

Logging metrics

Metrics, as opposite to parameters, are always numeric. The following table describes how to log specific numeric types:

LOGGED VALUE	EXAMPLE CODE	NOTES
Log a numeric value (int or float)	<pre>mlflow.log_metric("my_metric", 1)</pre>	
Log a numeric value (int or float) over time	<pre>mlflow.log_metric("my_metric", 1, step=1)</pre>	Use parameter <code>step</code> to indicate the step at which you are logging the metric value. It can be any integer number. It defaults to zero.
Log a boolean value	<pre>mlflow.log_metric("my_metric", 0)</pre>	0 = True, 1 = False

IMPORTANT

Performance considerations: If you need to log multiple metrics (or multiple values for the same metric) avoid making calls to `mlflow.log_metric` in loops. Better performance can be achieved by logging batch of metrics. Use the method `mlflow.log_metrics` which accepts a dictionary with all the metrics you want to log at once or use `mlflow.log_batch` which accepts multiple type of elements for logging.

Logging curves or list of values

Curves (or list of numeric values) can be logged with MLflow by logging the same metric multiple times. The following example shows how to do it:

```

list_to_log = [1, 2, 3, 2, 1, 2, 3, 2, 1]
from mlflow.entities import Metric
from mlflow.tracking import MlflowClient
import time

client = MlflowClient()
client.log_batch(mlflow.active_run().info.run_id,
                 metrics=[Metric(key="sample_list", value=val, timestamp=int(time.time() * 1000), step=0)
                           for val in list_to_log])

```

Logging images

MLflow supports two ways of logging images:

LOGGED VALUE	EXAMPLE CODE	NOTES
Log numpy metrics or PIL image objects	<code>mlflow.log_image(img, "figure.png")</code>	<code>img</code> should be an instance of <code>numpy.ndarray</code> or <code>PIL.Image.Image</code> . <code>figure.png</code> is the name of the artifact that will be generated inside of the run. It doesn't have to be an existing file.
Log matplotlib plot or image file	<code>mlflow.log_figure(fig, "figure.png")</code>	<code>figure.png</code> is the name of the artifact that will be generated inside of the run. It doesn't have to be an existing file.

Logging other types of data

LOGGED VALUE	EXAMPLE CODE	NOTES
Log text in a text file	<code>mlflow.log_text("text string", "notes.txt")</code>	Text is persisted inside of the run in a text file with name <code>notes.txt</code> .
Log dictionaries as <code>JSON</code> and <code>YAML</code> files	<code>mlflow.log_dict(dictionary, "file.yaml")</code>	<code>dictionary</code> is a dictionary object containing all the structure that you want to persist as <code>JSON</code> or <code>YAML</code> file.
Log a trivial file already existing	<code>mlflow.log_artifact("path/to/file.pkl")</code>	Files are always logged in the root of the run. If <code>artifact_path</code> is provided, then the file is logged in a folder as indicated in that parameter.
Log all the artifacts in an existing folder	<code>mlflow.log_artifacts("path/to/folder")</code>	Folder structure is copied to the run, but the root folder indicated is not included.

Logging models

MLflow introduces the concept of "models" as a way to package all the artifacts required for a given model to function. Models in MLflow are always a folder with an arbitrary number of files, depending on the framework used to generate the model. Logging models has the advantage of tracking all the elements of the model as a single entity that can be **registered** and then **deployed**. On top of that, MLflow models enjoy the benefit of **no-code deployment** and can be used with the [Responsible AI dashboard](#) in studio. Read the article [From artifacts](#)

to models in [MLflow](#) for more information.

To save the model from a training run, use the `log_model()` API for the framework you're working with. For example, `mlflow.sklearn.log_model()`. For more details about how to log MLflow models see [Logging MLflow models](#). For migrating existing models to MLflow, see [Convert custom models to MLflow](#).

Automatic logging

With Azure Machine Learning and MLflow, users can log metrics, model parameters and model artifacts automatically when training a model. Each framework decides what to track automatically for you. A [variety of popular machine learning libraries](#) are supported. [Learn more about Automatic logging with MLflow](#).

To enable [automatic logging](#) insert the following code before your training code:

```
mlflow.autolog()
```

TIP

You can control what gets automatically logged with autolog. For instance, if you indicate

```
mlflow.autolog(log_models=False)
```

MLflow will log everything but models for you. Such control is useful in cases where you want to log models manually but still enjoy automatic logging of metrics and parameters. Also notice that some frameworks may disable automatic logging of models if the trained model goes beyond specific boundaries. Such behavior depends on the flavor used and we recommend you to view their documentation if this is your case.

View jobs/runs information with MLflow

You can view the logged information using MLflow through the `MLflow.entities.Run` object:

```
import mlflow

run = mlflow.get_run(run_id=<RUN_ID>)
```

You can view the metrics, parameters, and tags for the run in the data field of the run object.

```
metrics = run.data.metrics
params = run.data.params
tags = run.data.tags
```

NOTE

The metrics dictionary returned by `mlflow.get_run` or `mlflow.search_runs` only returns the most recently logged value for a given metric name. For example, if you log a metric called `iteration` multiple times with values, `1`, then `2`, then `3`, then `4`, only `4` is returned when calling `run.data.metrics['iteration']`.

To get all metrics logged for a particular metric name, you can use `MLFlowClient.get_metric_history()` as explained in the example [Getting params and metrics from a run](#).

TIP

MLflow can retrieve metrics and parameters from multiple runs at the same time, allowing for quick comparisons across multiple trials. Learn about this in [Query & compare experiments and runs with MLflow](#).

Any artifact logged by a run can be queried by MLflow. Artifacts can't be accessed using the run object itself and the MLflow client should be used instead:

```
client = mlflow.tracking.MlflowClient()  
client.list_artifacts("<RUN_ID>")
```

The method above will list all the artifacts logged in the run, but they will remain stored in the artifacts store (Azure ML storage). To download any of them, use the method `download_artifact`:

```
file_path = client.download_artifacts("<RUN_ID>", path="feature_importance_weight.png")
```

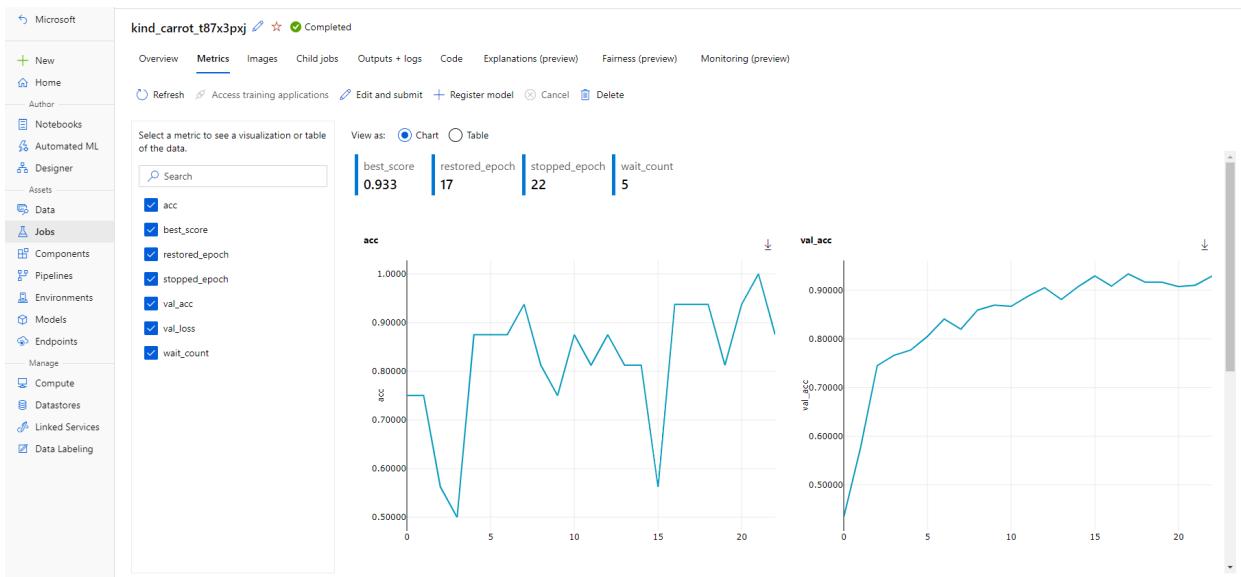
For more information please refer to [Getting metrics, parameters, artifacts and models](#).

View jobs/runs information in the studio

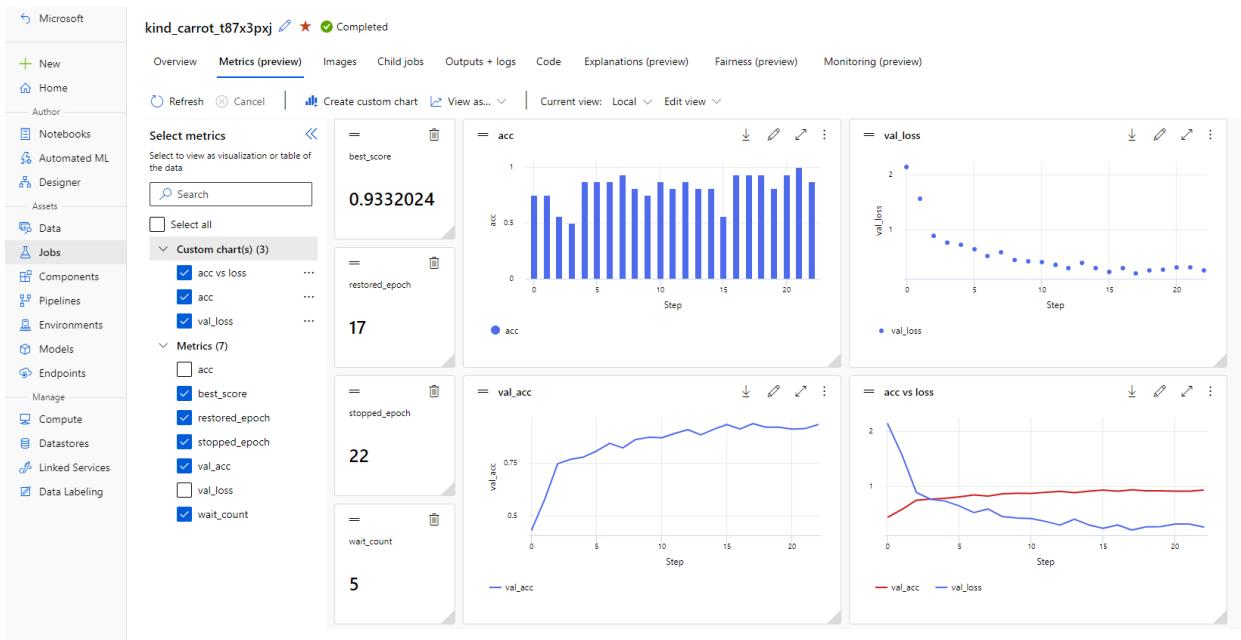
You can browse completed job records, including logged metrics, in the [Azure Machine Learning studio](#).

Navigate to the **Jobs** tab. To view all your jobs in your Workspace across Experiments, select the **All jobs** tab. You can drill down on jobs for specific Experiments by applying the Experiment filter in the top menu bar. Click on the job of interest to enter the details view, and then select the **Metrics** tab.

Select the logged metrics to render charts on the right side.



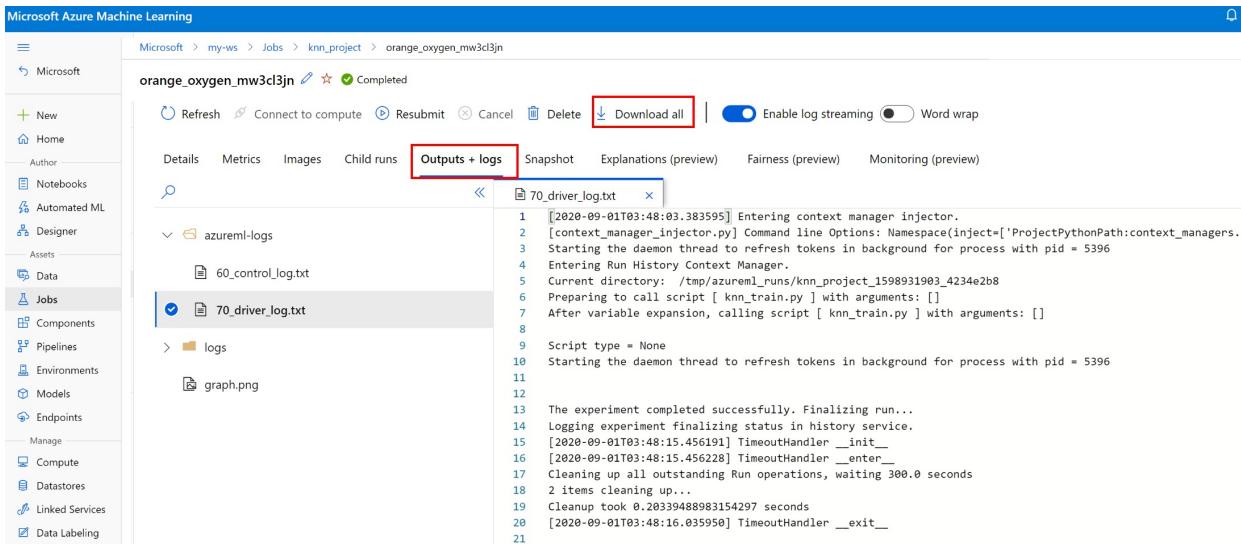
For a customizable view of your job metrics (preview), use the [preview panel](#) to enable the feature. Once enabled, you can add/remove charts and customize them by applying smoothing, changing the color, or plotting multiple metrics on a single graph. You can also resize and rearrange the layout as you wish. Once you have created your desired view, you can save it for future use and share it with your teammates using a direct link.



View and download diagnostic logs

Log files are an essential resource for debugging the Azure ML workloads. After submitting a training job, drill down to a specific run to view its logs and outputs:

1. Navigate to the **Jobs** tab.
2. Select the runID for a specific run.
3. Select **Outputs and logs** at the top of the page.
4. Select **Download all** to download all your logs into a zip folder.
5. You can also download individual log files by choosing the log file and selecting **Download**



user_logs folder

This folder contains information about the user generated logs. This folder is open by default, and the **std_log.txt** log is selected. The **std_log.txt** is where your code's logs (for example, print statements) show up. This file contains **stdout** log and **stderr** logs from your control script and training script, one per process. In most cases, you'll monitor the logs here.

system_logs folder

This folder contains the logs generated by Azure Machine Learning and it will be closed by default. The logs generated by the system are grouped into different folders, based on the stage of the job in the runtime.

Other folders

For jobs training on multi-compute clusters, logs are present for each node IP. The structure for each node is the

same as single node jobs. There's one more logs folder for overall execution, stderr, and stdout logs.

Azure Machine Learning logs information from various sources during training, such as AutoML or the Docker container that runs the training job. Many of these logs aren't documented. If you encounter problems and contact Microsoft support, they may be able to use these logs during troubleshooting.

Next steps

- [Train ML models with MLflow and Azure Machine Learning](#).
- [Migrate from SDK v1 logging to MLflow tracking](#).

Logging MLflow models

9/22/2022 • 10 minutes to read • [Edit Online](#)

The following article explains how to start logging your trained models (or artifacts) as MLflow models. It explores the different methods to customize the way MLflow packages your models and hence how it runs them.

Why logging models instead of artifacts?

If you are not familiar with MLflow, you may not be aware of the difference between logging artifacts or files vs. logging MLflow models. We recommend reading the article [From artifacts to models in MLflow](#) for an introduction to the topic.

A model in MLflow is also an artifact, but with a specific structure that serves as a contract between the person that created the model and the person that intends to use it. Such contract helps build the bridge about the artifacts themselves and what they mean.

Logging models has the following advantages:

- You don't need to provide a scoring script nor an environment for deployment.
- Swagger is enabled in endpoints automatically and the **Test** feature can be used in Azure ML studio.
- Models can be used as pipelines inputs directly.
- You can use the Responsible AI dashboard.

There are different ways to start using the model's concept in Azure Machine Learning with MLflow, as explained in the following sections:

Logging models using autolog

One of the simplest ways to start using this approach is by using MLflow autolog functionality. Autolog allows MLflow to instruct the framework associated to with the framework you are using to log all the metrics, parameters, artifacts and models that the framework considers relevant. By default, most models will be log if autolog is enabled. Some flavors may decide not to do that in specific situations. For instance, the flavor PySpark won't log models if they exceed a certain size.

You can turn on autologging by using either `mlflow.autolog()` or `mlflow.<flavor>.autolog()`. The following example uses `autolog()` for logging a classifier model trained with XGBoost:

```
import mlflow
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

mlflow.autolog()

model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

TIP

If you are using Machine Learning pipelines, like for instance [Scikit-Learn pipelines](#), use the `autolog` functionality of that flavor for logging models. Models are automatically logged when the `fit()` method is called on the pipeline object. The notebook [Training and tracking an XGBoost classifier with MLflow](#) demonstrates how to log a model with preprocessing using pipelines.

Logging models with a custom signature, environment or samples

You can log models manually using the method `mlflow.<flavor>.log_model` in MLflow. Such workflow has the advantages of retaining control of different aspects of how the model is logged.

Use this method when:

- You want to indicate pip packages or a conda environment different from the ones that are automatically detected.
- You want to include input examples.
- You want to include specific artifacts into the package that will be needed.
- Your signature is not correctly inferred by `autolog`. This is specifically important when you deal with inputs that are tensors where the signature needs specific shapes.
- Somehow the default behavior of `autolog` doesn't fill your purpose.

The following example code logs a model for an XGBoost classifier:

```
import mlflow
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from mlflow.models import infer_signature
from mlflow.utils.environment import _mlflow_conda_env

mlflow.autolog(log_models=False)

model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

# Signature
signature = infer_signature(X_test, y_test)

# Conda environment
custom_env = _mlflow_conda_env(
    additional_conda_deps=None,
    additional_pip_deps=["xgboost==1.5.2"],
    additional_conda_channels=None,
)

# Sample
input_example = X_train.sample(n=1)

# Log the model manually
mlflow.xgboost.log_model(model,
                        artifact_path="classifier",
                        conda_env=custom_env,
                        signature=signature,
                        input_example=input_example)
```

NOTE

- `log_models=False` is configured in `autolog`. This prevents MLflow to automatically log the model, as it is done manually later.
- `infer_signature` is a convenient method to try to infer the signature directly from inputs and outputs.
- `mlflow.utils.environment._mlflow_conda_env` is a private method in MLflow SDK and it may change in the future. This example uses it just for sake of simplicity, but it must be used with caution or generate the YAML definition manually as a Python dictionary.

Logging models with a different behavior in the predict method

When you log a model using either `mlflow.autolog` or using `mlflow.<flavor>.log_model`, the flavor used for the model decides how inference should be executed and what gets returned by the model. MLflow doesn't enforce any specific behavior in how the `predict` generate results. There are scenarios where you probably want to do some pre-processing or post-processing before and after your model is executed.

A solution to this scenario is to implement machine learning pipelines that moves from inputs to outputs directly. Although this is possible (and sometimes encourageable for performance considerations), it may be challenging to achieve. For those cases, you probably want to [customize how your model does inference using a custom models](#) as explained in the following section.

Logging custom models

MLflow provides support for a variety of [machine learning frameworks](#) including FastAI, MXNet Gluon, PyTorch, TensorFlow, XGBoost, CatBoost, h2o, Keras, LightGBM, MLeap, ONNX, Prophet, spaCy, Spark MLLib, Scikit-Learn, and statsmodels. However, they may be times where you need to change how a flavor works, log a model not natively supported by MLflow or even log a model that uses multiple elements from different frameworks. For those cases, you may need to create a custom model flavor.

For this type of models, MLflow introduces a flavor called `pyfunc` (standing from Python function). Basically this flavor allows you to log any object you want as a model, as long as it satisfies two conditions:

- You implement the method `predict` (at least).
- The Python object inherits from `mlflow.pyfunc.PythonModel`.

TIP

Serializable models that implements the Scikit-learn API can use the Scikit-learn flavor to log the model, regardless of whether the model was built with Scikit-learn. If your model can be persisted in Pickle format and the object has methods `predict()` and `predict_proba()` (at least), then you can use `mlflow.sklearn.log_model()` to log it inside a MLflow run.

- [Using a model wrapper](#)
- [Using artifacts](#)
- [Using a model loader](#)

The simplest way of creating your custom model's flavor is by creating a wrapper around your existing model object. MLflow will serialize it and package it for you. Python objects are serializable when the object can be stored in the file system as a file (generally in Pickle format). During runtime, the object can be materialized from such file and all the values, properties and methods available when it was saved will be restored.

Use this method when:

- Your model can be serialized in Pickle format.
- You want to retain the models state as it was just after training.
- You want to customize the way the `predict` function works.

The following sample wraps a model created with XGBoost to make it behaves in a different way to the default implementation of the XGBoost flavor (it returns the probabilities instead of the classes):

```
from mlflow.pyfunc import PythonModel, PythonModelContext

class ModelWrapper(PythonModel):
    def __init__(self, model):
        self._model = model

    def predict(self, context: PythonModelContext, data):
        # You don't have to keep the semantic meaning of `predict`. You can use here model.recommend(),
        model.forecast(), etc
        return self._model.predict_proba(data)

    # You can even add extra functions if you need to. Since the model is serialized,
    # all of them will be available when you load your model back.
    def predict_batch(self, data):
        pass
```

Then, a custom model can be logged in the run like this:

```
import mlflow
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from mlflow.models import infer_signature

mlflow.xgboost.autolog(log_models=False)

model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)
y_probs = model.predict_proba(X_test)

accuracy = accuracy_score(y_test, y_probs.argmax(axis=1))
mlflow.log_metric("accuracy", accuracy)

signature = infer_signature(X_test, y_probs)
mlflow.pyfunc.log_model("classifier",
                       python_model=ModelWrapper(model),
                       signature=signature)
```

TIP

Note how the `infer_signature` method now uses `y_probs` to infer the signature. Our target column has the target class, but our model now returns the two probabilities for each class.

Next steps

- [Deploy MLflow models](#)

Query & compare experiments and runs with MLflow

9/22/2022 • 8 minutes to read • [Edit Online](#)

Experiments and runs in Azure Machine Learning can be queried using MLflow. This removes the need of any Azure Machine Learning specific SDKs to manage anything that happens inside of a training job, allowing dependencies removal and creating a more seamless transition between local runs and cloud.

NOTE

The Azure Machine Learning Python SDK v2 (preview) does not provide native logging or tracking capabilities. This applies not just for logging but also for querying the metrics logged. Instead, we recommend to use MLflow to manage experiments and runs. This article explains how to use MLflow to manage experiments and runs in Azure ML.

MLflow allows you to:

- Create, delete and search for experiments in a workspace.
- Start, stop, cancel and query runs for experiments.
- Track and retrieve metrics, parameters, artifacts and models from runs.

In this article, you'll learn how to manage experiments and runs in your workspace using Azure ML and MLflow SDK in Python.

Using MLflow SDK in Azure ML

Use MLflow to query and manage all the experiments in Azure Machine Learning. The MLflow SDK has capabilities to query everything that happens inside of a training job in Azure Machine Learning. See [Support matrix for querying runs and experiments in Azure Machine Learning](#) for a detailed comparison between MLflow Open-Source and MLflow when connected to Azure Machine Learning.

Prerequisites

- Install `azureml-mlflow` plug-in.
- If you're running in a compute not hosted in Azure ML, configure MLflow to point to the Azure ML tracking URL. You can follow the instruction at [Track runs from your local machine](#).

Getting all the experiments

You can get all the active experiments in the workspace using MLFlow:

```
experiments = mlflow.list_experiments()
for exp in experiments:
    print(exp.name)
```

If you want to retrieve archived experiments too, then include the option `ViewType.ALL` in the `view_type` argument. The following sample shows how:

```
from mlflow.entities import ViewType

experiments = mlflow.list_experiments(view_type=ViewType.ALL)
for exp in experiments:
    print(exp.name)
```

Getting a specific experiment

Details about a specific experiment can be retrieved using the `get_experiment_by_name` method:

```
exp = mlflow.get_experiment_by_name(experiment_name)
print(exp)
```

Getting runs inside an experiment

MLflow allows searching runs inside of any experiment, including multiple experiments at the same time. By default, MLflow returns the data in Pandas `Dataframe` format, which makes it handy when doing further processing our analysis of the runs. Returned data includes columns with:

- Basic information about the run.
- Parameters with column's name `params.<parameter-name>`.
- Metrics (last logged value of each) with column's name `metrics.<metric-name>`.

Getting all the runs from an experiment

By experiment name:

```
mlflow.search_runs(experiment_names=[ "my_experiment" ])
```

By experiment ID:

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ])
```

TIP

Notice that `experiment_ids` supports providing an array of experiments, so you can search runs across multiple experiments if required. This may be useful in case you want to compare runs of the same model when it is being logged in different experiments (by different people, different project iterations, etc). You can also use `search_all_experiments=True` if you want to search across all the experiments in the workspace.

Another important point to notice is that get returning runs, all metrics are parameters are also returned for them. However, for metrics containing multiple values (for instance, a loss curve, or a PR curve), only the last value of the metric is returned. If you want to retrieve all the values of a given metric, uses `mlflow.get_metric_history` method.

Ordering runs

By default, experiments are ordered descending by `start_time`, which is the time the experiment was queue in Azure ML. However, you can change this default by using the parameter `order_by`.

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ], order_by=["start_time DESC"])
```

Use the argument `max_results` from `search_runs` to limit the number of runs returned. For instance, the following example returns the last run of the experiment:

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ], max_results=1, order_by=["start_time DESC"])
```

WARNING

Using `order_by` with expressions containing `metrics.*` in the parameter `order_by` is not supported by the moment. Please use `order_values` method from Pandas as shown in the next example.

You can also order by metrics to know which run generated the best results:

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ]).sort_values("metrics.accuracy", ascending=False)
```

Filtering runs

You can also look for a run with a specific combination in the hyperparameters using the parameter `filter_string`. Use `params` to access run's parameters and `metrics` to access metrics logged in the run. MLflow supports expressions joined by the AND keyword (the syntax does not support OR):

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ], filter_string="params.num_boost_round='100'")
```

Filter runs by status

You can also filter experiment by status. It becomes useful to find runs that are running, completed, canceled or failed. In MLflow, `status` is an `attribute`, so we can access this value using the expression `attributes.status`. The following table shows the possible values:

AZURE ML JOB STATUS	MLFLOW'S ATTRIBUTES.STATUS	MEANING
Not started	SCHEDULED	The job/run was just registered in Azure ML but it has processed it yet.
Queue	SCHEDULED	The job/run is scheduled for running, but it hasn't started yet.
Preparing	SCHEDULED	The job/run has not started yet, but a compute has been allocated for the execution and it is on building state.
Running	RUNNING	The job/run is currently under active execution.
Completed	FINISHED	The job/run has completed without errors.
Failed	FAILED	The job/run has completed with errors.
Canceled	KILLED	The job/run has been canceled or killed by the user/system.

WARNING

Expressions containing `attributes.status` in the parameter `filter_string` are not supported at the moment. Please use Pandas filtering expressions as shown in the next example.

The following example shows all the runs that have been completed:

```
runs = mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ])
runs[runs.status == "FINISHED"]
```

Getting metrics, parameters, artifacts and models

By default, MLflow returns runs as a Pandas `Dataframe` containing a limited amount of information. You can get Python objects if needed, which may be useful to get details about them. Use the `output_format` parameter to control how output is returned:

```
runs = mlflow.search_runs(
    experiment_ids=[ "1234-5678-90AB-CDEFG" ],
    filter_string="params.num_boost_round='100'",
    output_format="list",
)
```

Details can then be accessed from the `info` member. The following sample shows how to get the `run_id`:

```
last_run = runs[-1]
print("Last run ID:", last_run.info.run_id)
```

Getting params and metrics from a run

When runs are returned using `output_format="list"`, you can easily access parameters using the key `data`:

```
last_run.data.params
```

In the same way, you can query metrics:

```
last_run.data.metrics
```

For metrics that contain multiple values (for instance, a loss curve, or a PR curve), only the last logged value of the metric is returned. If you want to retrieve all the values of a given metric, use the `mlflow.get_metric_history` method. This method requires you to use the `MlflowClient`:

```
client = mlflow.tracking.MlflowClient()
client.get_metric_history("1234-5678-90AB-CDEFG", "log_loss")
```

Getting artifacts from a run

Any artifact logged by a run can be queried by MLflow. Artifacts can't be accessed using the run object itself and the MLflow client should be used instead:

```
client = mlflow.tracking.MlflowClient()
client.list_artifacts("1234-5678-90AB-CDEFG")
```

The method above will list all the artifacts logged in the run, but they will remain stored in the artifacts store (Azure ML storage). To download any of them, use the method `download_artifact`:

```
file_path = client.download_artifacts("1234-5678-90AB-CDEFG", path="feature_importance_weight.png")
```

Getting models from a run

Models can also be logged in the run and then retrieved directly from it. To retrieve it, you need to know the artifact's path where it is stored. The method `list_artifacts` can be used to find artifacts that are representing a model since MLflow models are always folders. You can download a model by indicating the path where the model is stored using the `download_artifact` method:

```
artifact_path="classifier"
model_local_path = client.download_artifacts("1234-5678-90AB-CDEFG", path=artifact_path)
```

You can then load the model back from the downloaded artifacts using the typical function `load_model`:

```
model = mlflow.xgboost.load_model(model_local_path)
```

NOTE

In the example above, we are assuming the model was created using `xgboost`. Change it to the flavor applies to your case.

MLflow also allows you to both operations at once and download and load the model in a single instruction. MLflow will download the model to a temporary folder and load it from there. This can be done using the `load_model` method which uses an URI format to indicate from where the model has to be retrieved. In the case of loading a model from a run, the URI structure is as follows:

```
model = mlflow.xgboost.load_model(f"runs:{last_run.info.run_id}/{artifact_path}")
```

TIP

You can also load models from the registry using MLflow. View [loading MLflow models with MLflow](#) for details.

Getting child (nested) runs

MLflow supports the concept of child (nested) runs. They are useful when you need to spin off training routines requiring being tracked independently from the main training process. This is the typical case of hyper-parameter tuning for instance. You can query all the child runs of a specific run using the property tag `mlflow.parentRunId`, which contains the run ID of the parent run.

```
hyperopt_run = mlflow.last_active_run()
child_runs = mlflow.search_runs(
    filter_string=f"tags.mlflow.parentRunId='{hyperopt_run.info.run_id}'"
)
```

Example notebooks

The [MLflow with Azure ML notebooks](#) demonstrate and expand upon concepts presented in this article.

- [Training and tracking a classifier with MLflow](#): Demonstrates how to track experiments using MLflow, log models and combine multiple flavors into pipelines.
- [Manage experiments and runs with MLflow](#): Demonstrates how to query experiments, runs, metrics, parameters and artifacts from Azure ML using MLflow.

Support matrix for querying runs and experiments

The MLflow SDK exposes several methods to retrieve runs, including options to control what is returned and how. Use the following table to learn about which of those methods are currently supported in MLflow when connected to Azure Machine Learning:

FEATURE	SUPPORTED BY MLFLOW	SUPPORTED BY AZURE ML
Ordering runs by run fields (like <code>start_time</code> , <code>end_time</code> , etc)	✓	✓
Ordering runs by attributes	✓	1
Ordering runs by metrics	✓	1
Ordering runs by parameters	✓	1
Ordering runs by tags	✓	1
Filtering runs by run fields (like <code>start_time</code> , <code>end_time</code> , etc)		1
Filtering runs by attributes	✓	1
Filtering runs by metrics	✓	✓
Filtering runs by metrics with special characters (escaped)	✓	
Filtering runs by parameters	✓	✓
Filtering runs by tags	✓	✓
Filtering runs with numeric comparators (metrics) including <code>=</code> , <code>!=</code> , <code>></code> , <code>>=</code> , <code><</code> , and <code><=</code>	✓	✓
Filtering runs with string comparators (params, tags, and attributes): <code>=</code> and <code>!=</code>	✓	✓ ²
Filtering runs with string comparators (params, tags, and attributes): <code>LIKE</code> / <code>ILIKE</code>	✓	
Filtering runs with comparators <code>AND</code>	✓	✓
Filtering runs with comparators <code>OR</code>		

FEATURE	SUPPORTED BY MLFLOW	SUPPORTED BY AZURE ML
Renaming experiments	✓	

NOTE

- ¹ Check the section [Getting runs inside an experiment](#) for instructions and examples on how to achieve the same functionality in Azure ML.
- ² for tags not supported.

Next steps

- [Manage your models with MLflow.](#)
- [Deploy models with MLflow.](#)

Visualize experiment jobs and metrics with TensorBoard and Azure Machine Learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, you learn how to view your experiment jobs and metrics in TensorBoard using the [tensorboard package](#) in the main Azure Machine Learning SDK. Once you've inspected your experiment jobs, you can better tune and retrain your machine learning models.

TensorBoard is a suite of web applications for inspecting and understanding your experiment structure and performance.

How you launch TensorBoard with Azure Machine Learning experiments depends on the type of experiment:

- If your experiment natively outputs log files that are consumable by TensorBoard, such as PyTorch, Chainer and TensorFlow experiments, then you can [launch TensorBoard directly](#) from experiment's job history.
- For experiments that don't natively output TensorBoard consumable files, such as like Scikit-learn or Azure Machine Learning experiments, use the [export_to_tensorboard\(\)](#) method to export the job histories as TensorBoard logs and launch TensorBoard from there.

TIP

The information in this document is primarily for data scientists and developers who want to monitor the model training process. If you are an administrator interested in monitoring resource usage and events from Azure Machine learning, such as quotas, completed training jobs, or completed model deployments, see [Monitoring Azure Machine Learning](#).

Prerequisites

- To launch TensorBoard and view your experiment job histories, your experiments need to have previously enabled logging to track its metrics and performance.
- The code in this document can be run in either of the following environments:
 - Azure Machine Learning compute instance - no downloads or installation necessary
 - Complete the [Quickstart: Get started with Azure Machine Learning](#) to create a dedicated notebook server pre-loaded with the SDK and the sample repository.
 - In the samples folder on the notebook server, find two completed and expanded notebooks by navigating to these directories:
 - `how-to-use-azureml > track-and-monitor-experiments > tensorboard > export-run-history-to-tensorboard > export-run-history-to-tensorboard.ipynb`
 - `how-to-use-azureml > track-and-monitor-experiments > tensorboard > tensorboard > tensorboard.ipynb`
 - Your own Jupyter notebook server
 - [Install the Azure Machine Learning SDK](#) with the `tensorboard` extra
 - [Create an Azure Machine Learning workspace](#).
 - [Create a workspace configuration file](#).

Option 1: Directly view job history in TensorBoard

This option works for experiments that natively outputs log files consumable by TensorBoard, such as PyTorch, Chainer, and TensorFlow experiments. If that is not the case of your experiment, use [the `export_to_tensorboard\(\)` method](#) instead.

The following example code uses the [MNIST demo experiment](#) from TensorFlow's repository in a remote compute target, Azure Machine Learning Compute. Next, we will configure and start a job for training the TensorFlow model, and then start TensorBoard against this TensorFlow experiment.

Set experiment name and create project folder

Here we name the experiment and create its folder.

```
from os import path, makedirs
experiment_name = 'tensorboard-demo'

# experiment folder
exp_dir = './sample_projects/' + experiment_name

if not path.exists(exp_dir):
    makedirs(exp_dir)
```

Download TensorFlow demo experiment code

TensorFlow's repository has an MNIST demo with extensive TensorBoard instrumentation. We do not, nor need to, alter any of this demo's code for it to work with Azure Machine Learning. In the following code, we download the MNIST code and save it in our newly created experiment folder.

```
import requests
import os

tf_code =
requests.get("https://raw.githubusercontent.com/tensorflow/tensorflow/r1.8/tensorflow/examples/tutorials/mnist/mnist_with_summaries.py")
with open(os.path.join(exp_dir, "mnist_with_summaries.py"), "w") as file:
    file.write(tf_code.text)
```

Throughout the MNIST code file, `mnist_with_summaries.py`, notice that there are lines that call

`tf.summary.scalar()`, `tf.summary.histogram()`, `tf.summary.FileWriter()` etc. These methods group, log, and tag key metrics of your experiments into job history. The `tf.summary.FileWriter()` is especially important as it serializes the data from your logged experiment metrics, which allows for TensorBoard to generate visualizations off of them.

Configure experiment

In the following, we configure our experiment and set up directories for logs and data. These logs will be uploaded to the job history, which TensorBoard accesses later.

NOTE

For this TensorFlow example, you will need to install TensorFlow on your local machine. Further, the TensorBoard module (that is, the one included with TensorFlow) must be accessible to this notebook's kernel, as the local machine is what runs TensorBoard.

```

import azureml.core
from azureml.core import Workspace
from azureml.core import Experiment

ws = Workspace.from_config()

# create directories for experiment logs and dataset
logs_dir = os.path.join(os.curdir, "logs")
data_dir = os.path.abspath(os.path.join(os.curdir, "mnist_data"))

if not path.exists(data_dir):
    makedirs(data_dir)

os.environ["TEST_TMPDIR"] = data_dir

# Writing logs to ./logs results in their being uploaded to the job history,
# and thus, made accessible to our TensorBoard instance.
args = ["--log_dir", logs_dir]

# Create an experiment
exp = Experiment(ws, experiment_name)

```

Create a cluster for your experiment

We create an AmlCompute cluster for this experiment, however your experiments can be created in any environment and you are still able to launch TensorBoard against the experiment job history.

```

from azureml.core.compute import ComputeTarget, AmlCompute

cluster_name = "cpu-cluster"

cts = ws.compute_targets
found = False
if cluster_name in cts and cts[cluster_name].type == 'AmlCompute':
    found = True
    print('Found existing compute target.')
    compute_target = cts[cluster_name]
if not found:
    print('Creating a new compute target...')
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                           max_nodes=4)

    # create the cluster
    compute_target = ComputeTarget.create(ws, cluster_name, compute_config)

compute_target.wait_for_completion(show_output=True, min_node_count=None)

# use get_status() to get a detailed status for the current cluster.
# print(compute_target.get_status().serialize())

```

NOTE

You may choose to use [low-priority VMs](#) to run some or all of your workloads. See how to [create a low-priority VM](#).

Configure and submit training job

Configure a training job by creating a ScriptRunConfig object.

```

from azureml.core import ScriptRunConfig
from azureml.core import Environment

# Here we will use the TensorFlow 2.2 curated environment
tf_env = Environment.get(ws, 'AzureML-TensorFlow-2.2-GPU')

src = ScriptRunConfig(source_directory=exp_dir,
                      script='mnist_with_summaries.py',
                      arguments=args,
                      compute_target=compute_target,
                      environment=tf_env)
run = exp.submit(src)

```

Launch TensorBoard

You can launch TensorBoard during your run or after it completes. In the following, we create a TensorBoard object instance, `tb`, that takes the experiment job history loaded in the `job`, and then launches TensorBoard with the `start()` method.

The [TensorBoard constructor](#) takes an array of jobs, so be sure and pass it in as a single-element array.

```

from azureml.tensorboard import Tensorboard

tb = Tensorboard([job])

# If successful, start() returns a string with the URI of the instance.
tb.start()

# After your job completes, be sure to stop() the streaming otherwise it will continue to run.
tb.stop()

```

NOTE

While this example used TensorFlow, TensorBoard can be used as easily with PyTorch or Chainer. TensorFlow must be available on the machine running TensorBoard, but is not necessary on the machine doing PyTorch or Chainer computations.

Option 2: Export history as log to view in TensorBoard

The following code sets up a sample experiment, begins the logging process using the Azure Machine Learning job history APIs, and exports the experiment job history into logs consumable by TensorBoard for visualization.

Set up experiment

The following code sets up a new experiment and names the job directory `root_run`.

```

from azureml.core import Workspace, Experiment
import azureml.core

# set experiment name and job name
ws = Workspace.from_config()
experiment_name = 'export-to-tensorboard'
exp = Experiment(ws, experiment_name)
root_run = exp.start_logging()

```

Here we load the diabetes dataset-- a built-in small dataset that comes with scikit-learn, and split it into test and training sets.

```

from sklearn.datasets import load_diabetes
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
X, y = load_diabetes(return_X_y=True)
columns = ['age', 'gender', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
data = {
    "train": {"x": x_train, "y": y_train},
    "test": {"x": x_test, "y": y_test}
}

```

Run experiment and log metrics

For this code, we train a linear regression model and log key metrics, the alpha coefficient, `alpha`, and mean squared error, `mse`, in run history.

```

from tqdm import tqdm
alphas = [.1, .2, .3, .4, .5, .6, .7]
# try a bunch of alpha values in a Linear Regression (aka Ridge regression) mode
for alpha in tqdm(alphas):
    # create child runs and fit lines for the resulting models
    with root_run.child_run("alpha" + str(alpha)) as run:

        reg = Ridge(alpha=alpha)
        reg.fit(data["train"]["x"], data["train"]["y"])

        preds = reg.predict(data["test"]["x"])
        mse = mean_squared_error(preds, data["test"]["y"])
        # End train and eval

    # log alpha, mean_squared_error and feature names in run history
    root_run.log("alpha", alpha)
    root_run.log("mse", mse)

```

Export jobs to TensorBoard

With the SDK's `export_to_tensorboard()` method, we can export the job history of our Azure machine learning experiment into TensorBoard logs, so we can view them via TensorBoard.

In the following code, we create the folder `logdir` in our current working directory. This folder is where we will export our experiment job history and logs from `root_run` and then mark that job as completed.

```

from azureml.tensorboard.export import export_to_tensorboard
import os

logdir = 'exportedTBlogs'
log_path = os.path.join(os.getcwd(), logdir)
try:
    os.stat(log_path)
except os.error:
    os.mkdir(log_path)
print(logdir)

# export job history for the project
export_to_tensorboard(root_run, logdir)

root_run.complete()

```

NOTE

You can also export a particular run to TensorBoard by specifying the name of the run

```
export_to_tensorboard(run_name, logdir)
```

Start and stop TensorBoard

Once our job history for this experiment is exported, we can launch TensorBoard with the [start\(\)](#) method.

```
from azureml.tensorboard import Tensorboard

# The TensorBoard constructor takes an array of jobs, so be sure and pass it in as a single-element array
# here
tb = Tensorboard([], local_root=logdir, port=6006)

# If successful, start() returns a string with the URI of the instance.
tb.start()
```

When you're done, make sure to call the [stop\(\)](#) method of the TensorBoard object. Otherwise, TensorBoard will continue to run until you shut down the notebook kernel.

```
tb.stop()
```

Next steps

In this how-to you, created two experiments and learned how to launch TensorBoard against their job histories to identify areas for potential tuning and retraining.

- If you are satisfied with your model, head over to our [How to deploy a model](#) article.
- Learn more about [hyperparameter tuning](#).

Migrate logging from SDK v1 to SDK v2 (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

The Azure Machine Learning Python SDK v2 does not provide native logging APIs. Instead, we recommend that you use [MLflow Tracking](#). If you're migrating from SDK v1 to SDK v2 (preview), use the information in this section to understand the MLflow equivalents of SDK v1 logging APIs.

Setup

To use MLflow tracking, import `mlflow` and optionally set the tracking URI for your workspace. If you're training on an Azure Machine Learning compute resource, such as a compute instance or compute cluster, the tracking URI is set automatically. If you're using a different compute resource, such as your laptop or desktop, you need to set the tracking URI.

```
import mlflow

# The rest of this is only needed if you are not using an Azure ML compute
## Construct AzureML MLFLOW TRACKING URI
def get_azureml_mlflow_tracking_uri(region, subscription_id, resource_group, workspace):
    return
    "azurerm://{}.api.azureml.ms/mlflow/v1.0/subscriptions/{}/resourceGroups/{}/providers/Microsoft.MachineLearn
    ingServices/workspaces{}".format(region, subscription_id, resource_group, workspace)

region='<REGION>' ## example: westus
subscription_id = '<SUBSCRIPTION_ID>' ## example: 11111111-1111-1111-1111-111111111111
resource_group = '<RESOURCE_GROUP>' ## example: myresourcegroup
workspace = '<AML_WORKSPACE_NAME>' ## example: myworkspacename

MLFLOW_TRACKING_URI = get_azureml_mlflow_tracking_uri(region, subscription_id, resource_group, workspace)

## Set the MLFLOW TRACKING URI
mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)
```

Experiments and runs

SDK v1

```
from azureml.core import Experiment

# create an AzureML experiment and start a run
experiment = Experiment(ws, "create-experiment-sdk-v1")
azureml_run = experiment.start_logging()
```

SDK v2 (preview) with MLflow

```
# Set the MLflow experiment and start a run
mlflow.set_experiment("logging-with-mlflow")
mlflow_run = mlflow.start_run()
```

Logging API comparison

Log an integer or float metric

SDK v1

```
azureml_run.log("sample_int_metric", 1)
```

SDK v2 (preview) with MLflow

```
mlflow.log_metric("sample_int_metric", 1)
```

Log a boolean metric

SDK v1

```
azureml_run.log("sample_boolean_metric", True)
```

SDK v2 (preview) with MLflow

```
mlflow.log_metric("sample_boolean_metric", 1)
```

Log a string metric

SDK v1

```
azureml_run.log("sample_string_metric", "a_metric")
```

SDK v2 (preview) with MLflow

```
mlflow.log_text("sample_string_text", "string.txt")
```

- The string will be logged as an *artifact*, not as a metric. In Azure Machine Learning studio, the value will be displayed in the **Outputs + logs** tab.

Log an image to a PNG or JPEG file

SDK v1

```
azureml_run.log_image("sample_image", path="Azure.png")
```

SDK v2 (preview) with MLflow

```
mlflow.log_artifact("Azure.png")
```

The image is logged as an artifact and will appear in the **Images** tab in Azure Machine Learning Studio.

Log a matplotlib.pyplot

SDK v1

```
import matplotlib.pyplot as plt  
  
plt.plot([1, 2, 3])  
azureml_run.log_image("sample_pyplot", plot=plt)
```

SDK v2 (preview) with MLflow

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3])
fig, ax = plt.subplots()
ax.plot([0, 1], [2, 3])
mlflow.log_figure(fig, "sample_pyplot.png")
```

- The image is logged as an artifact and will appear in the **Images** tab in Azure Machine Learning Studio.
- The `mlflow.log_figure` method is **experimental**.

Log a list of metrics

SDK v1

```
list_to_log = [1, 2, 3, 2, 1, 2, 3, 2, 1]
azureml_run.log_list('sample_list', list_to_log)
```

SDK v2 (preview) with MLflow

```
list_to_log = [1, 2, 3, 2, 1, 2, 3, 2, 1]
from mlflow.entities import Metric
from mlflow.tracking import MlflowClient
import time

metrics = [Metric(key="sample_list", value=val, timestamp=int(time.time() * 1000), step=0) for val in
list_to_log]
MlflowClient().log_batch(mlflow_run.info.run_id, metrics=metrics)
```

- Metrics appear in the **metrics** tab in Azure Machine Learning studio.
- Text values are not supported.

Log a row of metrics

SDK v1

```
azureml_run.log_row("sample_table", col1=5, col2=10)
```

SDK v2 (preview) with MLflow

```
metrics = {"sample_table.col1": 5, "sample_table.col2": 10}
mlflow.log_metrics(metrics)
```

- Metrics do not render as a table in Azure Machine Learning studio.
- Text values are not supported.
- Logged as an *artifact*, not as a metric.

Log a table

SDK v1

```
table = {
    "col1" : [1, 2, 3],
    "col2" : [4, 5, 6]
}
azureml_run.log_table("table", table)
```

SDK v2 (preview) with MLflow

```

# Add a metric for each column prefixed by metric name. Similar to log_row
row1 = {"table.col1": 5, "table.col2": 10}
# To be done for each row in the table
mlflow.log_metrics(row1)

# Using mlflow.log_artifact
import json

with open("table.json", 'w') as f:
    json.dump(table, f)
    mlflow.log_artifact("table.json")

```

- Logs metrics for each column.
- Metrics do not render as a table in Azure Machine Learning studio.
- Text values are not supported.
- Logged as an *artifact*, not as a metric.

Log an accuracy table

SDK v1

```

ACCURACY_TABLE = '{"schema_type": "accuracy_table", "schema_version": "v1", "data": {"probability_tables": ' + \
    '[[[114311, 385689, 0, 0], [0, 0, 385689, 114311]], [[67998, 432002, 0, 0], [0, 0, ' + \
    '432002, 67998]]], "percentile_tables": [[[114311, 385689, 0, 0], [1, 0, 385689, ' + \
    '114310]], [[67998, 432002, 0, 0], [1, 0, 432002, 67997]]]], "class_labels": ["0", "1"], ' + \
    '"probability_thresholds": [0.52], "percentile_thresholds": [0.09]}]'}

azureml_run.log_accuracy_table('v1_accuracy_table', ACCURACY_TABLE)

```

SDK v2 (preview) with MLflow

```

ACCURACY_TABLE = '{"schema_type": "accuracy_table", "schema_version": "v1", "data": {"probability_tables": ' + \
    '[[[114311, 385689, 0, 0], [0, 0, 385689, 114311]], [[67998, 432002, 0, 0], [0, 0, ' + \
    '432002, 67998]]], "percentile_tables": [[[114311, 385689, 0, 0], [1, 0, 385689, ' + \
    '114310]], [[67998, 432002, 0, 0], [1, 0, 432002, 67997]]]], "class_labels": ["0", "1"], ' + \
    '"probability_thresholds": [0.52], "percentile_thresholds": [0.09]}]'

mlflow.log_dict(ACCURACY_TABLE, 'mlflow_accuracy_table.json')

```

- Metrics do not render as an accuracy table in Azure Machine Learning studio.
- Logged as an *artifact*, not as a metric.
- The `mlflow.log_dict` method is *experimental*.

Log a confusion matrix

SDK v1

```

CONF_MATRIX = '{"schema_type": "confusion_matrix", "schema_version": "v1", "data": {"class_labels": ' + \
    '["0", "1", "2", "3"], "matrix": [[3, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1]]]}'

azureml_run.log_confusion_matrix('v1_confusion_matrix', json.loads(CONF_MATRIX))

```

SDK v2 (preview) with MLflow

```

CONF_MATRIX = '{"schema_type": "confusion_matrix", "schema_version": "v1", "data": {"class_labels": ' + \
    '[0", "1", "2", "3"], "matrix": [[3, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1]]}}'

mlflow.log_dict(CONF_MATRIX, 'mlflow_confusion_matrix.json')

```

- Metrics do not render as a confusion matrix in Azure Machine Learning studio.
- Logged as an *artifact*, not as a metric.
- The `mlflow.log_dict` method is *experimental*.

Log predictions

SDK v1

```

PREDICTIONS = '{"schema_type": "predictions", "schema_version": "v1", "data": {"bin_averages": [0.25, ' + \
    ' 0.75], "bin_errors": [0.013, 0.042], "bin_counts": [56, 34], "bin_edges": [0.0, 0.5, 1.0]}}'

azureml_run.log_predictions('test_predictions', json.loads(PREDICTIONS))

```

SDK v2 (preview) with MLflow

```

PREDICTIONS = '{"schema_type": "predictions", "schema_version": "v1", "data": {"bin_averages": [0.25, ' + \
    ' 0.75], "bin_errors": [0.013, 0.042], "bin_counts": [56, 34], "bin_edges": [0.0, 0.5, 1.0]}}'

mlflow.log_dict(PREDICTIONS, 'mlflow_predictions.json')

```

- Metrics do not render as a confusion matrix in Azure Machine Learning studio.
- Logged as an *artifact*, not as a metric.
- The `mlflow.log_dict` method is *experimental*.

Log residuals

SDK v1

```

RESIDUALS = '{"schema_type": "residuals", "schema_version": "v1", "data": {"bin_edges": [100, 200, 300], ' + \
    '\
    '"bin_counts": [0.88, 20, 30, 50.99]}}'

azureml_run.log_residuals('test_residuals', json.loads(RESIDUALS))

```

SDK v2 (preview) with MLflow

```

RESIDUALS = '{"schema_type": "residuals", "schema_version": "v1", "data": {"bin_edges": [100, 200, 300], ' + \
    '\
    '"bin_counts": [0.88, 20, 30, 50.99]}}'

mlflow.log_dict(RESIDUALS, 'mlflow_residuals.json')

```

- Metrics do not render as a confusion matrix in Azure Machine Learning studio.
- Logged as an *artifact*, not as a metric.
- The `mlflow.log_dict` method is *experimental*.

View run info and data

You can access run information using the MLflow run object's `data` and `info` properties. For more information, see [mlflow.entities.Run](#) reference.

The following example shows how to retrieve a finished run:

```
from mlflow.tracking import MlflowClient

# Use MLFlow to retrieve the run that was just completed
client = MlflowClient()
finished_mlflow_run = MlflowClient().get_run(mlflow_run.info.run_id)
```

The following example shows how to view the `metrics`, `tags`, and `params`:

```
metrics = finished_mlflow_run.data.metrics
tags = finished_mlflow_run.data.tags
params = finished_mlflow_run.data.params
```

NOTE

The `metrics` will only have the most recently logged value for a given metric. For example, if you log in order a value of `1`, then `2`, `3`, and finally `4` to a metric named `sample_metric`, only `4` will be present in the `metrics` dictionary. To get all metrics logged for a specific named metric, use [MlflowClient.get_metric_history](#):

```
with mlflow.start_run() as multiple_metrics_run:
    mlflow.log_metric("sample_metric", 1)
    mlflow.log_metric("sample_metric", 2)
    mlflow.log_metric("sample_metric", 3)
    mlflow.log_metric("sample_metric", 4)

print(client.get_run(multiple_metrics_run.info.run_id).data.metrics)
print(client.get_metric_history(multiple_metrics_run.info.run_id, "sample_metric"))
```

For more information, see the [MlflowClient](#) reference.

The `info` field provides general information about the run, such as start time, run ID, experiment ID, etc.:

```
run_start_time = finished_mlflow_run.info.start_time
run_experiment_id = finished_mlflow_run.info.experiment_id
run_id = finished_mlflow_run.info.run_id
```

View run artifacts

To view the artifacts of a run, use [MlflowClient.list_artifacts](#):

```
client.list_artifacts(finished_mlflow_run.info.run_id)
```

To download an artifact, use [MlflowClient.download_artifacts](#):

```
client.download_artifacts(finished_mlflow_run.info.run_id, "Azure.png")
```

Next steps

- [Track ML experiments and models with MLflow](#)
- [Log and view metrics](#)

Track ML experiments and models with MLflow

9/22/2022 • 8 minutes to read • [Edit Online](#)

In this article, learn how to enable [MLflow Tracking](#) to connect Azure Machine Learning as the backend of your MLflow experiments.

MLflow is an open-source library for managing the lifecycle of your machine learning experiments. MLflow Tracking is a component of MLflow that logs and tracks your training job metrics and model artifacts, no matter your experiment's environment--locally on your computer, on a remote compute target, a virtual machine, or an [Azure Databricks cluster](#).

See [MLflow and Azure Machine Learning](#) for all supported MLflow and Azure Machine Learning functionality including MLflow Project support (preview) and model deployment.

TIP

If you want to track experiments running on Azure Databricks or Azure Synapse Analytics, see the dedicated articles [Track Azure Databricks ML experiments with MLflow and Azure Machine Learning](#) or [Track Azure Synapse Analytics ML experiments with MLflow and Azure Machine Learning](#).

NOTE

The information in this document is primarily for data scientists and developers who want to monitor the model training process. If you are an administrator interested in monitoring resource usage and events from Azure Machine Learning, such as quotas, completed training jobs, or completed model deployments, see [Monitoring Azure Machine Learning](#).

Prerequisites

- Install the `mlflow` package.
 - You can use the [MLflow Skinny](#) which is a lightweight MLflow package without SQL storage, server, UI, or data science dependencies. This is recommended for users who primarily need the tracking and logging capabilities without importing the full suite of MLflow features including deployments.
- Install the `azureml-mlflow` package.
- [Create an Azure Machine Learning Workspace](#).
 - See which [access permissions](#) you need to perform your [MLflow operations with your workspace](#).
- (Optional) Install and [set up Azure ML CLI \(v2\)](#) and make sure you install the ml extension.
- (Optional) Install and set up Azure ML SDK(v2) for Python.

Track runs from your local machine or remote compute

Tracking using MLflow with Azure Machine Learning lets you store the logged metrics and artifacts runs that were executed on your local machine into your Azure Machine Learning workspace.

Set up tracking environment

To track a run that is not running on Azure Machine Learning compute (from now on referred to as "*local compute*"), you need to point your local compute to the Azure Machine Learning MLflow Tracking URI.

NOTE

When running on Azure Compute (Azure Notebooks, Jupyter Notebooks hosted on Azure Compute Instances or Compute Clusters) you don't have to configure the tracking URI. It's automatically configured for you.

- [Using the Azure ML SDK v2](#)
- [Using an environment variable](#)
- [Building the MLflow tracking URI](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

You can get the Azure ML MLflow tracking URI using the [Azure Machine Learning SDK v2 for Python](#). Ensure you have the library `azure-ai-ml` installed in the cluster you are using. The following sample gets the unique MLFlow tracking URI associated with your workspace. Then the method `set_tracking_uri()` points the MLflow tracking URI to that URI.

1. Using the workspace configuration file:

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

ml_client = MLClient.from_config(credential=DefaultAzureCredential())
azureml_mlflow_uri = ml_client.workspaces.get(ml_client.workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)
```

TIP

You can download the workspace configuration file by:

1. Navigate to [Azure ML studio](#)
2. Click on the upper-right corner of the page -> Download config file.
3. Save the file `config.json` in the same directory where you are working on.

2. Using the subscription ID, resource group name and workspace name:

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

#Enter details of your AzureML workspace
subscription_id = '<SUBSCRIPTION_ID>'
resource_group = '<RESOURCE_GROUP>'
workspace_name = '<AZUREML_WORKSPACE_NAME>'

ml_client = MLClient(credential=DefaultAzureCredential(),
                     subscription_id=subscription_id,
                     resource_group_name=resource_group)

azureml_mlflow_uri = ml_client.workspaces.get(workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)
```

IMPORTANT

`DefaultAzureCredential` will try to pull the credentials from the available context. If you want to specify credentials in a different way, for instance using the web browser in an interactive way, you can use `InteractiveBrowserCredential` or any other method available in `azure.identity` package.

Set experiment name

All MLflow runs are logged to the active experiment. By default, runs are logged to an experiment named `Default` that is automatically created for you. To configure the experiment you want to work on use MLflow command `mlflow.set_experiment()`.

```
experiment_name = 'experiment_with_mlflow'  
mlflow.set_experiment(experiment_name)
```

TIP

When submitting jobs using Azure ML CLI v2, you can set the experiment name using the property `experiment_name` in the YAML definition of the job. You don't have to configure it on your training script. See [YAML: display name, experiment name, description, and tags](#) for details.

You can also set one of the MLflow environment variables `MLFLOW_EXPERIMENT_NAME` or `MLFLOW_EXPERIMENT_ID` with the experiment name.

```
export MLFLOW_EXPERIMENT_NAME="experiment_with_mlflow"
```

Start training job

After you set the MLflow experiment name, you can start your training job with `start_run()`. Then use `log_metric()` to activate the MLflow logging API and begin logging your training job metrics.

```
import os  
from random import random  
  
with mlflow.start_run() as mlflow_run:  
    mlflow.log_param("hello_param", "world")  
    mlflow.log_metric("hello_metric", random())  
    os.system(f"echo 'hello world' > helloworld.txt")  
    mlflow.log_artifact("helloworld.txt")
```

For details about how to log metrics, parameters and artifacts in a run using MLflow view [How to log and view metrics](#).

Track jobs running on Azure Machine Learning

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Remote runs (jobs) let you train your models in a more robust and repetitive way. They can also leverage more powerful computes, such as Machine Learning Compute clusters. See [What are compute targets in Azure Machine Learning?](#) to learn about different compute options.

When submitting runs using jobs, Azure Machine Learning automatically configures MLflow to work with the workspace the job is running in. This means that there is no need to configure the MLflow tracking URI. On top of that, experiments are automatically named based on the details of the job.

IMPORTANT

When submitting training jobs to Azure Machine Learning, you don't have to configure the MLflow tracking URI on your training logic as it is already configured for you.

Creating a training routine

First, you should create a `src` subdirectory and create a file with your training code in a `hello_world.py` file in the `src` subdirectory. All your training code will go into the `src` subdirectory, including `train.py`.

The training code is taken from this [MLflow example](#) in the Azure Machine Learning example repo.

Copy this code into the file:

```
# imports
import os
import mlflow

from random import random

# define functions
def main():
    mlflow.log_param("hello_param", "world")
    mlflow.log_metric("hello_metric", random())
    os.system(f"echo 'hello world' > helloworld.txt")
    mlflow.log_artifact("helloworld.txt")

# run functions
if __name__ == "__main__":
    # run main function
    main()
```

NOTE

Note how this sample don't contains the instructions `mlflow.start_run` nor `mlflow.set_experiment`. This is automatically done by Azure Machine Learning.

Submitting the job

Use the [Azure Machine Learning CLI \(v2\)](#) to submit a remote run. When using the Azure Machine Learning CLI (v2), the MLflow tracking URI and experiment name are set automatically and directs the logging from MLflow to your workspace. Learn more about [logging Azure Machine Learning CLI \(v2\) experiments with MLflow](#)

Create a YAML file with your job definition in a `job.yml` file. This file should be created outside the `src` directory. Copy this code into the file:

```
$schema: https://azureschemas.azureedge.net/latest/commandJob.schema.json
command: python hello-mlflow.py
code: src
environment: azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azureml:cpu-cluster
```

Open your terminal and use the following to submit the job.

```
az ml job create -f job.yml --web
```

View metrics and artifacts in your workspace

The metrics and artifacts from MLflow logging are tracked in your workspace. To view them anytime, navigate to your workspace and find the experiment by name in your workspace in [Azure Machine Learning studio](#). Or run the below code.

Retrieve run metric using MLflow [get_run\(\)](#).

```
from mlflow.tracking import MlflowClient

# Use MLFlow to retrieve the job that was just completed
client = MlflowClient()
run_id = mlflow_run.info.run_id
finished_mlflow_run = MlflowClient().get_run(run_id)

metrics = finished_mlflow_run.data.metrics
tags = finished_mlflow_run.data.tags
params = finished_mlflow_run.data.params

print(metrics,tags,params)
```

To view the artifacts of a run, you can use [MlflowClient.list_artifacts\(\)](#)

```
client.list_artifacts(run_id)
```

To download an artifact to the current directory, you can use [MLflowClient.download_artifacts\(\)](#)

```
client.download_artifacts(run_id, "helloworld.txt", ".")
```

For more details about how to retrieve information from experiments and runs in Azure Machine Learning using MLflow view [Manage experiments and runs with MLflow](#).

Manage models

Register and track your models with the [Azure Machine Learning model registry](#), which supports the MLflow model registry. Azure Machine Learning models are aligned with the MLflow model schema making it easy to export and import these models across different workflows. The MLflow-related metadata, such as run ID, is also tracked with the registered model for traceability. Users can submit training jobs, register, and deploy models produced from MLflow runs.

If you want to deploy and register your production ready model in one step, see [Deploy and register MLflow models](#).

To register and view a model from a job, use the following steps:

- Once a job is complete, call the [register_model\(\)](#) method.

```
# the model folder produced from a job is registered. This includes the MLmodel file, model.pkl and
# the conda.yaml.
model_path = "model"
model_uri = 'runs:/{}{}'.format(run_id, model_path)
mlflow.register_model(model_uri,"registered_model_name")
```

- View the registered model in your workspace with [Azure Machine Learning studio](#).

In the following example the registered model, [my-model](#) has MLflow tracking metadata tagged.

Microsoft Azure Machine Learning Studio

Microsoft > my-ws > Models > my-model:1

my-model:1

Details Versions Artifacts Endpoints Data Explanations (preview) Fairness (preview) Responsible AI (preview)

Refresh Deploy Download all Create Responsible AI dashboard (preview)

Responsible AI dashboard currently only supports MLflow format models with scikit-learn flavor. Learn more about model conversion to MLflow.

Attributes

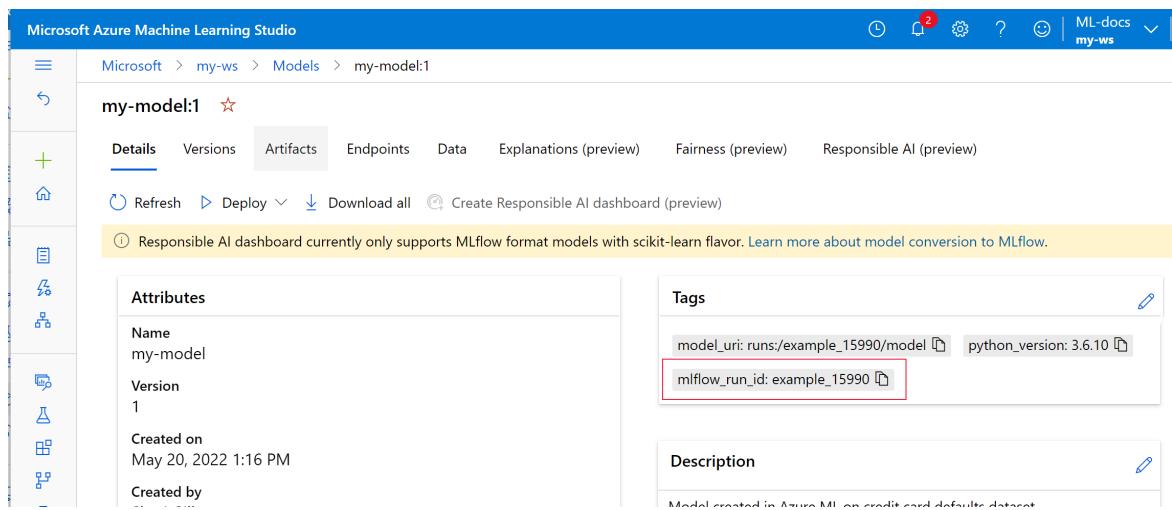
Name: my-model
Version: 1
Created on: May 20, 2022 1:16 PM
Created by: ...

Tags

model_uri: runs/example_15990/model
python_version: 3.6.10
mlflow_run_id: example_15990

Description

Model created in Azure ML on credit card default dataset



3. Select the **Artifacts** tab to see all the model files that align with the MLflow model schema (conda.yaml, MLmodel, model.pkl).

e2edemos > Models > my_model:1

my_model:1

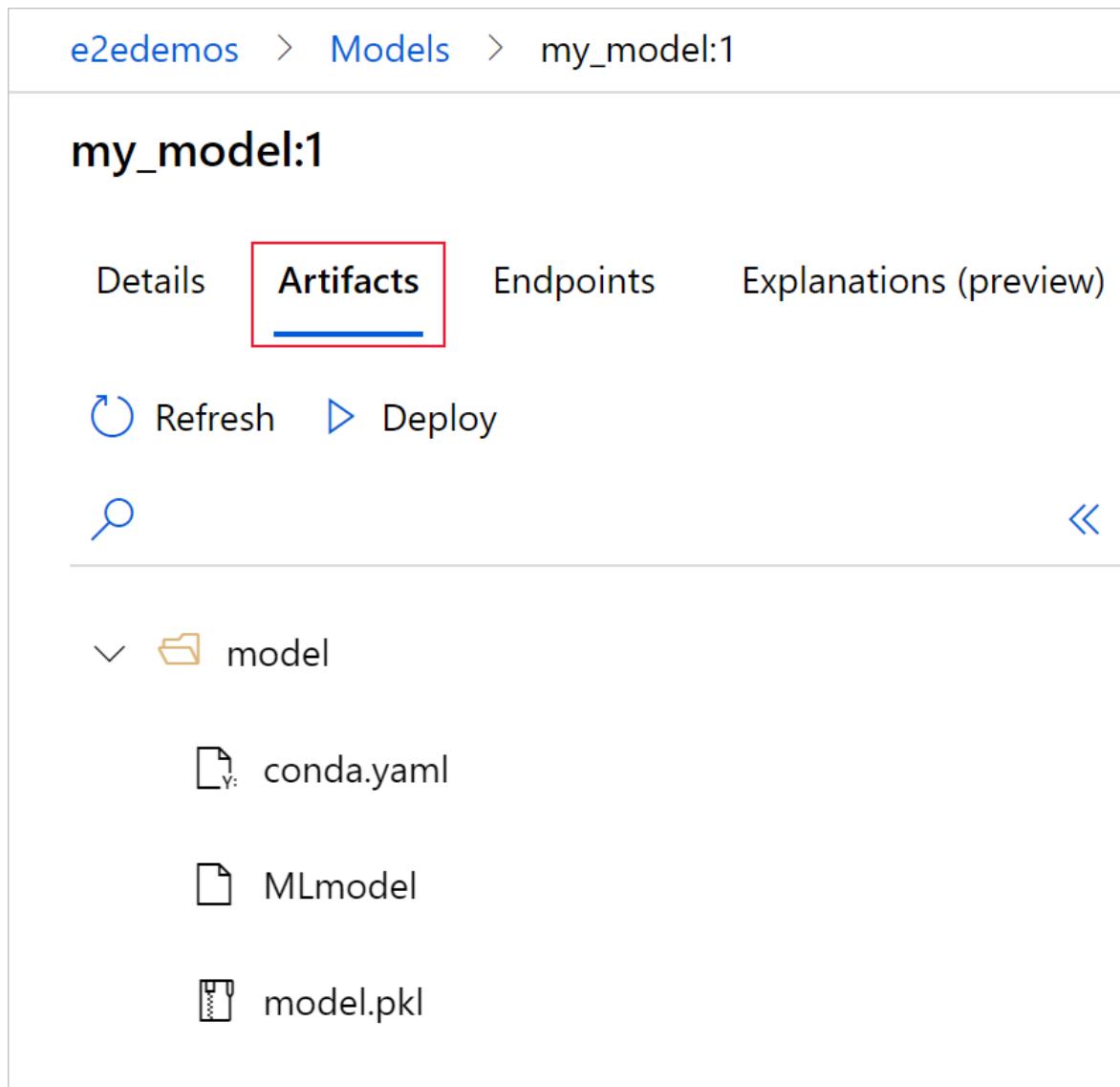
Details Artifacts Endpoints Explanations (preview)

Refresh Deploy

🔍 ⏪

model

- conda.yaml
- MLmodel
- model.pkl



4. Select MLmodel to see the MLmodel file generated by the job.

The screenshot shows the Azure Machine Learning studio interface. At the top, there's a navigation bar with 'e2edemos > Models > my_model:1'. Below it, a tab bar has 'Artifacts' selected, with other tabs like 'Details', 'Endpoints', 'Explanations (preview)', 'Fairness (preview)', and 'Datasets'. Underneath the tabs are two buttons: 'Refresh' and 'Deploy'. A search icon is also present. The main area is divided into two panes. The left pane shows a file tree with a 'model' folder containing 'conda.yaml', 'MLmodel' (which is selected and highlighted with a red box), and 'model.pkl'. The right pane displays the contents of the 'MLmodel' file in a code editor-like view:

```
1 artifact_path: model
2 flavors:
3   python_function:
4     env: conda.yaml
5     loader_module: mlflow.sklearn
6     model_path: model.pkl
7     python_version: 3.6.2
8     sklearn:
9       pickled_model: model.pkl
10      serialization_format:云pickle
11      sklearn_version: 0.23.2
12    run_id: example_15990
13    utc_time_created: '2020-09-02 16:54:26.734153'
14
```

Example files

[Using MLflow \(Jupyter Notebooks\)](#)

Limitations

Some methods available in the MLflow API may not be available when connected to Azure Machine Learning. For details about supported and unsupported operations please read [Support matrix for querying runs and experiments](#).

Next steps

- [Deploy MLflow models](#)).
- [Manage models with MLflow](#).

Track Azure Databricks ML experiments with MLflow and Azure Machine Learning

9/22/2022 • 10 minutes to read • [Edit Online](#)

In this article, learn how to enable MLflow to connect to Azure Machine Learning while working in an Azure Databricks workspace. You can leverage this configuration for tracking, model management and model deployment.

MLflow is an open-source library for managing the life cycle of your machine learning experiments. MLFlow Tracking is a component of MLflow that logs and tracks your training job metrics and model artifacts. Learn more about [Azure Databricks and MLflow](#).

See [MLflow and Azure Machine Learning](#) for additional MLflow and Azure Machine Learning functionality integrations.

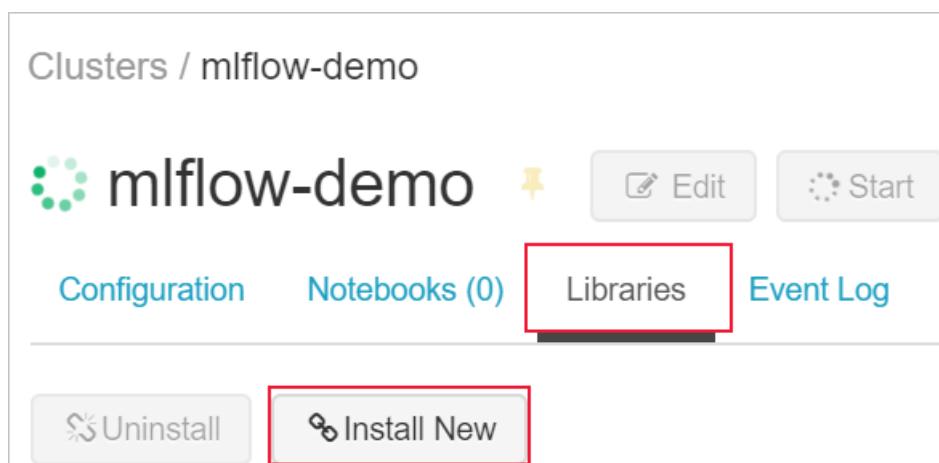
If you have an MLflow Project to train with Azure Machine Learning, see [Train ML models with MLflow Projects and Azure Machine Learning \(preview\)](#).

Prerequisites

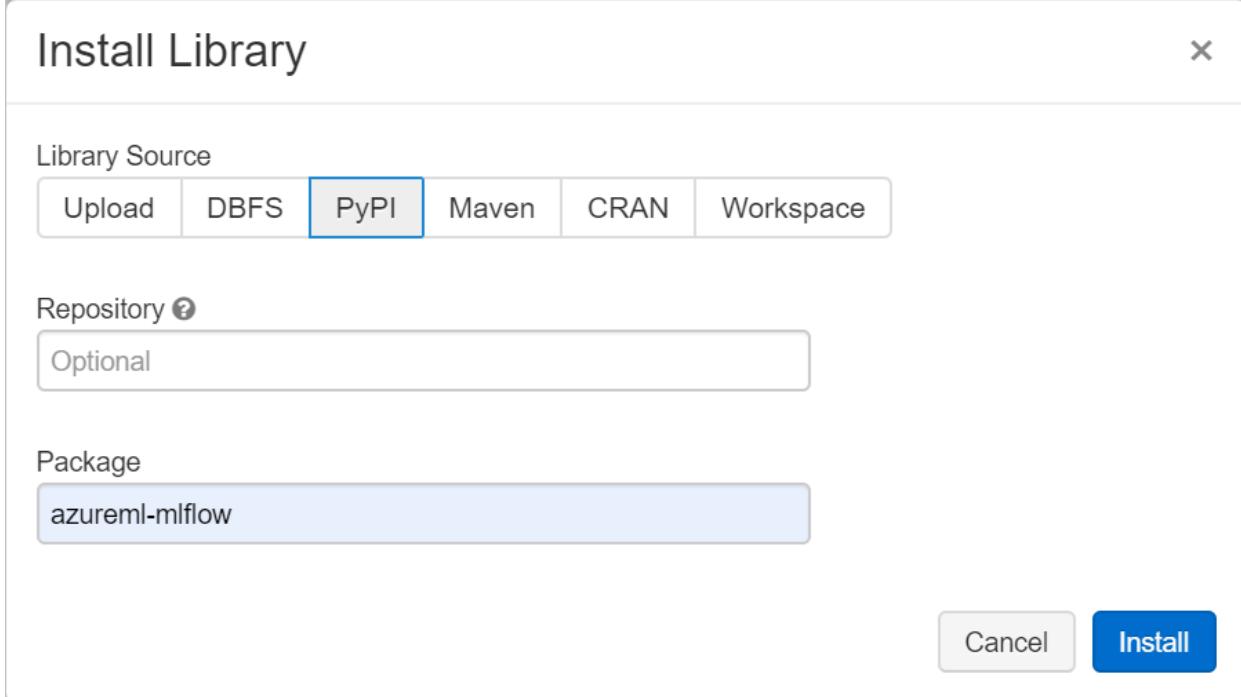
- Install the `azureml-mlflow` package, which handles the connectivity with Azure Machine Learning, including authentication.
- An [Azure Databricks workspace and cluster](#).
- [Create an Azure Machine Learning Workspace](#).
 - See which access permissions you need to perform your MLflow operations with your workspace.

Install libraries

To install libraries on your cluster, navigate to the **Libraries** tab and select **Install New**



In the **Package** field, type `azureml-mlflow` and then select install. Repeat this step as necessary to install other additional packages to your cluster for your experiment.



Track Azure Databricks runs with MLflow

Azure Databricks can be configured to track experiments using MLflow in two ways:

- [Track in both Azure Databricks workspace and Azure Machine Learning workspace \(dual-tracking\)](#)
- [Track exclusively on Azure Machine Learning](#)

By default, dual-tracking is configured for you when you linked your Azure Databricks workspace.

Dual-tracking on Azure Databricks and Azure Machine Learning

Linking your ADB workspace to your Azure Machine Learning workspace enables you to track your experiment data in the Azure Machine Learning workspace and Azure Databricks workspace at the same time. This is referred as Dual-tracking.

WARNING

Dual-tracking in a private link enabled Azure Machine Learning workspace is not supported by the moment. Configure exclusive tracking with your Azure Machine Learning workspace instead.

WARNING

Dual-tracking in not supported in Azure China by the moment. Configure exclusive tracking with your Azure Machine Learning workspace instead.

To link your ADB workspace to a new or existing Azure Machine Learning workspace,

1. Sign in to [Azure portal](#).
2. Navigate to your ADB workspace's [Overview](#) page.
3. Select the [Link Azure Machine Learning workspace](#) button on the bottom right.

The screenshot shows the Azure portal interface for managing a Databricks workspace named 'databricksws'. The left sidebar contains navigation links like Overview, Activity log, Access control (IAM), Tags, Settings, Virtual Network Peering, Encryption, Locks, Automation, Tasks, Export template, New support request, Documentation, Getting Started, Import Data from File, Import Data from Azure Storage, Notebook, Admin Guide, and Link Azure ML workspace. The 'Link Azure ML workspace' button is highlighted with a red box.

After you link your Azure Databricks workspace with your Azure Machine Learning workspace, MLflow Tracking is automatically set to be tracked in all of the following places:

- The linked Azure Machine Learning workspace.
- Your original ADB workspace.

You can use then MLflow in Azure Databricks in the same way as you're used to. The following example sets the experiment name as it is usually done in Azure Databricks and start logging some parameters:

```
import mlflow

experimentName = "/Users/{user_name}/{experiment_folder}/{experiment_name}"
mlflow.set_experiment(experimentName)

with mlflow.start_run():
    mlflow.log_param('epochs', 20)
    pass
```

NOTE

As opposite to tracking, model registries don't support registering models at the same time on both Azure Machine Learning and Azure Databricks. Either one or the other has to be used. Please read the section [Registering models in the registry with MLflow](#) for more details.

Tracking exclusively on Azure Machine Learning workspace

If you prefer to manage your tracked experiments in a centralized location, you can set MLflow tracking to **only** track in your Azure Machine Learning workspace. This configuration has the advantage of enabling easier path to deployment using Azure Machine Learning deployment options.

WARNING

For private link enabled Azure Machine Learning workspace, you have to deploy Azure Databricks in your own network (VNet injection) to ensure proper connectivity.

You have to configure the MLflow tracking URI to point exclusively to Azure Machine Learning, as it is demonstrated in the following example:

- [Using the Azure ML SDK v2](#)
- [Using an environment variable](#)
- [Building the MLflow tracking URI](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

You can get the Azure ML MLflow tracking URI using the [Azure Machine Learning SDK v2 for Python](#). Ensure you have the library `azure-ai-ml` installed in the cluster you are using. The following sample gets the unique MLFlow tracking URI associated with your workspace. Then the method `set_tracking_uri()` points the MLflow tracking URI to that URI.

a. Using the workspace configuration file:

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

ml_client = MLClient.from_config(credential=DefaultAzureCredential())
azureml_mlflow_uri = ml_client.workspaces.get(ml_client.workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)
```

TIP

You can download the workspace configuration file by:

1. Navigate to [Azure ML studio](#)
2. Click on the uper-right corner of the page -> Download config file.
3. Save the file `config.json` in the same directory where you are working on.

b. Using the subscription ID, resource group name and workspace name:

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

#Enter details of your AzureML workspace
subscription_id = '<SUBSCRIPTION_ID>'
resource_group = '<RESOURCE_GROUP>'
workspace_name = '<AZUREML_WORKSPACE_NAME>'

ml_client = MLClient(credential=DefaultAzureCredential(),
                     subscription_id=subscription_id,
                     resource_group_name=resource_group)

azureml_mlflow_uri = ml_client.workspaces.get(workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)
```

IMPORTANT

`DefaultAzureCredential` will try to pull the credentials from the available context. If you want to specify credentials in a different way, for instance using the web browser in an interactive way, you can use `InteractiveBrowserCredential` or any other method available in `azure.identity` package.

Experiment's names in Azure Machine Learning

When MLflow is configured to exclusively track experiments in Azure Machine Learning workspace, the experiment's naming convention has to follow the one used by Azure Machine Learning. In Azure Databricks, experiments are named with the path to where the experiment is saved like `/Users/alice@contoso.com/iris-classifier`. However, in Azure Machine Learning, you have to provide the experiment name directly. As in the previous example, the same experiment would be named `iris-classifier` directly:

```
mlflow.set_experiment(experiment_name="experiment-name")
```

Tracking parameters, metrics and artifacts

You can use then MLflow in Azure Databricks in the same way as you're used to. For details see [Log & view metrics and log files](#).

Logging models with MLflow

After your model is trained, you can log it to the tracking server with the `mlflow.<model_flavor>.log_model()` method. `<model_flavor>`, refers to the framework associated with the model. [Learn what model flavors are supported](#). In the following example, a model created with the Spark library MLLib is being registered:

```
mlflow.spark.log_model(model, artifact_path = "model")
```

It's worth to mention that the flavor `spark` doesn't correspond to the fact that we are training a model in a Spark cluster but because of the training framework it was used (you can perfectly train a model using TensorFlow with Spark and hence the flavor to use would be `tensorflow`).

Models are logged inside of the run being tracked. That means that models are available in either both Azure Databricks and Azure Machine Learning (default) or exclusively in Azure Machine Learning if you configured the tracking URI to point to it.

IMPORTANT

Notice that here the parameter `registered_model_name` has not been specified. Read the section [Registering models in the registry with MLflow](#) for more details about the implications of such parameter and how the registry works.

Registering models in the registry with MLflow

As opposite to tracking, **model registries can't operate** at the same time in Azure Databricks and Azure Machine Learning. Either one or the other has to be used. By default, the Azure Databricks workspace is used for model registries; unless you chose to [set MLflow Tracking to only track in your Azure Machine Learning workspace](#), then the model registry is the Azure Machine Learning workspace.

Then, considering you're using the default configuration, the following line will log a model inside the corresponding runs of both Azure Databricks and Azure Machine Learning, but it will register it only on Azure Databricks:

```
mlflow.spark.log_model(model, artifact_path = "model",
                       registered_model_name = 'model_name')
```

- If a registered model with the name doesn't exist, the method registers a new model, creates version 1, and returns a ModelVersion MLflow object.
- If a registered model with the name already exists, the method creates a new model version and returns the version object.

Using Azure Machine Learning Registry with MLflow

If you want to use Azure Machine Learning Model Registry instead of Azure Databricks, we recommend you to set [MLflow Tracking to only track in your Azure Machine Learning workspace](#). This will remove the ambiguity of where models are being registered and simplifies complexity.

However, if you want to continue using the dual-tracking capabilities but register models in Azure Machine Learning, you can instruct MLflow to use Azure ML for model registries by configuring the MLflow Model Registry URI. This URI has the exact same format and value that the MLflow tracking URI.

```
mlflow.set_registry_uri(azureml_mlflow_uri)
```

NOTE

The value of `azureml_mlflow_uri` was obtained in the same way it was demonstrated in [Set MLflow Tracking to only track in your Azure Machine Learning workspace](#)

For a complete example about this scenario please check the example [Training models in Azure Databricks and deploying them on Azure ML](#).

Deploying and consuming models registered in Azure Machine Learning

Models registered in Azure Machine Learning Service using MLflow can be consumed as:

- An Azure Machine Learning endpoint (real-time and batch): This deployment allows you to leverage Azure Machine Learning deployment capabilities for both real-time and batch inference in Azure Container Instances (ACI), Azure Kubernetes (AKS) or our Managed Inference Endpoints.
- MLFlow model objects or Pandas UDFs, which can be used in Azure Databricks notebooks in streaming or batch pipelines.

Deploy models to Azure Machine Learning endpoints

You can leverage the `azureml-mlflow` plugin to deploy a model to your Azure Machine Learning workspace. Check [How to deploy MLflow models](#) page for a complete detail about how to deploy models to the different targets.

IMPORTANT

Models need to be registered in Azure Machine Learning registry in order to deploy them. If your models happen to be registered in the MLflow instance inside Azure Databricks, you will have to register them again in Azure Machine Learning. If this is your case, please check the example [Training models in Azure Databricks and deploying them on Azure ML](#)

Deploy models to ADB for batch scoring using UDFs

You can choose Azure Databricks clusters for batch scoring. The MLFlow model is loaded and used as a Spark Pandas UDF to score new data.

```
from pyspark.sql.types import ArrayType, FloatType

model_uri = "runs:/"+last_run_id+ {model_path}

#Create a Spark UDF for the MLFlow model
pyfunc_udf = mlflow.pyfunc.spark_udf(spark, model_uri)

#Load Scoring Data into Spark Dataframe
scoreDf = spark.table({table_name}).where({required_conditions})

#Make Prediction
preds = (scoreDf
    .withColumn('target_column_name', pyfunc_udf('Input_column1', 'Input_column2', ' Input_column3',
...))
)
display(preds)
```

Clean up resources

If you wish to keep your Azure Databricks workspace, but no longer need the Azure ML workspace, you can delete the Azure ML workspace. This action results in unlinking your Azure Databricks workspace and the Azure ML workspace.

If you don't plan to use the logged metrics and artifacts in your workspace, the ability to delete them individually is unavailable at this time. Instead, delete the resource group that contains the storage account and workspace, so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with navigation links: 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES' (which includes 'All resources' and 'Resource groups'), 'App Services', 'Function Apps', and 'SQL databases'. The 'Resource groups' link is highlighted with a red box. The main content area is titled 'Resource groups' and shows a table with one item selected: 'aml-get-started-rg'. To the right of the table, there's a context menu with options: 'Pin to dashboard' (with a small icon), 'Delete resource group' (highlighted with a red box), and three vertical dots (...).

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name. Then select **Delete**.

Example notebooks

The [Training models in Azure Databricks and deploying them on Azure ML](#) demonstrates how to train models in Azure Databricks and deploy them in Azure ML. It also includes how to handle cases where you also want to

track the experiments and models with the MLflow instance in Azure Databricks and leverage Azure ML for deployment.

Next steps

- [Deploy MLflow models as an Azure web service.](#)
- [Manage your models.](#)
- [Track experiment jobs with MLflow and Azure Machine Learning.](#)
- Learn more about [Azure Databricks and MLflow](#).

Track Azure Synapse Analytics ML experiments with MLflow and Azure Machine Learning

9/22/2022 • 5 minutes to read • [Edit Online](#)

In this article, learn how to enable MLflow to connect to Azure Machine Learning while working in an Azure Synapse Analytics workspace. You can leverage this configuration for tracking, model management and model deployment.

MLflow is an open-source library for managing the life cycle of your machine learning experiments. MLFlow Tracking is a component of MLflow that logs and tracks your training run metrics and model artifacts. Learn more about [MLflow](#).

If you have an MLflow Project to train with Azure Machine Learning, see [Train ML models with MLflow Projects and Azure Machine Learning \(preview\)](#).

Prerequisites

- An [Azure Synapse Analytics workspace and cluster](#).
- An [Azure Machine Learning Workspace](#).

Install libraries

To install libraries on your dedicated cluster in Azure Synapse Analytics:

1. Create a `requirements.txt` file with the packages your experiments requires, but making sure it also includes the following packages:

`requirements.txt`

```
mlflow
azureml-mlflow
azure-ai-ml
```

2. Navigate to Azure Analytics Workspace portal.
3. Navigate to the **Manage** tab and select **Apache Spark Pools**.
4. Click the three dots next to the cluster name, and select **Packages**.

Name	Node size family	Size
dedicated	Memory Optimized	Small (4 vCores / 32 GB) - 3 to 3 nodes

- Scale settings
- Pause settings
- Packages**
- Apache Spark configuration
- Assign tags
- View role assignments
- Delete

5. On the **Requirements files** section, click on **Upload**.

6. Upload the `requirements.txt` file.

7. Wait for your cluster to restart.

Track experiments with MLflow

Azure Synapse Analytics can be configured to track experiments using MLflow to Azure Machine Learning workspace. Azure Machine Learning provides a centralized repository to manage the entire lifecycle of experiments, models and deployments. It also has the advantage of enabling easier path to deployment using Azure Machine Learning deployment options.

Configuring your notebooks to use MLflow connected to Azure Machine Learning

To use Azure Machine Learning as your centralized repository for experiments, you can leverage MLflow. On each notebook where you are working on, you have to configure the tracking URI to point to the workspace you will be using. The following example shows how it can be done:

- [Using the Azure ML SDK v2](#)
- [Building the MLflow tracking URI](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

You can get the Azure ML MLflow tracking URI using the [Azure Machine Learning SDK v2 for Python](#). Ensure you have the library `azure-ai-ml` installed in the cluster you are using. The following sample gets the unique MLFlow tracking URI associated with your workspace. Then the method `set_tracking_uri()` points the MLflow tracking URI to that URI.

a. Using the workspace configuration file:

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

ml_client = MLClient.from_config(credential=DefaultAzureCredential())
azureml_mlflow_uri = ml_client.workspaces.get(ml_client.workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)
```

TIP

You can download the workspace configuration file by:

1. Navigate to [Azure ML studio](#)
2. Click on the upper-right corner of the page -> Download config file.
3. Save the file `config.json` in the same directory where you are working on.

b. Using the subscription ID, resource group name and workspace name:

```

from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

#Enter details of your AzureML workspace
subscription_id = '<SUBSCRIPTION_ID>'
resource_group = '<RESOURCE_GROUP>'
workspace_name = '<AZUREML_WORKSPACE_NAME>'

ml_client = MLClient(credential=DefaultAzureCredential(),
                     subscription_id=subscription_id,
                     resource_group_name=resource_group)

azureml_mlflow_uri = ml_client.workspaces.get(workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)

```

IMPORTANT

`DefaultAzureCredential` will try to pull the credentials from the available context. If you want to specify credentials in a different way, for instance using the web browser in an interactive way, you can use `InteractiveBrowserCredential` or any other method available in `azure.identity` package.

Experiment's names in Azure Machine Learning

By default, Azure Machine Learning tracks runs in a default experiment called `Default`. It is usually a good idea to set the experiment you will be going to work on. Use the following syntax to set the experiment's name:

```
mlflow.set_experiment(experiment_name="experiment-name")
```

Tracking parameters, metrics and artifacts

You can use then MLflow in Azure Synapse Analytics in the same way as you're used to. For details see [Log & view metrics and log files](#).

Registering models in the registry with MLflow

Models can be registered in Azure Machine Learning workspace, which offers a centralized repository to manage their lifecycle. The following example logs a model trained with Spark MLLib and also registers it in the registry.

```
mlflow.spark.log_model(model,
                       artifact_path = "model",
                       registered_model_name = "model_name")
```

- **If a registered model with the name doesn't exist**, the method registers a new model, creates version 1, and returns a `ModelVersion` MLflow object.
- **If a registered model with the name already exists**, the method creates a new model version and returns the version object.

You can manage models registered in Azure Machine Learning using MLflow. View [Manage models registries in Azure Machine Learning with MLflow](#) for more details.

Deploying and consuming models registered in Azure Machine Learning

Models registered in Azure Machine Learning Service using MLflow can be consumed as:

- An Azure Machine Learning endpoint (real-time and batch): This deployment allows you to leverage Azure Machine Learning deployment capabilities for both real-time and batch inference in Azure Container Instances (ACI), Azure Kubernetes (AKS) or our Managed Endpoints.
- MLFlow model objects or Pandas UDFs, which can be used in Azure Synapse Analytics notebooks in streaming or batch pipelines.

Deploy models to Azure Machine Learning endpoints

You can leverage the `azureml-mlflow` plugin to deploy a model to your Azure Machine Learning workspace. Check [How to deploy MLflow models](#) page for a complete detail about how to deploy models to the different targets.

IMPORTANT

Models need to be registered in Azure Machine Learning registry in order to deploy them. Deployment of unregistered models is not supported in Azure Machine Learning.

Deploy models for batch scoring using UDFs

You can choose Azure Synapse Analytics clusters for batch scoring. The MLFlow model is loaded and used as a Spark Pandas UDF to score new data.

```
from pyspark.sql.types import ArrayType, FloatType

model_uri = "runs:/"+last_run_id+ {model_path}

#Create a Spark UDF for the MLflow model
pyfunc_udf = mlflow.pyfunc.spark_udf(spark, model_uri)

#Load Scoring Data into Spark Dataframe
scoreDf = spark.table({table_name}).where({required_conditions})

#Make Prediction
preds = (scoreDf
         .withColumn('target_column_name', pyfunc_udf('Input_column1', 'Input_column2', 'Input_column3',
...))
     )
display(preds)
```

Clean up resources

If you wish to keep your Azure Synapse Analytics workspace, but no longer need the Azure ML workspace, you can delete the Azure ML workspace. If you don't plan to use the logged metrics and artifacts in your workspace, the ability to delete them individually is unavailable at this time. Instead, delete the resource group that contains the storage account and workspace, so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.

The screenshot shows the Microsoft Azure portal interface. The left sidebar includes links for 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES' (with 'All resources' and 'Resource groups' listed), 'App Services', 'Function Apps', and 'SQL databases'. The main content area is titled 'Resource groups' and shows a list of items. At the top right are buttons for 'Add', 'Edit columns', 'Refresh', 'Assign tags', and 'Export to CSV'. Below this is a section for 'Subscriptions' with a link to 'Open Directory + Subscription settings'. The list of items shows 1 of 192 items selected. The columns are 'NAME' (sorted ascending), 'SUBSCRIPTION', and 'LOCATION'. A context menu is open for the item 'aml-get-started-rg', with options including 'Pin to dashboard' and 'Delete resource group'. The 'Delete resource group' option is highlighted with a red box.

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name. Then select **Delete**.

Next steps

- [Track experiment runs with MLflow and Azure Machine Learning.](#)
- [Deploy MLflow models in Azure Machine Learning.](#)
- [Manage your models with MLflow.](#)

Train ML models with MLflow Projects and Azure Machine Learning (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, learn how to enable MLflow's tracking URI and logging API, collectively known as [MLflow Tracking](#), to submit training jobs with [MLflow Projects](#) and Azure Machine Learning backend support. You can submit jobs locally with Azure Machine Learning tracking or migrate your runs to the cloud like via an [Azure Machine Learning Compute](#).

[MLflow Projects](#) allow for you to organize and describe your code to let other data scientists (or automated tools) run it. MLflow Projects with Azure Machine Learning enable you to track and manage your training runs in your workspace.

[MLflow](#) is an open-source library for managing the life cycle of your machine learning experiments. MLFlow Tracking is a component of MLflow that logs and tracks your training run metrics and model artifacts, no matter your experiment's environment--locally on your computer, on a remote compute target, a virtual machine, or an [Azure Databricks cluster](#).

[Learn more about the MLflow and Azure Machine Learning integration..](#)

TIP

The information in this document is primarily for data scientists and developers who want to monitor the model training process. If you are an administrator interested in monitoring resource usage and events from Azure Machine Learning, such as quotas, completed training runs, or completed model deployments, see [Monitoring Azure Machine Learning](#).

Prerequisites

- Install the `azureml-mlflow` package.
- [Create an Azure Machine Learning Workspace](#).
 - See which [access permissions you need to perform your MLflow operations with your workspace](#).
- Configure MLflow for tracking in Azure Machine Learning, as explained in the next section.

Set up tracking environment

To configure MLflow for working with Azure Machine Learning, you need to point your MLflow environment to the Azure Machine Learning MLflow Tracking URI.

NOTE

When running on Azure Compute (Azure Notebooks, Jupyter Notebooks hosted on Azure Compute Instances or Compute Clusters) you don't have to configure the tracking URI. It's automatically configured for you.

- [Using the Azure ML SDK v2](#)
- [Using an environment variable](#)
- [Building the MLflow tracking URI](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

You can get the Azure ML MLflow tracking URI using the [Azure Machine Learning SDK v2 for Python](#). Ensure you have the library `azure-ai-ml` installed in the cluster you are using. The following sample gets the unique MLFlow tracking URI associated with your workspace. Then the method `set_tracking_uri()` points the MLflow tracking URI to that URI.

1. Using the workspace configuration file:

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

ml_client = MLClient.from_config(credential=DefaultAzureCredential())
azureml_mlflow_uri = ml_client.workspaces.get(ml_client.workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)
```

TIP

You can download the workspace configuration file by:

1. Navigate to [Azure ML studio](#)
2. Click on the upper-right corner of the page -> Download config file.
3. Save the file `config.json` in the same directory where you are working on.

2. Using the subscription ID, resource group name and workspace name:

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
import mlflow

#Enter details of your AzureML workspace
subscription_id = '<SUBSCRIPTION_ID>'
resource_group = '<RESOURCE_GROUP>'
workspace_name = '<AZUREML_WORKSPACE_NAME>'

ml_client = MLClient(credential=DefaultAzureCredential(),
                     subscription_id=subscription_id,
                     resource_group_name=resource_group)

azureml_mlflow_uri = ml_client.workspaces.get(workspace_name).mlflow_tracking_uri
mlflow.set_tracking_uri(azureml_mlflow_uri)
```

IMPORTANT

`DefaultAzureCredential` will try to pull the credentials from the available context. If you want to specify credentials in a different way, for instance using the web browser in an interactive way, you can use `InteractiveBrowserCredential` or any other method available in `azure.identity` package.

Train MLflow Projects on local compute

This example shows how to submit MLflow projects locally with Azure Machine Learning.

Create the backend configuration object to store necessary information for the integration such as, the compute target and which type of managed environment to use.

```
backend_config = {"USE_CONDA": False}
```

Add the `azureml-mlflow` package as a pip dependency to your environment configuration file in order to track metrics and key artifacts in your workspace.

```
name: mlflow-example
channels:
  - defaults
  - anaconda
  - conda-forge
dependencies:
  - python=3.6
  - scikit-learn=0.19.1
  - pip
  - pip:
    - mlflow
    - azureml-mlflow
```

Submit the local run and ensure you set the parameter `backend = "azureml"`. With this setting, you can submit runs locally and get the added support of automatic output tracking, log files, snapshots, and printed errors in your workspace.

View your runs and metrics in the [Azure Machine Learning studio](#).

```
local_env_run = mlflow.projects.run(uri=".",
                                      parameters={"alpha":0.3},
                                      backend = "azureml",
                                      use_conda=False,
                                      backend_config = backend_config,
                                      )
```

Train MLflow projects with remote compute

This example shows how to submit MLflow projects on a remote compute with Azure Machine Learning tracking.

Create the backend configuration object to store necessary information for the integration such as, the compute target and which type of managed environment to use.

The integration accepts "COMPUTE" and "USE_CONDA" as parameters where "COMPUTE" is set to the name of your remote compute cluster and "USE_CONDA" which creates a new environment for the project from the environment configuration file. If "COMPUTE" is present in the object, the project will be automatically submitted to the remote compute and ignore "USE_CONDA". MLflow accepts a dictionary object or a JSON file.

```
# dictionary
backend_config = {"COMPUTE": "cpu-cluster", "USE_CONDA": False}
```

Add the `azureml-mlflow` package as a pip dependency to your environment configuration file in order to track metrics and key artifacts in your workspace.

```

name: mlflow-example
channels:
  - defaults
  - anaconda
  - conda-forge
dependencies:
  - python=3.6
  - scikit-learn=0.19.1
  - pip
  - pip:
    - mlflow
  - azureml-mlflow

```

Submit the mlflow project run and ensure you set the parameter `backend = "azureml"`. With this setting, you can submit your run to your remote compute and get the added support of automatic output tracking, log files, snapshots, and printed errors in your workspace.

View your runs and metrics in the [Azure Machine Learning studio](#).

```

remote_mlflow_run = mlflow.projects.run(uri=".",
                                         parameters={"alpha":0.3},
                                         backend = "azureml",
                                         backend_config = backend_config,
                                         )

```

Clean up resources

If you don't plan to use the logged metrics and artifacts in your workspace, the ability to delete them individually is currently unavailable. Instead, delete the resource group that contains the storage account and workspace, so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.

NAME	SUBSCRIPTION	LOCATION
aml-get-started-rg		

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name. Then select **Delete**.

Example notebooks

The [MLflow with Azure ML notebooks](#) demonstrate and expand upon concepts presented in this article.

- [Train an MLflow project on a local compute](#)
- [Train an MLflow project on remote compute.](#)

NOTE

A community-driven repository of examples using mlflow can be found at <https://github.com/Azure/azureml-examples>.

Next steps

- [Deploy models with MLflow.](#)
- Monitor your production models for [data drift](#).
- [Track Azure Databricks runs with MLflow.](#)
- [Manage your models.](#)

Query & compare experiments and runs with MLflow

9/22/2022 • 8 minutes to read • [Edit Online](#)

Experiments and runs in Azure Machine Learning can be queried using MLflow. This removes the need of any Azure Machine Learning specific SDKs to manage anything that happens inside of a training job, allowing dependencies removal and creating a more seamless transition between local runs and cloud.

NOTE

The Azure Machine Learning Python SDK v2 (preview) does not provide native logging or tracking capabilities. This applies not just for logging but also for querying the metrics logged. Instead, we recommend to use MLflow to manage experiments and runs. This article explains how to use MLflow to manage experiments and runs in Azure ML.

MLflow allows you to:

- Create, delete and search for experiments in a workspace.
- Start, stop, cancel and query runs for experiments.
- Track and retrieve metrics, parameters, artifacts and models from runs.

In this article, you'll learn how to manage experiments and runs in your workspace using Azure ML and MLflow SDK in Python.

Using MLflow SDK in Azure ML

Use MLflow to query and manage all the experiments in Azure Machine Learning. The MLflow SDK has capabilities to query everything that happens inside of a training job in Azure Machine Learning. See [Support matrix for querying runs and experiments in Azure Machine Learning](#) for a detailed comparison between MLflow Open-Source and MLflow when connected to Azure Machine Learning.

Prerequisites

- Install `azureml-mlflow` plug-in.
- If you're running in a compute not hosted in Azure ML, configure MLflow to point to the Azure ML tracking URL. You can follow the instruction at [Track runs from your local machine](#).

Getting all the experiments

You can get all the active experiments in the workspace using MLFlow:

```
experiments = mlflow.list_experiments()
for exp in experiments:
    print(exp.name)
```

If you want to retrieve archived experiments too, then include the option `ViewType.ALL` in the `view_type` argument. The following sample shows how:

```
from mlflow.entities import ViewType

experiments = mlflow.list_experiments(view_type=ViewType.ALL)
for exp in experiments:
    print(exp.name)
```

Getting a specific experiment

Details about a specific experiment can be retrieved using the `get_experiment_by_name` method:

```
exp = mlflow.get_experiment_by_name(experiment_name)
print(exp)
```

Getting runs inside an experiment

MLflow allows searching runs inside of any experiment, including multiple experiments at the same time. By default, MLflow returns the data in Pandas `Dataframe` format, which makes it handy when doing further processing our analysis of the runs. Returned data includes columns with:

- Basic information about the run.
- Parameters with column's name `params.<parameter-name>`.
- Metrics (last logged value of each) with column's name `metrics.<metric-name>`.

Getting all the runs from an experiment

By experiment name:

```
mlflow.search_runs(experiment_names=[ "my_experiment" ])
```

By experiment ID:

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ])
```

TIP

Notice that `experiment_ids` supports providing an array of experiments, so you can search runs across multiple experiments if required. This may be useful in case you want to compare runs of the same model when it is being logged in different experiments (by different people, different project iterations, etc). You can also use `search_all_experiments=True` if you want to search across all the experiments in the workspace.

Another important point to notice is that get returning runs, all metrics are parameters are also returned for them. However, for metrics containing multiple values (for instance, a loss curve, or a PR curve), only the last value of the metric is returned. If you want to retrieve all the values of a given metric, uses `mlflow.get_metric_history` method.

Ordering runs

By default, experiments are ordered descending by `start_time`, which is the time the experiment was queue in Azure ML. However, you can change this default by using the parameter `order_by`.

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ], order_by=["start_time DESC"])
```

Use the argument `max_results` from `search_runs` to limit the number of runs returned. For instance, the following example returns the last run of the experiment:

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ], max_results=1, order_by=["start_time DESC"])
```

WARNING

Using `order_by` with expressions containing `metrics.*` in the parameter `order_by` is not supported by the moment. Please use `order_values` method from Pandas as shown in the next example.

You can also order by metrics to know which run generated the best results:

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ]).sort_values("metrics.accuracy", ascending=False)
```

Filtering runs

You can also look for a run with a specific combination in the hyperparameters using the parameter `filter_string`. Use `params` to access run's parameters and `metrics` to access metrics logged in the run. MLflow supports expressions joined by the AND keyword (the syntax does not support OR):

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ], filter_string="params.num_boost_round='100'")
```

Filter runs by status

You can also filter experiment by status. It becomes useful to find runs that are running, completed, canceled or failed. In MLflow, `status` is an `attribute`, so we can access this value using the expression `attributes.status`. The following table shows the possible values:

AZURE ML JOB STATUS	MLFLOW'S ATTRIBUTES.STATUS	MEANING
Not started	SCHEDULED	The job/run was just registered in Azure ML but it has processed it yet.
Queue	SCHEDULED	The job/run is scheduled for running, but it hasn't started yet.
Preparing	SCHEDULED	The job/run has not started yet, but a compute has been allocated for the execution and it is on building state.
Running	RUNNING	The job/run is currently under active execution.
Completed	FINISHED	The job/run has completed without errors.
Failed	FAILED	The job/run has completed with errors.
Canceled	KILLED	The job/run has been canceled or killed by the user/system.

WARNING

Expressions containing `attributes.status` in the parameter `filter_string` are not supported at the moment. Please use Pandas filtering expressions as shown in the next example.

The following example shows all the runs that have been completed:

```
runs = mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ])
runs[runs.status == "FINISHED"]
```

Getting metrics, parameters, artifacts and models

By default, MLflow returns runs as a Pandas `Dataframe` containing a limited amount of information. You can get Python objects if needed, which may be useful to get details about them. Use the `output_format` parameter to control how output is returned:

```
runs = mlflow.search_runs(
    experiment_ids=[ "1234-5678-90AB-CDEFG" ],
    filter_string="params.num_boost_round='100'",
    output_format="list",
)
```

Details can then be accessed from the `info` member. The following sample shows how to get the `run_id`:

```
last_run = runs[-1]
print("Last run ID:", last_run.info.run_id)
```

Getting params and metrics from a run

When runs are returned using `output_format="list"`, you can easily access parameters using the key `data`:

```
last_run.data.params
```

In the same way, you can query metrics:

```
last_run.data.metrics
```

For metrics that contain multiple values (for instance, a loss curve, or a PR curve), only the last logged value of the metric is returned. If you want to retrieve all the values of a given metric, use the `mlflow.get_metric_history` method. This method requires you to use the `MlflowClient`:

```
client = mlflow.tracking.MlflowClient()
client.get_metric_history("1234-5678-90AB-CDEFG", "log_loss")
```

Getting artifacts from a run

Any artifact logged by a run can be queried by MLflow. Artifacts can't be accessed using the run object itself and the MLflow client should be used instead:

```
client = mlflow.tracking.MlflowClient()
client.list_artifacts("1234-5678-90AB-CDEFG")
```

The method above will list all the artifacts logged in the run, but they will remain stored in the artifacts store (Azure ML storage). To download any of them, use the method `download_artifact`:

```
file_path = client.download_artifacts("1234-5678-90AB-CDEFG", path="feature_importance_weight.png")
```

Getting models from a run

Models can also be logged in the run and then retrieved directly from it. To retrieve it, you need to know the artifact's path where it is stored. The method `list_artifacts` can be used to find artifacts that are representing a model since MLflow models are always folders. You can download a model by indicating the path where the model is stored using the `download_artifact` method:

```
artifact_path="classifier"
model_local_path = client.download_artifacts("1234-5678-90AB-CDEFG", path=artifact_path)
```

You can then load the model back from the downloaded artifacts using the typical function `load_model`:

```
model = mlflow.xgboost.load_model(model_local_path)
```

NOTE

In the example above, we are assuming the model was created using `xgboost`. Change it to the flavor applies to your case.

MLflow also allows you to both operations at once and download and load the model in a single instruction. MLflow will download the model to a temporary folder and load it from there. This can be done using the `load_model` method which uses an URI format to indicate from where the model has to be retrieved. In the case of loading a model from a run, the URI structure is as follows:

```
model = mlflow.xgboost.load_model(f"runs:{last_run.info.run_id}/{artifact_path}")
```

TIP

You can also load models from the registry using MLflow. View [loading MLflow models with MLflow](#) for details.

Getting child (nested) runs

MLflow supports the concept of child (nested) runs. They are useful when you need to spin off training routines requiring being tracked independently from the main training process. This is the typical case of hyper-parameter tuning for instance. You can query all the child runs of a specific run using the property tag `mlflow.parentRunId`, which contains the run ID of the parent run.

```
hyperopt_run = mlflow.last_active_run()
child_runs = mlflow.search_runs(
    filter_string=f"tags.mlflow.parentRunId='{hyperopt_run.info.run_id}'"
)
```

Example notebooks

The [MLflow with Azure ML notebooks](#) demonstrate and expand upon concepts presented in this article.

- [Training and tracking a classifier with MLflow](#): Demonstrates how to track experiments using MLflow, log models and combine multiple flavors into pipelines.
- [Manage experiments and runs with MLflow](#): Demonstrates how to query experiments, runs, metrics, parameters and artifacts from Azure ML using MLflow.

Support matrix for querying runs and experiments

The MLflow SDK exposes several methods to retrieve runs, including options to control what is returned and how. Use the following table to learn about which of those methods are currently supported in MLflow when connected to Azure Machine Learning:

FEATURE	SUPPORTED BY MLFLOW	SUPPORTED BY AZURE ML
Ordering runs by run fields (like <code>start_time</code> , <code>end_time</code> , etc)	✓	✓
Ordering runs by attributes	✓	1
Ordering runs by metrics	✓	1
Ordering runs by parameters	✓	1
Ordering runs by tags	✓	1
Filtering runs by run fields (like <code>start_time</code> , <code>end_time</code> , etc)		1
Filtering runs by attributes	✓	1
Filtering runs by metrics	✓	✓
Filtering runs by metrics with special characters (escaped)	✓	
Filtering runs by parameters	✓	✓
Filtering runs by tags	✓	✓
Filtering runs with numeric comparators (metrics) including <code>=</code> , <code>!=</code> , <code>></code> , <code>>=</code> , <code><</code> , and <code><=</code>	✓	✓
Filtering runs with string comparators (params, tags, and attributes): <code>=</code> and <code>!=</code>	✓	✓ ²
Filtering runs with string comparators (params, tags, and attributes): <code>LIKE</code> / <code>ILIKE</code>	✓	
Filtering runs with comparators <code>AND</code>	✓	✓
Filtering runs with comparators <code>OR</code>		

FEATURE	SUPPORTED BY MLFLOW	SUPPORTED BY AZURE ML
Renaming experiments	✓	

NOTE

- ¹ Check the section [Getting runs inside an experiment](#) for instructions and examples on how to achieve the same functionality in Azure ML.
- ² for tags not supported.

Next steps

- [Manage your models with MLflow.](#)
- [Deploy models with MLflow.](#)

Set up AutoML training with the Azure ML Python SDK v2 (preview)

9/22/2022 • 15 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this guide, learn how to set up an automated machine learning, AutoML, training job with the [Azure Machine Learning Python SDK v2 \(preview\)](#). Automated ML picks an algorithm and hyperparameters for you and generates a model ready for deployment. This guide provides details of the various options that you can use to configure automated ML experiments.

If you prefer a no-code experience, you can also [Set up no-code AutoML training in the Azure Machine Learning studio](#).

If you prefer to submit training jobs with the Azure Machine learning CLI v2 extension, see [Train models with the CLI \(v2\)](#).

Prerequisites

For this article you need:

- An Azure Machine Learning workspace. To create the workspace, see [Create workspace resources](#).
- The Azure Machine Learning Python SDK v2 (preview) installed. To install the SDK you can either,
 - Create a compute instance, which already has installed the latest AzureML Python SDK and is pre-configured for ML workflows. See [Create and manage an Azure Machine Learning compute instance](#) for more information.
 - Use the followings commands to install Azure ML Python SDK v2:
 - Uninstall previous preview version:

```
pip uninstall azure-ai-ml
```

- Install the Azure ML Python SDK v2:

```
pip install azure-ai-ml
```

IMPORTANT

The Python commands in this article require the latest `azureml-train-automl` package version.

- [Install the latest `azureml-train-automl` package to your local environment.](#)
- For details on the latest `azureml-train-automl` package, see the [release notes](#).

Setup your workspace

To connect to a workspace, you need to provide a subscription, resource group and workspace name. These details are used in the MLClient from `azure.ai.ml` to get a handle to the required Azure Machine Learning workspace.

In the following example, the default Azure authentication is used along with the default workspace configuration or from any `config.json` file you might have copied into the folders structure. If no `config.json` is found, then you need to manually introduce the `subscription_id`, `resource_group` and `workspace` when creating MLClient.

```
from azure.identity import DefaultAzureCredential
from azure.ai.ml import MLClient

credential = DefaultAzureCredential()
ml_client = None
try:
    ml_client = MLClient.from_config(credential)
except Exception as ex:
    print(ex)
# Enter details of your AzureML workspace
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
workspace = "<AZUREML_WORKSPACE_NAME>"
ml_client = MLClient(credential, subscription_id, resource_group, workspace)
```

Data source and format

In order to provide training data to AutoML in SDK v2 you need to upload it into the cloud through an **MLTable**.

Requirements for loading data into an MLTable:

- Data must be in tabular form.
- The value to predict, target column, must be in the data.

Training data must be accessible from the remote compute. Automated ML v2 (Python SDK and CLI/YAML) accepts MLTable data assets (v2), although for backwards compatibility it also supports v1 Tabular Datasets from v1 (a registered Tabular Dataset) through the same input dataset properties. However the recommendation is to use MLTable available in v2.

The following YAML code is the definition of a MLTable that could be placed in a local folder or a remote folder in the cloud, along with the data file (.CSV or Parquet file).

```

# MLTable definition file

paths:
- file: ./bank_marketing_train_data.csv
transformations:
- read_delimited:
  delimiter: ','
  encoding: 'ascii'

```

Therefore, the MLTable folder would have the MLTable definition file plus the data file (the bank_marketing_train_data.csv file in this case).

The following shows two ways of creating an MLTable.

- A. Providing your training data and MLTable definition file from your local folder and it'll be automatically uploaded into the cloud (default Workspace Datastore)
- B. Providing a MLTable already registered and uploaded into the cloud.

```

from azure.ai.ml.constants import AssetTypes
from azure.ai.ml import automl, Input

# A. Create MLTable for training data from your local directory
my_training_data_input = Input(
    type=AssetTypes.MLTABLE, path="./data/training-mltable-folder"
)

# B. Remote MLTable definition
my_training_data_input = Input(type=AssetTypes.MLTABLE,
path="azureml://datastores/workspaceblobstore/paths/Classification/Train")

```

Training, validation, and test data

You can specify separate **training data** and **validation data** sets, however training data must be provided to the `training_data` parameter in the factory function of your automated ML job.

If you do not explicitly specify a `validation_data` or `n_cross_validation` parameter, automated ML applies default techniques to determine how validation is performed. This determination depends on the number of rows in the dataset assigned to your `training_data` parameter.

TRAINING DATA SIZE	VALIDATION TECHNIQUE
Larger than 20,000 rows	Train/validation data split is applied. The default is to take 10% of the initial training data set as the validation set. In turn, that validation set is used for metrics calculation.
Smaller than or equal to 20,000 rows	Cross-validation approach is applied. The default number of folds depends on the number of rows. If the dataset is fewer than 1,000 rows, 10 folds are used. If the rows are equal to or between 1,000 and 20,000, then three folds are used.

Compute to run experiment

Automated ML jobs with the Python SDK v2 (or CLI v2) are currently only supported on Azure ML remote compute (cluster or compute instance).

[Learn more about creating compute with the Python SDKv2 \(or CLIV2\)..](#)

Configure your experiment settings

There are several options that you can use to configure your automated ML experiment. These configuration parameters are set in your task method. You can also set job training settings and [exit criteria](#) with the `set_training()` and `set_limits()` functions, respectively.

The following example shows the required parameters for a classification task that specifies accuracy as the [primary metric](#) and 5 cross-validation folds.

```
# note that the below is a code snippet -- you might have to modify the variable values to run it
successfully
classification_job = automl.classification(
    compute=my_compute_name,
    experiment_name=my_exp_name,
    training_data=my_training_data_input,
    target_column_name="y",
    primary_metric="accuracy",
    n_cross_validations=5,
    enable_model_explainability=True,
    tags={"my_custom_tag": "My custom value"}
)

# Limits are all optional

classification_job.set_limits(
    timeout_minutes=600,
    trial_timeout_minutes=20,
    max_trials=5,
    enable_early_termination=True,
)

# Training properties are optional
classification_job.set_training(
    blocked_training_algorithms=["LogisticRegression"],
    enable_onnx_compatible_models=True
)
```

Select your machine learning task type (ML problem)

Before you can submit your automated ML job, you need to determine the kind of machine learning problem you are solving. This problem determines which function your automated ML job uses and what model algorithms it applies.

Automated ML supports tabular data based tasks (classification, regression, forecasting), computer vision tasks (such as Image Classification and Object Detection), and natural language processing tasks (such as Text classification and Entity Recognition tasks). Learn more about [task types](#).

Supported algorithms

Automated machine learning tries different models and algorithms during the automation and tuning process. As a user, there is no need for you to specify the algorithm.

The task method determines the list of algorithms/models, to apply. Use the `allowed_algorithms` or `blocked_training_algorithms` parameters in the `set_training()` setter function to further modify iterations with the available models to include or exclude.

In the following list of links you can explore the supported algorithms per machine learning task listed below.

CLASSIFICATION	REGRESSION	TIME SERIES FORECASTING
Logistic Regression*	Elastic Net*	AutoARIMA

CLASSIFICATION	REGRESSION	TIME SERIES FORECASTING
Light GBM*	Light GBM*	Prophet
Gradient Boosting*	Gradient Boosting*	Elastic Net
Decision Tree*	Decision Tree*	Light GBM
K Nearest Neighbors*	K Nearest Neighbors*	K Nearest Neighbors
Linear SVC*	LARS Lasso*	Decision Tree
Support Vector Classification (SVC)*	Stochastic Gradient Descent (SGD)*	Arimax
Random Forest*	Random Forest	LARS Lasso
Extremely Randomized Trees*	Extremely Randomized Trees*	Extremely Randomized Trees*
Xgboost*	Xgboost*	Random Forest
Naive Bayes*	Xgboost	ForecastTCN
Stochastic Gradient Descent (SGD)*	Stochastic Gradient Descent (SGD)	Gradient Boosting
		ExponentialSmoothing
		SeasonalNaive
		Average
		Naive
		SeasonalAverage

With additional algorithms below.

- [Image Classification Multi-class Algorithms](#)
- [Image Classification Multi-label Algorithms](#)
- [Image Object Detection Algorithms](#)
- [NLP Text Classification Multi-label Algorithms](#)
- [NLP Text Named Entity Recognition \(NER\) Algorithms](#)

Follow [this link](#) for example notebooks of each task type.

Primary metric

The `primary_metric` parameter determines the metric to be used during model training for optimization. The available metrics you can select is determined by the task type you choose.

Choosing a primary metric for automated ML to optimize depends on many factors. We recommend your primary consideration be to choose a metric that best represents your business needs. Then consider if the metric is suitable for your dataset profile (data size, range, class distribution, etc.). The following sections summarize the recommended primary metrics based on task type and business scenario.

Learn about the specific definitions of these metrics in [Understand automated machine learning results](#).

Metrics for classification multi-class scenarios

These metrics apply for all classification scenarios, including tabular data, images/computer-vision and NLP-Text.

Threshold-dependent metrics, like `accuracy`, `recall_score_weighted`, `norm_macro_recall`, and `precision_score_weighted` may not optimize as well for datasets that are small, have very large class skew (class imbalance), or when the expected metric value is very close to 0.0 or 1.0. In those cases, `AUC_weighted` can be a better choice for the primary metric. After automated ML completes, you can choose the winning model based on the metric best suited to your business needs.

METRIC	EXAMPLE USE CASE(S)
<code>accuracy</code>	Image classification, Sentiment analysis, Churn prediction
<code>AUC_weighted</code>	Fraud detection, Image classification, Anomaly detection/spam detection
<code>average_precision_score_weighted</code>	Sentiment analysis
<code>norm_macro_recall</code>	Churn prediction
<code>precision_score_weighted</code>	

Metrics for classification multi-label scenarios

- For Text classification multi-label currently 'Accuracy' is the only primary metric supported.
- For Image classification multi-label, the primary metrics supported are defined in the `ClassificationMultilabelPrimaryMetrics` Enum

Metrics for NLP Text NER (Named Entity Recognition) scenarios

- For NLP Text NER (Named Entity Recognition) currently 'Accuracy' is the only primary metric supported.

Metrics for regression scenarios

`r2_score`, `normalized_mean_absolute_error` and `normalized_root_mean_squared_error` are all trying to minimize prediction errors. `r2_score` and `normalized_root_mean_squared_error` are both minimizing average squared errors while `normalized_mean_absolute_error` is minimizing the average absolute value of errors. Absolute value treats errors at all magnitudes alike and squared errors will have a much larger penalty for errors with larger absolute values. Depending on whether larger errors should be punished more or not, one can choose to optimize squared error or absolute error.

The main difference between `r2_score` and `normalized_root_mean_squared_error` is the way they are normalized and their meanings. `normalized_root_mean_squared_error` is root mean squared error normalized by range and can be interpreted as the average error magnitude for prediction. `r2_score` is mean squared error normalized by an estimate of variance of data. It is the proportion of variation that can be captured by the model.

NOTE

`r2_score` and `normalized_root_mean_squared_error` also behave similarly as primary metrics. If a fixed validation set is applied, these two metrics are optimizing the same target, mean squared error, and will be optimized by the same model. When only a training set is available and cross-validation is applied, they would be slightly different as the normalizer for `normalized_root_mean_squared_error` is fixed as the range of training set, but the normalizer for `r2_score` would vary for every fold as it's the variance for each fold.

If the rank, instead of the exact value is of interest, `spearman_correlation` can be a better choice as it measures the rank correlation between real values and predictions.

However, currently no primary metrics for regression addresses relative difference. All of `r2_score`, `normalized_mean_absolute_error`, and `normalized_root_mean_squared_error` treat a \$20k prediction error the same for a worker with a \$30k salary as a worker making \$20M, if these two data points belongs to the same dataset for regression, or the same time series specified by the time series identifier. While in reality, predicting only \$20k off from a \$20M salary is very close (a small 0.1% relative difference), whereas \$20k off from \$30k is not close (a large 67% relative difference). To address the issue of relative difference, one can train a model with available primary metrics, and then select the model with best `mean_absolute_percentage_error` or `root_mean_squared_log_error`.

METRIC	EXAMPLE USE CASE(S)
<code>spearman_correlation</code>	
<code>normalized_root_mean_squared_error</code>	Price prediction (house/product/tip), Review score prediction
<code>r2_score</code>	Airline delay, Salary estimation, Bug resolution time
<code>normalized_mean_absolute_error</code>	

Metrics for Time Series Forecasting scenarios

The recommendations are similar to those noted for regression scenarios.

METRIC	EXAMPLE USE CASE(S)
<code>normalized_root_mean_squared_error</code>	Price prediction (forecasting), Inventory optimization, Demand forecasting
<code>r2_score</code>	Price prediction (forecasting), Inventory optimization, Demand forecasting
<code>normalized_mean_absolute_error</code>	

Metrics for Image Object Detection scenarios

- For Image Object Detection, the primary metrics supported are defined in the `ObjectDetectionPrimaryMetrics` Enum

Metrics for Image Instance Segmentation scenarios

- For Image Instance Segmentation scenarios, the primary metrics supported are defined in the `InstanceSegmentationPrimaryMetrics` Enum

Data featurization

In every automated ML experiment, your data is automatically transformed to numbers and vectors of numbers plus (i.e. converting text to numeric) also scaled and normalized to help *certain* algorithms that are sensitive to features that are on different scales. This data transformation, scaling and normalization is referred to as featurization.

NOTE

Automated machine learning featurization steps (feature normalization, handling missing data, converting text to numeric, etc.) become part of the underlying model. When using the model for predictions, the same featurization steps applied during training are applied to your input data automatically.

When configuring your automated ML jobs, you can enable/disable the `featurization` settings by using the

`.set_featurization()` setter function.

The following table shows the accepted settings for featurization.

FEATURIZATION CONFIGURATION	DESCRIPTION
<code>"mode": "auto"</code>	Indicates that as part of preprocessing, data guardrails and featurization steps are performed automatically. Default setting .
<code>"mode": "off"</code>	Indicates featurization step shouldn't be done automatically.
<code>"mode": "custom"</code>	Indicates customized featurization step should be used.

The following code shows how custom featurization can be provided in this case for a regression job.

```
from azure.ai.ml.automl import ColumnTransformer

transformer_params = {
    "imputer": [
        ColumnTransformer(fields=["CACH"], parameters={"strategy": "most_frequent"}),
        ColumnTransformer(fields=["PRP"], parameters={"strategy": "most_frequent"}),
    ],
}
regression_job.set_featurization(
    mode="custom",
    transformer_params=transformer_params,
    blocked_transformers=["LabelEncoding"],
    column_name_and_types={"CHMIN": "Categorical"},
)
```

Exit criteria

There are a few options you can define in the `set_limits()` function to end your experiment prior to job completion.

CRITERIA	DESCRIPTION
No criteria	If you do not define any exit parameters the experiment continues until no further progress is made on your primary metric.
<code>timeout</code>	Defines how long, in minutes, your experiment should continue to run. If not specified, the default job's total timeout is 6 days (8,640 minutes). To specify a timeout less than or equal to 1 hour (60 minutes), make sure your dataset's size is not greater than 10,000,000 (rows times column) or an error results. This timeout includes setup, featurization and training runs but does not include the ensembling and model explainability runs at the end of the process since those actions need to happen once all the trials (children jobs) are done.
<code>trial_timeout_minutes</code>	Maximum time in minutes that each trial (child job) can run for before it terminates. If not specified, a value of 1 month or 43200 minutes is used

CRITERIA	DESCRIPTION
<code>enable_early_termination</code>	Whether to end the job if the score is not improving in the short term
<code>max_trials</code>	The maximum number of trials/runs each with a different combination of algorithm and hyperparameters to try during an AutoML job. If not specified, the default is 1000 trials. If using <code>enable_early_termination</code> the number of trials used can be smaller.
<code>max_concurrent_trials</code>	Represents the maximum number of trials (children jobs) that would be executed in parallel. It's a good practice to match this number with the number of nodes your cluster

Run experiment

NOTE

If you run an experiment with the same configuration settings and primary metric multiple times, you'll likely see variation in each experiments final metrics score and generated models. The algorithms automated ML employs have inherent randomness that can cause slight variation in the models output by the experiment and the recommended model's final metrics score, like accuracy. You'll likely also see results with the same model name, but different hyperparameters used.

WARNING

If you have set rules in firewall and/or Network Security Group over your workspace, verify that required permissions are given to inbound and outbound network traffic as defined in [Configure inbound and outbound network traffic](#).

Submit the experiment to run and generate a model. With the MLClient created in the prerequisites,you can run the following command in the workspace.

```
# Submit the AutoML job
returned_job = ml_client.jobs.create_or_update(
    classification_job
) # submit the job to the backend

print(f"Created job: {returned_job}")

# Get a URL for the status of the job
returned_job.services["Studio"].endpoint
```

Multiple child runs on clusters

Automated ML experiment child runs can be performed on a cluster that is already running another experiment. However, the timing depends on how many nodes the cluster has, and if those nodes are available to run a different experiment.

Each node in the cluster acts as an individual virtual machine (VM) that can accomplish a single training run; for automated ML this means a child run. If all the nodes are busy, the new experiment is queued. But if there are free nodes, the new experiment will run automated ML child runs in parallel in the available nodes/VMs.

To help manage child runs and when they can be performed, we recommend you create a dedicated cluster per

experiment, and match the number of `max_concurrent_iterations` of your experiment to the number of nodes in the cluster. This way, you use all the nodes of the cluster at the same time with the number of concurrent child runs/iterations you want.

Configure `max_concurrent_iterations` in the `.set_limits()` setter function. If it is not configured, then by default only one concurrent child run/iteration is allowed per experiment. In case of compute instance, `max_concurrent_trials` can be set to be the same as number of cores on the compute instance VM.

Explore models and metrics

Automated ML offers options for you to monitor and evaluate your training results.

- For definitions and examples of the performance charts and metrics provided for each run, see [Evaluate automated machine learning experiment results](#).
- To get a featurization summary and understand what features were added to a particular model, see [Featurization transparency](#).

From Azure Machine Learning UI at the model's page you can also view the hyperparameters used when training a particular model and also view and customize the internal model's training code used.

Register and deploy models

After you test a model and confirm you want to use it in production, you can register it for later use.

TIP

For registered models, one-click deployment is available via the [Azure Machine Learning studio](#). See [how to deploy registered models from the studio](#).

AutoML in pipelines

To leverage AutoML in your MLOps workflows, you can add AutoML Job steps to your [AzureML Pipelines](#). This allows you to automate your entire workflow by hooking up your data prep scripts to AutoML and then registering and validating the resulting best model.

Below is a [sample pipeline](#) with an AutoML classification component and a command component that shows the resulting AutoML output. Note how the inputs (training & validation data) and the outputs (best model) are referenced in different steps.

```

# Define pipeline
@pipeline(
    description="AutoML Classification Pipeline",
)
def automl_classification(
    classification_train_data,
    classification_validation_data
):
    # define the automl classification task with automl function
    classification_node = classification(
        training_data=classification_train_data,
        validation_data=classification_validation_data,
        target_column_name="y",
        primary_metric="accuracy",
        # currently need to specify outputs "mlflow_model" explicitly to reference it in following nodes
        outputs={"best_model": Output(type="mlflow_model")},
    )
    # set limits and training
    classification_node.set_limits(max_trials=1)
    classification_node.set_training(enable_stack_ensemble=False, enable_vote_ensemble=False)

    command_func = command(
        inputs=dict(
            automl_output=Input(type="mlflow_model")
        ),
        command="ls ${inputs.automl_output}",
        environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:1"
    )
    show_output = command_func(automl_output=classification_node.outputs.best_model)

pipeline_classification = automl_classification(
    classification_train_data=Input(path=".//training-mltable-folder/", type="mltable"),
    classification_validation_data=Input(path=".//validation-mltable-folder/", type="mltable"),
)
# ...
# Note that the above is only a snippet from the bankmarketing example you can find in our examples repo ->
# https://github.com/Azure/azureml-examples/tree/sdk-preview/sdk/jobs/pipelines/1h_automl_in_pipeline/automl-
# classification-bankmarketing-in-pipeline

```

For more examples on how to do include AutoML in your pipelines, please check out our [examples repo](#).

Next steps

- Learn more about [how and where to deploy a model](#).

Set up no-code AutoML training with the studio UI

9/22/2022 • 15 minutes to read • [Edit Online](#)

In this article, you learn how to set up AutoML training jobs without a single line of code using Azure Machine Learning automated ML in the [Azure Machine Learning studio](#).

Automated machine learning, AutoML, is a process in which the best machine learning algorithm to use for your specific data is selected for you. This process enables you to generate machine learning models quickly. [Learn more about how Azure Machine Learning implements automated machine learning](#).

For an end to end example, try the [Tutorial: AutoML- train no-code classification models](#).

For a Python code-based experience, [configure your automated machine learning experiments](#) with the Azure Machine Learning SDK.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- An Azure Machine Learning workspace. See [Create workspace resources](#).

Get started

1. Sign in to [Azure Machine Learning studio](#).
2. Select your subscription and workspace.
3. Navigate to the left pane. Select **Automated ML** under the **Author** section.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The top navigation bar includes links for Microsoft, my-ws, and Automated ML, along with various icons for settings and help. The left sidebar, titled 'Author', has a tree view with 'Microsoft' expanded, showing 'New', 'Home', 'Notebooks', and 'Automated ML'. The 'Automated ML' item is highlighted with a red box. The main content area is titled 'Automated ML' and contains instructions: 'Let Automated ML train and find the best model based on your data without writing a single line of code.' It features a button labeled '+ New Automated ML job' with a red box around it, and a 'Refresh' button. Below this, a message says 'No recent Automated ML jobs to display.' and 'Click "New Automated ML job" to create your first job.' A link 'Learn more about creating Automated ML jobs' is also present. At the bottom, there's a 'Documentation' section with links to 'Concept: What is Automated ML?', 'Tutorial: Create your first classification model with Automated ML', and 'Blog: Build more accurate forecasts with new capabilities in Automated ML'. A 'View all documentation' link is at the top right of this section.

If this is your first time doing any experiments, you'll see an empty list and links to documentation.

Otherwise, you'll see a list of your recent automated ML experiments, including those created with the SDK.

Create and run experiment

1. Select + **New automated ML job** and populate the form.
2. Select a data asset from your storage container, or create a new data asset. Data asset can be created from local files, web urls, datastores, or Azure open datasets. Learn more about [data asset creation](#).

IMPORTANT

Requirements for training data:

- Data must be in tabular form.
- The value you want to predict (target column) must be present in the data.

- a. To create a new dataset from a file on your local computer, select + **Create dataset** and then select **From local file**.
- b. In the **Basic info** form, give your dataset a unique name and provide an optional description.
- c. Select **Next** to open the **Datastore and file selection form**. On this form you select where to upload your dataset; the default storage container that's automatically created with your workspace, or choose a storage container that you want to use for the experiment.
 - a. If your data is behind a virtual network, you need to enable the **skip the validation** function to ensure that the workspace can access your data. For more information, see [Use Azure Machine Learning studio in an Azure virtual network](#).
- d. Select **Browse** to upload the data file for your dataset.
- e. Review the **Settings and preview** form for accuracy. The form is intelligently populated based on the file type.

FIELD	DESCRIPTION
File format	Defines the layout and type of data stored in a file.
Delimiter	One or more characters for specifying the boundary between separate, independent regions in plain text or other data streams.
Encoding	Identifies what bit to character schema table to use to read your dataset.
Column headers	Indicates how the headers of the dataset, if any, will be treated.
Skip rows	Indicates how many, if any, rows are skipped in the dataset.

Select **Next**.

- f. The **Schema** form is intelligently populated based on the selections in the **Settings and preview** form. Here configure the data type for each column, review the column names, and select which columns to **Not include** for your experiment.

Select **Next**.

g. The **Confirm details** form is a summary of the information previously populated in the **Basic info** and **Settings and preview** forms. You also have the option to create a data profile for your dataset using a profiling enabled compute. Learn more about [data profiling](#).

Select **Next**.

3. Select your newly created dataset once it appears. You are also able to view a preview of the dataset and sample statistics.
4. On the **Configure job** form, select **Create new** and enter **Tutorial-automl-deploy** for the experiment name.
5. Select a target column; this is the column that you would like to do predictions on.
6. Select a compute type for the data profiling and training job. You can select a [compute cluster](#) or [compute instance](#).
7. Select a compute from the dropdown list of your existing computes. To create a new compute, follow the instructions in step 8.
8. Select **Create a new compute** to configure your compute context for this experiment.

FIELD	DESCRIPTION
Compute name	Enter a unique name that identifies your compute context.
Virtual machine priority	Low priority virtual machines are cheaper but don't guarantee the compute nodes.
Virtual machine type	Select CPU or GPU for virtual machine type.
Virtual machine size	Select the virtual machine size for your compute.
Min / Max nodes	To profile data, you must specify 1 or more nodes. Enter the maximum number of nodes for your compute. The default is 6 nodes for an AzureML Compute.
Advanced settings	These settings allow you to configure a user account and existing virtual network for your experiment.

Select **Create**. Creation of a new compute can take a few minutes.

NOTE

Your compute name will indicate if the compute you select/create is *profiling enabled*. (See the section [data profiling](#) for more details).

Select **Next**.

9. On the **Task type and settings** form, select the task type: classification, regression, or forecasting. See [supported task types](#) for more information.
 - a. For **classification**, you can also enable deep learning.

If deep learning is enabled, validation is limited to *train_validation split*. Learn more about [validation options](#).

- b. For **forecasting** you can,
- Enable deep learning.
 - Select *time column*: This column contains the time data to be used.
 - Select *forecast horizon*: Indicate how many time units (minutes/hours/days/weeks/months/years) will the model be able to predict to the future. The further the model is required to predict into the future, the less accurate it becomes. [Learn more about forecasting and forecast horizon](#).
10. (Optional) View addition configuration settings: additional settings you can use to better control the training job. Otherwise, defaults are applied based on experiment selection and data.
- | ADDITIONAL CONFIGURATIONS | DESCRIPTION |
|---------------------------|--|
| Primary metric | Main metric used for scoring your model. Learn more about model metrics . |
| Explain best model | Select to enable or disable, in order to show explanations for the recommended best model. This functionality is not currently available for certain forecasting algorithms . |
| Blocked algorithm | Select algorithms you want to exclude from the training job.

Allowing algorithms is only available for SDK experiments . See the supported algorithms for each task type . |
| Exit criterion | When any of these criteria are met, the training job is stopped.
<i>Training job time (hours)</i> : How long to allow the training job to run.
<i>Metric score threshold</i> : Minimum metric score for all pipelines. This ensures that if you have a defined target metric you want to reach, you do not spend more time on the training job than necessary. |
| Concurrency | <i>Max concurrent iterations</i> : Maximum number of pipelines (iterations) to test in the training job. The job will not run more than the specified number of iterations. Learn more about how automated ML performs multiple child jobs on clusters . |
11. (Optional) View featurization settings: if you choose to enable **Automatic featurization** in the **Additional configuration settings** form, default featurization techniques are applied. In the **View featurization settings** you can change these defaults and customize accordingly. Learn how to [customize featurizations](#).

Select task type

Select the machine learning task type for the experiment. Additional settings are available to fine tune the experiment if needed.

Classification

-  To predict one of several categories in the target column. 
yes/no, blue, red, green.

- Enable deep learning (preview) 

Regression

-  To predict continuous numeric values

Time series forecasting

- To predict values based on time

 [View additional configuration settings](#)

 [View featurization settings](#)

[Back](#)

[Finish](#)

[Cancel](#)

12. The [Optional] Validate and test form allows you to do the following.

- a. Specify the type of validation to be used for your training job. [Learn more about cross validation](#).
 - a. Forecasting tasks only supports k-fold cross validation.
- b. Provide a test dataset (preview) to evaluate the recommended model that automated ML generates for you at the end of your experiment. When you provide test data, a test job is automatically triggered at the end of your experiment. This test job is only job on the best model that was recommended by automated ML. Learn how to get the [results of the remote test job](#).

IMPORTANT

Providing a test dataset to evaluate generated models is a preview feature. This capability is an [experimental](#) preview feature, and may change at any time.

- Test data is considered a separate from training and validation, so as to not bias the results of the test job of the recommended model. [Learn more about bias during model validation](#).
- You can either provide your own test dataset or opt to use a percentage of your training dataset. Test data must be in the form of an [Azure Machine Learning TabularDataset](#).
- The schema of the test dataset should match the training dataset. The target column is optional, but if no target column is indicated no test metrics are calculated.
- The test dataset should not be the same as the training dataset or the validation dataset.
- Forecasting jobs do not support train/test split.

Create a new Automated ML run

[Optional] Select the validation and test type
You can choose a validation type and select a test dataset as an optional step. Providing your own validation and test datasets are currently preview features.

Validation type ⓘ
Auto

Test dataset (preview) ⓘ
No test dataset required

Back **Finish** **Cancel**

Customize featurization

In the **Featurization** form, you can enable/disable automatic featurization and customize the automatic featurization settings for your experiment. To open this form, see step 10 in the [Create and run experiment](#) section.

The following table summarizes the customizations currently available via the studio.

COLUMN	CUSTOMIZATION
Included	Specifies which columns to include for training.
Feature type	Change the value type for the selected column.
Impute with	Select what value to impute missing values with in your data.

Featurization X

Feature selection identifies the actions performed on the dataset to prepare the data for training. This will not impact the input data needed for inferencing i.e., if columns are excluded from training, the excluded columns will still be required as input for inferencing on the model. [Learn more about Automated ML's featurization](#)

Enable featurization

Column name	Included	Feature type	Impute with	Data example
instant	<input checked="" type="checkbox"/>	Auto	Auto	1, 2, 3
date	<input checked="" type="checkbox"/>	Auto	Auto	2011-01-01 00:00:00, 2011
season	<input checked="" type="checkbox"/>	Auto	Auto	1, 1, 1
yr	<input checked="" type="checkbox"/>	Auto	Auto	0, 0, 0
mnth	<input checked="" type="checkbox"/>	Auto	Auto	1, 1, 1

Save

Run experiment and view results

Select **Finish** to run your experiment. The experiment preparing process can take up to 10 minutes. Training

jobs can take an additional 2-3 minutes more for each pipeline to finish running.

NOTE

The algorithms automated ML employs have inherent randomness that can cause slight variation in a recommended model's final metrics score, like accuracy. Automated ML also performs operations on data such as train-test split, train-validation split or cross-validation when necessary. So if you run an experiment with the same configuration settings and primary metric multiple times, you'll likely see variation in each experiments final metrics score due to these factors.

View experiment details

The **Job Detail** screen opens to the **Details** tab. This screen shows you a summary of the experiment job including a status bar at the top next to the job number.

The **Models** tab contains a list of the models created ordered by the metric score. By default, the model that scores the highest based on the chosen metric is at the top of the list. As the training job tries out more models, they are added to the list. Use this to get a quick comparison of the metrics for the models produced so far.

The screenshot shows the 'Bank class (parent)' experiment details. The left pane displays general experiment metadata: Status (Completed), Created (Oct 27, 2021 3:40 PM), Started (Oct 27, 2021 3:40 PM), Duration (33m 18.08s), Compute duration (33m 18.08s), Compute target (test-compute), Run ID (AutoML_e1d071ab-5336-46b1-9cbc-038932134f46), Script name (--), and users (Nina Baccam). Input datasets are listed as 'Input name: training_data, Dataset: bank marketing: Version 1'. Output datasets are listed as 'None'. The right pane shows the 'Best model summary' which includes the algorithm name (VotingEnsemble), ensemble details (View ensemble details), AUC weighted (0.94836, View all other metrics), Sampling (100.00 %), Registered models (No registration yet), and Deploy status (No deployment yet). The 'Run summary' section lists Task type (Classification, View configuration settings), Featureization (Auto), Primary metric (AUC weighted), and Experiment name (new-test).

View training job details

Drill down on any of the completed models to see training job details. On the **Model** tab view details like a model summary and the hyperparameters used for the selected model.

... > my-ws > Jobs > automl > joyful_boniato_1pntr7zx > witty_button_m517m8xj

witty_button_m517m8xj

Refresh Deploy Download Explain model Cancel Delete

Details **Model** Explanations (preview) Metrics Data transformation (preview) Outputs + logs ...

Model summary

Algorithm name
StandardScalerWrapper, XGBoostClassifier

Hyperparameters
[View hyperparameters](#)

AUC weighted
0.94667 [View all other metrics](#)

Sampling
100.00 %

Registered models
No registration yet

Deploy status
No deployment yet

You can also see model specific performance metric charts on the **Metrics** tab. [Learn more about charts.](#)

Run 5

Refresh Deploy Download Explain model Cancel

Details Model Explanations (preview) **Metrics** Outputs + logs Images Child runs Snapshot

Select a metric to see a visualization or table of the data.

View as: Chart Table

confusion_matrix

accuracy_table

weighted_accuracy

precision_score_weighted

precision_score_micro

recall_score_weighted

average_precision_score_micro

average_precision_score_weighted

AUC_weighted

AUC_macro

recall_score_macro

Confusion Matrix

		Predicted Label	
		no	yes
True Label	no	28202	1056
	yes	1910	1782

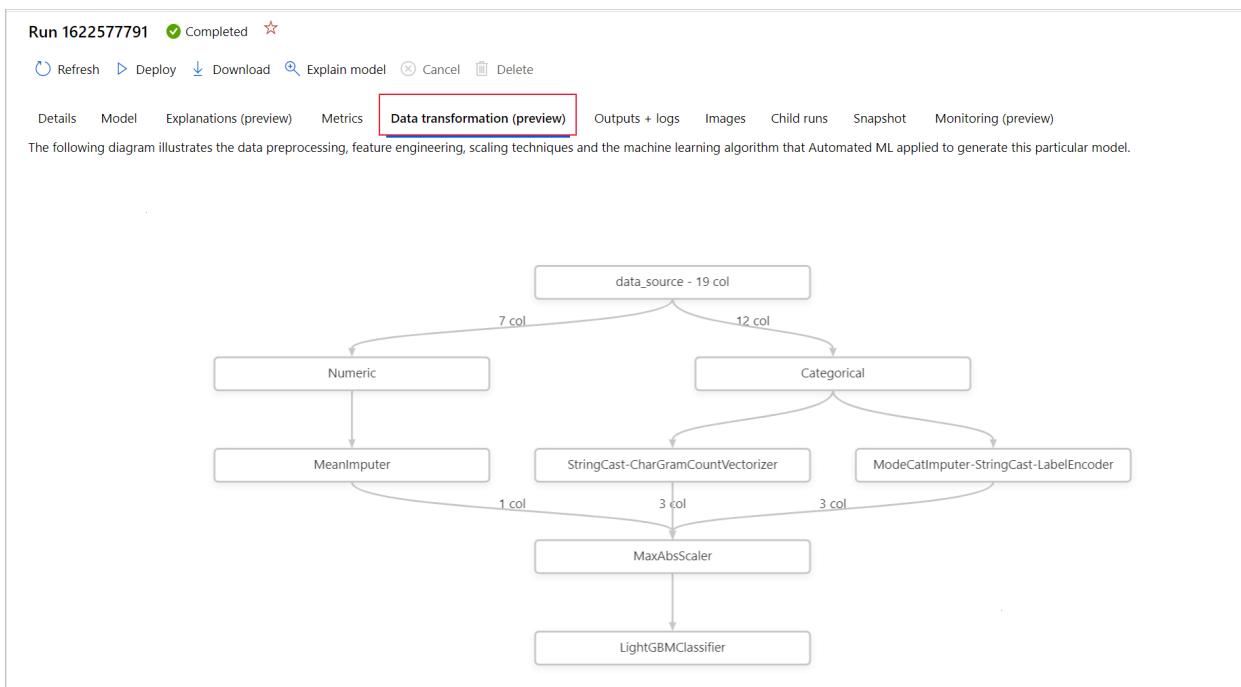
Raw Confusion Matrix

20k
10k

On the Data transformation tab, you can see a diagram of what data preprocessing, feature engineering, scaling techniques and the machine learning algorithm that were applied to generate this model.

IMPORTANT

The Data transformation tab is in preview. This capability should be considered [experimental](#) and may change at any time.



View remote test job results (preview)

If you specified a test dataset or opted for a train/test split during your experiment setup-- on the **Validate and test** form, automated ML automatically tests the recommended model by default. As a result, automated ML calculates test metrics to determine the quality of the recommended model and its predictions.

IMPORTANT

Testing your models with a test dataset to evaluate generated models is a preview feature. This capability is an [experimental](#) preview feature, and may change at any time.

WARNING

This feature is not available for the following automated ML scenarios

- Computer vision tasks (preview)
- Many models and hierarchical time series forecasting training (preview)
- Forecasting tasks where deep learning neural networks (DNN) are enabled
- Automated ML jobs from local computers or Azure Databricks clusters

To view the test job metrics of the recommended model,

1. Navigate to the **Models** page, select the best model.
2. Select the **Test results (preview)** tab.
3. Select the job you want, and view the **Metrics** tab.

The screenshot shows the Azure Machine Learning studio interface. At the top, there are several navigation links: Refresh, Deploy, Download, Explain model, View generated code, Test model (preview), Cancel, and Delete. Below these are tabs for Details, Model, Explanations (preview), Metrics, Data transformation (preview), Test results (preview) (which is highlighted with a red border), Outputs + logs, and Images. A message states: "Testing your model gives you the opportunity to see how your model performs before deployment. Whenever you test your model, your results will be displayed here." A box highlights the "Algorithm name: VotingEnsemble". Below this, a table lists a single row of test results:

Display name	Normalized root mean s...	Dataset used	Created
funny_ring_w2p9k58w	0.36621	20521ab4-7199-4cf0-96bb-eb57f1e1df40	Sep 27, 2021 5:16 PM

To view the test predictions used to calculate the test metrics,

1. Navigate to the bottom of the page and select the link under **Outputs dataset** to open the dataset.
2. On the **Datasets** page, select the **Explore** tab to view the predictions from the test job.
 - a. Alternatively, the prediction file can also be viewed/downloaded from the **Outputs + logs** tab, expand the **Predictions** folder to locate your `predicted.csv` file.

Alternatively, the predictions file can also be viewed/downloaded from the **Outputs + logs** tab, expand **Predictions** folder to locate your `predictions.csv` file.

The model test job generates the `predictions.csv` file that's stored in the default datastore created with the workspace. This datastore is visible to all users with the same subscription. Test jobs are not recommended for scenarios if any of the information used for or created by the test job needs to remain private.

Test an existing automated ML model (preview)

IMPORTANT

Testing your models with a test dataset to evaluate generated models is a preview feature. This capability is an [experimental](#) preview feature, and may change at any time.

WARNING

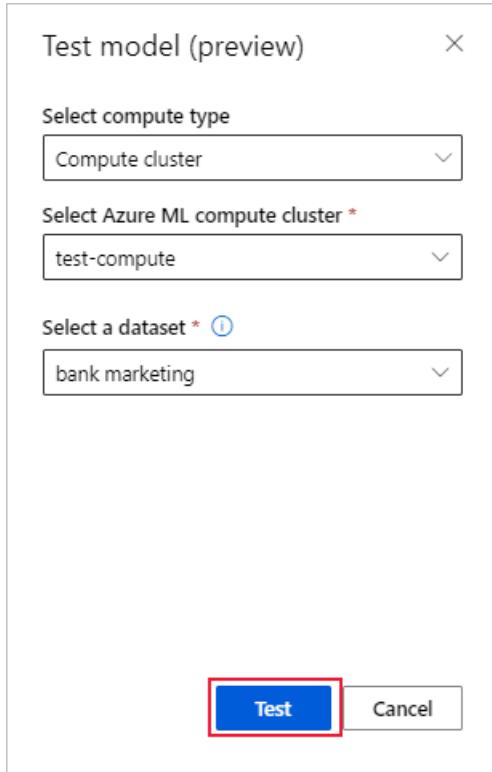
This feature is not available for the following automated ML scenarios

- Computer vision tasks (preview)
- Many models and hierarchical time series forecasting training (preview)
- Forecasting tasks where deep learning neural networks (DNN) are enabled
- Automated ML runs from local computes or Azure Databricks clusters

After your experiment completes, you can test the model(s) that automated ML generates for you. If you want to test a different automated ML generated model, not the recommended model, you can do so with the following steps.

1. Select an existing automated ML experiment job.
2. Navigate to the **Models** tab of the job and select the completed model you want to test.

3. On the model **Details** page, select the **Test model(preview)** button to open the **Test model** pane.
4. On the **Test model** pane, select the compute cluster and a test dataset you want to use for your test job.
5. Select the **Test** button. The schema of the test dataset should match the training dataset, but the **target column** is optional.
6. Upon successful creation of model test job, the **Details** page displays a success message. Select the **Test results** tab to see the progress of the job.
7. To view the results of the test job, open the **Details** page and follow the steps in the [view results of the remote test job](#) section.



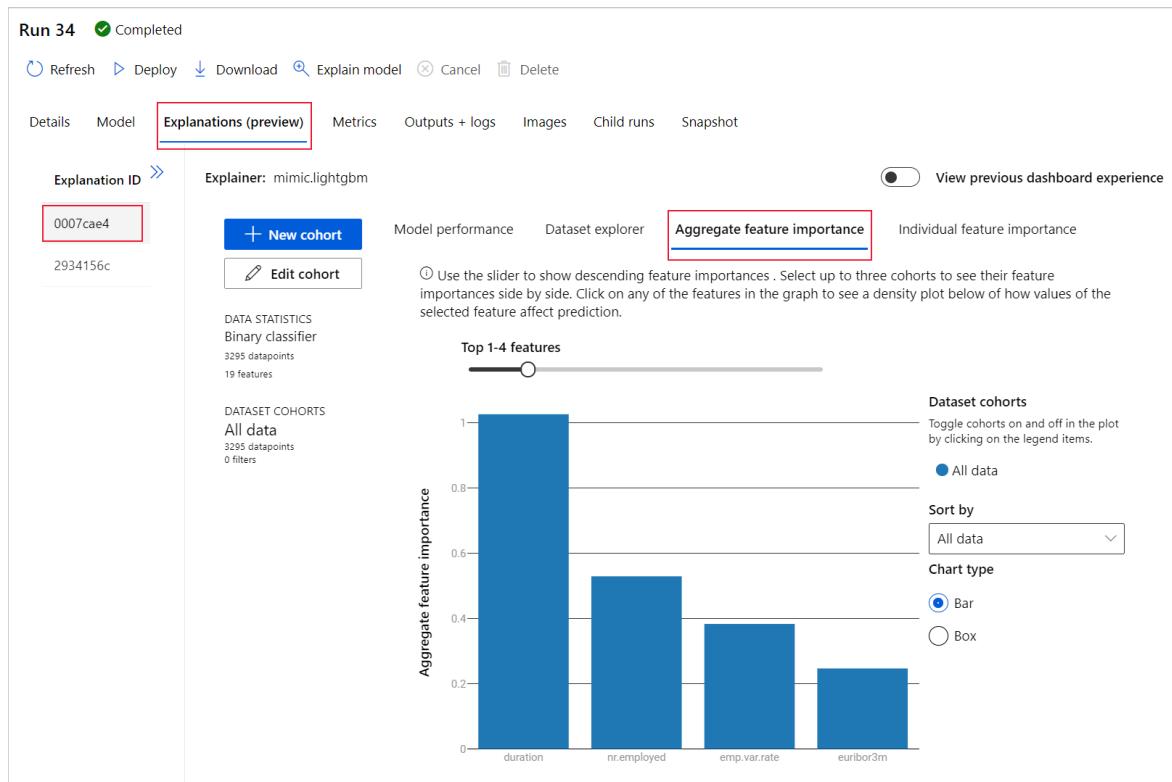
Model explanations (preview)

To better understand your model, you can see which data features (raw or engineered) influenced the model's predictions with the model explanations dashboard.

The model explanations dashboard provides an overall analysis of the trained model along with its predictions and explanations. It also lets you drill into an individual data point and its individual feature importance. [Learn more about the explanation dashboard visualizations](#).

To get explanations for a particular model,

1. On the **Models** tab, select the model you want to understand.
2. Select the **Explain model** button, and provide a compute that can be used to generate the explanations.
3. Check the **Child jobs** tab for the status.
4. Once complete, navigate to the **Explanations (preview)** tab which contains the explanations dashboard.



Edit and submit jobs (preview)

IMPORTANT

The ability to copy, edit and submit a new experiment based on an existing experiment is a preview feature. This capability is an [experimental](#) preview feature, and may change at any time.

In scenarios where you would like to create a new experiment based on the settings of an existing experiment, automated ML provides the option to do so with the **Edit and submit** button in the studio UI.

This functionality is limited to experiments initiated from the studio UI and requires the data schema for the new experiment to match that of the original experiment.

The **Edit and submit** button opens the **Create a new Automated ML job** wizard with the data, compute and experiment settings pre-populated. You can go through each form and edit selections as needed for your new experiment.

Deploy your model

Once you have the best model at hand, it is time to deploy it as a web service to predict on new data.

TIP

If you are looking to deploy a model that was generated via the `automl` package with the Python SDK, you must [register your model](#) to the workspace.

Once your model is registered, find it in the studio by selecting **Models** on the left pane. Once you open your model, you can select the **Deploy** button at the top of the screen, and then follow the instructions as described in [step 2](#) of the [Deploy your model](#) section.

Automated ML helps you with deploying the model without writing code:

1. You have a couple options for deployment.

- Option 1: Deploy the best model, according to the metric criteria you defined.
 - After the experiment is complete, navigate to the parent job page by selecting **Job 1** at the top of the screen.
 - Select the model listed in the **Best model summary** section.
 - Select **Deploy** on the top left of the window.
- Option 2: To deploy a specific model iteration from this experiment.
 - Select the desired model from the **Models** tab
 - Select **Deploy** on the top left of the window.

2. Populate the **Deploy model** pane.

FIELD	VALUE
Name	Enter a unique name for your deployment.
Description	Enter a description to better identify what this deployment is for.
Compute type	Select the type of endpoint you want to deploy: Azure Kubernetes Service (AKS) or Azure Container Instance (ACI) .
Compute name	<i>Applies to AKS only:</i> Select the name of the AKS cluster you wish to deploy to.
Enable authentication	Select to allow for token-based or key-based authentication.
Use custom deployment assets	Enable this feature if you want to upload your own scoring script and environment file. Otherwise, automated ML provides these assets for you by default. Learn more about scoring scripts .

IMPORTANT

File names must be under 32 characters and must begin and end with alphanumerics. May include dashes, underscores, dots, and alphanumerics between. Spaces are not allowed.

The *Advanced* menu offers default deployment features such as [data collection](#) and resource utilization settings. If you wish to override these defaults do so in this menu.

3. Select **Deploy**. Deployment can take about 20 minutes to complete. Once deployment begins, the **Model summary** tab appears. See the deployment progress under the **Deploy status** section.

Now you have an operational web service to generate predictions! You can test the predictions by querying the service from [Power BI's built in Azure Machine Learning support](#).

Next steps

- [Learn how to consume a web service](#).
- [Understand automated machine learning results](#).
- [Learn more about automated machine learning](#) and Azure Machine Learning.

Set up a development environment with Azure Databricks and AutoML in Azure Machine Learning

9/22/2022 • 4 minutes to read • [Edit Online](#)

Learn how to configure a development environment in Azure Machine Learning that uses Azure Databricks and automated ML.

Azure Databricks is ideal for running large-scale intensive machine learning workflows on the scalable Apache Spark platform in the Azure cloud. It provides a collaborative Notebook-based environment with a CPU or GPU-based compute cluster.

For information on other machine learning development environments, see [Set up Python development environment](#).

Prerequisite

Azure Machine Learning workspace. To create one, use the steps in the [Create workspace resources](#) article.

Azure Databricks with Azure Machine Learning and AutoML

Azure Databricks integrates with Azure Machine Learning and its AutoML capabilities.

You can use Azure Databricks:

- To train a model using Spark MLlib and deploy the model to ACI/AKS.
- With [automated machine learning](#) capabilities using an Azure ML SDK.
- As a compute target from an [Azure Machine Learning pipeline](#).

Set up a Databricks cluster

Create a [Databricks cluster](#). Some settings apply only if you install the SDK for automated machine learning on Databricks.

It takes few minutes to create the cluster.

Use these settings:

SETTING	APPLIES TO	VALUE
Cluster Name	always	yourclustername
Databricks Runtime Version	always	9.1 LTS
Python version	always	3
Worker Type (determines max # of concurrent iterations)	Automated ML only	Memory optimized VM preferred
Workers	always	2 or higher

SETTING	APPLIES TO	VALUE
Enable Autoscaling	Automated ML only	Uncheck

Wait until the cluster is running before proceeding further.

Add the Azure ML SDK to Databricks

Once the cluster is running, [create a library](#) to attach the appropriate Azure Machine Learning SDK package to your cluster.

To use automated ML, skip to [Add the Azure ML SDK with AutoML](#).

1. Right-click the current Workspace folder where you want to store the library. Select **Create > Library**.

TIP

If you have an old SDK version, deselect it from cluster's installed libraries and move to trash. Install the new SDK version and restart the cluster. If there is an issue after the restart, detach and reattach your cluster.

2. Choose the following option (no other SDK installations are supported)

SDK PACKAGE EXTRAS	SOURCE	PYPI NAME
For Databricks	Upload Python Egg or PyPI	azureml-sdk[databricks]

WARNING

No other SDK extras can be installed. Choose only the [`databricks`] option .

- Do not select **Attach automatically to all clusters**.
- Select **Attach** next to your cluster name.

3. Monitor for errors until status changes to **Attached**, which may take several minutes. If this step fails:

Try restarting your cluster by:

- In the left pane, select **Clusters**.
- In the table, select your cluster name.
- On the **Libraries** tab, select **Restart**.

A successful install looks like the following:

Name	Type	Status	Source
azureml-sdk[databricks]	PyPI	Installed	

Add the Azure ML SDK with AutoML to Databricks

If the cluster was created with Databricks Runtime 7.3 LTS (*not* ML), run the following command in the first cell of your notebook to install the AzureML SDK.

```
%pip install --upgrade --force-reinstall -r https://aka.ms/automl_linux_requirements.txt
```

AutoML config settings

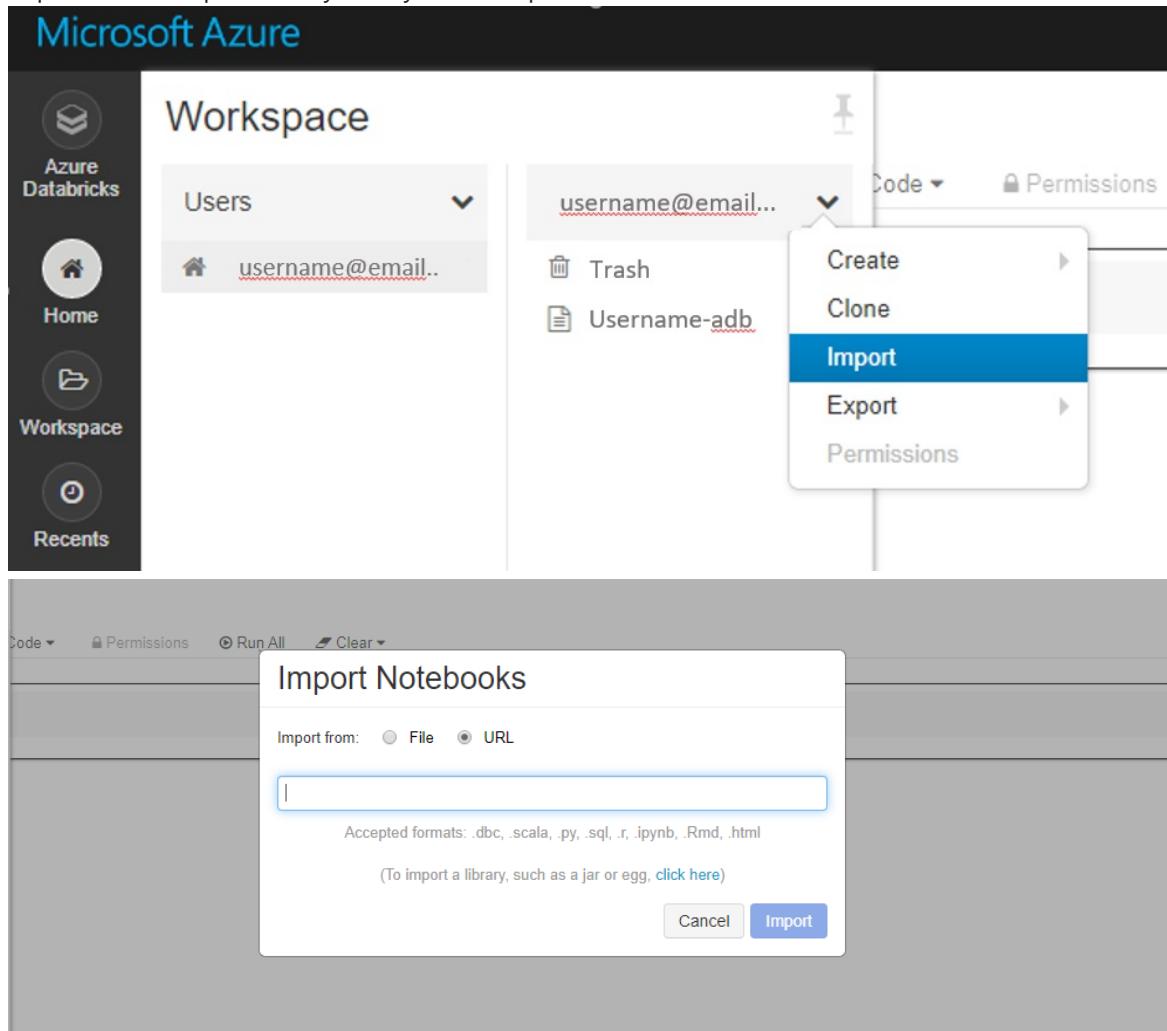
In AutoML config, when using Azure Databricks add the following parameters:

- `max_concurrent_iterations` is based on number of worker nodes in your cluster.
- `spark_context=sc` is based on the default spark context.

ML notebooks that work with Azure Databricks

Try it out:

- While many sample notebooks are available, only [these sample notebooks](#) work with Azure Databricks.
- Import these samples directly from your workspace. See below:



- Learn how to [create a pipeline with Databricks as the training compute](#).

Troubleshooting

- **Databricks cancel an automated machine learning run:** When you use automated machine

learning capabilities on Azure Databricks, to cancel a run and start a new experiment run, restart your Azure Databricks cluster.

- **Databricks >10 iterations for automated machine learning:** In automated machine learning settings, if you have more than 10 iterations, set `show_output` to `False` when you submit the run.
- **Databricks widget for the Azure Machine Learning SDK and automated machine learning:** The Azure Machine Learning SDK widget isn't supported in a Databricks notebook because the notebooks can't parse HTML widgets. You can view the widget in the portal by using this Python code in your Azure Databricks notebook cell:

```
displayHTML("<a href={} target='_blank'>Azure Portal: {}</a>".format(local_run.get_portal_url(),  
local_run.id))
```

- **Failure when installing packages**

Azure Machine Learning SDK installation fails on Azure Databricks when more packages are installed. Some packages, such as `psutil`, can cause conflicts. To avoid installation errors, install packages by freezing the library version. This issue is related to Databricks and not to the Azure Machine Learning SDK. You might experience this issue with other libraries, too. Example:

```
psutil cryptography==1.5 pyopenssl==16.0.0 ipython==2.2.0
```

Alternatively, you can use init scripts if you keep facing install issues with Python libraries. This approach isn't officially supported. For more information, see [Cluster-scoped init scripts](#).

- **Import error: cannot import name `Timedelta` from `pandas._libs.tslibs`:** If you see this error when you use automated machine learning, run the two following lines in your notebook:

```
%sh rm -rf /databricks/python/lib/python3.7/site-packages/pandas-0.23.4.dist-info  
/databricks/python/lib/python3.7/site-packages/pandas  
%sh /databricks/python/bin/pip install pandas==0.23.4
```

- **Import error: No module named 'pandas.core.indexes':** If you see this error when you use automated machine learning:

1. Run this command to install two packages in your Azure Databricks cluster:

```
scikit-learn==0.19.1  
pandas==0.22.0
```

2. Detach and then reattach the cluster to your notebook.

If these steps don't solve the issue, try restarting the cluster.

- **FailToSendFeather:** If you see a `FailToSendFeather` error when reading data on Azure Databricks cluster, refer to the following solutions:
 - Upgrade `azureml-sdk[automl]` package to the latest version.
 - Add `azureml-dataprep` version 1.1.8 or above.
 - Add `pyarrow` version 0.11 or above.

Next steps

- [Train and deploy a model](#) on Azure Machine Learning with the MNIST dataset.

- See the [Azure Machine Learning SDK for Python](#) reference.

Set up AutoML to train a time-series forecasting model with Python

9/22/2022 • 17 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, you learn how to set up AutoML training for time-series forecasting models with Azure Machine Learning automated ML in the [Azure Machine Learning Python SDK](#).

To do so, you:

- Prepare data for time series modeling.
- Configure specific time-series parameters in an `AutoMLConfig` object.
- Run predictions with time-series data.

For a low code experience, see the [Tutorial: Forecast demand with automated machine learning](#) for a time-series forecasting example using automated ML in the [Azure Machine Learning studio](#).

Unlike classical time series methods, in automated ML, past time-series values are "pivoted" to become additional dimensions for the regressor together with other predictors. This approach incorporates multiple contextual variables and their relationship to one another during training. Since multiple factors can influence a forecast, this method aligns itself well with real world forecasting scenarios. For example, when forecasting sales, interactions of historical trends, exchange rate, and price all jointly drive the sales outcome.

Prerequisites

For this article you need,

- An Azure Machine Learning workspace. To create the workspace, see [Create workspace resources](#).
- This article assumes some familiarity with setting up an automated machine learning experiment. Follow the [how-to](#) to see the main automated machine learning experiment design patterns.

IMPORTANT

The Python commands in this article require the latest `azureml-train-automl` package version.

- [Install the latest `azureml-train-automl` package to your local environment](#).
- For details on the latest `azureml-train-automl` package, see the [release notes](#).

Training and validation data

The most important difference between a forecasting regression task type and regression task type within automated ML is including a feature in your training data that represents a valid time series. A regular time series has a well-defined and consistent frequency and has a value at every sample point in a continuous time span.

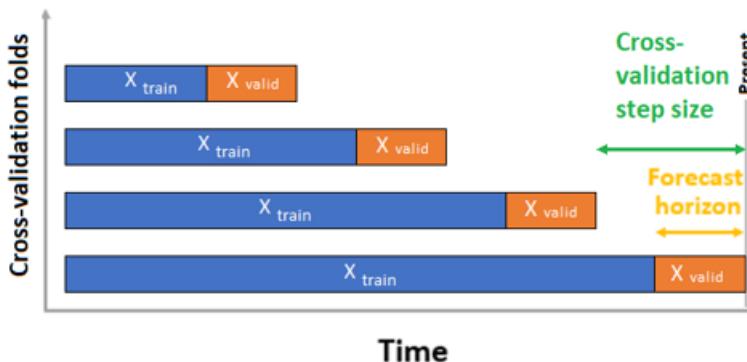
IMPORTANT

When training a model for forecasting future values, ensure all the features used in training can be used when running predictions for your intended horizon.

For example, when creating a demand forecast, including a feature for current stock price could massively increase training accuracy. However, if you intend to forecast with a long horizon, you may not be able to accurately predict future stock values corresponding to future time-series points, and model accuracy could suffer.

You can specify separate [training data and validation data](#) directly in the `AutoMLConfig` object. Learn more about the [AutoMLConfig](#).

For time series forecasting, only **Rolling Origin Cross Validation (ROCV)** is used for validation by default. ROCV divides the series into training and validation data using an origin time point. Sliding the origin in time generates the cross-validation folds. This strategy preserves the time series data integrity and eliminates the risk of data leakage.



Pass your training and validation data as one dataset to the parameter `training_data`. Set the number of cross validation folds with the parameter `n_cross_validations` and set the number of periods between two consecutive cross-validation folds with `cv_step_size`. You can also leave either or both parameters empty and AutoML will set them automatically.

APPLIES TO: [Python SDK azureml v1](#)

```
automl_config = AutoMLConfig(task='forecasting',
                             training_data= training_data,
                             n_cross_validations="auto", # Could be customized as an integer
                             cv_step_size = "auto", # Could be customized as an integer
                             ...
                             **time_series_settings)
```

You can also bring your own validation data, learn more in [Configure data splits and cross-validation in AutoML](#).

Learn more about how AutoML applies cross validation to [prevent over-fitting models](#).

Configure experiment

The `AutoMLConfig` object defines the settings and data necessary for an automated machine learning task. Configuration for a forecasting model is similar to the setup of a standard regression model, but certain models, configuration options, and featurization steps exist specifically for time-series data.

Supported models

Automated machine learning automatically tries different models and algorithms as part of the model creation and tuning process. As a user, there is no need for you to specify the algorithm. For forecasting experiments, both native time-series and deep learning models are part of the recommendation system.

TIP

Traditional regression models are also tested as part of the recommendation system for forecasting experiments. See a complete list of the [supported models](#) in the SDK reference documentation.

Configuration settings

Similar to a regression problem, you define standard training parameters like task type, number of iterations, training data, and number of cross-validations. Forecasting tasks require the `time_column_name` and `forecast_horizon` parameters to configure your experiment. If the data includes multiple time series, such as sales data for multiple stores or energy data across different states, automated ML automatically detects this and sets the `time_series_id_column_names` parameter (preview) for you. You can also include additional parameters to better configure your run, see the [optional configurations](#) section for more detail on what can be included.

IMPORTANT

Automatic time series identification is currently in public preview. This preview version is provided without a service-level agreement. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

PARAMETER NAME	DESCRIPTION
<code>time_column_name</code>	Used to specify the datetime column in the input data used for building the time series and inferring its frequency.
<code>forecast_horizon</code>	Defines how many periods forward you would like to forecast. The horizon is in units of the time series frequency. Units are based on the time interval of your training data, for example, monthly, weekly that the forecaster should predict out.

The following code,

- Leverages the `ForecastingParameters` class to define the forecasting parameters for your experiment training
- Sets the `time_column_name` to the `day_datetime` field in the data set.
- Sets the `forecast_horizon` to 50 in order to predict for the entire test set.

```
from azureml.automl.core.forecasting_parameters import ForecastingParameters

forecasting_parameters = ForecastingParameters(time_column_name='day_datetime',
                                                forecast_horizon=50,
                                                freq='W')
```

These `forecasting_parameters` are then passed into your standard `AutoMLConfig` object along with the `forecasting` task type, primary metric, exit criteria, and training data.

```

from azureml.core.workspace import Workspace
from azureml.core.experiment import Experiment
from azureml.train.automl import AutoMLConfig
import logging

automl_config = AutoMLConfig(task='forecasting',
                             primary_metric='normalized_root_mean_squared_error',
                             experiment_timeout_minutes=15,
                             enable_early_stopping=True,
                             training_data=train_data,
                             label_column_name=label,
                             n_cross_validations="auto", # Could be customized as an integer
                             cv_step_size = "auto", # Could be customized as an integer
                             enable_ensembling=False,
                             verbosity=logging.INFO,
                             forecasting_parameters=forecasting_parameters)

```

The amount of data required to successfully train a forecasting model with automated ML is influenced by the `forecast_horizon`, `n_cross_validations`, and `target_lags` or `target_rolling_window_size` values specified when you configure your `AutoMLConfig`.

The following formula calculates the amount of historic data that would be needed to construct time series features.

Minimum historic data required: $(2 \times \text{forecast_horizon}) + \# \text{n_cross_validations} + \max(\max(\text{target_lags}), \text{target_rolling_window_size})$

An `Error exception` is raised for any series in the dataset that does not meet the required amount of historic data for the relevant settings specified.

Featurization steps

In every automated machine learning experiment, automatic scaling and normalization techniques are applied to your data by default. These techniques are types of **featurization** that help *certain* algorithms that are sensitive to features on different scales. Learn more about default featurization steps in [Featurization in AutoML](#)

However, the following steps are performed only for `forecasting` task types:

- Detect time-series sample frequency (for example, hourly, daily, weekly) and create new records for absent time points to make the series continuous.
- Impute missing values in the target (via forward-fill) and feature columns (using median column values)
- Create features based on time series identifiers to enable fixed effects across different series
- Create time-based features to assist in learning seasonal patterns
- Encode categorical variables to numeric quantities
- Detect the non-stationary time series and automatically differencing them to mitigate the impact of unit roots.

To view the full list of possible engineered features generated from time series data, see [TimeIndexFeaturizer Class](#).

NOTE

Automated machine learning featurization steps (feature normalization, handling missing data, converting text to numeric, etc.) become part of the underlying model. When using the model for predictions, the same featurization steps applied during training are applied to your input data automatically.

Customize featurization

You also have the option to customize your featurization settings to ensure that the data and features that are

used to train your ML model result in relevant predictions.

Supported customizations for `forecasting` tasks include:

CUSTOMIZATION	DEFINITION
Column purpose update	Override the auto-detected feature type for the specified column.
Transformer parameter update	Update the parameters for the specified transformer. Currently supports <code>Imputer</code> (<code>fill_value</code> and <code>median</code>).
Drop columns	Specifies columns to drop from being featurized.

To customize featurizations with the SDK, specify `"featurization": FeaturizationConfig` in your `AutoMLConfig` object. Learn more about [custom featurizations](#).

NOTE

The `drop columns` functionality is deprecated as of SDK version 1.19. Drop columns from your dataset as part of data cleansing, prior to consuming it in your automated ML experiment.

```
featurization_config = FeaturizationConfig()

# `logQuantity` is a leaky feature, so we remove it.
featurization_config.drop_columns = ['logQuantity']

# Force the CPWVOL5 feature to be of numeric type.
featurization_config.add_column_purpose('CPWVOL5', 'Numeric')

# Fill missing values in the target column, Quantity, with zeroes.
featurization_config.add_transformer_params('Imputer', ['Quantity'], {"strategy": "constant", "fill_value": 0})

# Fill missing values in the `INCOME` column with median value.
featurization_config.add_transformer_params('Imputer', ['INCOME'], {"strategy": "median"})
```

If you're using the Azure Machine Learning studio for your experiment, see [how to customize featurization in the studio](#).

Optional configurations

Additional optional configurations are available for forecasting tasks, such as enabling deep learning and specifying a target rolling window aggregation. A complete list of additional parameters is available in the [ForecastingParameters SDK reference documentation](#).

Frequency & target data aggregation

Leverage the frequency, `freq`, parameter to help avoid failures caused by irregular data, that is data that doesn't follow a set cadence, like hourly or daily data.

For highly irregular data or for varying business needs, users can optionally set their desired forecast frequency, `freq`, and specify the `target_aggregation_function` to aggregate the target column of the time series. Leverage these two settings in your `AutoMLConfig` object can help save some time on data preparation.

Supported aggregation operations for target column values include:

FUNCTION	DESCRIPTION
sum	Sum of target values
mean	Mean or average of target values
min	Minimum value of a target
max	Maximum value of a target

Enable deep learning

NOTE

DNN support for forecasting in Automated Machine Learning is in [preview](#) and not supported for local runs or runs initiated in Databricks.

You can also apply deep learning with deep neural networks, DNNs, to improve the scores of your model. Automated ML's deep learning allows for forecasting univariate and multivariate time series data.

Deep learning models have three intrinsic capabilities:

1. They can learn from arbitrary mappings from inputs to outputs
2. They support multiple inputs and outputs
3. They can automatically extract patterns in input data that spans over long sequences.

To enable deep learning, set the `enable_dnn=True` in the `AutoMLConfig` object.

```
automl_config = AutoMLConfig(task='forecasting',
                             enable_dnn=True,
                             ...
                             forecasting_parameters=forecasting_parameters)
```

WARNING

When you enable DNN for experiments created with the SDK, [best model explanations](#) are disabled.

To enable DNN for an AutoML experiment created in the Azure Machine Learning studio, see the [task type settings in the studio UI how-to](#).

Target rolling window aggregation

Often the best information a forecaster can have is the recent value of the target. Target rolling window aggregations allow you to add a rolling aggregation of data values as features. Generating and using these features as extra contextual data helps with the accuracy of the train model.

For example, say you want to predict energy demand. You might want to add a rolling window feature of three days to account for thermal changes of heated spaces. In this example, create this window by setting

`target_rolling_window_size= 3` in the `AutoMLConfig` constructor.

The table shows resulting feature engineering that occurs when window aggregation is applied. Columns for **minimum**, **maximum**, and **sum** are generated on a sliding window of three based on the defined settings. Each row has a new calculated feature, in the case of the timestamp for September 8, 2017 4:00am the maximum, minimum, and sum values are calculated using the **demand values** for September 8, 2017 1:00AM

- 3:00AM. This window of three shifts along to populate data for the remaining rows.

TimeStamp	Precip	Temp	Demand	Demand_min_window3D	Demand_max_window3D	Demand_sum_window3D
2017-08-08 01:00:00	0.00	68.30	5100	nan	nan	nan
2017-08-08 02:00:00	0.00	68.44	4900	nan	nan	nan
2017-08-08 03:00:00	0.00	67.78	4800	nan	nan	nan
2017-08-08 04:00:00	0.00	68.70	4900	4800	5100	14800

View a Python code example applying the [target rolling window aggregate feature](#).

Short series handling

Automated ML considers a time series a **short series** if there are not enough data points to conduct the train and validation phases of model development. The number of data points varies for each experiment, and depends on the max_horizon, the number of cross validation splits, and the length of the model lookback, that is the maximum of history that's needed to construct the time-series features.

Automated ML offers short series handling by default with the `short_series_handling_configuration` parameter in the `ForecastingParameters` object.

To enable short series handling, the `freq` parameter must also be defined. To define an hourly frequency, we will set `freq='H'`. View the frequency string options by visiting the [pandas Time series page DataOffset objects section](#). To change the default behavior, `short_series_handling_configuration = 'auto'`, update the `short_series_handling_configuration` parameter in your `ForecastingParameter` object.

```
from azureml.automl.core.forecasting_parameters import ForecastingParameters

forecast_parameters = ForecastingParameters(time_column_name='day_datetime',
                                             forecast_horizon=50,
                                             short_series_handling_configuration='auto',
                                             freq = 'H',
                                             target_lags='auto')
```

The following table summarizes the available settings for `short_series_handling_config`.

SETTING	DESCRIPTION
<code>auto</code>	The following is the default behavior for short series handling <ul style="list-style-type: none">• If all series are short, pad the data.• If not all series are short, drop the short series.
<code>pad</code>	If <code>short_series_handling_config = pad</code> , then automated ML adds random values to each short series found. The following lists the column types and what they are padded with: <ul style="list-style-type: none">• Object columns with NaNs• Numeric columns with 0• Boolean/logic columns with False• The target column is padded with random values with mean of zero and standard deviation of 1.
<code>drop</code>	If <code>short_series_handling_config = drop</code> , then automated ML drops the short series, and it will not be used for training or prediction. Predictions for these series will return NaN's.

SETTING	DESCRIPTION
None	No series is padded or dropped

WARNING

Padding may impact the accuracy of the resulting model, since we are introducing artificial data just to get past training without failures.

If many of the series are short, then you may also see some impact in explainability results

Run the experiment

When you have your `AutoMLConfig` object ready, you can submit the experiment. After the model finishes, retrieve the best run iteration.

```
ws = Workspace.from_config()
experiment = Experiment(ws, "Tutorial-automl-forecasting")
local_run = experiment.submit(automl_config, show_output=True)
best_run, fitted_model = local_run.get_output()
```

Forecasting with best model

Use the best model iteration to forecast values for data that wasn't used to train the model.

The `forecast_quantiles()` function allows specifications of when predictions should start, unlike the `predict()` method, which is typically used for classification and regression tasks. The `forecast_quantiles()` method by default generates a point forecast or a mean/median forecast which doesn't have a cone of uncertainty around it. Learn more in the [Forecasting away from training data notebook](#).

In the following example, you first replace all values in `y_pred` with `NaN`. The forecast origin is at the end of training data in this case. However, if you replaced only the second half of `y_pred` with `NaN`, the function would leave the numerical values in the first half unmodified, but forecast the `NaN` values in the second half. The function returns both the forecasted values and the aligned features.

You can also use the `forecast_destination` parameter in the `forecast_quantiles()` function to forecast values up to a specified date.

```
label_query = test_labels.copy().astype(np.float)
label_query.fill(np.nan)
label_fcst, data_trans = fitted_model.forecast_quantiles(
    test_dataset, label_query, forecast_destination=pd.Timestamp(2019, 1, 8))
```

Often customers want to understand the predictions at a specific quantile of the distribution. For example, when the forecast is used to control inventory like grocery items or virtual machines for a cloud service. In such cases, the control point is usually something like "we want the item to be in stock and not run out 99% of the time". The following demonstrates how to specify which quantiles you'd like to see for your predictions, such as 50th or 95th percentile. If you don't specify a quantile, like in the aforementioned code example, then only the 50th percentile predictions are generated.

```
# specify which quantiles you would like
fitted_model.quantiles = [0.05,0.5, 0.9]
fitted_model.forecast_quantiles(
    test_dataset, label_query, forecast_destination=pd.Timestamp(2019, 1, 8))
```

You can calculate model metrics like, root mean squared error (RMSE) or mean absolute percentage error (MAPE) to help you estimate the models performance. See the Evaluate section of the [Bike share demand notebook](#) for an example.

After the overall model accuracy has been determined, the most realistic next step is to use the model to forecast unknown future values.

Supply a data set in the same format as the test set `test_dataset` but with future datetimes, and the resulting prediction set is the forecasted values for each time-series step. Assume the last time-series records in the data set were for 12/31/2018. To forecast demand for the next day (or as many periods as you need to forecast, `<= forecast_horizon`), create a single time series record for each store for 01/01/2019.

```
day_datetime,store,week_of_year
01/01/2019,A,1
01/01/2019,A,1
```

Repeat the necessary steps to load this future data to a dataframe and then run

```
best_run.forecast_quantiles(test_dataset)
```

to predict future values.

NOTE

In-sample predictions are not supported for forecasting with automated ML when `target_lags` and/or `target_rolling_window_size` are enabled.

Forecasting at scale

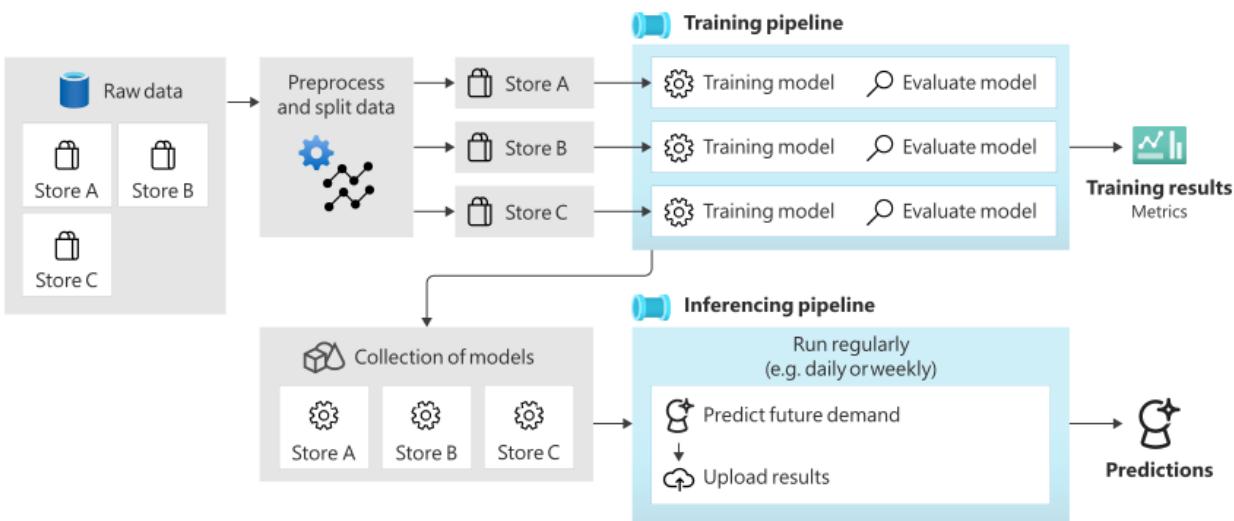
There are scenarios where a single machine learning model is insufficient and multiple machine learning models are needed. For instance, predicting sales for each individual store for a brand, or tailoring an experience to individual users. Building a model for each instance can lead to improved results on many machine learning problems.

Grouping is a concept in time series forecasting that allows time series to be combined to train an individual model per group. This approach can be particularly helpful if you have time series which require smoothing, filling or entities in the group that can benefit from history or trends from other entities. Many models and hierarchical time series forecasting are solutions powered by automated machine learning for these large scale forecasting scenarios.

Many models

The Azure Machine Learning many models solution with automated machine learning allows users to train and manage millions of models in parallel. Many models The solution accelerator leverages [Azure Machine Learning pipelines](#) to train the model. Specifically, a [Pipeline](#) object and [ParallelRunStep](#) are used and require specific configuration parameters set through the [ParallelRunConfig](#).

The following diagram shows the workflow for the many models solution.



The following code demonstrates the key parameters users need to set up their many models run. See the [Many Models- Automated ML notebook](#) for a many models forecasting example

```

from azureml.train.automl.runtime._many_models.many_models_parameters import ManyModelsTrainParameters

partition_column_names = ['Store', 'Brand']
automl_settings = {"task" : 'forecasting',
                  "primary_metric" : 'normalized_root_mean_squared_error',
                  "iteration_timeout_minutes" : 10, #This needs to be changed based on the dataset. Explore how long training is taking before setting this value
                  "iterations" : 15,
                  "experiment_timeout_hours" : 1,
                  "label_column_name" : 'Quantity',
                  "n_cross_validations" : "auto", # Could be customized as an integer
                  "cv_step_size" : "auto", # Could be customized as an integer
                  "time_column_name": 'WeekStarting',
                  "max_horizon" : 6,
                  "track_child_runs": False,
                  "pipeline_fetch_max_batch_size": 15,}

mm_params = ManyModelsTrainParameters(automl_settings=automl_settings,
                                      partition_column_names=partition_column_names)

```

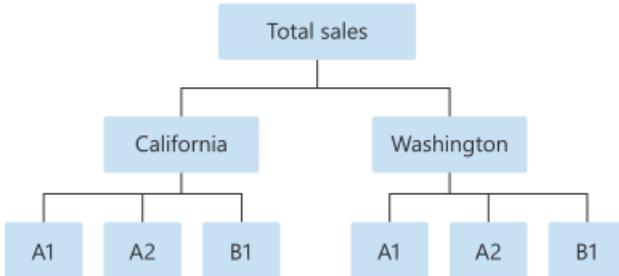
Hierarchical time series forecasting

In most applications, customers have a need to understand their forecasts at a macro and micro level of the business; whether that be predicting sales of products at different geographic locations, or understanding the expected workforce demand for different organizations at a company. The ability to train a machine learning model to intelligently forecast on hierarchy data is essential.

A hierarchical time series is a structure in which each of the unique series are arranged into a hierarchy based on dimensions such as, geography or product type. The following example shows data with unique attributes that form a hierarchy. Our hierarchy is defined by: the product type such as headphones or tablets, the product category which splits product types into accessories and devices, and the region the products are sold in.

State	Store_ID	Product_Category	SKU	Price	Quantity
California	1	A	A1	7.98	5
Washington	1	A	A1	7.92	8
California	2	A	A2	4.3	9
Washington	2	A	A2	4.37	11
California	3	B	B1	3.63	7
Washington	3	B	B1	3.73	8

To further visualize this, the leaf levels of the hierarchy contain all the time series with unique combinations of attribute values. Each higher level in the hierarchy considers one less dimension for defining the time series and aggregates each set of child nodes from the lower level into a parent node.



The hierarchical time series solution is built on top of the Many Models Solution and share a similar configuration setup.

The following code demonstrates the key parameters to set up your hierarchical time series forecasting runs. See the [Hierarchical time series- Automated ML notebook](#), for an end to end example.

```
from azureml.train.automl.runtime._hts.hts_parameters import HTSTrainParameters

model_explainability = True

engineered_explanations = False # Define your hierarchy. Adjust the settings below based on your dataset.
hierarchy = ["state", "store_id", "product_category", "SKU"]
training_level = "SKU"# Set your forecast parameters. Adjust the settings below based on your dataset.
time_column_name = "date"
label_column_name = "quantity"
forecast_horizon = 7

automl_settings = {"task" : "forecasting",
                   "primary_metric" : "normalized_root_mean_squared_error",
                   "label_column_name": label_column_name,
                   "time_column_name": time_column_name,
                   "forecast_horizon": forecast_horizon,
                   "hierarchy_column_names": hierarchy,
                   "hierarchy_training_level": training_level,
                   "track_child_runs": False,
                   "pipeline_fetch_max_batch_size": 15,
                   "model_explainability": model_explainability,# The following settings are specific to
this sample and should be adjusted according to your own needs.
                   "iteration_timeout_minutes" : 10,
                   "iterations" : 10,
                   "n_cross_validations" : "auto", # Could be customized as an integer
                   "cv_step_size" : "auto", # Could be customized as an integer
                   }

hts_parameters = HTSTrainParameters(
    automl_settings=automl_settings,
    hierarchy_column_names=hierarchy,
    training_level=training_level,
    enable_engineered_explanations=engineered_explanations
)
```

Example notebooks

See the [forecasting sample notebooks](#) for detailed code examples of advanced forecasting configuration including:

- [holiday detection and featurization](#)
- [rolling-origin cross validation](#)
- [configurable lags](#)
- [rolling window aggregate features](#)

Next steps

- Learn more about [How to deploy an AutoML model to an online endpoint](#).
- Learn about [Interpretability: model explanations in automated machine learning \(preview\)](#).

Prepare data for computer vision tasks with automated machine learning (preview)

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

IMPORTANT

Support for training computer vision models with automated ML in Azure Machine Learning is an experimental public preview feature. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you learn how to prepare image data for training computer vision models with [automated machine learning in Azure Machine Learning](#).

To generate models for computer vision tasks with automated machine learning, you need to bring labeled image data as input for model training in the form of an [MLTable](#).

You can create an [MLTable](#) from labeled training data in JSONL format. If your labeled training data is in a different format (like, pascal VOC or COCO), you can use a conversion script to first convert it to JSONL, and then create an [MLTable](#). Alternatively, you can use Azure Machine Learning's [data labeling tool](#) to manually label images, and export the labeled data to use for training your AutoML model.

Prerequisites

- Familiarize yourself with the accepted [schemas for JSONL files for AutoML computer vision experiments](#).

Get labeled data

In order to train computer vision models using AutoML, you need to first get labeled training data. The images need to be uploaded to the cloud and label annotations need to be in JSONL format. You can either use the Azure ML Data Labeling tool to label your data or you could start with pre-labeled image data.

Using Azure ML Data Labeling tool to label your training data

If you don't have pre-labeled data, you can use Azure Machine Learning's [data labeling tool](#) to manually label images. This tool automatically generates the data required for training in the accepted format.

It helps to create, manage, and monitor data labeling tasks for

- Image classification (multi-class and multi-label)
- Object detection (bounding box)
- Instance segmentation (polygon)

If you already have a data labeling project and you want to use that data, you can [export your labeled data as an Azure ML Dataset](#). You can then access the exported dataset under the 'Datasets' tab in Azure ML Studio, and download the underlying JSONL file from the Dataset details page under Data sources. The downloaded JSONL file can then be used to create an [MLTable](#) that can be used by automated ML for training computer vision models.

Using pre-labeled training data

If you have previously labeled data that you would like to use to train your model, you will first need to upload the images to the default Azure Blob Storage of your Azure ML Workspace and register it as a data asset.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Create a .yml file with the following configuration.

```
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: fridge-items-images-object-detection
description: Fridge-items images Object detection
path: ./data/odFridgeObjects
type: uri_folder
```

To upload the images as a data asset, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

```
az ml data create -f [PATH_TO_YML_FILE] --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Next, you will need to get the label annotations in JSONL format. The schema of labeled data depends on the computer vision task at hand. Refer to [schemas for JSONL files for AutoML computer vision experiments](#) to learn more about the required JSONL schema for each task type.

If your training data is in a different format (like, pascal VOC or COCO), [helper scripts](#) to convert the data to JSONL are available in [notebook examples](#).

Create MLTable

Once you have your labeled data in JSONL format, you can use it to create [MLTable](#) as shown below. MLTable packages your data into a consumable object for training.

```
paths:
  - file: ./train_annotations.jsonl
transformations:
  - read_json_lines:
      encoding: utf8
      invalid_lines: error
      include_path_column: false
  - convert_column_types:
      - columns: image_url
        column_type: stream_info
```

You can then pass in the [MLTable](#) as a data input for your AutoML training job.

Next steps

- [Train computer vision models with automated machine learning.](#)
- [Train a small object detection model with automated machine learning.](#)
- [Tutorial: Train an object detection model \(preview\) with AutoML and Python.](#)

Set up AutoML to train computer vision models

9/22/2022 • 19 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you learn how to train computer vision models on image data with automated ML with the Azure Machine Learning CLI extension v2 or the Azure Machine Learning Python SDK v2 (preview).

Automated ML supports model training for computer vision tasks like image classification, object detection, and instance segmentation. Authoring AutoML models for computer vision tasks is currently supported via the Azure Machine Learning Python SDK. The resulting experimentation runs, models, and outputs are accessible from the Azure Machine Learning studio UI. [Learn more about automated ml for computer vision tasks on image data](#).

Prerequisites

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO: Azure CLI ml extension v2 (current)

- An Azure Machine Learning workspace. To create the workspace, see [Create workspace resources](#).
- Install and [set up CLI \(v2\)](#) and make sure you install the `ml` extension.

Select your task type

Automated ML for images supports the following task types:

TASK TYPE	AUTOML JOB SYNTAX
image classification	CLI v2: <code>image_classification</code> SDK v2: <code>image_classification()</code>
image classification multi-label	CLI v2: <code>image_classification_multilabel</code> SDK v2: <code>image_classification_multilabel()</code>
image object detection	CLI v2: <code>image_object_detection</code> SDK v2: <code>image_object_detection()</code>
image instance segmentation	CLI v2: <code>image_instance_segmentation</code> SDK v2: <code>image_instance_segmentation()</code>

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

This task type is a required parameter and can be set using the `task` key.

For example:

```
task: image_object_detection
```

Training and validation data

In order to generate computer vision models, you need to bring labeled image data as input for model training in the form of an `MLTable`. You can create an `MLTable` from training data in JSONL format.

If your training data is in a different format (like, pascal VOC or COCO), you can apply the helper scripts included with the sample notebooks to convert the data to JSONL. Learn more about how to [prepare data for computer vision tasks with automated ML](#).

NOTE

The training data needs to have at least 10 images in order to be able to submit an AutoML run.

WARNING

Creation of `MLTable` from data in JSONL format is supported using the SDK and CLI only, for this capability. Creating the `MLTable` via UI is not supported at this time. As of now, the UI doesn't recognize the StreamInfo datatype, which is the datatype used for image URLs in JSONL format.

JSONL schema samples

The structure of the TabularDataset depends upon the task at hand. For computer vision task types, it consists of the following fields:

FIELD	DESCRIPTION
<code>image_url</code>	Contains filepath as a StreamInfo object
<code>image_details</code>	Image metadata information consists of height, width, and format. This field is optional and hence may or may not exist.
<code>label</code>	A json representation of the image label, based on the task type.

The following is a sample JSONL file for image classification:

```
{
  "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/image_data/Image_01.png",
  "image_details":
  {
    "format": "png",
    "width": "2230px",
    "height": "4356px"
  },
  "label": "cat"
}
{
  "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/image_data/Image_02.jpeg",
  "image_details":
  {
    "format": "jpeg",
    "width": "3456px",
    "height": "3467px"
  },
  "label": "dog"
}
```

The following code is a sample JSONL file for object detection:

```
{
  "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/image_data/Image_01.png",
  "image_details":
  {
    "format": "png",
    "width": "2230px",
    "height": "4356px"
  },
  "label":
  {
    "label": "cat",
    "topX": "1",
    "topY": "0",
    "bottomX": "0",
    "bottomY": "1",
    "isCrowd": "true",
  }
}
{
  "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/image_data/Image_02.png",
  "image_details":
  {
    "format": "jpeg",
    "width": "1230px",
    "height": "2356px"
  },
  "label":
  {
    "label": "dog",
    "topX": "0",
    "topY": "1",
    "bottomX": "0",
    "bottomY": "1",
    "isCrowd": "false",
  }
}
```

Consume data

Once your data is in JSONL format, you can create training and validation `MLTable` as shown below.

```
paths:
- file: ./train_annotations.jsonl
transformations:
- read_json_lines:
  encoding: utf8
  invalid_lines: error
  include_path_column: false
- convert_column_types:
  - columns: image_url
  column_type: stream_info
```

Automated ML doesn't impose any constraints on training or validation data size for computer vision tasks. Maximum dataset size is only limited by the storage layer behind the dataset (i.e. blob store). There's no minimum number of images or labels. However, we recommend starting with a minimum of 10-15 samples per label to ensure the output model is sufficiently trained. The higher the total number of labels/classes, the more samples you need per label.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Training data is a required parameter and is passed in using the `training` key of the data section. You can optionally specify another `MLtable` as a validation data with the `validation` key. If no validation data is specified, 20% of your training data will be used for validation by default, unless you pass `validation_data_size` argument with a different value.

Target column name is a required parameter and used as target for supervised ML task. It's passed in using the `target_column_name` key in the data section. For example,

```
target_column_name: label
training_data:
  path: data/training-mltable-folder
  type: mltable
validation_data:
  path: data/validation-mltable-folder
  type: mltable
```

Compute to run experiment

Provide a `compute target` for automated ML to conduct model training. Automated ML models for computer vision tasks require GPU SKUs and support NC and ND families. We recommend the NCv3-series (with v100 GPUs) for faster training. A compute target with a multi-GPU VM SKU leverages multiple GPUs to also speed up training. Additionally, when you set up a compute target with multiple nodes you can conduct faster model training through parallelism when tuning hyperparameters for your model.

The compute target is passed in using the `compute` parameter. For example:

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
compute: azureml:gpu-cluster
```

Configure model algorithms and hyperparameters

With support for computer vision tasks, you can control the model algorithm and sweep hyperparameters. These model algorithms and hyperparameters are passed in as the parameter space for the sweep.

The model algorithm is required and is passed in via `model_name` parameter. You can either specify a single `model_name` or choose between multiple.

Supported model algorithms

The following table summarizes the supported models for each computer vision task.

TASK	MODEL ALGORITHMS	STRING LITERAL SYNTAX <small>DEFAULT_MODEL * DENOTED WITH *</small>
Image classification (multi-class and multi-label)	MobileNet : Light-weighted models for mobile applications ResNet : Residual networks ResNeSt : Split attention networks SE-ResNeXt50 : Squeeze-and-Excitation networks ViT : Vision transformer networks	<code>mobilenetv2</code> <code>resnet18</code> <code>resnet34</code> <code>resnet50</code> <code>resnet101</code> <code>resnet152</code> <code>resnest50</code> <code>resnest101</code> <code>seresnext</code> <code>vits16r224</code> (small) <code>vitb16r224</code> * (base) <code>vitl16r224</code> (large)
Object detection	YOLOv5 : One stage object detection model Faster RCNN ResNet FPN : Two stage object detection models RetinaNet ResNet FPN : address class imbalance with Focal Loss <i>Note: Refer to <code>model_size</code> hyperparameter for YOLOv5 model sizes.</i>	<code>yolov5</code> * <code>fasterrcnn_resnet18_fpn</code> <code>fasterrcnn_resnet34_fpn</code> <code>fasterrcnn_resnet50_fpn</code> <code>fasterrcnn_resnet101_fpn</code> <code>fasterrcnn_resnet152_fpn</code> <code>retinanet_resnet50_fpn</code>
Instance segmentation	MaskRCNN ResNet FPN	<code>maskrcnn_resnet18_fpn</code> <code>maskrcnn_resnet34_fpn</code> <code>maskrcnn_resnet50_fpn</code> * <code>maskrcnn_resnet101_fpn</code> <code>maskrcnn_resnet152_fpn</code>

In addition to controlling the model algorithm, you can also tune hyperparameters used for model training. While many of the hyperparameters exposed are model-agnostic, there are instances where hyperparameters are task-specific or model-specific. [Learn more about the available hyperparameters for these instances](#).

Data augmentation

In general, deep learning model performance can often improve with more data. Data augmentation is a practical technique to amplify the data size and variability of a dataset which helps to prevent overfitting and improve the model's generalization ability on unseen data. Automated ML applies different data augmentation techniques based on the computer vision task, before feeding input images to the model. Currently, there's no exposed hyperparameter to control data augmentations.

Task	Impacted Dataset	Data Augmentation Technique(s) Applied
Image classification (multi-class and multi-label)	Training	Random resize and crop, horizontal flip, color jitter (brightness, contrast, saturation, and hue), normalization using channel-wise ImageNet's mean and standard deviation
	Validation & Test	Resize, center crop, normalization
Object detection, instance segmentation	Training	Random crop around bounding boxes, expand, horizontal flip, normalization, resize
	Validation & Test	Normalization, resize
Object detection using yolov5	Training	Mosaic, random affine (rotation, translation, scale, shear), horizontal flip
	Validation & Test	Letterbox resizing

Configure your experiment settings

Before doing a large sweep to search for the optimal models and hyperparameters, we recommend trying the default values to get a first baseline. Next, you can explore multiple hyperparameters for the same model before sweeping over multiple models and their parameters. This way, you can employ a more iterative approach, because with multiple models and multiple hyperparameters for each, the search space grows exponentially and you need more iterations to find optimal configurations.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

If you wish to use the default hyperparameter values for a given algorithm (say yolov5), you can specify it using `model_name` key in `image_model` section. For example,

```
image_model:
  model_name: "yolov5"
```

Once you've built a baseline model, you might want to optimize model performance in order to sweep over the model algorithm and hyperparameter space. You can use the following sample config to sweep over the hyperparameters for each algorithm, choosing from a range of values for `learning_rate`, `optimizer`, `lr_scheduler`, etc., to generate a model with the optimal primary metric. If hyperparameter values aren't specified, then default values are used for the specified algorithm.

Primary metric

The primary metric used for model optimization and hyperparameter tuning depends on the task type. Using other primary metric values is currently not supported.

- `accuracy` for IMAGE_CLASSIFICATION
- `iou` for IMAGE_CLASSIFICATION_MULTILABEL

- `mean_average_precision` for IMAGE_OBJECT_DETECTION
- `mean_average_precision` for IMAGE_INSTANCE_SEGMENTATION

Experiment budget

You can optionally specify the maximum time budget for your AutoML Vision training job using the `timeout` parameter in the `limits` - the amount of time in minutes before the experiment terminates. If none specified, default experiment timeout is seven days (maximum 60 days). For example,

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

```
limits:
  timeout: 60
```

Sweeping hyperparameters for your model

When training computer vision models, model performance depends heavily on the hyperparameter values selected. Often, you might want to tune the hyperparameters to get optimal performance. With support for computer vision tasks in automated ML, you can sweep hyperparameters to find the optimal settings for your model. This feature applies the hyperparameter tuning capabilities in Azure Machine Learning. [Learn how to tune hyperparameters](#).

Define the parameter search space

You can define the model algorithms and hyperparameters to sweep in the parameter space.

- See [Configure model algorithms and hyperparameters](#) for the list of supported model algorithms for each task type.
- See [Hyperparameters for computer vision tasks](#) hyperparameters for each computer vision task type.
- See [details on supported distributions for discrete and continuous hyperparameters](#).

Sampling methods for the sweep

When sweeping hyperparameters, you need to specify the sampling method to use for sweeping over the defined parameter space. Currently, the following sampling methods are supported with the `sampling_algorithm` parameter:

SAMPLING TYPE	AUTOML JOB SYNTAX
Random Sampling	<code>random</code>
Grid Sampling	<code>grid</code>
Bayesian Sampling	<code>bayesian</code>

NOTE

Currently only random sampling supports conditional hyperparameter spaces.

Early termination policies

You can automatically end poorly performing runs with an early termination policy. Early termination improves computational efficiency, saving compute resources that would have been otherwise spent on less promising

configurations. Automated ML for images supports the following early termination policies using the `early_termination` parameter. If no termination policy is specified, all configurations are run to completion.

EARLY TERMINATION POLICY	AUTOML JOB SYNTAX
Bandit policy	CLI v2: <code>bandit</code> SDK v2: <code>BanditPolicy()</code>
Median stopping policy	CLI v2: <code>median_stopping</code> SDK v2: <code>MedianStoppingPolicy()</code>
Truncation selection policy	CLI v2: <code>truncation_selection</code> SDK v2: <code>TruncationSelectionPolicy()</code>

Learn more about [how to configure the early termination policy for your hyperparameter sweep](#).

Resources for the sweep

You can control the resources spent on your hyperparameter sweep by specifying the `max_trials` and the `max_concurrent_trials` for the sweep.

NOTE

For a complete sweep configuration sample, please refer to this [tutorial](#).

PARAMETER	DETAIL
<code>max_trials</code>	Required parameter for maximum number of configurations to sweep. Must be an integer between 1 and 1000. When exploring just the default hyperparameters for a given model algorithm, set this parameter to 1.
<code>max_concurrent_trials</code>	Maximum number of runs that can run concurrently. If not specified, all runs launch in parallel. If specified, must be an integer between 1 and 100. NOTE: The number of concurrent runs is gated on the resources available in the specified compute target. Ensure that the compute target has the available resources for the desired concurrency.

You can configure all the sweep related parameters as shown in the example below.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
sweep:  
  limits:  
    max_trials: 10  
    max_concurrent_trials: 2  
  sampling_algorithm: random  
  early_termination:  
    type: bandit  
    evaluation_interval: 2  
    slack_factor: 0.2  
    delay_evaluation: 6
```

Fixed settings

You can pass fixed settings or parameters that don't change during the parameter space sweep as shown below.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
image_model:  
  early_stopping: True  
  evaluation_frequency: 1
```

Incremental training (optional)

Once the training run is done, you have the option to further train the model by loading the trained model checkpoint. You can either use the same dataset or a different one for incremental training.

Pass the checkpoint via run ID

You can pass the run ID that you want to load the checkpoint from.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
image_model:  
  checkpoint_run_id : "target_checkpoint_run_id"
```

Submit the AutoML job

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

To submit your AutoML job, you run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

```
az ml job create --file ./hello-automl-job-basic.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Outputs and evaluation metrics

The automated ML training runs generates output model files, evaluation metrics, logs and deployment artifacts like the scoring file and the environment file which can be viewed from the outputs and logs and metrics tab of the child runs.

TIP

Check how to navigate to the run results from the [View run results](#) section.

For definitions and examples of the performance charts and metrics provided for each run, see [Evaluate automated machine learning experiment results](#).

Register and deploy model

Once the run completes, you can register the model that was created from the best run (configuration that resulted in the best primary metric). You can either register the model after downloading or by specifying the azureml path with corresponding jobid. Note: If you want to change the inference settings that are described below you need to download the model and change settings.json and register using the updated model folder.

Get the best run

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

CLI example not available, please use Python SDK.

register the model

Register the model either using the azureml path or your locally downloaded path.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml model create --name od-fridge-items-mlflow-model --version 1 --path
azureml://jobs/$best_run/outputs/artifacts/outputs/mlflow-model/ --type mlflow_model --workspace-name
[YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

After you register the model you want to use, you can deploy it using the managed online endpoint [deploy-managed-online-endpoint](#)

Configure online endpoint

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: od-fridge-items-endpoint
auth_mode: key
```

Create the endpoint

Using the `MLClient` created earlier, we'll now create the Endpoint in the workspace. This command will start the endpoint creation and return a confirmation response while the endpoint creation continues.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml online-endpoint create --file .\create_endpoint.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Configure online deployment

A deployment is a set of resources required for hosting the model that does the actual inferencing. We'll create a deployment for our endpoint using the `ManagedOnlineDeployment` class. You can use either GPU or CPU VM SKUs for your deployment cluster.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
name: od-fridge-items-mlflow-deploy
endpoint_name: od-fridge-items-endpoint
model: azureml:od-fridge-items-mlflow-model@latest
instance_type: Standard_DS3_v2
instance_count: 1
liveness_probe:
    failure_threshold: 30
    success_threshold: 1
    timeout: 2
    period: 10
    initial_delay: 2000
readiness_probe:
    failure_threshold: 10
    success_threshold: 1
    timeout: 10
    period: 10
    initial_delay: 2000
```

Create the deployment

Using the `MLClient` created earlier, we'll now create the deployment in the workspace. This command will start the deployment creation and return a confirmation response while the deployment creation continues.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml online-deployment create --file .\create_deployment.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

update traffic:

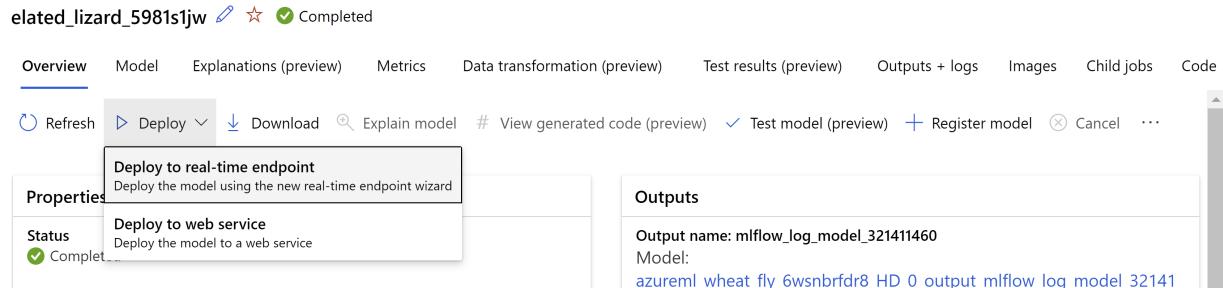
By default the current deployment is set to receive 0% traffic. you can set the traffic percentage current deployment should receive. Sum of traffic percentages of all the deployments with one end point shouldn't exceed 100%.

- Azure CLI
- Python SDK

APPLIES TO:  Azure CLI ml extension v2 (current)

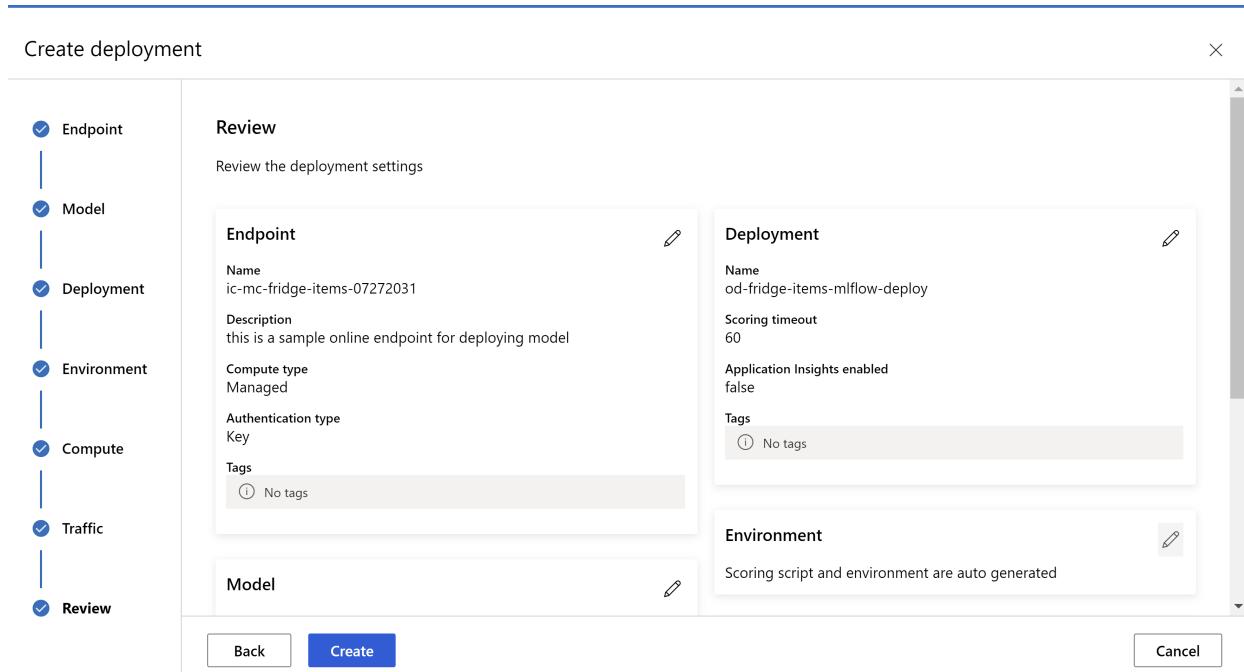
```
az ml online-endpoint update --name 'od-fridge-items-endpoint' --traffic 'od-fridge-items-mlflow-deploy=100' --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Alternatively You can deploy the model from the [Azure Machine Learning studio UI](#). Navigate to the model you wish to deploy in the **Models** tab of the automated ML run and select on **Deploy** and select **Deploy to real-time endpoint**.



The screenshot shows the Azure Machine Learning studio UI. At the top, there's a navigation bar with tabs like Overview, Model, Explanations (preview), Metrics, Data transformation (preview), Test results (preview), Outputs + logs, Images, Child jobs, and Code. Below the navigation bar, there's a toolbar with Refresh, Deploy, Download, Explain model, View generated code (preview), Test model (preview), Register model, Cancel, and more. A dropdown menu under the Deploy button is open, showing two options: 'Deploy to real-time endpoint' (selected) and 'Deploy to web service'. On the left, there's a sidebar with Properties, Status (Completed), and a list of steps: Endpoint, Model, Deployment, Environment, Compute, Traffic, and Review. The 'Review' step is currently selected. On the right, there's a section titled 'Outputs' with the output name 'mlflow_log_model_321411460' and the model path 'azureml_wheat_fly_6wsnbrdr8_HD_0_output_mlflow_log_model_32141'.

this is how your review page looks like. we can select instance type, instance count and set traffic percentage for the current deployment.



The screenshot shows the 'Create deployment' review page. On the left, there's a sidebar with steps: Endpoint, Model, Deployment, Environment, Compute, Traffic, and Review. The 'Review' step is selected. The main area has a 'Review' title and a sub-section 'Review the deployment settings'. It shows the configuration for an endpoint and a deployment. The endpoint details include Name: 'ic-mc-fridge-items-07272031', Description: 'this is a sample online endpoint for deploying model', Compute type: 'Managed', Authentication type: 'Key', and Tags: 'No tags'. The deployment details include Name: 'od-fridge-items-mlflow-deploy', Scoring timeout: '60', Application Insights enabled: 'false', and Tags: 'No tags'. Below these, there's a 'Model' section with the message 'Scoring script and environment are auto generated'. At the bottom, there are 'Back', 'Create', and 'Cancel' buttons.

Create deployment

Key
Tags
No tags

Model

Name: ic-mc-fridge-items-model
Version: 1

Compute

Name: Standard_DS2_v2
Instances: 3

Environment

Scoring script and environment are auto generated

Deployment traffic

od-fridge-items-mlflow-deploy
100%

Back Create Cancel

Update inference settings

In the previous step, we downloaded a file `mlflow-model/artifacts/settings.json` from the best model, which can be used to update the inference settings before registering the model. Although it's recommended to use the same parameters as training for best performance.

Each of the tasks (and some models) has a set of parameters. By default, we use the same values for the parameters that were used during the training and validation. Depending on the behavior that we need when using the model for inference, we can change these parameters. Below you can find a list of parameters for each task type and model.

TASK	PARAMETER NAME	DEFAULT
Image classification (multi-class and multi-label)	<code>valid_resize_size</code> <code>valid_crop_size</code>	256 224
Object detection	<code>min_size</code> <code>max_size</code> <code>box_score_thresh</code> <code>nms_iou_thresh</code> <code>box_detections_per_img</code>	600 1333 0.3 0.5 100
Object detection using <code>yolov5</code>	<code>img_size</code> <code>model_size</code> <code>box_score_thresh</code> <code>nms_iou_thresh</code>	640 medium 0.1 0.5
Instance segmentation	<code>min_size</code> <code>max_size</code> <code>box_score_thresh</code> <code>nms_iou_thresh</code> <code>box_detections_per_img</code> <code>mask_pixel_score_threshold</code> <code>max_number_of_polygon_points</code> <code>export_as_image</code> <code>image_type</code>	600 1333 0.3 0.5 100 0.5 100 False JPG

For a detailed description on task specific hyperparameters, please refer to [Hyperparameters for computer vision tasks in automated machine learning](#).

If you want to use tiling, and want to control tiling behavior, the following parameters are available:

`tile_grid_size`, `tile_overlap_ratio` and `tile_predictions_nms_thresh`. For more details on these parameters please check [Train a small object detection model using AutoML](#).

Test the deployment

Please check this [Test the deployment](#) section to test the deployment and visualize the detections from the model.

Example notebooks

Review detailed code examples and use cases in the [GitHub notebook repository for automated machine learning samples](#). Please check the folders with 'automl-image-' prefix for samples specific to building computer vision models.

Code examples

- [Azure CLI](#)
- [Python SDK](#)

Review detailed code examples and use cases in the [azureml-examples repository for automated machine learning samples](#).

Next steps

- [Tutorial: Train an object detection model \(preview\) with AutoML and Python.](#)
- [Troubleshoot automated ML experiments.](#)

Train a small object detection model with AutoML (preview)

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you'll learn how to train an object detection model to detect small objects in high-resolution images with [automated ML](#) in Azure Machine Learning.

Typically, computer vision models for object detection work well for datasets with relatively large objects. However, due to memory and computational constraints, these models tend to under-perform when tasked to detect small objects in high-resolution images. Because high-resolution images are typically large, they are resized before input into the model, which limits their capability to detect smaller objects--relative to the initial image size.

To help with this problem, automated ML supports tiling as part of the public preview computer vision capabilities. The tiling capability in automated ML is based on the concepts in [The Power of Tiling for Small Object Detection](#).

When tiling, each image is divided into a grid of tiles. Adjacent tiles overlap with each other in width and height dimensions. The tiles are cropped from the original as shown in the following image.



Prerequisites

- An Azure Machine Learning workspace. To create the workspace, see [Create workspace resources](#).
- This article assumes some familiarity with how to configure an [automated machine learning experiment for computer vision tasks](#).

Supported models

Small object detection using tiling is currently supported for the following models:

- fasterrcnn_resnet18_fpn
- fasterrcnn_resnet50_fpn
- fasterrcnn_resnet34_fpn
- fasterrcnn_resnet101_fpn
- fasterrcnn_resnet152_fpn
- retinanet_resnet50_fpn

Enable tiling during training

To enable tiling, you can set the `tile_grid_size` parameter to a value like `(3, 2)`; where 3 is the number of tiles along the width dimension and 2 is the number of tiles along the height dimension. When this parameter is set to `(3, 2)`, each image is split into a grid of 3×2 tiles. Each tile overlaps with the adjacent tiles, so that any objects that fall on the tile border are included completely in one of the tiles. This overlap can be controlled by the `tile_overlap_ratio` parameter, which defaults to 25%.

When tiling is enabled, the entire image and the tiles generated from it are passed through the model. These images and tiles are resized according to the `min_size` and `max_size` parameters before feeding to the model. The computation time increases proportionally because of processing this extra data.

For example, when the `tile_grid_size` parameter is `(3, 2)`, the computation time would be approximately seven times when compared to no tiling.

You can specify the value for `tile_grid_size` in your hyperparameter space as a string.

```
parameter_space = {
    'model_name': choice('fasterrcnn_resnet50_fpn'),
    'tile_grid_size': choice('(3, 2)'),
    ...
}
```

The value for `tile_grid_size` parameter depends on the image dimensions and size of objects within the image. For example, larger number of tiles would be helpful when there are smaller objects in the images.

To choose the optimal value for this parameter for your dataset, you can use hyperparameter search. To do so, you can specify a choice of values for this parameter in your hyperparameter space.

```
parameter_space = {
    'model_name': choice('fasterrcnn_resnet50_fpn'),
    'tile_grid_size': choice('(2, 1)', '(3, 2)', '(5, 3)'),
    ...
}
```

Tiling during inference

When a model trained with tiling is deployed, tiling also occurs during inference. Automated ML uses the `tile_grid_size` value from training to generate the tiles during inference. The entire image and corresponding tiles are passed through the model, and the object proposals from them are merged to output final predictions, like in the following image.



NOTE

It's possible that the same object is detected from multiple tiles, duplication detection is done to remove such duplicates.

Duplicate detection is done by running NMS on the proposals from the tiles and the image. When multiple proposals overlap, the one with the highest score is picked and others are discarded as duplicates. Two proposals are considered to be overlapping when the intersection over union (iou) between them is greater than the `tile_predictions_nms_thresh` parameter.

You also have the option to enable tiling only during inference without enabling it in training. To do so, set the `tile_grid_size` parameter only during inference, not for training.

Doing so, may improve performance for some datasets, and won't incur the extra cost that comes with tiling at training time.

Tiling hyperparameters

The following are the parameters you can use to control the tiling feature.

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>tile_grid_size</code>	<p>The grid size to use for tiling each image. Available for use during training, validation, and inference.</p> <p>Tuple of two integers passed as a string, e.g <code>'(3, 2)'</code></p> <p><i>Note: Setting this parameter increases the computation time proportionally, since all tiles and images are processed by the model.</i></p>	no default value

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>tile_overlap_ratio</code>	<p>Controls the overlap ratio between adjacent tiles in each dimension. When the objects that fall on the tile boundary are too large to fit completely in one of the tiles, increase the value of this parameter so that the objects fit in at least one of the tiles completely.</p> <p>Must be a float in [0, 1].</p>	0.25
<code>tile_predictions_nms_thresh</code>	<p>The intersection over union threshold to use to do non-maximum suppression (nms) while merging predictions from tiles and image. Available during validation and inference. Change this parameter if there are multiple boxes detected per object in the final predictions.</p> <p>Must be float in [0, 1].</p>	0.25

Example notebooks

See the [object detection sample notebook](#) for detailed code examples of setting up and training an object detection model.

NOTE

All images in this article are made available in accordance with the permitted use section of the [MIT licensing agreement](#). Copyright © 2020 Roboflow, Inc.

Next steps

- Learn more about [how and where to deploy a model](#).
- For definitions and examples of the performance charts and metrics provided for each job, see [Evaluate automated machine learning experiment results](#).
- [Tutorial: Train an object detection model \(preview\) with AutoML and Python](#).
- See [what hyperparameters are available for computer vision tasks](#). *[Make predictions with ONNX on computer vision models from AutoML](#)

Set up AutoML to train a natural language processing model (preview)

9/22/2022 • 12 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you learn how to train natural language processing (NLP) models with [automated ML](#) in Azure Machine Learning. You can create NLP models with automated ML via the Azure Machine Learning Python SDK v2 (preview) or the Azure Machine Learning CLI v2.

Automated ML supports NLP which allows ML professionals and data scientists to bring their own text data and build custom models for tasks such as, multi-class text classification, multi-label text classification, and named entity recognition (NER).

You can seamlessly integrate with the [Azure Machine Learning data labeling](#) capability to label your text data or bring your existing labeled data. Automated ML provides the option to use distributed training on multi-GPU compute clusters for faster model training. The resulting model can be operationalized at scale by leveraging Azure ML's MLOps capabilities.

Prerequisites

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO: Azure CLI ml extension v2 (current)

- Azure subscription. If you don't have an Azure subscription, sign up to try the [free or paid version of Azure Machine Learning](#) today.
- An Azure Machine Learning workspace with a GPU training compute. To create the workspace, see [Create workspace resources](#). See [GPU optimized virtual machine sizes](#) for more details of GPU instances provided by Azure.

WARNING

Support for multilingual models and the use of models with longer max sequence length is necessary for several NLP use cases, such as non-english datasets and longer range documents. As a result, these scenarios may require higher GPU memory for model training to succeed, such as the NC_v3 series or the ND series.

- The Azure Machine Learning CLI v2 installed. For guidance to update and install the latest version, see the [Install and set up CLI \(v2\)](#).
- This article assumes some familiarity with setting up an automated machine learning experiment. Follow the [how-to](#) to see the main automated machine learning experiment design patterns.

Select your NLP task

Determine what NLP task you want to accomplish. Currently, automated ML supports the following deep neural network NLP tasks.

TASK	AUTOML JOB SYNTAX	DESCRIPTION
Multi-class text classification	CLI v2: <code>text_classification</code> SDK v2 (preview): <code>text_classification()</code>	<p>There are multiple possible classes and each sample can be classified as exactly one class. The task is to predict the correct class for each sample.</p> <p>For example, classifying a movie script as "Comedy" or "Romantic".</p>
Multi-label text classification	CLI v2: <code>text_classification_multilabel</code> SDK v2 (preview): <code>text_classification_multilabel()</code>	<p>There are multiple possible classes and each sample can be assigned any number of classes. The task is to predict all the classes for each sample</p> <p>For example, classifying a movie script as "Comedy", or "Romantic", or "Comedy and Romantic".</p>
Named Entity Recognition (NER)	CLI v2: <code>text_ner</code> SDK v2 (preview): <code>text_ner()</code>	<p>There are multiple possible tags for tokens in sequences. The task is to predict the tags for all the tokens for each sequence.</p> <p>For example, extracting domain-specific entities from unstructured text, such as contracts or financial documents</p>

Preparing data

For NLP experiments in automated ML, you can bring your data in `.csv` format for multi-class and multi-label classification tasks. For NER tasks, two-column `.txt` files that use a space as the separator and adhere to the CoNLL format are supported. The following sections provide additional detail for the data format accepted for each task.

Multi-class

For multi-class classification, the dataset can contain several text columns and exactly one label column. The following example has only one text column.

```
text,labels
"I love watching Chicago Bulls games.", "NBA"
"Tom Brady is a great player.", "NFL"
"There is a game between Yankees and Orioles tonight", "MLB"
"Stephen Curry made the most number of 3-Pointers", "NBA"
```

Multi-label

For multi-label classification, the dataset columns would be the same as multi-class, however there are special format requirements for data in the label column. The two accepted formats and examples are in the following table.

LABEL COLUMN FORMAT OPTIONS	MULTIPLE LABELS	ONE LABEL	NO LABELS
Plain text	"label1, label2, label3"	"label1"	""
Python list with quotes	"['label1','label2','label3']"	["'label1'"]	["[]"]

IMPORTANT

Different parsers are used to read labels for these formats. If you are using the plain text format, only use alphabetical, numerical and `'_'` in your labels. All other characters are recognized as the separator of labels.

For example, if your label is `"cs.AI"`, it's read as `"cs"` and `"AI"`. Whereas with the Python list format, the label would be `["'cs.AI'"]`, which is read as `"cs.AI"`.

Example data for multi-label in plain text format.

```
text,labels
"I love watching Chicago Bulls games.", "basketball"
"The four most popular leagues are NFL, MLB, NBA and NHL", "football,baseball,basketball,hockey"
"I like drinking beer.", ""
```

Example data for multi-label in Python list with quotes format.

```
text,labels
"I love watching Chicago Bulls games.", ["'basketball'"]
"The four most popular leagues are NFL, MLB, NBA and NHL", ["'football','baseball','basketball','hockey'"]
"I like drinking beer.", []"
```

Named entity recognition (NER)

Unlike multi-class or multi-label, which takes `.csv` format datasets, named entity recognition requires CoNLL format. The file must contain exactly two columns and in each row, the token and the label is separated by a single space.

For example,

```
Hudson B-loc
Square I-loc
is O
a O
famous O
place O
in O
New B-loc
York I-loc
City I-loc

Stephen B-per
Curry I-per
got O
three O
championship O
rings O
```

Data validation

Before training, automated ML applies data validation checks on the input data to ensure that the data can be preprocessed correctly. If any of these checks fail, the run fails with the relevant error message. The following are the requirements to pass data validation checks for each task.

NOTE

Some data validation checks are applicable to both the training and the validation set, whereas others are applicable only to the training set. If the test dataset could not pass the data validation, that means that automated ML couldn't capture it and there is a possibility of model inference failure, or a decline in model performance.

TASK	DATA VALIDATION CHECK
All tasks	At least 50 training samples are required
Multi-class and Multi-label	The training data and validation data must have <ul style="list-style-type: none">- The same set of columns- The same order of columns from left to right- The same data type for columns with the same name- At least two unique labels- Unique column names within each dataset (For example, the training set can't have multiple columns named Age)
Multi-class only	None
Multi-label only	<ul style="list-style-type: none">- The label column format must be in accepted format- At least one sample should have 0 or 2+ labels, otherwise it should be a <code>multiclass</code> task- All labels should be in <code>str</code> or <code>int</code> format, with no overlapping. You should not have both label <code>1</code> and label <code>'1'</code>
NER only	<ul style="list-style-type: none">- The file should not start with an empty line- Each line must be an empty line, or follow format <code>{token} {label}</code>, where there is exactly one space between the token and the label and no white space after the label- All labels must start with <code>I-</code>, <code>B-</code>, or be exactly <code>O</code>. Case sensitive- Exactly one empty line between two samples- Exactly one empty line at the end of the file

Configure experiment

Automated ML's NLP capability is triggered through task specific `automl` type jobs, which is the same workflow for submitting automated ML experiments for classification, regression and forecasting tasks. You would set parameters as you would for those experiments, such as `experiment_name`, `compute_name` and data inputs.

However, there are key differences:

- You can ignore `primary_metric`, as it is only for reporting purposes. Currently, automated ML only trains one model per run for NLP and there is no model selection.
- The `label_column_name` parameter is only required for multi-class and multi-label text classification tasks.
- If the majority of the samples in your dataset contain more than 128 words, it's considered long range. By default, automated ML considers all samples long range text. To disable this feature, include the `enable_long_range_text=False` parameter in your `AutoMLConfig`.

- If you enable long range text, then a GPU with higher memory is required such as, [NCv3](#) series or [ND](#) series.
 - The `enable_long_range_text` parameter is only available for multi-class classification tasks.
- [Azure CLI](#)
 - [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

For CLI v2 AutoML jobs you configure your experiment in a YAML file like the following.

Language settings

As part of the NLP functionality, automated ML supports 104 languages leveraging language specific and multilingual pre-trained text DNN models, such as the BERT family of models. Currently, language selection defaults to English.

The following table summarizes what model is applied based on task type and language. See the full list of [supported languages and their codes](#).

TASK TYPE	SYNTAX FOR <code>DATASET_LANGUAGE</code>	TEXT MODEL ALGORITHM
Multi-label text classification	<pre>"eng" "deu" "mul"</pre>	English BERT uncased German BERT Multilingual BERT For all other languages, automated ML applies multilingual BERT
Multi-class text classification	<pre>"eng" "deu" "mul"</pre>	English BERT cased Multilingual BERT For all other languages, automated ML applies multilingual BERT
Named entity recognition (NER)	<pre>"eng" "deu" "mul"</pre>	English BERT cased German BERT Multilingual BERT For all other languages, automated ML applies multilingual BERT

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

You can specify your dataset language in the featurization section of your configuration YAML file. BERT is also used in the featurization process of automated ML experiment training, learn more about [BERT integration and featurization in automated ML](#).

```
featurization:
  dataset_language: "eng"
```

Distributed training

You can also run your NLP experiments with distributed training on an Azure ML compute cluster.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Submit the AutoML job

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

To submit your AutoML job, you can run the following CLI v2 command with the path to your .yml file, workspace name, resource group and subscription ID.

```
az ml job create --file ./hello-automl-job-basic.yml --workspace-name [YOUR_AZURE_WORKSPACE] --resource-group [YOUR_AZURE_RESOURCE_GROUP] --subscription [YOUR_AZURE_SUBSCRIPTION]
```

Code examples

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

See the following sample YAML files for each NLP task.

- [Multi-class text classification](#)
- [Multi-label text classification](#)
- [Named entity recognition](#)

Next steps

- [Deploy AutoML models to an online \(real-time inference\) endpoint](#)
- [Troubleshoot automated ML experiments](#)

Configure training, validation, cross-validation and test data in automated machine learning

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, you learn the different options for configuring training data and validation data splits along with cross-validation settings for your automated machine learning, automated ML, experiments.

In Azure Machine Learning, when you use automated ML to build multiple ML models, each child run needs to validate the related model by calculating the quality metrics for that model, such as accuracy or AUC weighted. These metrics are calculated by comparing the predictions made with each model with real labels from past observations in the validation data. [Learn more about how metrics are calculated based on validation type](#).

Automated ML experiments perform model validation automatically. The following sections describe how you can further customize validation settings with the [Azure Machine Learning Python SDK](#).

For a low-code or no-code experience, see [Create your automated machine learning experiments in Azure Machine Learning studio](#).

Prerequisites

For this article you need,

- An Azure Machine Learning workspace. To create the workspace, see [Create workspace resources](#).
- Familiarity with setting up an automated machine learning experiment with the Azure Machine Learning SDK. Follow the [tutorial](#) or [how-to](#) to see the fundamental automated machine learning experiment design patterns.
- An understanding of train/validation data splits and cross-validation as machine learning concepts. For a high-level explanation,
 - [About training, validation and test data in machine learning](#)
 - [Understand Cross Validation in machine learning](#)

IMPORTANT

The Python commands in this article require the latest `azureml-train-automl` package version.

- [Install the latest `azureml-train-automl` package to your local environment](#).
- For details on the latest `azureml-train-automl` package, see the [release notes](#).

Default data splits and cross-validation in machine learning

Use the `AutoMLConfig` object to define your experiment and training settings. In the following code snippet, notice that only the required parameters are defined, that is the parameters for `n_cross_validations` or `validation_data` are **not** included.

NOTE

The default data splits and cross-validation are not supported in forecasting scenarios.

```
data = "https://automlsamplenotebookdata.blob.core.windows.net/automl-sample-notebook-data/creditcard.csv"

dataset = Dataset.Tabular.from_delimited_files(data)

automl_config = AutoMLConfig(compute_target = aml_remote_compute,
                             task = 'classification',
                             primary_metric = 'AUC_weighted',
                             training_data = dataset,
                             label_column_name = 'Class'
                            )
```

If you do not explicitly specify either a `validation_data` or `n_cross_validations` parameter, automated ML applies default techniques depending on the number of rows provided in the single dataset `training_data`.

TRAINING DATA SIZE	VALIDATION TECHNIQUE
Larger than 20,000 rows	Train/validation data split is applied. The default is to take 10% of the initial training data set as the validation set. In turn, that validation set is used for metrics calculation.
Smaller than 20,000 rows	Cross-validation approach is applied. The default number of folds depends on the number of rows. If the dataset is less than 1,000 rows , 10 folds are used. If the rows are between 1,000 and 20,000 , then three folds are used.

Provide validation data

In this case, you can either start with a single data file and split it into training data and validation data sets or you can provide a separate data file for the validation set. Either way, the `validation_data` parameter in your `AutoMLConfig` object assigns which data to use as your validation set. This parameter only accepts data sets in the form of an [Azure Machine Learning dataset](#) or pandas dataframe.

NOTE

The `validation_data` parameter requires the `training_data` and `label_column_name` parameters to be set as well. You can only set one validation parameter, that is you can only specify either `validation_data` or `n_cross_validations`, not both.

The following code example explicitly defines which portion of the provided data in `dataset` to use for training and validation.

```
data = "https://automlsamplenotebookdata.blob.core.windows.net/automl-sample-notebook-data/creditcard.csv"

dataset = Dataset.Tabular.from_delimited_files(data)

training_data, validation_data = dataset.random_split(percentage=0.8, seed=1)

automl_config = AutoMLConfig(compute_target = aml_remote_compute,
                             task = 'classification',
                             primary_metric = 'AUC_weighted',
                             training_data = training_data,
                             validation_data = validation_data,
                             label_column_name = 'Class'
                            )
```

Provide validation set size

In this case, only a single dataset is provided for the experiment. That is, the `validation_data` parameter is **not** specified, and the provided dataset is assigned to the `training_data` parameter.

In your `AutoMLConfig` object, you can set the `validation_size` parameter to hold out a portion of the training data for validation. This means that the validation set will be split by automated ML from the initial `training_data` provided. This value should be between 0.0 and 1.0 non-inclusive (for example, 0.2 means 20% of the data is held out for validation data).

NOTE

The `validation_size` parameter is not supported in forecasting scenarios.

See the following code example:

```
data = "https://automlsamplenotebookdata.blob.core.windows.net/automl-sample-notebook-data/creditcard.csv"

dataset = Dataset.Tabular.from_delimited_files(data)

automl_config = AutoMLConfig(compute_target = aml_remote_compute,
                             task = 'classification',
                             primary_metric = 'AUC_weighted',
                             training_data = dataset,
                             validation_size = 0.2,
                             label_column_name = 'Class'
                            )
```

K-fold cross-validation

To perform k-fold cross-validation, include the `n_cross_validations` parameter and set it to a value. This parameter sets how many cross validations to perform, based on the same number of folds.

NOTE

The `n_cross_validations` parameter is not supported in classification scenarios that use deep neural networks. For forecasting scenarios, see how cross validation is applied in [Set up AutoML to train a time-series forecasting model](#).

In the following code, five folds for cross-validation are defined. Hence, five different trainings, each training using 4/5 of the data, and each validation using 1/5 of the data with a different holdout fold each time.

As a result, metrics are calculated with the average of the five validation metrics.

```
data = "https://automlsamplenotebookdata.blob.core.windows.net/automl-sample-notebook-data/creditcard.csv"

dataset = Dataset.Tabular.from_delimited_files(data)

automl_config = AutoMLConfig(compute_target = aml_remote_compute,
                             task = 'classification',
                             primary_metric = 'AUC_weighted',
                             training_data = dataset,
                             n_cross_validations = 5
                             label_column_name = 'Class'
                             )
```

Monte Carlo cross-validation

To perform Monte Carlo cross validation, include both the `validation_size` and `n_cross_validations` parameters in your `AutoMLConfig` object.

For Monte Carlo cross validation, automated ML sets aside the portion of the training data specified by the `validation_size` parameter for validation, and then assigns the rest of the data for training. This process is then repeated based on the value specified in the `n_cross_validations` parameter; which generates new training and validation splits, at random, each time.

NOTE

The Monte Carlo cross-validation is not supported in forecasting scenarios.

The follow code defines, 7 folds for cross-validation and 20% of the training data should be used for validation. Hence, 7 different trainings, each training uses 80% of the data, and each validation uses 20% of the data with a different holdout fold each time.

```
data = "https://automlsamplenotebookdata.blob.core.windows.net/automl-sample-notebook-data/creditcard.csv"

dataset = Dataset.Tabular.from_delimited_files(data)

automl_config = AutoMLConfig(compute_target = aml_remote_compute,
                             task = 'classification',
                             primary_metric = 'AUC_weighted',
                             training_data = dataset,
                             n_cross_validations = 7
                             validation_size = 0.2,
                             label_column_name = 'Class'
                             )
```

Specify custom cross-validation data folds

You can also provide your own cross-validation (CV) data folds. This is considered a more advanced scenario because you are specifying which columns to split and use for validation. Include custom CV split columns in your training data, and specify which columns by populating the column names in the `cv_split_column_names` parameter. Each column represents one cross-validation split, and is filled with integer values 1 or 0--where 1 indicates the row should be used for training and 0 indicates the row should be used for validation.

NOTE

The `cv_split_column_names` parameter is not supported in forecasting scenarios.

The following code snippet contains bank marketing data with two CV split columns 'cv1' and 'cv2'.

```
data = "https://automlsamplenotebookdata.blob.core.windows.net/automl-sample-notebook-data/bankmarketing_with_cv.csv"

dataset = Dataset.Tabular.from_delimited_files(data)

automl_config = AutoMLConfig(compute_target = aml_remote_compute,
                             task = 'classification',
                             primary_metric = 'AUC_weighted',
                             training_data = dataset,
                             label_column_name = 'y',
                             cv_split_column_names = ['cv1', 'cv2']
                            )
```

NOTE

To use `cv_split_column_names` with `training_data` and `label_column_name`, please upgrade your Azure Machine Learning Python SDK version 1.6.0 or later. For previous SDK versions, please refer to using `cv_splits_indices`, but note that it is used with `x` and `y` dataset input only.

Metric calculation for cross validation in machine learning

When either k-fold or Monte Carlo cross validation is used, metrics are computed on each validation fold and then aggregated. The aggregation operation is an average for scalar metrics and a sum for charts. Metrics computed during cross validation are based on all folds and therefore all samples from the training set. [Learn more about metrics in automated machine learning](#).

When either a custom validation set or an automatically selected validation set is used, model evaluation metrics are computed from only that validation set, not the training data.

Provide test data (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

You can also provide test data to evaluate the recommended model that automated ML generates for you upon completion of the experiment. When you provide test data it's considered a separate from training and validation, so as to not bias the results of the test run of the recommended model. [Learn more about training, validation and test data in automated ML](#).

WARNING

This feature is not available for the following automated ML scenarios

- Computer vision tasks (preview)
- Many models and hierarchical time series forecasting training (preview)
- Forecasting tasks where deep learning neural networks (DNN) are enabled
- Automated ML runs from local computes or Azure Databricks clusters

Test datasets must be in the form of an [Azure Machine Learning TabularDataset](#). You can specify a test dataset

with the `test_data` and `test_size` parameters in your `AutoMLConfig` object. These parameters are mutually exclusive and can not be specified at the same time or with `cv_split_column_names` or `cv_splits_indices`.

With the `test_data` parameter, specify an existing dataset to pass into your `AutoMLConfig` object.

```
automl_config = AutoMLConfig(task='forecasting',
    ...
    # Provide an existing test dataset
    test_data=test_dataset,
    ...
    forecasting_parameters=forecasting_parameters)
```

To use a train/test split instead of providing test data directly, use the `test_size` parameter when creating the `AutoMLConfig`. This parameter must be a floating point value between 0.0 and 1.0 exclusive, and specifies the percentage of the training dataset that should be used for the test dataset.

```
automl_config = AutoMLConfig(task = 'regression',
    ...
    # Specify train/test split
    training_data=training_data,
    test_size=0.2)
```

NOTE

For regression tasks, random sampling is used.

For classification tasks, stratified sampling is used, but random sampling is used as a fall back when stratified sampling is not feasible.

Forecasting does not currently support specifying a test dataset using a train/test split with the `test_size` parameter.

Passing the `test_data` or `test_size` parameters into the `AutoMLConfig`, automatically triggers a remote test run upon completion of your experiment. This test run uses the provided test data to evaluate the best model that automated ML recommends. Learn more about [how to get the predictions from the test run](#).

Next steps

- [Prevent imbalanced data and overfitting](#).
- How to [Auto-train a time-series forecast model](#).

Data featurization in automated machine learning

9/22/2022 • 15 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azureml v1](#)

Learn about the data featurization settings in Azure Machine Learning, and how to customize those features for [automated machine learning experiments](#).

Feature engineering and featurization

Training data consists of rows and columns. Each row is an observation or record, and the columns of each row are the features that describe each record. Typically, the features that best characterize the patterns in the data are selected to create predictive models.

Although many of the raw data fields can be used directly to train a model, it's often necessary to create additional (engineered) features that provide information that better differentiates patterns in the data. This process is called **feature engineering**, where the use of domain knowledge of the data is leveraged to create features that, in turn, help machine learning algorithms to learn better.

In Azure Machine Learning, data-scaling and normalization techniques are applied to make feature engineering easier. Collectively, these techniques and this feature engineering are called **featurization** in automated ML experiments.

Prerequisites

This article assumes that you already know how to configure an automated ML experiment.

IMPORTANT

The Python commands in this article require the latest `azureml-train-automl` package version.

- [Install the latest `azureml-train-automl` package to your local environment](#).
- For details on the latest `azureml-train-automl` package, see the [release notes](#).

For information about configuration, see the following articles:

- For a code-first experience: [Configure automated ML experiments by using the Azure Machine Learning SDK for Python](#).
- For a low-code or no-code experience: [Create, review, and deploy automated machine learning models by using the Azure Machine Learning studio](#).

Configure featurization

In every automated machine learning experiment, [automatic scaling and normalization techniques](#) are applied to your data by default. These techniques are types of featurization that help *certain* algorithms that are sensitive to features on different scales. You can enable more featurization, such as *missing-values imputation*, *encoding*, and *transforms*.

NOTE

Steps for automated machine learning featurization (such as feature normalization, handling missing data, or converting text to numeric) become part of the underlying model. When you use the model for predictions, the same featurization steps that are applied during training are applied to your input data automatically.

For experiments that you configure with the Python SDK, you can enable or disable the featurization setting and further specify the featurization steps to be used for your experiment. If you're using the Azure Machine Learning studio, see the [steps to enable featurization](#).

The following table shows the accepted settings for `featurization` in the [AutoMLConfig class](#):

FEATURIZATION CONFIGURATION	DESCRIPTION
<code>"featurization": "auto"</code>	Specifies that, as part of preprocessing, data guardrails and featurization steps are to be done automatically. This setting is the default.
<code>"featurization": "off"</code>	Specifies that featurization steps are not to be done automatically.
<code>"featurization": "FeaturizationConfig"</code>	Specifies that customized featurization steps are to be used. Learn how to customize featurization .

Automatic featurization

The following table summarizes techniques that are automatically applied to your data. These techniques are applied for experiments that are configured by using the SDK or the studio UI. To disable this behavior, set

`"featurization": "off"` in your `AutoMLConfig` object.

NOTE

If you plan to export your AutoML-created models to an [ONNX model](#), only the featurization options indicated with an asterisk (*) are supported in the ONNX format. Learn more about [converting models to ONNX](#).

FEATURIZATION STEPS	DESCRIPTION
Drop high cardinality or no variance features*	Drop these features from training and validation sets. Applies to features with all values missing, with the same value across all rows, or with high cardinality (for example, hashes, IDs, or GUIDs).
Impute missing values*	For numeric features, impute with the average of values in the column. For categorical features, impute with the most frequent value.

FEATURIZATION STEPS	DESCRIPTION
Generate more features*	<p>For DateTime features: Year, Month, Day, Day of week, Day of year, Quarter, Week of the year, Hour, Minute, Second.</p> <p><i>For forecasting tasks</i>, these additional DateTime features are created: ISO year, Half - half-year, Calendar month as string, Week, Day of week as string, Day of quarter, Day of year, AM/PM (0 if hour is before noon (12 pm), 1 otherwise), AM/PM as string, Hour of day (12-hr basis)</p> <p>For Text features: Term frequency based on unigrams, bigrams, and trigrams. Learn more about how this is done with BERT.</p>
Transform and encode*	<p>Transform numeric features that have few unique values into categorical features.</p> <p>One-hot encoding is used for low-cardinality categorical features. One-hot-hash encoding is used for high-cardinality categorical features.</p>
Word embeddings	A text featurizer converts vectors of text tokens into sentence vectors by using a pre-trained model. Each word's embedding vector in a document is aggregated with the rest to produce a document feature vector.
Cluster Distance	Trains a k-means clustering model on all numeric columns. Produces k new features (one new numeric feature per cluster) that contain the distance of each sample to the centroid of each cluster.

In every automated machine learning experiment, your data is automatically scaled or normalized to help algorithms perform well. During model training, one of the following scaling or normalization techniques are applied to each model.

SCALING & PROCESSING	DESCRIPTION
StandardScaleWrapper	Standardize features by removing the mean and scaling to unit variance
MinMaxScalar	Transforms features by scaling each feature by that column's minimum and maximum
MaxAbsScaler	Scale each feature by its maximum absolute value
RobustScalar	Scales features by their quantile range
PCA	Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space
TruncatedSVDWrapper	This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition, which means it can work with <code>scipy.sparse</code> matrices efficiently

SCALING & PROCESSING	DESCRIPTION
SparseNormalizer	Each sample (that is, each row of the data matrix) with at least one non-zero component is rescaled independently of other samples so that its norm (l1 or l2) equals one

Data guardrails

Data guardrails help you identify potential issues with your data (for example, missing values or [class imbalance](#)). They also help you take corrective actions for improved results.

Data guardrails are applied:

- **For SDK experiments:** When the parameters `"featurization": "auto"` or `validation=auto` are specified in your `AutoMLConfig` object.
- **For studio experiments:** When automatic featurization is enabled.

You can review the data guardrails for your experiment:

- By setting `show_output=True` when you submit an experiment by using the SDK.
- In the studio, on the **Data guardrails** tab of your automated ML run.

Data guardrail states

Data guardrails display one of three states:

STATE	DESCRIPTION
Passed	No data problems were detected and no action is required by you.
Done	Changes were applied to your data. We encourage you to review the corrective actions that AutoML took, to ensure that the changes align with the expected results.
Alerted	A data issue was detected but couldn't be remedied. We encourage you to revise and fix the issue.

Supported data guardrails

The following table describes the data guardrails that are currently supported and the associated statuses that you might see when you submit your experiment:

GUARDRAIL	STATUS	CONDITION FOR TRIGGER
Missing feature values imputation	Passed	No missing feature values were detected in your training data. Learn more about missing-value imputation .
	Done	Missing feature values were detected in your training data and were imputed.

GUARDRAIL	STATUS	CONDITION FOR TRIGGER
High cardinality feature detection	Passed	Your inputs were analyzed, and no high-cardinality features were detected.
	Done	High-cardinality features were detected in your inputs and were handled.
Validation split handling	Done	<p>The validation configuration was set to <code>'auto'</code> and the training data contained <i>fewer than 20,000 rows</i>. Each iteration of the trained model was validated by using cross-validation. Learn more about validation data.</p> <p>The validation configuration was set to <code>'auto'</code>, and the training data contained <i>more than 20,000 rows</i>. The input data has been split into a training dataset and a validation dataset for validation of the model.</p>
Class balancing detection	Passed	Your inputs were analyzed, and all classes are balanced in your training data. A dataset is considered to be balanced if each class has good representation in the dataset, as measured by number and ratio of samples.
	Alerted	Imbalanced classes were detected in your inputs. To fix model bias, fix the balancing problem. Learn more about imbalanced data .
	Done	Imbalanced classes were detected in your inputs and the sweeping logic has determined to apply balancing.
Memory issues detection	Passed	The selected values (horizon, lag, rolling window) were analyzed, and no potential out-of-memory issues were detected. Learn more about time-series forecasting configurations .
	Done	The selected values (horizon, lag, rolling window) were analyzed and will potentially cause your experiment to run out of memory. The lag or rolling-window configurations have been turned off.

GUARDRAIL	STATUS	CONDITION FOR TRIGGER
Frequency detection	Passed	The time series was analyzed, and all data points are aligned with the detected frequency.
	Done	The time series was analyzed, and data points that don't align with the detected frequency were detected. These data points were removed from the dataset.
Cross validation	Done	In order to accurately evaluate the model(s) trained by AutoML, we leverage a dataset that the model is not trained on. Hence, if the user doesn't provide an explicit validation dataset, a part of the training dataset is used to achieve this. For smaller datasets (fewer than 20,000 samples), cross-validation is leveraged, else a single hold-out set is split from the training data to serve as the validation dataset. Hence, for your input data we leverage cross-validation with 10 folds, if the number of training samples are fewer than 1000, and 3 folds in all other cases.
Train-Test data split	Done	In order to accurately evaluate the model(s) trained by AutoML, we leverage a dataset that the model is not trained on. Hence, if the user doesn't provide an explicit validation dataset, a part of the training dataset is used to achieve this. For smaller datasets (fewer than 20,000 samples), cross-validation is leveraged, else a single hold-out set is split from the training data to serve as the validation dataset. Hence, your input data has been split into a training dataset and a holdout validation dataset.
Time Series ID detection	Passed	The data set was analyzed, and no duplicate time index were detected.
	Fixed	Multiple time series were found in the dataset, and the time series identifiers were automatically created for your dataset.
Time series aggregation	Passed	The dataset frequency is aligned with the user specified frequency. No aggregation was performed.
	Fixed	The data was aggregated to comply with user provided frequency.

GUARDRAIL	STATUS	CONDITION FOR TRIGGER
Short series handling	Passed	Automated ML detected enough data points for each series in the input data to continue with training.
	Fixed	Automated ML detected that some series did not contain enough data points to train a model. To continue with training, these short series have been dropped or padded.

Customize featurization

You can customize your featurization settings to ensure that the data and features that are used to train your ML model result in relevant predictions.

To customize featurizations, specify `"featurization": FeaturizationConfig` in your `AutoMLConfig` object. If you're using the Azure Machine Learning studio for your experiment, see the [how-to article](#). To customize featurization for forecastings task types, refer to the [forecasting how-to](#).

Supported customizations include:

CUSTOMIZATION	DEFINITION
Column purpose update	Override the autodetected feature type for the specified column.
Transformer parameter update	Update the parameters for the specified transformer. Currently supports <i>Imputer</i> (mean, most frequent, and median) and <i>HashOneHotEncoder</i> .
Drop columns	Specifies columns to drop from being featurized.
Block transformers	Specifies block transformers to be used in the featurization process.

NOTE

The `drop_columns` functionality is deprecated as of SDK version 1.19. Drop columns from your dataset as part of data cleansing, prior to consuming it in your automated ML experiment.

Create the `FeaturizationConfig` object by using API calls:

```
featurization_config = FeaturizationConfig()
featurization_config.blocked_transformers = ['LabelEncoder']
featurization_config.drop_columns = ['aspiration', 'stroke']
featurization_config.add_column_purpose('engine-size', 'Numeric')
featurization_config.add_column_purpose('body-style', 'CategoricalHash')
#default strategy mean, add transformer param for for 3 columns
featurization_config.add_transformer_params('Imputer', ['engine-size'], {"strategy": "median"})
featurization_config.add_transformer_params('Imputer', ['city-mpg'], {"strategy": "median"})
featurization_config.add_transformer_params('Imputer', ['bore'], {"strategy": "most_frequent"})
featurization_config.add_transformer_params('HashOneHotEncoder', [], {"number_of_bits": 3})
```

Featurization transparency

Every AutoML model has featurization automatically applied. Featurization includes automated feature engineering (when `"featurization": "auto"`) and scaling and normalization, which then impacts the selected algorithm and its hyperparameter values. AutoML supports different methods to ensure you have visibility into what was applied to your model.

Consider this forecasting example:

- There are four input features: A (Numeric), B (Numeric), C (Numeric), D (DateTime).
- Numeric feature C is dropped because it is an ID column with all unique values.
- Numeric features A and B have missing values and hence are imputed by the mean.
- DateTime feature D is featurized into 11 different engineered features.

To get this information, use the `fitted_model` output from your automated ML experiment run.

```
automl_config = AutoMLConfig(...)  
automl_run = experiment.submit(automl_config ...)  
best_run, fitted_model = automl_run.get_output()
```

Automated feature engineering

The `get_engineered_feature_names()` returns a list of engineered feature names.

NOTE

Use 'timeseriestransformer' for task='forecasting', else use 'datatransformer' for 'regression' or 'classification' task.

```
fitted_model.named_steps['timeseriestransformer'].get_engineered_feature_names ()
```

This list includes all engineered feature names.

```
['A', 'B', 'A_WASNUL', 'B_WASNUL', 'year', 'half', 'quarter', 'month', 'day', 'hour', 'am_pm', 'hour12',  
'wday', 'qday', 'week']
```

The `get_featurization_summary()` gets a featurization summary of all the input features.

```
fitted_model.named_steps['timeseriestransformer'].get_featurization_summary()
```

Output

```
[{'RawFeatureName': 'A',
 'TypeDetected': 'Numeric',
 'Dropped': 'No',
 'EngineeredFeatureCount': 2,
 'Transformations': ['MeanImputer', 'ImputationMarker']},
 {'RawFeatureName': 'B',
 'TypeDetected': 'Numeric',
 'Dropped': 'No',
 'EngineeredFeatureCount': 2,
 'Transformations': ['MeanImputer', 'ImputationMarker']},
 {'RawFeatureName': 'C',
 'TypeDetected': 'Numeric',
 'Dropped': 'Yes',
 'EngineeredFeatureCount': 0,
 'Transformations': []},
 {'RawFeatureName': 'D',
 'TypeDetected': 'DateTime',
 'Dropped': 'No',
 'EngineeredFeatureCount': 11,
 'Transformations':
 ['DateTime','DateTime','DateTime','DateTime','DateTime','DateTime','DateTime','DateTime','DateTime','DateTime','DateTime']}]
```

OUTPUT	DEFINITION
RawFeatureName	Input feature/column name from the dataset provided.
TypeDetected	Detected datatype of the input feature.
Dropped	Indicates if the input feature was dropped or used.
EngineeringFeatureCount	Number of features generated through automated feature engineering transforms.
Transformations	List of transformations applied to input features to generate engineered features.

Scaling and normalization

To understand the scaling/normalization and the selected algorithm with its hyperparameter values, use

```
fitted_model.steps
```

The following sample output is from running `fitted_model.steps` for a chosen run:

```
[('RobustScaler',
  RobustScaler(copy=True,
    quantile_range=[10, 90],
    with_centering=True,
    with_scaling=True)),

  ('LogisticRegression',
  LogisticRegression(C=0.18420699693267145, class_weight='balanced',
    dual=False,
    fit_intercept=True,
    intercept_scaling=1,
    max_iter=100,
    multi_class='multinomial',
    n_jobs=1, penalty='l2',
    random_state=None,
    solver='newton-cg',
    tol=0.0001,
    verbose=0,
    warm_start=False))]
```

To get more details, use this helper function:

```
from pprint import pprint

def print_model(model, prefix=""):
    for step in model.steps:
        print(prefix + step[0])
        if hasattr(step[1], 'estimators') and hasattr(step[1], 'weights'):
            pprint({'estimators': list(e[0] for e in step[1].estimators), 'weights': step[1].weights})
            print()
            for estimator in step[1].estimators:
                print_model(estimator[1], estimator[0]+ ' - ')
        elif hasattr(step[1], '_base_learners') and hasattr(step[1], '_meta_learner'):
            print("\nMeta Learner")
            pprint(step[1]._meta_learner)
            print()
            for estimator in step[1]._base_learners:
                print_model(estimator[1], estimator[0]+ ' - ')
        else:
            pprint(step[1].get_params())
            print()
```

This helper function returns the following output for a particular run using

`LogisticRegression with RobustScalar` as the specific algorithm.

```
RobustScaler
{'copy': True,
'quantile_range': [10, 90],
'with_centering': True,
'with_scaling': True}

LogisticRegression
{'C': 0.18420699693267145,
'class_weight': 'balanced',
'dual': False,
'fit_intercept': True,
'intercept_scaling': 1,
'max_iter': 100,
'multi_class': 'multinomial',
'n_jobs': 1,
'penalty': 'l2',
'random_state': None,
'solver': 'newton-cg',
'tol': 0.0001,
'verbose': 0,
'warm_start': False}
```

Predict class probability

Models produced using automated ML all have wrapper objects that mirror functionality from their open-source origin class. Most classification model wrapper objects returned by automated ML implement the `predict_proba()` function, which accepts an array-like or sparse matrix data sample of your features (X values), and returns an n-dimensional array of each sample and its respective class probability.

Assuming you have retrieved the best run and fitted model using the same calls from above, you can call `predict_proba()` directly from the fitted model, supplying an `X_test` sample in the appropriate format depending on the model type.

```
best_run, fitted_model = automl_run.get_output()
class_prob = fitted_model.predict_proba(X_test)
```

If the underlying model does not support the `predict_proba()` function or the format is incorrect, a model class-specific exception will be thrown. See the [RandomForestClassifier](#) and [XGBoost](#) reference docs for examples of how this function is implemented for different model types.

BERT integration in automated ML

[BERT](#) is used in the featurization layer of automated ML. In this layer, if a column contains free text or other types of data like timestamps or simple numbers, then featurization is applied accordingly.

For BERT, the model is fine-tuned and trained utilizing the user-provided labels. From here, document embeddings are output as features alongside others, like timestamp-based features, day of week.

Learn how to [set up natural language processing \(NLP\) experiments that also use BERT with automated ML](#).

Steps to invoke BERT

In order to invoke BERT, set `enable_dnn: True` in your `automl_settings` and use a GPU compute (`vm_size = "STANDARD_NC6"` or a higher GPU). If a CPU compute is used, then instead of BERT, AutoML enables the BiLSTM DNN featurizer.

Automated ML takes the following steps for BERT.

1. **Preprocessing and tokenization of all text columns.** For example, the "StringCast" transformer can

be found in the final model's featurization summary. An example of how to produce the model's featurization summary can be found in [this notebook](#).

2. **Concatenate all text columns into a single text column**, hence the `StringConcatTransformer` in the final model.

Our implementation of BERT limits total text length of a training sample to 128 tokens. That means, all text columns when concatenated, should ideally be at most 128 tokens in length. If multiple columns are present, each column should be pruned so this condition is satisfied. Otherwise, for concatenated columns of length > 128 tokens BERT's tokenizer layer truncates this input to 128 tokens.

3. **As part of feature sweeping, AutoML compares BERT against the baseline (bag of words features) on a sample of the data.** This comparison determines if BERT would give accuracy improvements. If BERT performs better than the baseline, AutoML then uses BERT for text featurization for the whole data. In that case, you will see the `PretrainedTextDNNTransformer` in the final model.

BERT generally runs longer than other featurizers. For better performance, we recommend using "STANDARD_NC24r" or "STANDARD_NC24rs_V3" for their RDMA capabilities.

AutoML will distribute BERT training across multiple nodes if they are available (upto a max of eight nodes). This can be done in your `AutoMLConfig` object by setting the `max_concurrent_iterations` parameter to higher than 1.

Supported languages for BERT in AutoML

AutoML currently supports around 100 languages and depending on the dataset's language, AutoML chooses the appropriate BERT model. For German data, we use the German BERT model. For English, we use the English BERT model. For all other languages, we use the multilingual BERT model.

In the following code, the German BERT model is triggered, since the dataset language is specified to `'deu'`, the three letter language code for German according to [ISO classification](#):

```
from azureml.automl.core.featurization import FeaturizationConfig

featurization_config = FeaturizationConfig(dataset_language='deu')

automl_settings = {
    "experiment_timeout_minutes": 120,
    "primary_metric": 'accuracy',
    # All other settings you want to use
    "featurization": featurization_config,

    "enable_dnn": True, # This enables BERT DNN featurizer
    "enable_voting_ensemble": False,
    "enable_stack_ensemble": False
}
```

Next steps

- Learn how to set up your automated ML experiments:
 - For a code-first experience: [Configure automated ML experiments by using the Azure Machine Learning SDK](#).
 - For a low-code or no-code experience: [Create your automated ML experiments in the Azure Machine Learning studio](#).
- Learn more about [how and where to deploy a model](#).
- Learn more about [how to train a regression model by using automated machine learning](#) or [how to train by using automated machine learning on a remote resource](#).

Evaluate automated machine learning experiment results

9/22/2022 • 25 minutes to read • [Edit Online](#)

In this article, learn how to evaluate and compare models trained by your automated machine learning (automated ML) experiment. Over the course of an automated ML experiment, many jobs are created and each job creates a model. For each model, automated ML generates evaluation metrics and charts that help you measure the model's performance.

For example, automated ML generates the following charts based on experiment type.

CLASSIFICATION	REGRESSION/FORECASTING
Confusion matrix	Residuals histogram
Receiver operating characteristic (ROC) curve	Predicted vs. true
Precision-recall (PR) curve	Forecast horizon (preview)
Lift curve	
Cumulative gains curve	
Calibration curve	

Prerequisites

- An Azure subscription. (If you don't have an Azure subscription, [create a free account](#) before you begin)
- An Azure Machine Learning experiment created with either:
 - The [Azure Machine Learning studio](#) (no code required)
 - The [Azure Machine Learning Python SDK](#)

View job results

After your automated ML experiment completes, a history of the jobs can be found via:

- A browser with [Azure Machine Learning studio](#)
- A Jupyter notebook using the [JobDetails Jupyter widget](#)

The following steps and video, show you how to view the run history and model evaluation metrics and charts in the studio:

1. [Sign into the studio](#) and navigate to your workspace.
2. In the left menu, select **Experiments**.
3. Select your experiment from the list of experiments.
4. In the table at the bottom of the page, select an automated ML job.
5. In the **Models** tab, select the **Algorithm name** for the model you want to evaluate.
6. In the **Metrics** tab, use the checkboxes on the left to view metrics and charts.

The screenshot shows the Microsoft Azure Machine Learning studio interface. On the left, there is a navigation sidebar with various options like 'New', 'Home', 'Notebooks', 'Automated ML', 'Designer', 'Datasets', 'Runs', 'Pipelines', 'Models', 'Endpoints', 'Compute', 'Datastores', and 'Data Labeling'. The 'Home' option is selected. The main area is titled 'Azure Machine Learning studio' and contains four cards: 'Create new' (with a plus icon), 'Notebooks' (with a document icon), 'Automated ML' (with a gear icon), and 'Designer' (with a cube icon). Below these cards is a section titled 'My recent resources' with a table titled 'Runs'. The table has columns for Run, Run ID, Experiment, Status, Submitted time, Submitted by, and Run type. It shows one entry: 'Run 1' with Run ID 'AutoML_6a5762d5-d3b6...', Experiment 'automl_test', Status 'Completed', Submitted time 'Oct 6, 2020 11:34 AM', Submitted by 'Nina Baccam', and Run type 'Automated...'. There are navigation arrows at the bottom of the table.

Classification metrics

Automated ML calculates performance metrics for each classification model generated for your experiment. These metrics are based on the scikit learn implementation.

Many classification metrics are defined for binary classification on two classes, and require averaging over classes to produce one score for multi-class classification. Scikit-learn provides several averaging methods, three of which automated ML exposes: **macro**, **micro**, and **weighted**.

- **Macro** - Calculate the metric for each class and take the unweighted average
- **Micro** - Calculate the metric globally by counting the total true positives, false negatives, and false positives (independent of classes).
- **Weighted** - Calculate the metric for each class and take the weighted average based on the number of samples per class.

While each averaging method has its benefits, one common consideration when selecting the appropriate method is class imbalance. If classes have different numbers of samples, it might be more informative to use a macro average where minority classes are given equal weighting to majority classes. Learn more about [binary vs multiclass metrics in automated ML](#).

The following table summarizes the model performance metrics that automated ML calculates for each classification model generated for your experiment. For more detail, see the scikit-learn documentation linked in the **Calculation** field of each metric.

NOTE

Refer to [image metrics](#) section for additional details on metrics for image classification models.

METRIC	DESCRIPTION	CALCULATION
--------	-------------	-------------

METRIC	DESCRIPTION	CALCULATION
AUC	<p>AUC is the Area under the Receiver Operating Characteristic Curve.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p> <p>Supported metric names include,</p> <ul style="list-style-type: none"> • <code>AUC_macro</code>, the arithmetic mean of the AUC for each class. • <code>AUC_micro</code>, computed by counting the total true positives, false negatives, and false positives. • <code>AUC_weighted</code>, arithmetic mean of the score for each class, weighted by the number of true instances in each class. • <code>AUC_binary</code>, the value of AUC by treating one specific class as <code>true</code> class and combine all other classes as <code>false</code> class. 	Calculation
accuracy	<p>Accuracy is the ratio of predictions that exactly match the true class labels.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p>	Calculation
average_precision	<p>Average precision summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p> <p>Supported metric names include,</p> <ul style="list-style-type: none"> • <code>average_precision_score_macro</code>, the arithmetic mean of the average precision score of each class. • <code>average_precision_score_micro</code>, computed by counting the total true positives, false negatives, and false positives. • <code>average_precision_score_weighted</code>, the arithmetic mean of the average precision score for each class, weighted by the number of true instances in each class. • <code>average_precision_score_binary</code>, the value of average precision by treating one specific class as <code>true</code> class and combine all other classes as <code>false</code> class. 	Calculation

METRIC	DESCRIPTION	CALCULATION
balanced_accuracy	<p>Balanced accuracy is the arithmetic mean of recall for each class.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p>	Calculation
f1_score	<p>F1 score is the harmonic mean of precision and recall. It is a good balanced measure of both false positives and false negatives. However, it does not take true negatives into account.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p> <p>Supported metric names include,</p> <ul style="list-style-type: none"> • <code>f1_score_macro</code> : the arithmetic mean of F1 score for each class. • <code>f1_score_micro</code> : computed by counting the total true positives, false negatives, and false positives. • <code>f1_score_weighted</code> : weighted mean by class frequency of F1 score for each class. • <code>f1_score_binary</code> , the value of f1 by treating one specific class as <code>true</code> class and combine all other classes as <code>false</code> class. 	Calculation
log_loss	<p>This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of the true labels given a probabilistic classifier's predictions.</p> <p>Objective: Closer to 0 the better Range: [0, inf)</p>	Calculation
norm_macro_recall	<p>Normalized macro recall is recall macro-averaged and normalized, so that random performance has a score of 0, and perfect performance has a score of 1.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p>	$\frac{(\text{recall_score_macro} - R)}{(1 - R)}$ <p>where, <code>R</code> is the expected value of <code>recall_score_macro</code> for random predictions.</p> <p><code>R = 0.5</code> for binary classification. <code>R = (1 / C)</code> for C-class classification problems.</p>

METRIC	DESCRIPTION	CALCULATION
matthews_correlation	<p>Matthews correlation coefficient is a balanced measure of accuracy, which can be used even if one class has many more samples than another. A coefficient of 1 indicates perfect prediction, 0 random prediction, and -1 inverse prediction.</p> <p>Objective: Closer to 1 the better Range: [-1, 1]</p>	Calculation
precision	<p>Precision is the ability of a model to avoid labeling negative samples as positive.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p> <p>Supported metric names include,</p> <ul style="list-style-type: none"> • <code>precision_score_macro</code>, the arithmetic mean of precision for each class. • <code>precision_score_micro</code>, computed globally by counting the total true positives and false positives. • <code>precision_score_weighted</code>, the arithmetic mean of precision for each class, weighted by number of true instances in each class. • <code>precision_score_binary</code>, the value of precision by treating one specific class as <code>true</code> class and combine all other classes as <code>false</code> class. 	Calculation
recall	<p>Recall is the ability of a model to detect all positive samples.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p> <p>Supported metric names include,</p> <ul style="list-style-type: none"> • <code>recall_score_macro</code> : the arithmetic mean of recall for each class. • <code>recall_score_micro</code> : computed globally by counting the total true positives, false negatives and false positives. • <code>recall_score_weighted</code> : the arithmetic mean of recall for each class, weighted by number of true instances in each class. • <code>recall_score_binary</code> , the value of recall by treating one specific class as <code>true</code> class and combine all other classes as <code>false</code> class. 	Calculation

METRIC	DESCRIPTION	CALCULATION
weighted_accuracy	<p>Weighted accuracy is accuracy where each sample is weighted by the total number of samples belonging to the same class.</p> <p>Objective: Closer to 1 the better Range: [0, 1]</p>	Calculation

Binary vs. multiclass classification metrics

Automated ML automatically detects if the data is binary and also allows users to activate binary classification metrics even if the data is multiclass by specifying a `true` class. Multiclass classification metrics will be reported no matter if a dataset has two classes or more than two classes. Binary classification metrics will only be reported when the data is binary, or the users activate the option.

NOTE

When a binary classification task is detected, we use `numpy.unique` to find the set of labels and the later label will be used as the `true` class. Since there is a sorting procedure in `numpy.unique`, the choice of `true` class will be stable.

Note that multiclass classification metrics are intended for multiclass classification. When applied to a binary dataset, these metrics won't treat any class as the `true` class, as you might expect. Metrics that are clearly meant for multiclass are suffixed with `micro`, `macro`, or `weighted`. Examples include `average_precision_score`, `f1_score`, `precision_score`, `recall_score`, and `AUC`. For example, instead of calculating recall as $\frac{tp}{tp + fn}$, the multiclass averaged recall (`micro`, `macro`, or `weighted`) averages over both classes of a binary classification dataset. This is equivalent to calculating the recall for the `true` class and the `false` class separately, and then taking the average of the two.

Besides, although automatic detection of binary classification is supported, it is still recommended to always specify the `true` class manually to make sure the binary classification metrics are calculated for the correct class.

To activate metrics for binary classification datasets when the dataset itself is multiclass, users only need to specify the class to be treated as `true` class and these metrics will be calculated.

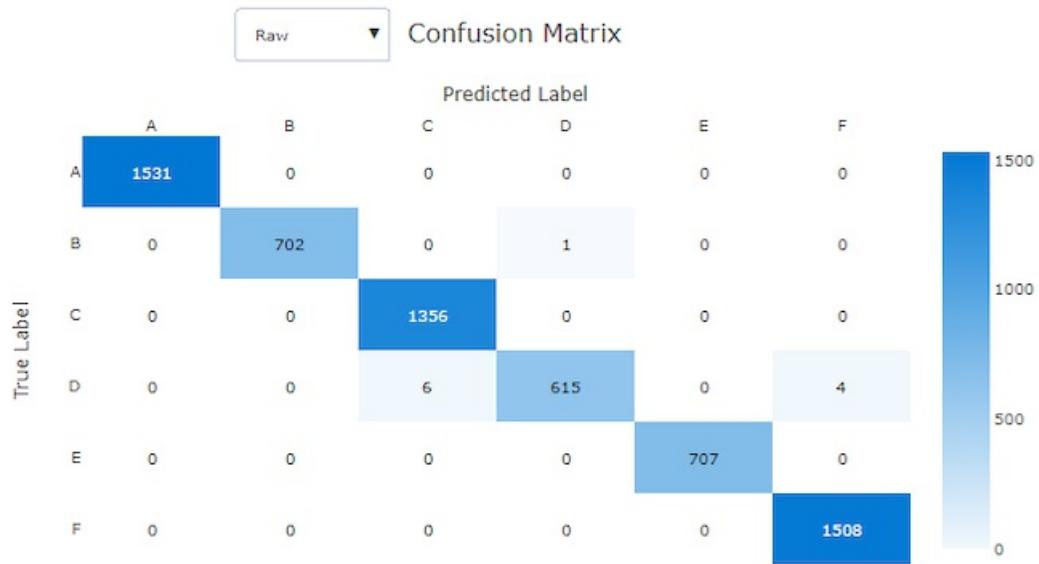
Confusion matrix

Confusion matrices provide a visual for how a machine learning model is making systematic errors in its predictions for classification models. The word "confusion" in the name comes from a model "confusing" or mislabeling samples. A cell at row `i` and column `j` in a confusion matrix contains the number of samples in the evaluation dataset that belong to class `c_i` and were classified by the model as class `c_j`.

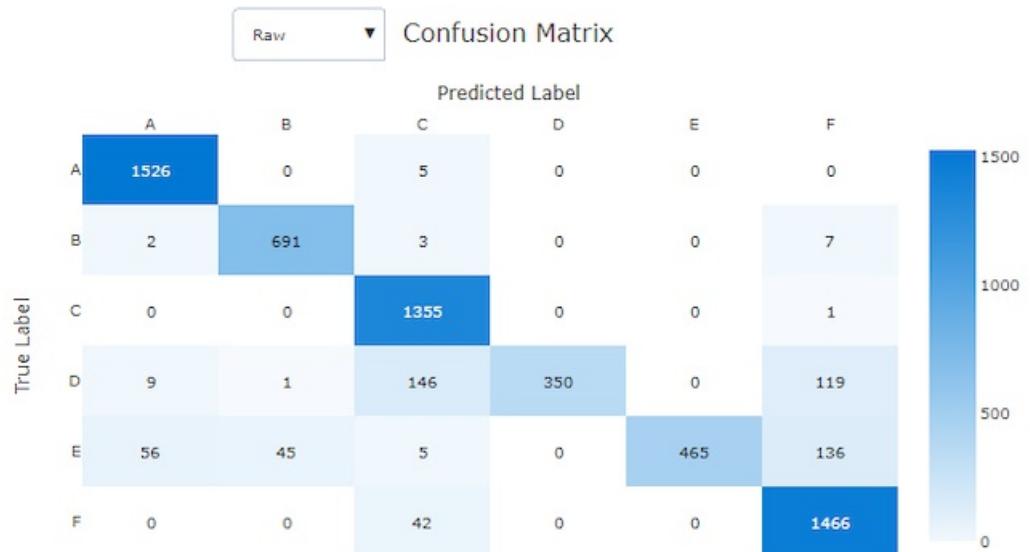
In the studio, a darker cell indicates a higher number of samples. Selecting **Normalized** view in the dropdown will normalize over each matrix row to show the percent of class `c_i` predicted to be class `c_j`. The benefit of the default **Raw** view is that you can see whether imbalance in the distribution of actual classes caused the model to misclassify samples from the minority class, a common issue in imbalanced datasets.

The confusion matrix of a good model will have most samples along the diagonal.

Confusion matrix for a good model



Confusion matrix for a bad model



ROC curve

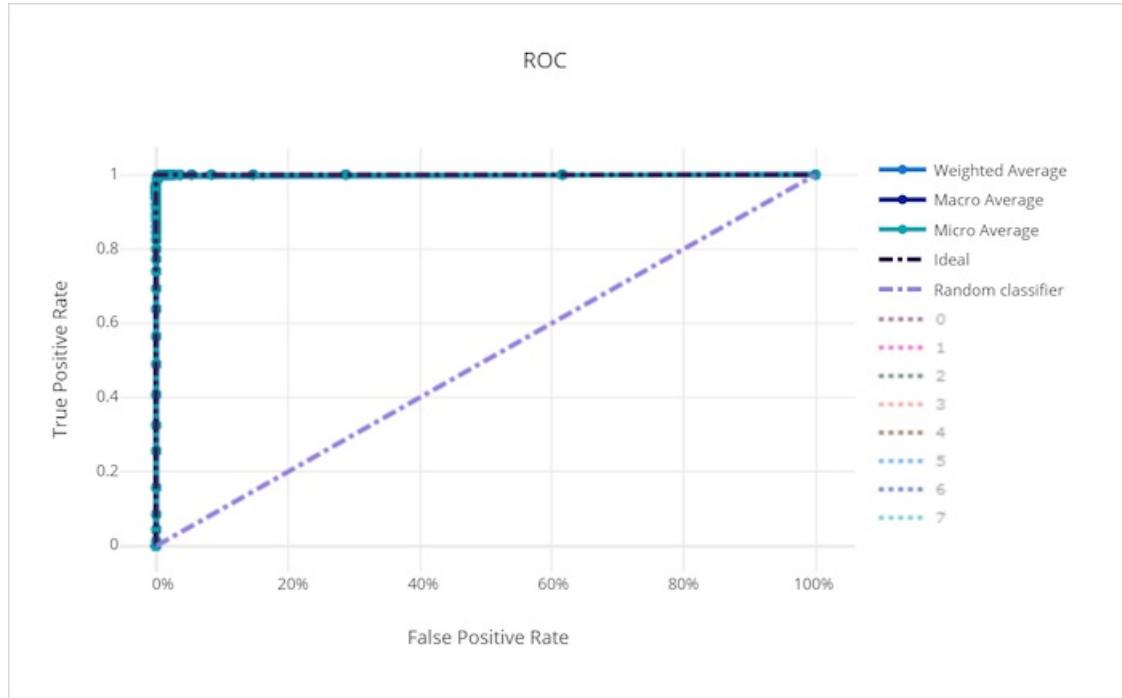
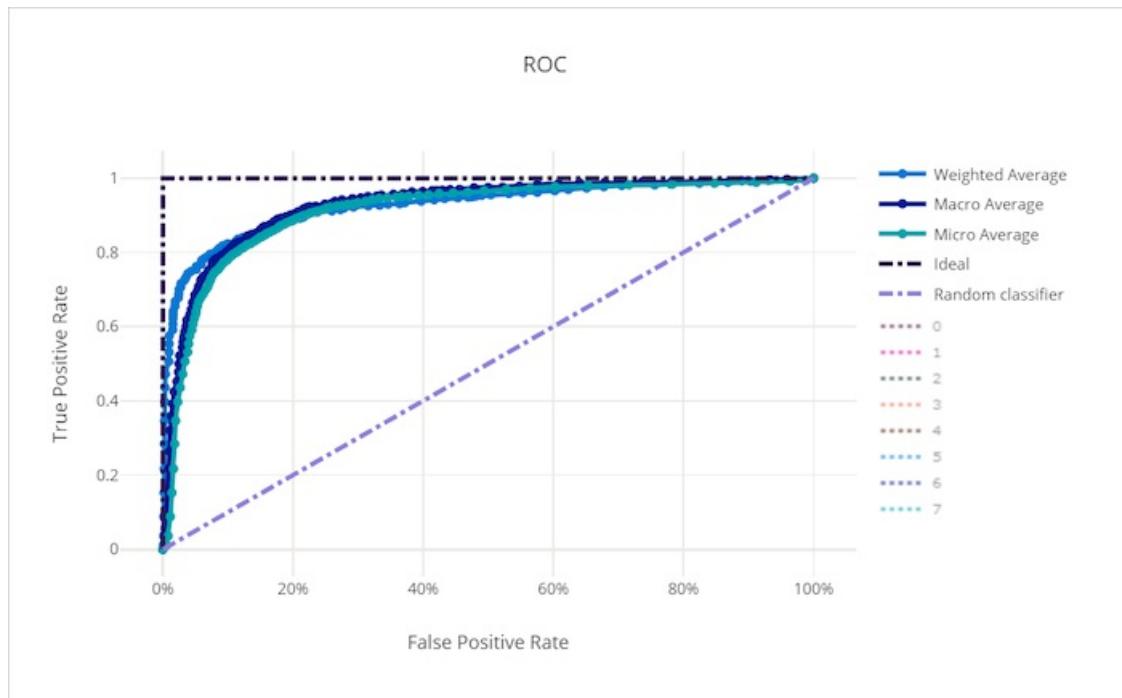
The receiver operating characteristic (ROC) curve plots the relationship between true positive rate (TPR) and false positive rate (FPR) as the decision threshold changes. The ROC curve can be less informative when training models on datasets with high class imbalance, as the majority class can drown out contributions from minority classes.

The area under the curve (AUC) can be interpreted as the proportion of correctly classified samples. More precisely, the AUC is the probability that the classifier ranks a randomly chosen positive sample higher than a randomly chosen negative sample. The shape of the curve gives an intuition for relationship between TPR and FPR as a function of the classification threshold or decision boundary.

A curve that approaches the top-left corner of the chart is approaching a 100% TPR and 0% FPR, the best possible model. A random model would produce an ROC curve along the $y = x$ line from the bottom-left corner to the top-right. A worse than random model would have an ROC curve that dips below the $y = x$ line.

TIP

For classification experiments, each of the line charts produced for automated ML models can be used to evaluate the model per-class or averaged over all classes. You can switch between these different views by clicking on class labels in the legend to the right of the chart.

ROC curve for a good model**ROC curve for a bad model**

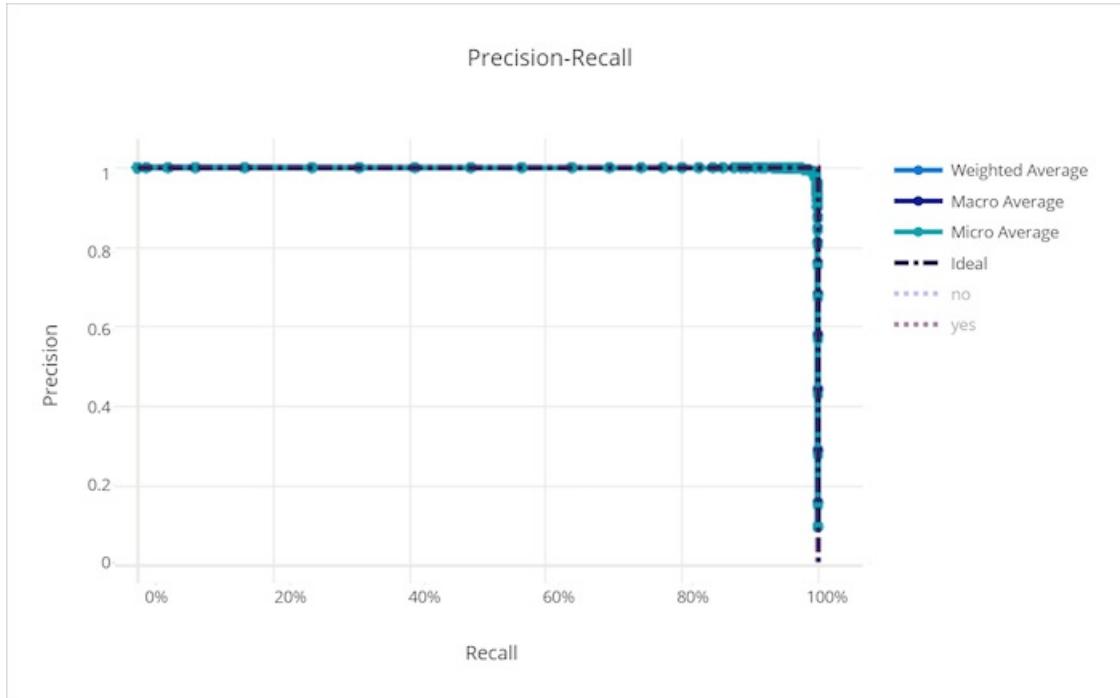
Precision-recall curve

The precision-recall curve plots the relationship between precision and recall as the decision threshold changes. Recall is the ability of a model to detect all positive samples and precision is the ability of a model to avoid labeling negative samples as positive. Some business problems might require higher recall and some higher precision depending on the relative importance of avoiding false negatives vs false positives.

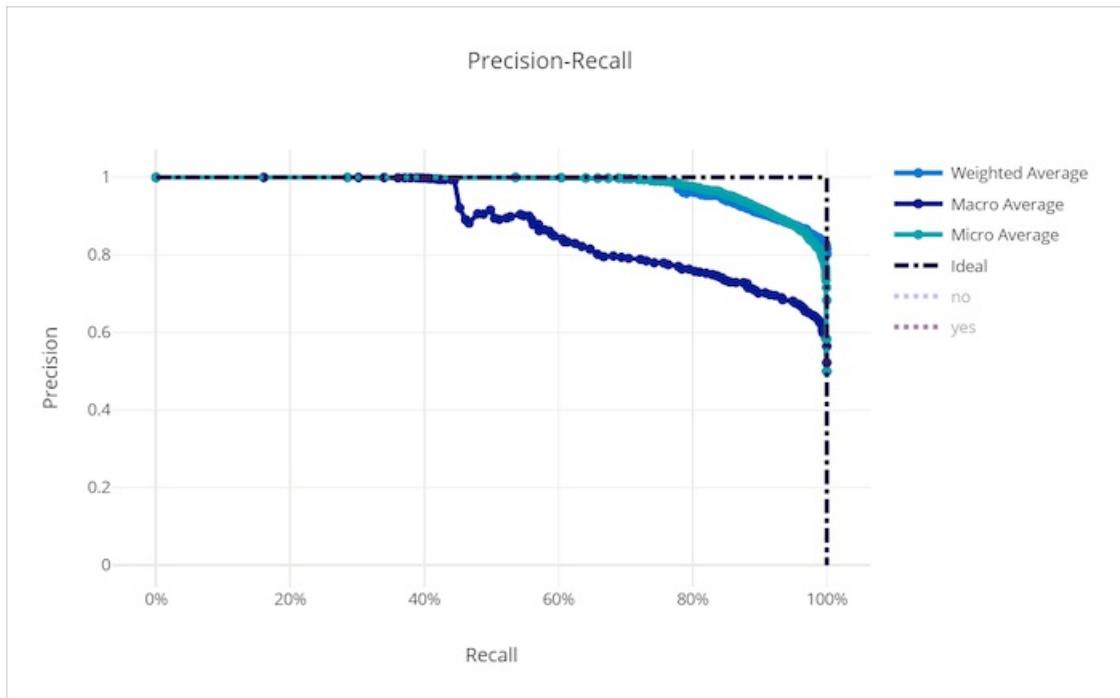
TIP

For classification experiments, each of the line charts produced for automated ML models can be used to evaluate the model per-class or averaged over all classes. You can switch between these different views by clicking on class labels in the legend to the right of the chart.

Precision-recall curve for a good model



Precision-recall curve for a bad model



Cumulative gains curve

The cumulative gains curve plots the percent of positive samples correctly classified as a function of the percent of samples considered where we consider samples in the order of predicted probability.

To calculate gain, first sort all samples from highest to lowest probability predicted by the model. Then take $x\%$ of the highest confidence predictions. Divide the number of positive samples detected in that $x\%$ by the total

number of positive samples to get the gain. Cumulative gain is the percent of positive samples we detect when considering some percent of the data that is most likely to belong to the positive class.

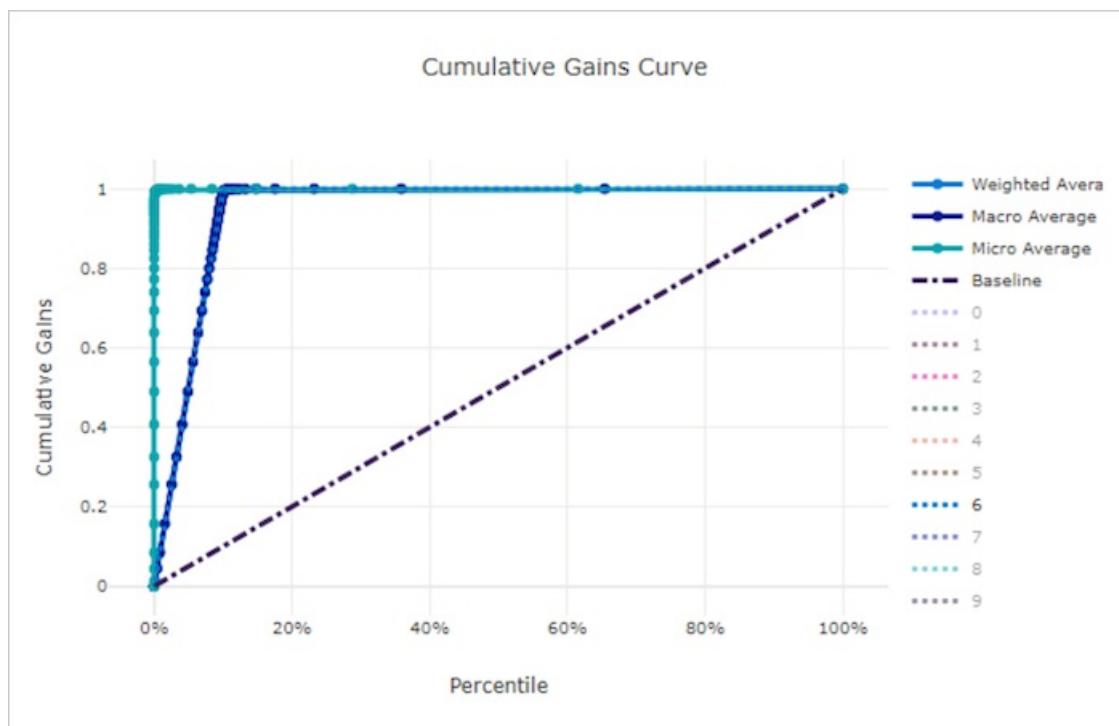
A perfect model will rank all positive samples above all negative samples giving a cumulative gains curve made up of two straight segments. The first is a line with slope $1 / x$ from $(0, 0)$ to $(x, 1)$ where x is the fraction of samples that belong to the positive class ($1 / \text{num_classes}$ if classes are balanced). The second is a horizontal line from $(x, 1)$ to $(1, 1)$. In the first segment, all positive samples are classified correctly and cumulative gain goes to 100% within the first $x\%$ of samples considered.

The baseline random model will have a cumulative gains curve following $y = x$ where for $x\%$ of samples considered only about $x\%$ of the total positive samples were detected. A perfect model for a balanced dataset will have a micro average curve and a macro average line that has slope num_classes until cumulative gain is 100% and then horizontal until the data percent is 100.

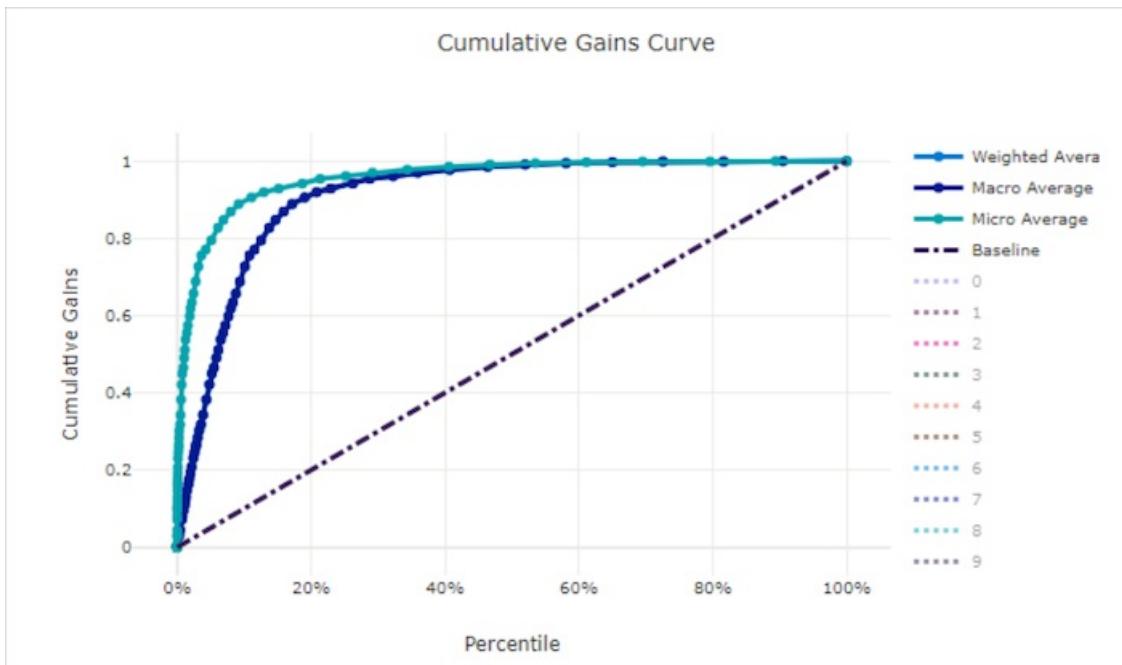
TIP

For classification experiments, each of the line charts produced for automated ML models can be used to evaluate the model per-class or averaged over all classes. You can switch between these different views by clicking on class labels in the legend to the right of the chart.

Cumulative gains curve for a good model



Cumulative gains curve for a bad model



Lift curve

The lift curve shows how many times better a model performs compared to a random model. Lift is defined as the ratio of cumulative gain to the cumulative gain of a random model (which should always be $y = 1$).

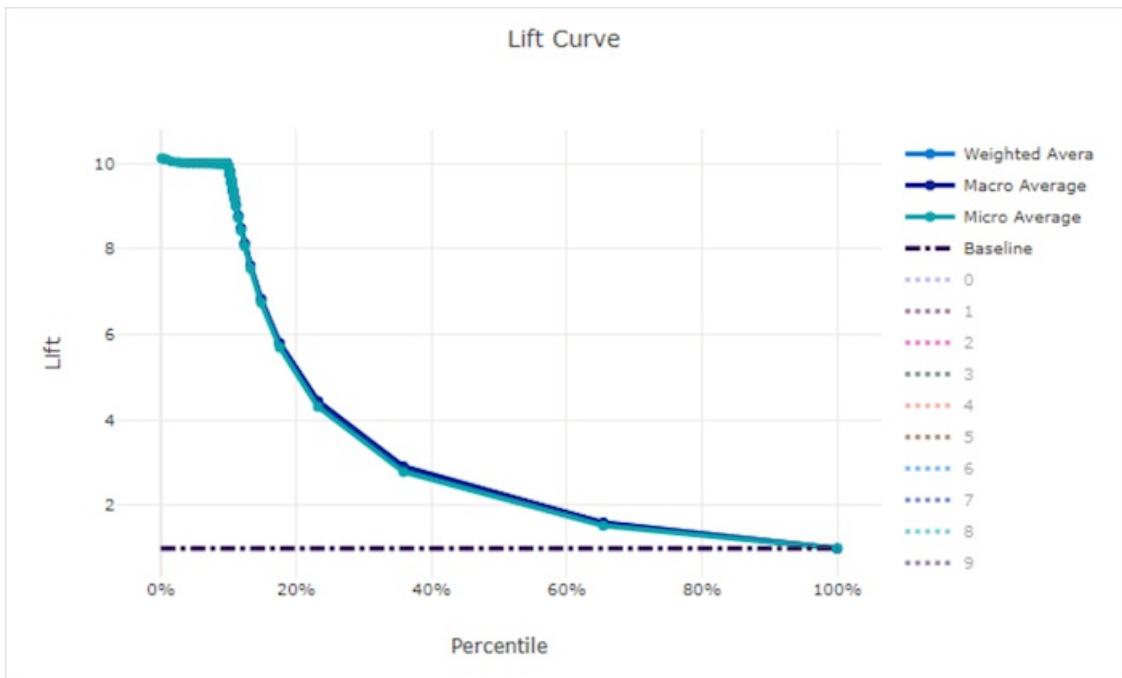
This relative performance takes into account the fact that classification gets harder as you increase the number of classes. (A random model incorrectly predicts a higher fraction of samples from a dataset with 10 classes compared to a dataset with two classes)

The baseline lift curve is the $y = 1$ line where the model performance is consistent with that of a random model. In general, the lift curve for a good model will be higher on that chart and farther from the x-axis, showing that when the model is most confident in its predictions it performs many times better than random guessing.

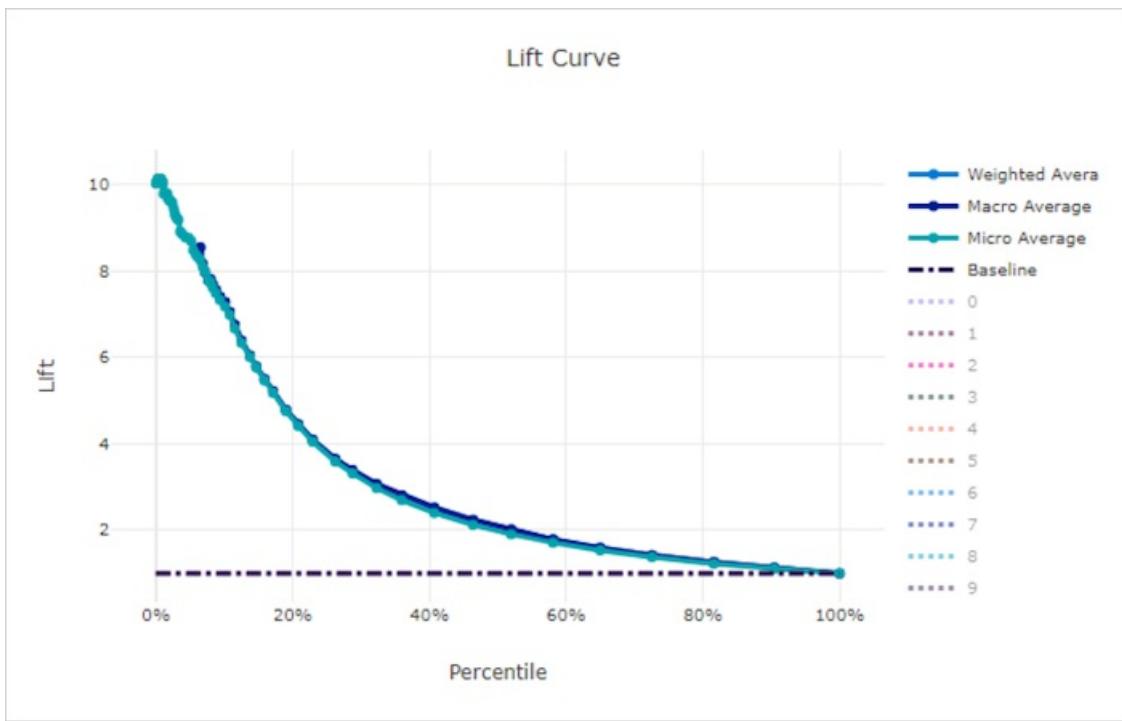
TIP

For classification experiments, each of the line charts produced for automated ML models can be used to evaluate the model per-class or averaged over all classes. You can switch between these different views by clicking on class labels in the legend to the right of the chart.

Lift curve for a good model



Lift curve for a bad model



Calibration curve

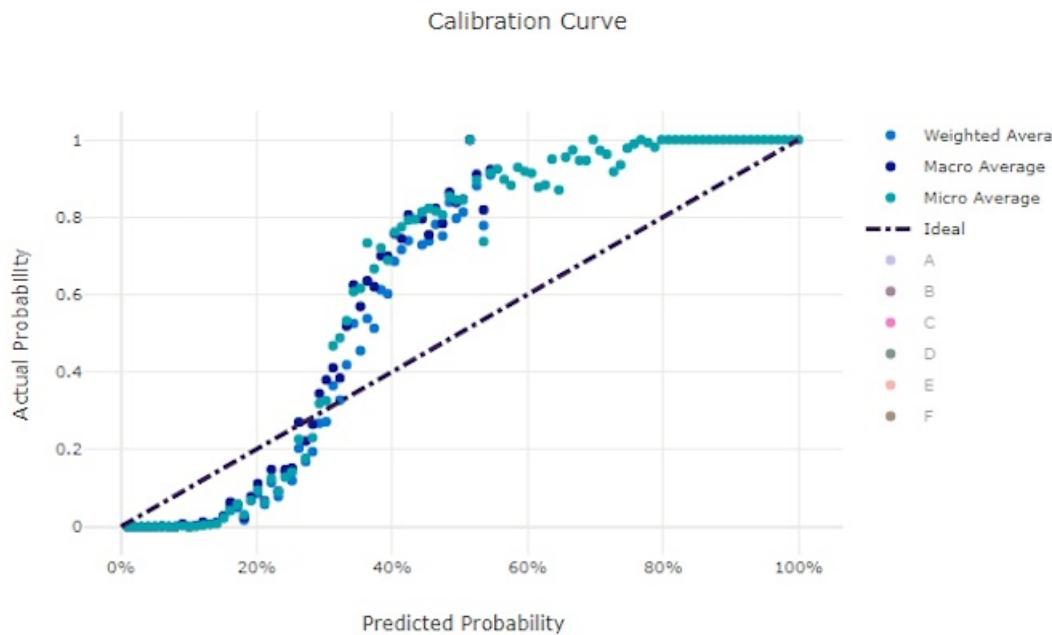
The calibration curve plots a model's confidence in its predictions against the proportion of positive samples at each confidence level. A well-calibrated model will correctly classify 100% of the predictions to which it assigns 100% confidence, 50% of the predictions it assigns 50% confidence, 20% of the predictions it assigns a 20% confidence, and so on. A perfectly calibrated model will have a calibration curve following the $y = x$ line where the model perfectly predicts the probability that samples belong to each class.

An over-confident model will over-predict probabilities close to zero and one, rarely being uncertain about the class of each sample and the calibration curve will look similar to backward "S". An under-confident model will assign a lower probability on average to the class it predicts and the associated calibration curve will look similar to an "S". The calibration curve does not depict a model's ability to classify correctly, but instead its ability to correctly assign confidence to its predictions. A bad model can still have a good calibration curve if the model correctly assigns low confidence and high uncertainty.

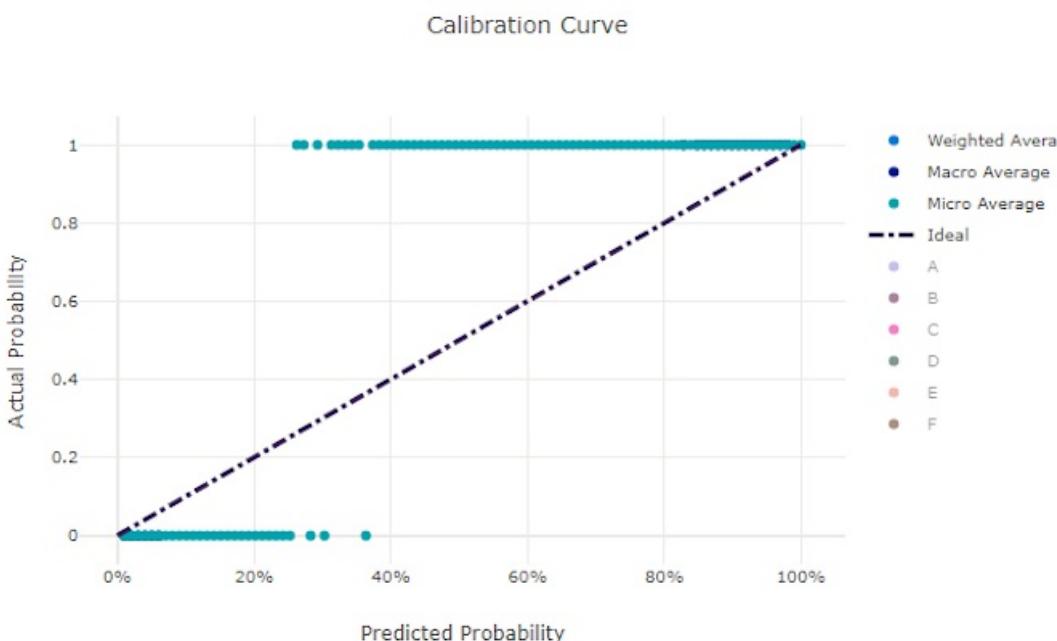
NOTE

The calibration curve is sensitive to the number of samples, so a small validation set can produce noisy results that can be hard to interpret. This does not necessarily mean that the model is not well-calibrated.

Calibration curve for a good model



Calibration curve for a bad model



Regression/forecasting metrics

Automated ML calculates the same performance metrics for each model generated, regardless if it is a regression or forecasting experiment. These metrics also undergo normalization to enable comparison between models trained on data with different ranges. To learn more, see [metric normalization](#).

The following table summarizes the model performance metrics generated for regression and forecasting

experiments. Like classification metrics, these metrics are also based on the scikit learn implementations. The appropriate scikit learn documentation is linked accordingly, in the **Calculation** field.

METRIC	DESCRIPTION	CALCULATION
explained_variance	<p>Explained variance measures the extent to which a model accounts for the variation in the target variable. It is the percent decrease in variance of the original data to the variance of the errors. When the mean of the errors is 0, it is equal to the coefficient of determination (see r2_score below).</p> <p>Objective: Closer to 1 the better Range: (-inf, 1]</p>	Calculation
mean_absolute_error	<p>Mean absolute error is the expected value of absolute value of difference between the target and the prediction.</p> <p>Objective: Closer to 0 the better Range: [0, inf)</p> <p>Types: mean_absolute_error normalized_mean_absolute_error, the mean_absolute_error divided by the range of the data.</p>	Calculation
mean_absolute_percentage_error	<p>Mean absolute percentage error (MAPE) is a measure of the average difference between a predicted value and the actual value.</p> <p>Objective: Closer to 0 the better Range: [0, inf)</p>	
median_absolute_error	<p>Median absolute error is the median of all absolute differences between the target and the prediction. This loss is robust to outliers.</p> <p>Objective: Closer to 0 the better Range: [0, inf)</p> <p>Types: median_absolute_error normalized_median_absolute_error : the median_absolute_error divided by the range of the data.</p>	Calculation

METRIC	DESCRIPTION	CALCULATION
r2_score	<p>R² (the coefficient of determination) measures the proportional reduction in mean squared error (MSE) relative to the total variance of the observed data.</p> <p>Objective: Closer to 1 the better Range: [-1, 1]</p> <p>Note: R² often has the range (-inf, 1]. The MSE can be larger than the observed variance, so R² can have arbitrarily large negative values, depending on the data and the model predictions. Automated ML clips reported R² scores at -1, so a value of -1 for R² likely means that the true R² score is less than -1. Consider the other metrics values and the properties of the data when interpreting a negative R² score.</p>	Calculation
root_mean_squared_error	<p>Root mean squared error (RMSE) is the square root of the expected squared difference between the target and the prediction. For an unbiased estimator, RMSE is equal to the standard deviation.</p> <p>Objective: Closer to 0 the better Range: [0, inf)</p> <p>Types: <input type="checkbox"/> <code>root_mean_squared_error</code> <input checked="" type="checkbox"/> <code>normalized_root_mean_squared_error</code> : the root_mean_squared_error divided by the range of the data.</p>	Calculation
root_mean_squared_log_error	<p>Root mean squared log error is the square root of the expected squared logarithmic error.</p> <p>Objective: Closer to 0 the better Range: [0, inf)</p> <p>Types: <input type="checkbox"/> <code>root_mean_squared_log_error</code> <input checked="" type="checkbox"/> <code>normalized_root_mean_squared_log_error</code> : the root_mean_squared_log_error divided by the range of the data.</p>	Calculation

METRIC	DESCRIPTION	CALCULATION
spearman_correlation	<p>Spearman correlation is a nonparametric measure of the monotonicity of the relationship between two datasets. Unlike the Pearson correlation, the Spearman correlation does not assume that both datasets are normally distributed. Like other correlation coefficients, Spearman varies between -1 and 1 with 0 implying no correlation. Correlations of -1 or 1 imply an exact monotonic relationship.</p> <p>Spearman is a rank-order correlation metric meaning that changes to predicted or actual values will not change the Spearman result if they do not change the rank order of predicted or actual values.</p> <p>Objective: Closer to 1 the better Range: [-1, 1]</p>	Calculation

Metric normalization

Automated ML normalizes regression and forecasting metrics which enables comparison between models trained on data with different ranges. A model trained on a data with a larger range has higher error than the same model trained on data with a smaller range, unless that error is normalized.

While there is no standard method of normalizing error metrics, automated ML takes the common approach of dividing the error by the range of the data: `normalized_error = error / (y_max - y_min)`

NOTE

The range of data is not saved with the model. If you do inference with the same model on a holdout test set, `y_min` and `y_max` may change according to the test data and the normalized metrics may not be directly used to compare the model's performance on training and test sets. You can pass in the value of `y_min` and `y_max` from your training set to make the comparison fair.

When evaluating a forecasting model on time series data, automated ML takes extra steps to ensure that normalization happens per time series ID (grain), because each time series likely has a different distribution of target values.

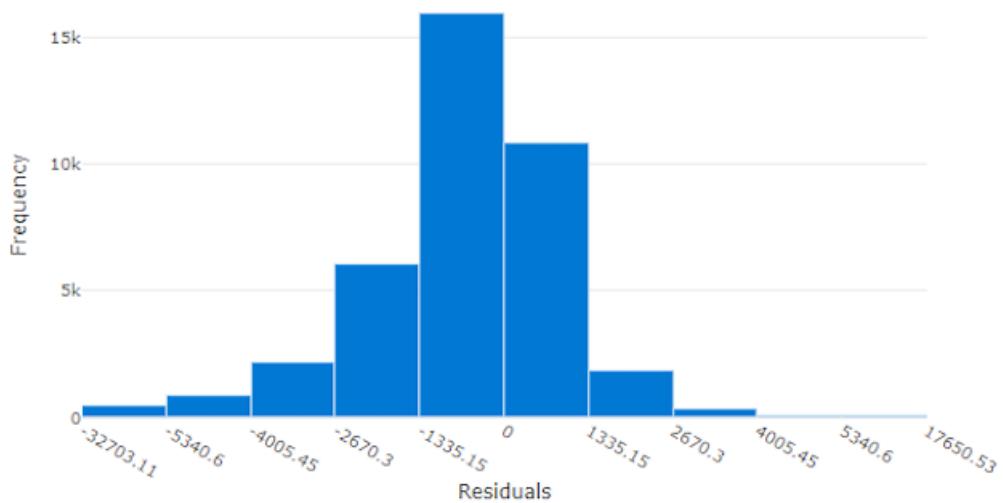
Residuals

The residuals chart is a histogram of the prediction errors (residuals) generated for regression and forecasting experiments. Residuals are calculated as `y_predicted - y_true` for all samples and then displayed as a histogram to show model bias.

In this example, note that both models are slightly biased to predict lower than the actual value. This is not uncommon for a dataset with a skewed distribution of actual targets, but indicates worse model performance. A good model will have a residuals distribution that peaks at zero with few residuals at the extremes. A worse model will have a spread out residuals distribution with fewer samples around zero.

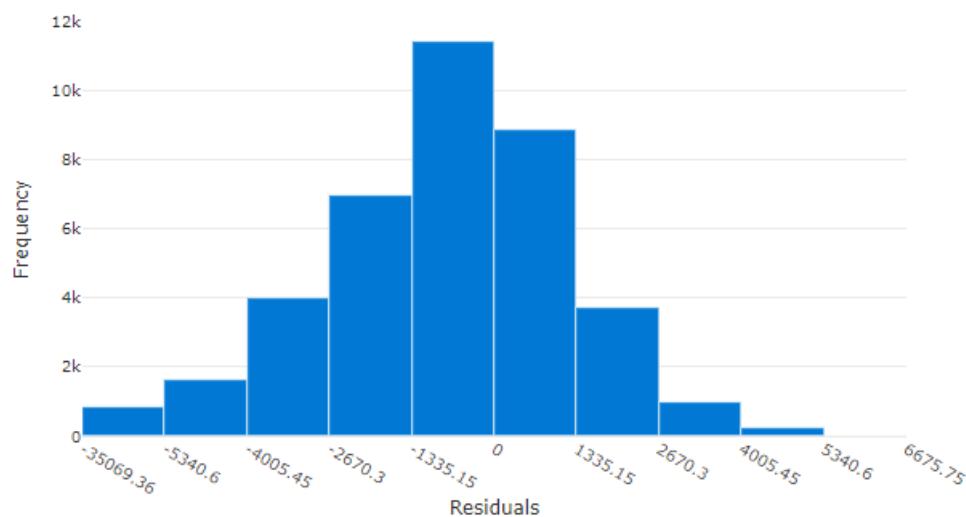
Residuals chart for a good model

Residual Histogram



Residuals chart for a bad model

Residual Histogram



Predicted vs. true

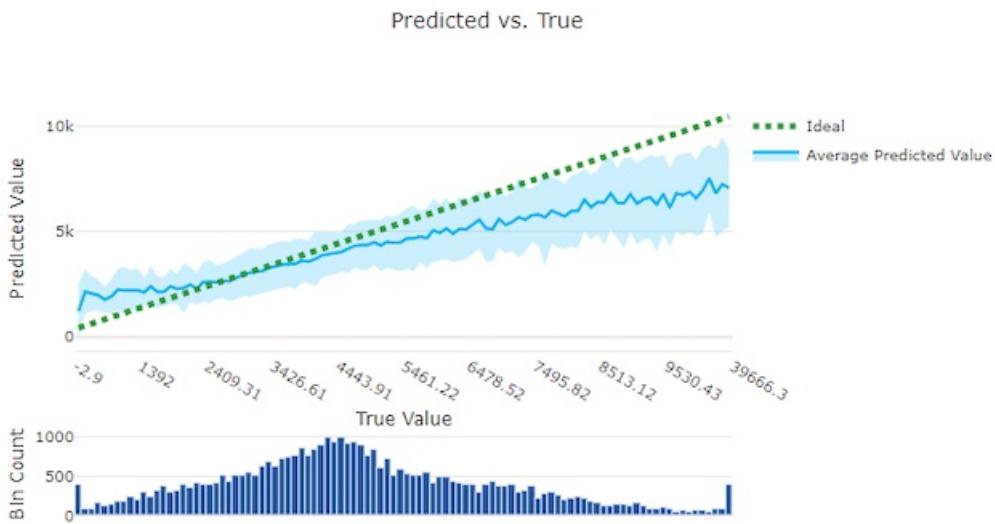
For regression and forecasting experiment the predicted vs. true chart plots the relationship between the target feature (true/actual values) and the model's predictions. The true values are binned along the x-axis and for each bin the mean predicted value is plotted with error bars. This allows you to see if a model is biased toward predicting certain values. The line displays the average prediction and the shaded area indicates the variance of predictions around that mean.

Often, the most common true value will have the most accurate predictions with the lowest variance. The distance of the trend line from the ideal $y = x$ line where there are few true values is a good measure of model performance on outliers. You can use the histogram at the bottom of the chart to reason about the actual data distribution. Including more data samples where the distribution is sparse can improve model performance on

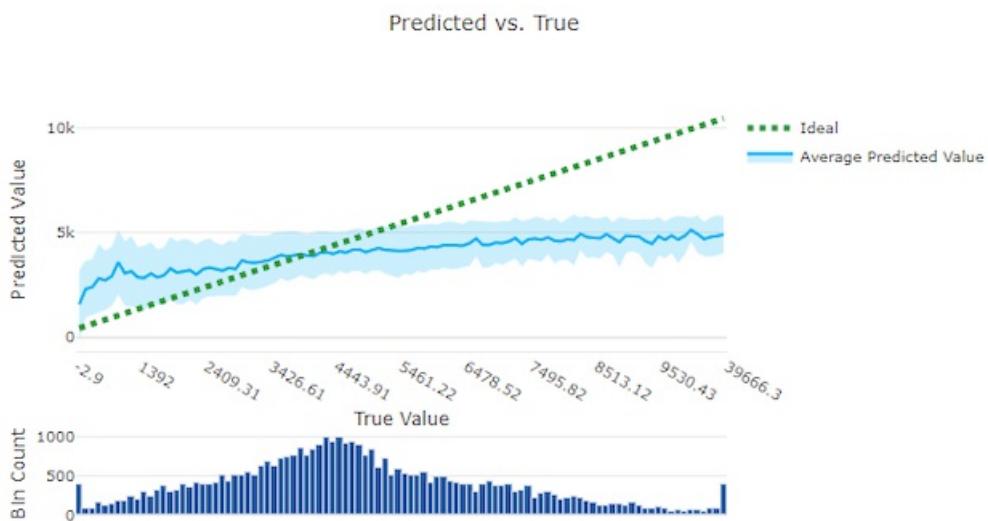
unseen data.

In this example, note that the better model has a predicted vs. true line that is closer to the ideal $y = x$ line.

Predicted vs. true chart for a good model



Predicted vs. true chart for a bad model



Forecast horizon (preview)

For forecasting experiments, the forecast horizon chart plots the relationship between the models predicted value and the actual values mapped over time per cross validation fold, up to 5 folds. The x axis maps time based on the frequency you provided during training setup. The vertical line in the chart marks the forecast horizon point also referred to as the horizon line, which is the time period at which you would want to start generating predictions. To the left of the forecast horizon line, you can view historic training data to better visualize past trends. To the right of the forecast horizon, you can visualize the predictions (the purple line) against the actuals (the blue line) for the different cross validation folds and time series identifiers. The shaded purple area indicates the confidence intervals or variance of predictions around that mean.

You can choose which cross validation fold and time series identifier combinations to display by clicking the edit pencil icon on the top right corner of the chart. Select from the first 5 cross validation folds and up to 20

different time series identifiers to visualize the chart for your various time series.

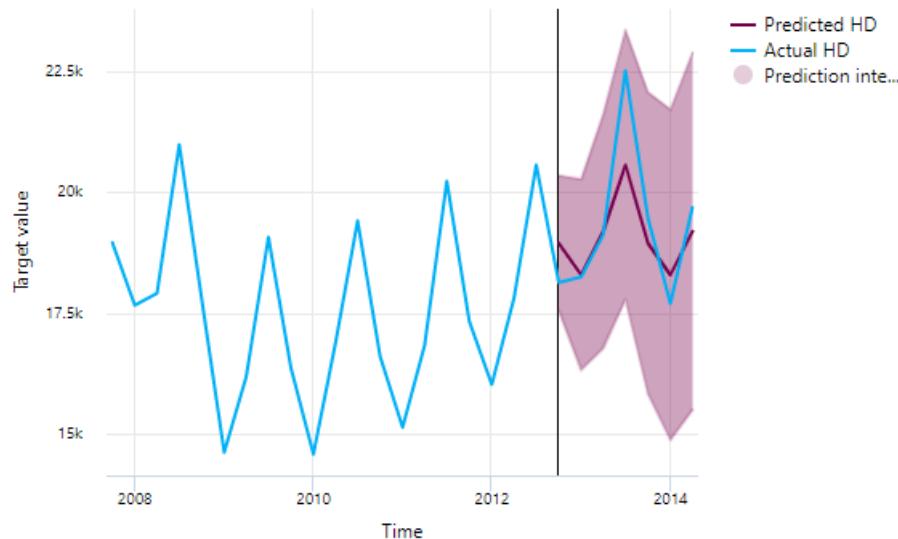
IMPORTANT

This chart is only available for models generated from training and validation data. We allow up to 20 data points before and up to 80 data points after the forecast origin. Visuals for models based on test data are not supported at this time.

Selected cross validation: 3



Forecast horizon (preview)



Metrics for image models (preview)

Automated ML uses the images from the validation dataset for evaluating the performance of the model. The performance of the model is measured at an **epoch-level** to understand how the training progresses. An epoch elapses when an entire dataset is passed forward and backward through the neural network exactly once.

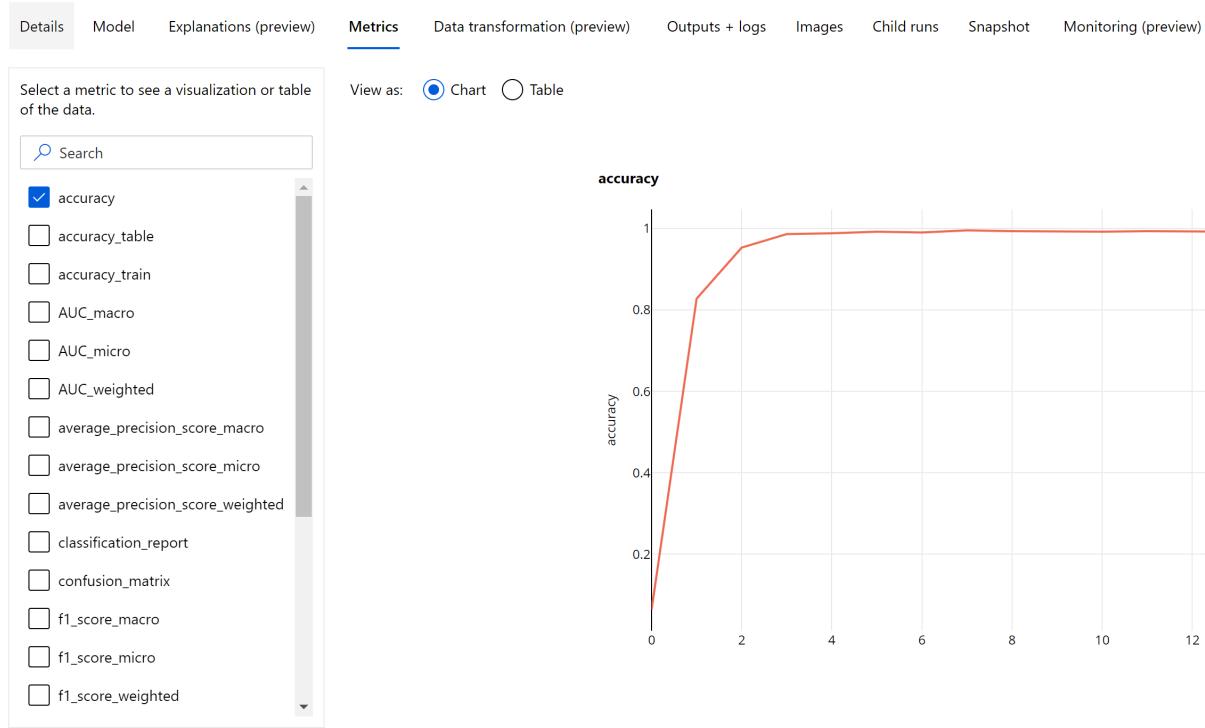
Image classification metrics

The primary metric for evaluation is **accuracy** for binary and multi-class classification models and **IoU (Intersection over Union)** for multilabel classification models. The classification metrics for image classification models are same as those defined in the [classification metrics](#) section. The loss values associated with an epoch are also logged which can help monitor how the training progresses and determine if the model is over-fitting or under-fitting.

Every prediction from a classification model is associated with a confidence score, which indicates the level of confidence with which the prediction was made. Multilabel image classification models are by default evaluated with a score threshold of 0.5 which means only predictions with at least this level of confidence will be considered as a positive prediction for the associated class. Multiclass classification does not use a score threshold but instead, the class with the maximum confidence score is considered as the prediction.

Epoch-level metrics for image classification

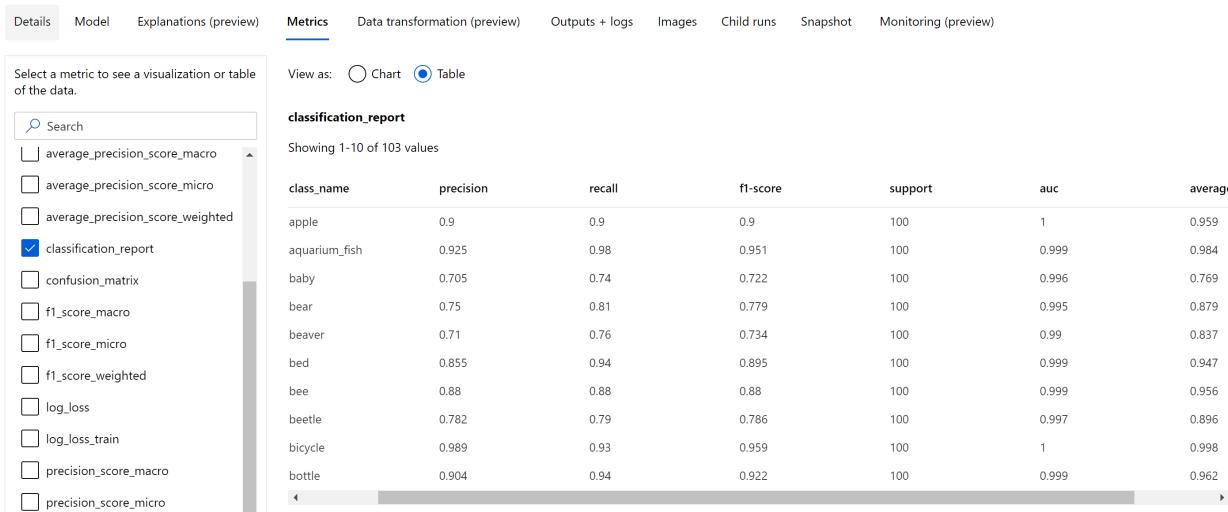
Unlike the classification metrics for tabular datasets, image classification models log all the classification metrics at an epoch-level as shown below.



Summary metrics for image classification

Apart from the scalar metrics that are logged at the epoch level, image classification model also log summary metrics like [confusion matrix](#), [classification charts](#) including ROC curve, precision-recall curve and classification report for the model from the best epoch at which we get the highest primary metric (accuracy) score.

Classification report provides the class-level values for metrics like precision, recall, f1-score, support, auc and average_precision with various level of averaging - micro, macro and weighted as shown below. Please refer to the metrics definitions from the [classification metrics](#) section.



Object detection and instance segmentation metrics

Every prediction from an image object detection or instance segmentation model is associated with a confidence score. The predictions with confidence score greater than score threshold are output as predictions and used in the metric calculation, the default value of which is model specific and can be referred from the [hyperparameter tuning](#) page(`box_score_threshold` hyperparameter).

The metric computation of an image object detection and instance segmentation model is based on an overlap measurement defined by a metric called **IoU** ([Intersection over Union](#)) which is computed by dividing the area of overlap between the ground-truth and the predictions by the area of union of the ground-truth and the predictions. The IoU computed from every prediction is compared with an **overlap threshold** called an IoU threshold which determines how much a prediction should overlap with a user-annotated ground-truth in order

to be considered as a positive prediction. If the IoU computed from the prediction is less than the overlap threshold the prediction would not be considered as a positive prediction for the associated class.

The primary metric for the evaluation of image object detection and instance segmentation models is the **mean average precision (mAP)**. The mAP is the average value of the average precision(AP) across all the classes. Automated ML object detection models support the computation of mAP using the below two popular methods.

Pascal VOC metrics:

Pascal VOC mAP is the default way of mAP computation for object detection/instance segmentation models. Pascal VOC style mAP method calculates the area under a version of the precision-recall curve. First $p(r_i)$, which is precision at recall i is computed for all unique recall values. $p(r_i)$ is then replaced with maximum precision obtained for any recall $r' \geq r_i$. The precision value is monotonically decreasing in this version of the curve. Pascal VOC mAP metric is by default evaluated with an IoU threshold of 0.5. A detailed explanation of this concept is available in this [blog](#).

COCO metrics:

COCO evaluation method uses a 101-point interpolated method for AP calculation along with averaging over ten IoU thresholds. AP@[.5:.95] corresponds to the average AP for IoU from 0.5 to 0.95 with a step size of 0.05. Automated ML logs all the twelve metrics defined by the COCO method including the AP and AR(average recall) at various scales in the application logs while the metrics user interface shows only the mAP at an IoU threshold of 0.5.

TIP

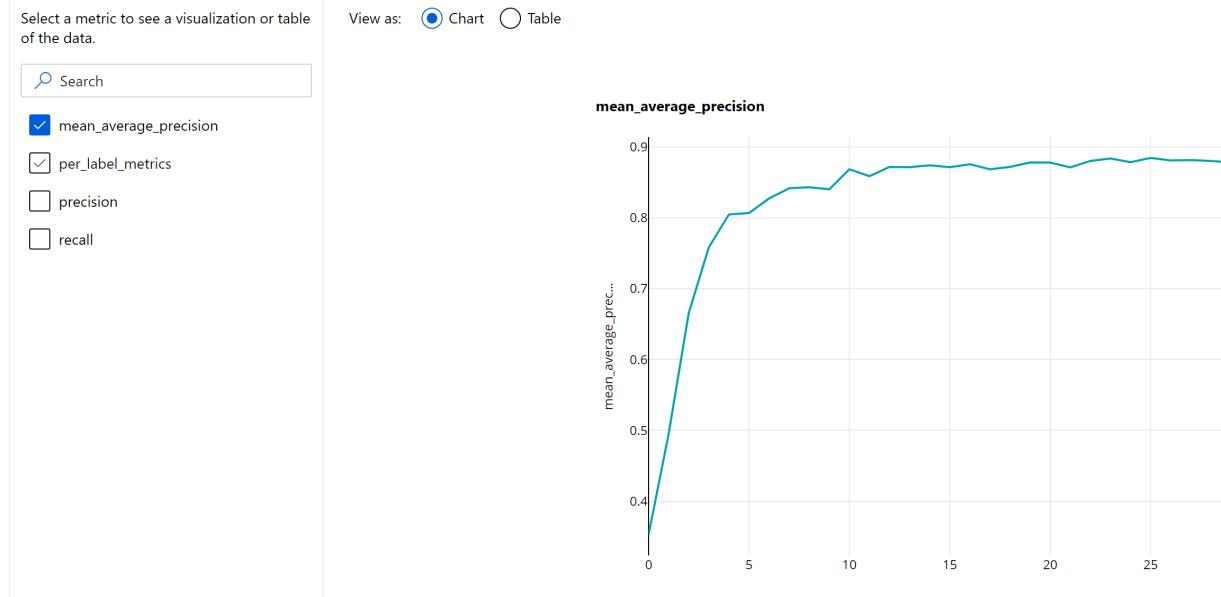
The image object detection model evaluation can use coco metrics if the `validation_metric_type` hyperparameter is set to be 'coco' as explained in the [hyperparameter tuning](#) section.

Epoch-level metrics for object detection and instance segmentation

The mAP, precision and recall values are logged at an epoch-level for image object detection/instance segmentation models. The mAP, precision and recall metrics are also logged at a class level with the name 'per_label_metrics'. The 'per_label_metrics' should be viewed as a table.

NOTE

Epoch-level metrics for precision, recall and per_label_metrics are not available when using the 'coco' method.



Model explanations and feature importances

While model evaluation metrics and charts are good for measuring the general quality of a model, inspecting which dataset features a model used to make its predictions is essential when practicing responsible AI. That's why automated ML provides a model explanations dashboard to measure and report the relative contributions of dataset features. See how to [view the explanations dashboard in the Azure Machine Learning studio](#).

For a code first experience, see how to set up [model explanations for automated ML experiments with the Azure Machine Learning Python SDK](#).

NOTE

Interpretability, best model explanation, is not available for automated ML forecasting experiments that recommend the following algorithms as the best model or ensemble:

- TCNForecaster
- AutoArima
- ExponentialSmoothing
- Prophet
- Average
- Naive
- Seasonal Average
- Seasonal Naive

Next steps

- Try the [automated machine learning model explanation sample notebooks](#).
- For automated ML specific questions, reach out to askautomatedml@microsoft.com.

View automated ML model's training code (preview)

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO: Python SDK azureml v1

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you learn how to view the generated training code from any automated machine learning trained model.

Code generation for automated ML trained models allows you to see the following details that automated ML uses to train and build the model for a specific run.

- Data preprocessing
- Algorithm selection
- Featurization
- Hyperparameters

You can select any automated ML trained model, recommended or child run, and view the generated Python training code that created that specific model.

With the generated model's training code you can,

- **Learn** what featurization process and hyperparameters the model algorithm uses.
- **Track/version/audit** trained models. Store versioned code to track what specific training code is used with the model that's to be deployed to production.
- **Customize** the training code by changing hyperparameters or applying your ML and algorithms skills/experience, and retrain a new model with your customized code.

The following diagram illustrates that you can generate the code for automated ML experiments with all task types. First select a model. The model you selected will be highlighted, then Azure Machine Learning copies the code files used to create the model, and displays them into your notebooks shared folder. From here, you can view and customize the code as needed.

Microsoft > public-test > Jobs > SFDATA > witty_worm_5b6rmqjf

witty_worm_5b6rmqjf Completed

Overview Data guardrails **Models** Outputs + logs Child jobs

Refresh Edit and submit (preview) Create model Cancel Delete Deploy Download Explain model **# View generated code (preview)** ...

Search Created on All filters Clear all

Showing 1-2 of 2 models Page size: 25

Algorithm name	Explained	AUC weighted	Sampling	Created on	Duration
MaxAbsScaler, XGBoostClassifier		0.58901	100.00 %	Jul 5, 2022 9:43 AM	3h 52m 44s
MaxAbsScaler, LightGBM		0.50717	100.00 %	Jul 5, 2022 9:43 AM	1h 56m 26s

Prerequisites

- An Azure Machine Learning workspace. To create the workspace, see [Create workspace resources](#).
- This article assumes some familiarity with setting up an automated machine learning experiment. Follow the [tutorial](#) or [how-to](#) to see the main automated machine learning experiment design patterns.
- Automated ML code generation is only available for experiments run on remote Azure ML compute targets. Code generation isn't supported for local runs.
- To enable code generation with the SDK, you have the following options:
 - You can run your code via a Jupyter notebook in an [Azure Machine Learning compute instance](#), which contains the latest Azure ML SDK already installed. The compute instance comes with a ready-to-use Conda environment that is compatible with the automated ML code generation (preview) capability.
 - Alternatively, you can create a new local Conda environment on your local machine and then install the latest Azure ML SDK. [How to install AutoML client SDK in Conda environment with the automl package](#).

Code generation with the SDK

By default, each automated ML trained model generates its training code after training completes. Automated ML saves this code in the experiment's `outputs/generated_code` for that specific model. You can view them in the Azure ML studio UI on the **Outputs + logs** tab of the selected model.

You can also explicitly enable code generation for your automated ML experiments in your `AutoMLConfig` object with the `enable_code_generation=True` parameter. This parameter must be set prior to submitting your experiment.

Confirm that you call `experiment.submit()` from a Conda environment that contains the latest Azure ML SDK with automated ML. This ensures that code generation is triggered properly for the experiments that are run on a remote compute target.

```
config = AutoMLConfig( task="classification",
                        training_data=data,
                        label_column_name="label",
                        compute_target=compute_target,
                        enable_code_generation=True
                    )
```

In some troubleshooting cases, you might want to disable code generation. Before you submit your automated ML experiment, you can disable code generation in your `AutoMLConfig` object with the `enable_code_generation=False` parameter.

```
# Disabling Code Generation
config = AutoMLConfig( task="classification",
                        training_data=data,
                        label_column_name="label",
                        compute_target=compute_target,
                        enable_code_generation=False
                    )
```

There are two main files with the generated code,

- `script.py` This is the model's training code that you likely want to analyze with the featurization steps,

specific algorithm used, and hyperparameters.

- `script_run_notebook.ipynb` Notebook with boiler-plate code to run the model's training code (`script.py`) in AzureML compute through Azure ML SDK classes such as `ScriptRunConfig`.

Get generated code and model artifacts

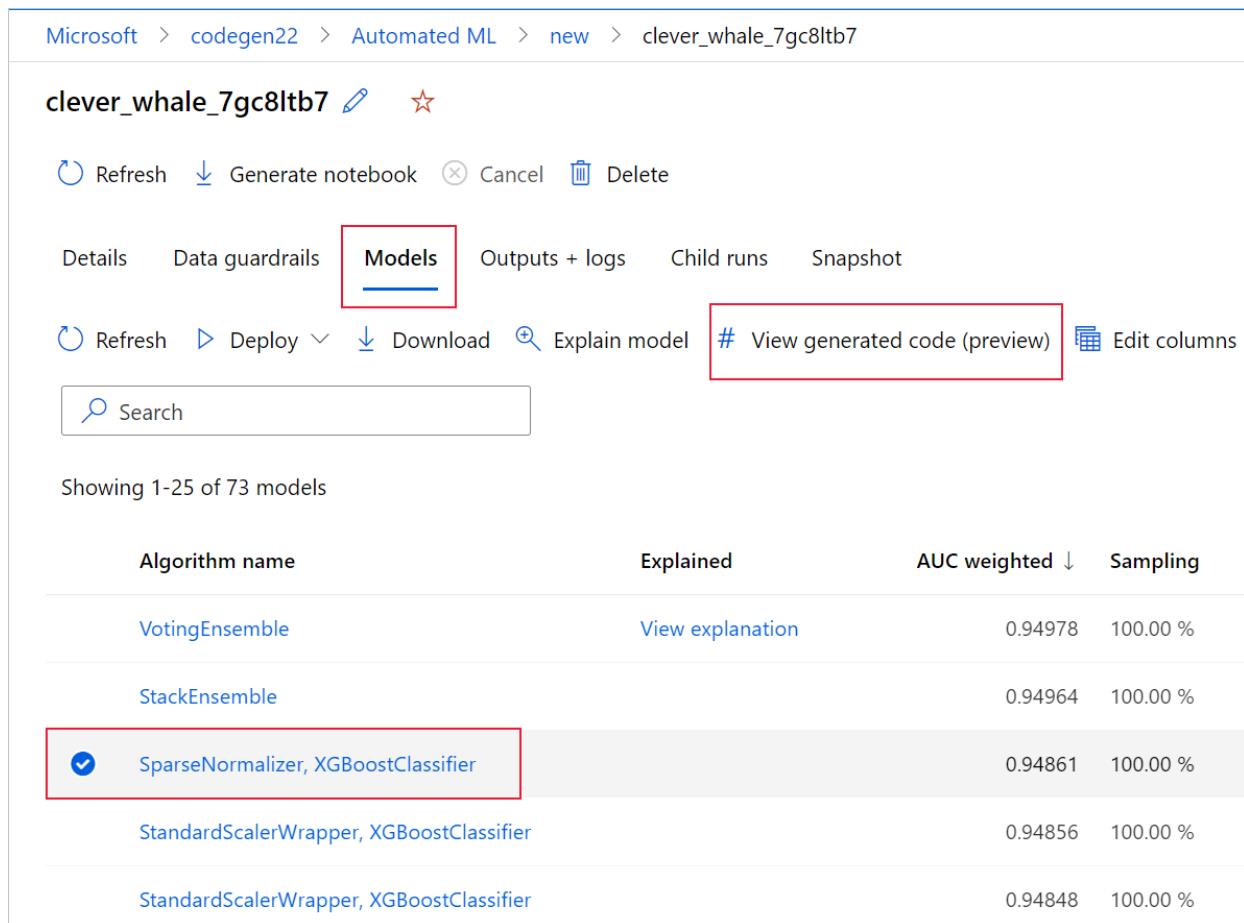
After the automated ML training run completes, you can get the `script.py` and the `script_run_notebook.ipynb` files. The following code gets the best child run and downloads both files.

```
best_run = remote_run.get_best_child()

best_run.download_file("outputs/generated_code/script.py", "script.py")
best_run.download_file("outputs/generated_code/script_run_notebook.ipynb", "script_run_notebook.ipynb")
```

You also can view the generated code and prepare it for code customization via the Azure Machine Learning studio UI.

To do so, navigate to the **Models** tab of the automated ML experiment parent run page. After you select one of the trained models, you can select the **View generated code (preview)** button. This button redirects you to the **Notebooks** portal extension, where you can view, edit and run the generated code for that particular selected model.



The screenshot shows the Azure Machine Learning studio interface. At the top, there is a breadcrumb navigation: Microsoft > codegen22 > Automated ML > new > clever_whale_7gc8ltb7. Below the navigation, the experiment name "clever_whale_7gc8ltb7" is displayed with edit and star icons. A toolbar contains Refresh, Generate notebook, Cancel, and Delete buttons. Below the toolbar, there are tabs: Details, Data guardrails, **Models**, Outputs + logs, Child runs, and Snapshot. The Models tab is currently selected. Another toolbar below the tabs includes Refresh, Deploy, Download, Explain model, View generated code (preview) (which is highlighted with a red box), and Edit columns. A search bar is also present. The main area shows a table of 73 models, with the first few rows listed:

Algorithm name	Explained	AUC weighted ↓	Sampling
VotingEnsemble	View explanation	0.94978	100.00 %
StackEnsemble		0.94964	100.00 %
SparseNormalizer, XGBoostClassifier		0.94861	100.00 %
StandardScalerWrapper, XGBoostClassifier		0.94856	100.00 %
StandardScalerWrapper, XGBoostClassifier		0.94848	100.00 %

Alternatively, you can also access to the model's generated code from the top of the child run's page once you navigate into that child run's page of a particular model.

The screenshot shows the Azure Machine Learning studio interface. At the top, there's a navigation bar with links for Refresh, Deploy, Download, Explain model, View generated code (preview) (which is highlighted with a red box), Cancel, and Delete. Below the navigation bar, there are tabs for Details, Model (which is selected and underlined), Explanations (preview), Metrics, Data transformation (preview), Outputs + logs, Images, and Child runs. The main content area is titled "Model summary". It contains sections for Algorithm name (SparseNormalizer, XGBoostClassifier), Hyperparameters (with a "View hyperparameters" link), AUC weighted (0.94861, with a "View all other metrics" link), Sampling (100.00 %), Registered models (No registration yet), and Deploy status (No deployment yet).

script.py

The `script.py` file contains the core logic needed to train a model with the previously used hyperparameters. While intended to be executed in the context of an Azure ML script run, with some modifications, the model's training code can also be run standalone in your own on-premises environment.

The script can roughly be broken down into several the following parts: data loading, data preparation, data featurization, preprocessor/algorithm specification, and training.

Data loading

The function `get_training_dataset()` loads the previously used dataset. It assumes that the script is run in an AzureML script run under the same workspace as the original experiment.

```
def get_training_dataset(dataset_id):
    from azureml.core.dataset import Dataset
    from azureml.core.run import Run

    logger.info("Running get_training_dataset")
    ws = Run.get_context().experiment.workspace
    dataset = Dataset.get_by_id(workspace=ws, id=dataset_id)
    return dataset.to_pandas_dataframe()
```

When running as part of a script run, `Run.get_context().experiment.workspace` retrieves the correct workspace. However, if this script is run inside of a different workspace or run locally without using `ScriptRunConfig`, you need to modify the script to [explicitly specify the appropriate workspace](#).

Once the workspace has been retrieved, the original dataset is retrieved by its ID. Another dataset with exactly the same structure could also be specified by ID or name with the `get_by_id()` or `get_by_name()`, respectively. You can find the ID later on in the script, in a similar section as the following code.

```

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--training_dataset_id', type=str, default='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx',
    help='Default training dataset id is populated from the parent run')
    args = parser.parse_args()

    main(args.training_dataset_id)

```

You can also opt to replace this entire function with your own data loading mechanism; the only constraints are that the return value must be a Pandas dataframe and that the data must have the same shape as in the original experiment.

Data preparation code

The function `prepare_data()` cleans the data, splits out the feature and sample weight columns and prepares the data for use in training. This function can vary depending on the type of dataset and the experiment task type: classification, regression, or time-series forecasting.

The following example shows that in general, the dataframe from the data loading step is passed in. The label column and sample weights, if originally specified, are extracted and rows containing `NaN` are dropped from the input data.

```

def prepare_data(dataframe):
    from azureml.training.tabular.preprocessing import data_cleaning

    logger.info("Running prepare_data")
    label_column_name = 'y'

    # extract the features, target and sample weight arrays
    y = dataframe[label_column_name].values
    X = dataframe.drop([label_column_name], axis=1)
    sample_weights = None
    X, y, sample_weights = data_cleaning._remove_nan_rows_in_X_y(X, y, sample_weights,
        is_timeseries=False, target_column=label_column_name)

    return X, y, sample_weights

```

If you want to do any additional data preparation, it can be done in this step by adding your custom data preparation code.

Data featurization code

The function `generate_data_transformation_config()` specifies the featurization step in the final scikit-learn pipeline. The featurizers from the original experiment are reproduced here, along with their parameters.

For example, possible data transformation that can happen in this function can be based on imputers like, `SimpleImputer()` and `CatImputer()`, or transformers such as `StringCastTransformer()` and `LabelEncoderTransformer()`.

The following is a transformer of type `StringCastTransformer()` that can be used to transform a set of columns. In this case, the set indicated by `column_names`.

```

def get_mapper_c6ba98(column_names):
    # ... Multiple imports to package dependencies, removed for simplicity ...

    definition = gen_features(
        columns=column_names,
        classes=[

            {
                'class': StringCastTransformer,
            },
            {

                'class': CountVectorizer,
                'analyzer': 'word',
                'binary': True,
                'decode_error': 'strict',
                'dtype': numpy.uint8,
                'encoding': 'utf-8',
                'input': 'content',
                'lowercase': True,
                'max_df': 1.0,
                'max_features': None,
                'min_df': 1,
                'ngram_range': (1, 1),
                'preprocessor': None,
                'stop_words': None,
                'strip_accents': None,
                'token_pattern': '(?u)\\b\\w\\w+\\b',
                'tokenizer': wrap_in_lst,
                'vocabulary': None,
            },
        ],
    )
    mapper = DataFrameMapper(features=definition, input_df=True, sparse=True)

    return mapper

```

Be aware that if you have many columns that need to have the same featurization/transformation applied (for example, 50 columns in several column groups), these columns are handled by grouping based on type.

In the following example, notice that each group has a unique mapper applied. This mapper is then applied to each of the columns of that group.

```

def generate_data_transformation_config():
    from sklearn.pipeline import FeatureUnion

    column_group_1 = [['id'], ['ps_reg_01'], ['ps_reg_02'], ['ps_reg_03'], ['ps_car_11_cat'], ['ps_car_12'],
    ['ps_car_13'], ['ps_car_14'], ['ps_car_15'], ['ps_calc_01'], ['ps_calc_02'], ['ps_calc_03']]

    column_group_2 = ['ps_ind_06_bin', 'ps_ind_07_bin', 'ps_ind_08_bin', 'ps_ind_09_bin', 'ps_ind_10_bin',
    'ps_ind_11_bin', 'ps_ind_12_bin', 'ps_ind_13_bin', 'ps_ind_16_bin', 'ps_ind_17_bin', 'ps_ind_18_bin',
    'ps_car_08_cat', 'ps_calc_15_bin', 'ps_calc_16_bin', 'ps_calc_17_bin', 'ps_calc_18_bin', 'ps_calc_19_bin',
    'ps_calc_20_bin']

    column_group_3 = ['ps_ind_01', 'ps_ind_02_cat', 'ps_ind_03', 'ps_ind_04_cat', 'ps_ind_05_cat',
    'ps_ind_14', 'ps_ind_15', 'ps_car_01_cat', 'ps_car_02_cat', 'ps_car_03_cat', 'ps_car_04_cat',
    'ps_car_05_cat', 'ps_car_06_cat', 'ps_car_07_cat', 'ps_car_09_cat', 'ps_car_10_cat', 'ps_car_11',
    'ps_calc_04', 'ps_calc_05', 'ps_calc_06', 'ps_calc_07', 'ps_calc_08', 'ps_calc_09', 'ps_calc_10',
    'ps_calc_11', 'ps_calc_12', 'ps_calc_13', 'ps_calc_14']

    feature_union = FeatureUnion([
        ('mapper_ab1045', get_mapper_ab1045(column_group_1)),
        ('mapper_c6ba98', get_mapper_c6ba98(column_group_3)),
        ('mapper_9133f9', get_mapper_9133f9(column_group_2)),
    ])
    return feature_union

```

This approach allows you to have a more streamlined code, by not having a transformer's code-block for each column, which can be especially cumbersome even when you have tens or hundreds of columns in your dataset.

With classification and regression tasks, [`FeatureUnion`] is used for featurizers. For time-series forecasting models, multiple time series-aware featurizers are collected into a scikit-learn pipeline, then wrapped in the `TimeSeriesTransformer`. Any user provided featurizations for time series forecasting models happens before the ones provided by automated ML.

Preprocessor specification code

The function `generate_preprocessor_config()`, if present, specifies a preprocessing step to be done after featurization in the final scikit-learn pipeline.

Normally, this preprocessing step only consists of data standardization/normalization that's accomplished with `sklearn.preprocessing`.

Automated ML only specifies a preprocessing step for non-ensemble classification and regression models.

Here's an example of a generated preprocessor code:

```
def generate_preprocessor_config():
    from sklearn.preprocessing import MaxAbsScaler

    preproc = MaxAbsScaler(
        copy=True
    )

    return preproc
```

Algorithm and hyperparameters specification code

The algorithm and hyperparameters specification code is likely what many ML professionals are most interested in.

The `generate_algorithm_config()` function specifies the actual algorithm and hyperparameters for training the model as the last stage of the final scikit-learn pipeline.

The following example uses an XGBoostClassifier algorithm with specific hyperparameters.

```

def generate_algorithm_config():
    from xgboost.sklearn import XGBClassifier

    algorithm = XGBClassifier(
        base_score=0.5,
        booster='gbtree',
        colsample_bylevel=1,
        colsample_bynode=1,
        colsample_bytree=1,
        gamma=0,
        learning_rate=0.1,
        max_delta_step=0,
        max_depth=3,
        min_child_weight=1,
        missing=np.nan,
        n_estimators=100,
        n_jobs=-1,
        nthread=None,
        objective='binary:logistic',
        random_state=0,
        reg_alpha=0,
        reg_lambda=1,
        scale_pos_weight=1,
        seed=None,
        silent=None,
        subsample=1,
        verbosity=0,
        tree_method='auto',
        verbose=-10
    )

    return algorithm

```

The generated code in most cases uses open source software (OSS) packages and classes. There are instances where intermediate wrapper classes are used to simplify more complex code. For example, XGBoost classifier and other commonly used libraries like LightGBM or Scikit-Learn algorithms can be applied.

As an ML Professional, you are able to customize that algorithm's configuration code by tweaking its hyperparameters as needed based on your skills and experience for that algorithm and your particular ML problem.

For ensemble models, `generate_preprocessor_config_N()` (if needed) and `generate_algorithm_config_N()` are defined for each learner in the ensemble model, where `N` represents the placement of each learner in the ensemble model's list. For stack ensemble models, the meta learner `generate_algorithm_config_meta()` is defined.

End to end training code

Code generation emits `build_model_pipeline()` and `train_model()` for defining the scikit-learn pipeline and for calling `fit()` on it, respectively.

```

def build_model_pipeline():
    from sklearn.pipeline import Pipeline

    logger.info("Running build_model_pipeline")
    pipeline = Pipeline(
        steps=[
            ('featurization', generate_data_transformation_config()),
            ('preproc', generate_preprocessor_config()),
            ('model', generate_algorithm_config()),
        ]
    )

    return pipeline

```

The scikit-learn pipeline includes the featurization step, a preprocessor (if used), and the algorithm or model.

For time-series forecasting models, the scikit-learn pipeline is wrapped in a `ForecastingPipelineWrapper`, which has some additional logic needed to properly handle time-series data depending on the applied algorithm. For all task types, we use `PipelineWithYTransformer` in cases where the label column needs to be encoded.

Once you have the scikit-Learn pipeline, all that is left to call is the `fit()` method to train the model:

```

def train_model(X, y, sample_weights):

    logger.info("Running train_model")
    model_pipeline = build_model_pipeline()

    model = model_pipeline.fit(X, y)
    return model

```

The return value from `train_model()` is the model fitted/trained on the input data.

The main code that runs all the previous functions is the following:

```

def main(training_dataset_id=None):
    from azureml.core.run import Run

    # The following code is for when running this code as part of an AzureML script run.
    run = Run.get_context()
    setup_instrumentation(run)

    df = get_training_dataset(training_dataset_id)
    X, y, sample_weights = prepare_data(df)
    split_ratio = 0.1
    try:
        (X_train, y_train, sample_weights_train), (X_valid, y_valid, sample_weights_valid) =
        split_dataset(X, y, sample_weights, split_ratio, should_stratify=True)
    except Exception:
        (X_train, y_train, sample_weights_train), (X_valid, y_valid, sample_weights_valid) =
        split_dataset(X, y, sample_weights, split_ratio, should_stratify=False)

    model = train_model(X_train, y_train, sample_weights_train)

    metrics = calculate_metrics(model, X, y, sample_weights, X_test=X_valid, y_test=y_valid)

    print(metrics)
    for metric in metrics:
        run.log(metric, metrics[metric])

```

Once you have the trained model, you can use it for making predictions with the `predict()` method. If your experiment is for a time series model, use the `forecast()` method for predictions.

```
y_pred = model.predict(X)
```

Finally, the model is serialized and saved as a `.pkl` file named "model.pkl":

```
with open('model.pkl', 'wb') as f:  
    pickle.dump(model, f)  
run.upload_file('outputs/model.pkl', 'model.pkl')
```

script_run_notebook.ipynb

The `script_run_notebook.ipynb` notebook serves as an easy way to execute `script.py` on an Azure ML compute. This notebook is similar to the existing automated ML sample notebooks however, there are a couple of key differences as explained in the following sections.

Environment

Typically, the training environment for an automated ML run is automatically set by the SDK. However, when running a custom script run like the generated code, automated ML is no longer driving the process, so the environment must be specified for the script run to succeed.

Code generation reuses the environment that was used in the original automated ML experiment, if possible. Doing so guarantees that the training script run doesn't fail due to missing dependencies, and has a side benefit of not needing a Docker image rebuild, which saves time and compute resources.

If you make changes to `script.py` that require additional dependencies, or you would like to use your own environment, you need to update the `Create environment` cell in `script_run_notebook.ipynb` accordingly.

For more information about AzureML environments, see [the Environment class documentation](#).

Submit the experiment

Since the generated code isn't driven by automated ML anymore, instead of creating an `AutoMLConfig` and then passing it to `experiment.submit()`, you need to create a `ScriptRunConfig` and provide the generated code (`script.py`) to it.

The following example contains the parameters and regular dependencies needed to run `ScriptRunConfig`, such as compute, environment, etc. For more information on how to use `ScriptRunConfig`, see [Configure and submit training runs](#).

```
from azureml.core import ScriptRunConfig  
  
src = ScriptRunConfig(source_directory=project_folder,  
                      script='script.py',  
                      compute_target=cpu_cluster,  
                      environment=myenv,  
                      docker_runtime_config=docker_config)  
  
run = experiment.submit(config=src)
```

Download and load the serialized trained model in-memory

Once you have a trained model, you can save/serialize it to a `.pkl` file with `pickle.dump()` and `pickle.load()`. You can also use `joblib.dump()` and `joblib.load()`.

The following example is how you download and load a model in-memory that was trained in AzureML compute with `ScriptRunConfig`. This code can run in the same notebook you used the Azure ML SDK `ScriptRunConfig`.

```

import joblib

# Load the fitted model from the script run.

# Note that if training dependencies are not installed on the machine
# this notebook is being run from, this step can fail.
try:
    # Download the model from the run in the Workspace
    run.download_file("outputs/model.pkl", "model.pkl")

    # Load the model into memory
    model = joblib.load("model.pkl")

except ImportError:
    print('Required dependencies are missing; please run pip install azureml-automl-runtime.')
    raise

```

Making predictions with the model in-memory

Finally, you can load test data in a Pandas dataframe and use the model to make predictions.

```

import os
import numpy as np
import pandas as pd

DATA_DIR = "."
filepath = os.path.join(DATA_DIR, 'porto_seguro_safe_driver_test_dataset.csv')

test_data_df = pd.read_csv(filepath)

print(test_data_df.shape)
test_data_df.head(5)

#test_data_df is a Pandas dataframe with test data
y_predictions = model.predict(test_data_df)

```

In an Azure ML compute instance you have all the automated ML dependencies, so you're able to load the model and predict from any notebook in a compute instance recently created.

However, in order to load that model in a notebook in your custom local Conda environment, you need to have all the dependencies coming from the environment used when training (AutoML environment) installed.

Next steps

- Learn more about [how and where to deploy a model](#).
- See how to [enable interpretability features](#) specifically within automated ML experiments.

Make predictions with an AutoML ONNX model in .NET

9/22/2022 • 7 minutes to read • [Edit Online](#)

In this article, you learn how to use an Automated ML (AutoML) Open Neural Network Exchange (ONNX) model to make predictions in a C# .NET Core console application with ML.NET.

ML.NET is an open-source, cross-platform, machine learning framework for the .NET ecosystem that allows you to train and consume custom machine learning models using a code-first approach in C# or F# as well as through low-code tooling like [Model Builder](#) and the [ML.NET CLI](#). The framework is also extensible and allows you to leverage other popular machine learning frameworks like TensorFlow and ONNX.

ONNX is an open-source format for AI models. ONNX supports interoperability between frameworks. This means you can train a model in one of the many popular machine learning frameworks like PyTorch, convert it into ONNX format, and consume the ONNX model in a different framework like ML.NET. To learn more, visit the [ONNX website](#).

Prerequisites

- [.NET Core SDK 3.1 or greater](#)
- Text Editor or IDE (such as [Visual Studio](#) or [Visual Studio Code](#))
- ONNX model. To learn how to train an AutoML ONNX model, see the following [bank marketing classification notebook](#).
- [Netron](#) (optional)

Create a C# console application

In this sample, you use the .NET Core CLI to build your application but you can do the same tasks using Visual Studio. Learn more about the [.NET Core CLI](#).

1. Open a terminal and create a new C# .NET Core console application. In this example, the name of the application is `AutoMLONNXConsoleApp`. A directory is created by that same name with the contents of your application.

```
dotnet new console -o AutoMLONNXConsoleApp
```

2. In the terminal, navigate to the `AutoMLONNXConsoleApp` directory.

```
cd AutoMLONNXConsoleApp
```

Add software packages

1. Install the **Microsoft.ML**, **Microsoft.ML.OnnxRuntime**, and **Microsoft.ML.OnnxTransformer** NuGet packages using the .NET Core CLI.

```
dotnet add package Microsoft.ML
dotnet add package Microsoft.ML.OnnxRuntime
dotnet add package Microsoft.ML.OnnxTransformer
```

These packages contain the dependencies required to use an ONNX model in a .NET application. ML.NET provides an API that uses the [ONNX runtime](#) for predictions.

2. Open the `Program.cs` file and add the following `using` statements at the top to reference the appropriate packages.

```
using System.Linq;
using Microsoft.ML;
using Microsoft.ML.Data;
using Microsoft.ML.Transforms.Onnx;
```

Add a reference to the ONNX model

A way for the console application to access the ONNX model is to add it to the build output directory. To learn more about MSBuild common items, see the [MSBuild guide](#).

Add a reference to your ONNX model file in your application

1. Copy your ONNX model to your application's `AutoMLONNXConsoleApp` root directory.
2. Open the `AutoMLONNXConsoleApp.csproj` file and add the following content inside the `Project` node.

```
<ItemGroup>
  <None Include="automl-model.onnx">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
</ItemGroup>
```

In this case, the name of the ONNX model file is `automl-model.onnx`.

3. Open the `Program.cs` file and add the following line inside the `Program` class.

```
static string ONNX_MODEL_PATH = "automl-model.onnx";
```

Initialize MLContext

Inside the `Main` method of your `Program` class, create a new instance of `MLContext`.

```
MLContext mlContext = new MLContext();
```

The `MLContext` class is a starting point for all ML.NET operations, and initializing `mlContext` creates a new ML.NET environment that can be shared across the model lifecycle. It's similar, conceptually, to `DbContext` in Entity Framework.

Define the model data schema

Your model expects your input and output data in a specific format. ML.NET allows you to define the format of your data via classes. Sometimes you may already know what that format looks like. In cases when you don't know the data format, you can use tools like Netron to inspect your ONNX model.

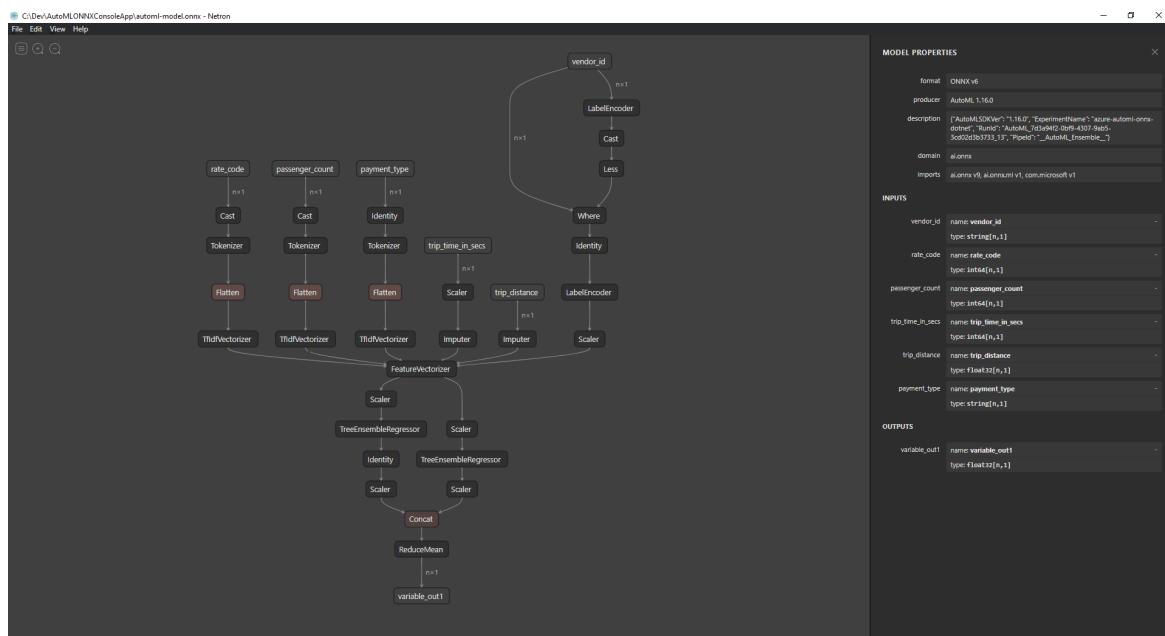
The model used in this sample uses data from the NYC TLC Taxi Trip dataset. A sample of the data can be seen below:

VENDOR_ID	RATE_CODE	PASSENGER_COUNT	TRIP_TIME_IN_SECS	TRIP_DISTANCE	PAYMENT_TYPE	FARE_AMOUNT
VTS	1	1	1140	3.75	CRD	15.5
VTS	1	1	480	2.72	CRD	10.0
VTS	1	1	1680	7.8	CSH	26.5

Inspect the ONNX model (optional)

Use a tool like Netron to inspect your model's inputs and outputs.

1. Open Netron.
2. In the top menu bar, select **File > Open** and use the file browser to select your model.
3. Your model opens. For example, the structure of the *automl-model.onnx* model looks like the following:



4. Select the last node at the bottom of the graph (`variable_out1` in this case) to display the model's metadata. The inputs and outputs on the sidebar show you the model's expected inputs, outputs, and data types. Use this information to define the input and output schema of your model.

Define model input schema

Create a new class called `OnnxInput` with the following properties inside the *Program.cs* file.

```

public class OnnxInput
{
    [ColumnName("vendor_id")]
    public string VendorId { get; set; }

    [ColumnName("rate_code"), OnnxMapType(typeof(Int64), typeof(Single))]
    public Int64 RateCode { get; set; }

    [ColumnName("passenger_count"), OnnxMapType(typeof(Int64), typeof(Single))]
    public Int64 PassengerCount { get; set; }

    [ColumnName("trip_time_in_secs"), OnnxMapType(typeof(Int64), typeof(Single))]
    public Int64 TripTimeInSecs { get; set; }

    [ColumnName("trip_distance")]
    public float TripDistance { get; set; }

    [ColumnName("payment_type")]
    public string PaymentType { get; set; }
}

```

Each of the properties maps to a column in the dataset. The properties are further annotated with attributes.

The `ColumnName` attribute lets you specify how ML.NET should reference the column when operating on the data. For example, although the `TripDistance` property follows standard .NET naming conventions, the model only knows of a column or feature known as `trip_distance`. To address this naming discrepancy, the `ColumnName` attribute maps the `TripDistance` property to a column or feature by the name `trip_distance`.

For numerical values, ML.NET only operates on `Single` value types. However, the original data type of some of the columns are integers. The `OnnxMapType` attribute maps types between ONNX and ML.NET.

To learn more about data attributes, see the [ML.NET load data guide](#).

Define model output schema

Once the data is processed, it produces an output of a certain format. Define your data output schema. Create a new class called `OnnxOutput` with the following properties inside the `Program.cs` file.

```

public class OnnxOutput
{
    [ColumnName("variable_out1")]
    public float[] PredictedFare { get; set; }
}

```

Similar to `OnnxInput`, use the `ColumnName` attribute to map the `variable_out1` output to a more descriptive name `PredictedFare`.

Define a prediction pipeline

A pipeline in ML.NET is typically a series of chained transformations that operate on the input data to produce an output. To learn more about data transformations, see the [ML.NET data transformation guide](#).

1. Create a new method called `GetPredictionPipeline` inside the `Program` class

```

static ITransformer GetPredictionPipeline(MLContext mlContext)
{
}

```

2. Define the name of the input and output columns. Add the following code inside the `GetPredictionPipeline` method.

```
var inputColumns = new string []
{
    "vendor_id", "rate_code", "passenger_count", "trip_time_in_secs", "trip_distance", "payment_type"
};

var outputColumns = new string [] { "variable_out1" };
```

3. Define your pipeline. An `IEstimator` provides a blueprint of the operations, input, and output schemas of your pipeline.

```
var onnxPredictionPipeline =
    mlContext
        .Transforms
        .ApplyOnnxModel(
            outputColumnNames: outputColumns,
            inputColumnNames: inputColumns,
            ONNX_MODEL_PATH);
```

In this case, `ApplyOnnxModel` is the only transform in the pipeline, which takes in the names of the input and output columns as well as the path to the ONNX model file.

4. An `IEstimator` only defines the set of operations to apply to your data. What operates on your data is known as an `ITransformer`. Use the `Fit` method to create one from your `onnxPredictionPipeline`.

```
var emptyDv = mlContext.Data.LoadFromEnumerable(new OnnxInput[] {});

return onnxPredictionPipeline.Fit(emptyDv);
```

The `Fit` method expects an `IDataView` as input to perform the operations on. An `IDataView` is a way to represent data in ML.NET using a tabular format. Since in this case the pipeline is only used for predictions, you can provide an empty `IDataView` to give the `ITransformer` the necessary input and output schema information. The fitted `ITransformer` is then returned for further use in your application.

TIP

In this sample, the pipeline is defined and used within the same application. However, it is recommended that you use separate applications to define and use your pipeline to make predictions. In ML.NET your pipelines can be serialized and saved for further use in other .NET end-user applications. ML.NET supports various deployment targets such as desktop applications, web services, WebAssembly applications*, and many more. To learn more about saving pipelines, see the [ML.NET save and load trained models guide](#).

*WebAssembly is only supported in .NET Core 5 or greater

5. Inside the `Main` method, call the `GetPredictionPipeline` method with the required parameters.

```
var onnxPredictionPipeline = GetPredictionPipeline(mlContext);
```

Use the model to make predictions

Now that you have a pipeline, it's time to use it to make predictions. ML.NET provides a convenience API for making predictions on a single data instance called `PredictionEngine`.

1. Inside the `Main` method, create a `PredictionEngine` by using the `CreatePredictionEngine` method.

```
var onnxPredictionEngine = mlContext.Model.CreatePredictionEngine<OnnxInput, OnnxOutput>(onnxPredictionPipeline);
```

2. Create a test data input.

```
var testInput = new OnnxInput
{
    VendorId = "CMT",
    RateCode = 1,
    PassengerCount = 1,
    TripTimeInSecs = 1271,
    TripDistance = 3.8f,
    PaymentType = "CRD"
};
```

3. Use the `predictionEngine` to make predictions based on the new `testInput` data using the `Predict` method.

```
var prediction = onnxPredictionEngine.Predict(testInput);
```

4. Output the result of your prediction to the console.

```
Console.WriteLine($"Predicted Fare: {prediction.PredictedFare.First()}");
```

5. Use the .NET Core CLI to run your application.

```
dotnet run
```

The result should look as similar to the following output:

```
Predicted Fare: 15.621523
```

To learn more about making predictions in ML.NET, see the [use a model to make predictions guide](#).

Next steps

- [Deploy your model as an ASP.NET Core Web API](#)
- [Deploy your model as a serverless .NET Azure Function](#)

Make predictions with ONNX on computer vision models from AutoML

9/22/2022 • 34 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK `azure-ai-ml v2 (preview)`

In this article, you will learn how to use Open Neural Network Exchange (ONNX) to make predictions on computer vision models generated from automated machine learning (AutoML) in Azure Machine Learning.

To use ONNX for predictions, you need to:

1. Download ONNX model files from an AutoML training run.
2. Understand the inputs and outputs of an ONNX model.
3. Preprocess your data so that it's in the required format for input images.
4. Perform inference with ONNX Runtime for Python.
5. Visualize predictions for object detection and instance segmentation tasks.

ONNX is an open standard for machine learning and deep learning models. It enables model import and export (interoperability) across the popular AI frameworks. For more details, explore the [ONNX GitHub project](#).

ONNX Runtime is an open-source project that supports cross-platform inference. ONNX Runtime provides APIs across programming languages (including Python, C++, C#, C, Java, and JavaScript). You can use these APIs to perform inference on input images. After you have the model that has been exported to ONNX format, you can use these APIs on any programming language that your project needs.

In this guide, you'll learn how to use [Python APIs for ONNX Runtime](#) to make predictions on images for popular vision tasks. You can use these ONNX exported models across languages.

Prerequisites

- Get an AutoML-trained computer vision model for any of the supported image tasks: classification, object detection, or instance segmentation. [Learn more about AutoML support for computer vision tasks](#).
- Install the `onnxruntime` package. The methods in this article have been tested with versions 1.3.0 to 1.8.0.

Download ONNX model files

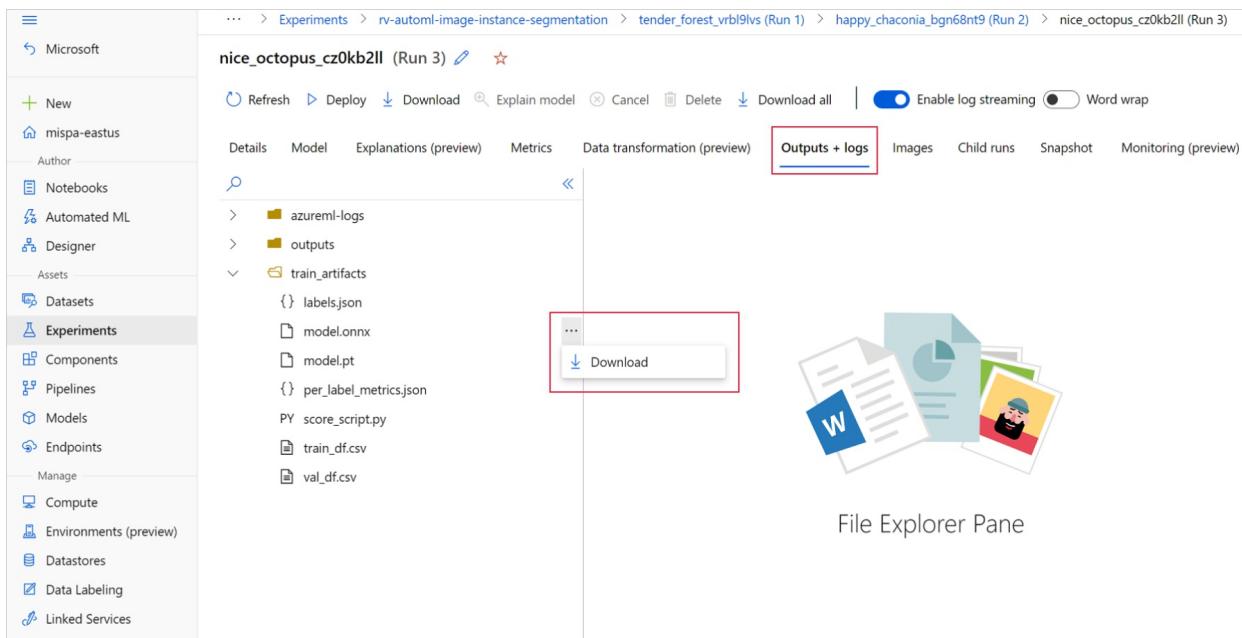
You can download ONNX model files from AutoML runs by using the Azure Machine Learning studio UI or the Azure Machine Learning Python SDK. We recommend downloading via the SDK with the experiment name and parent run ID.

Azure Machine Learning studio

On Azure Machine Learning studio, go to your experiment by using the hyperlink to the experiment generated in the training notebook, or by selecting the experiment name on the **Experiments** tab under **Assets**. Then select the best child run.

Within the best child run, go to **Outputs+logs > train_artifacts**. Use the **Download** button to manually download the following files:

- `labels.json`: File that contains all the classes or labels in the training dataset.
- `model.onnx`: Model in ONNX format.



Save the downloaded model files in a directory. The example in this article uses the `./automl_models` directory.

Azure Machine Learning Python SDK

With the SDK, you can select the best child run (by primary metric) with the experiment name and parent run ID. Then, you can download the `labels.json` and `model.onnx` files.

The following code returns the best child run based on the relevant primary metric.

```
from azure.identity import DefaultAzureCredential
from azure.ai.ml import MLClient

credential = DefaultAzureCredential()
ml_client = None
try:
    ml_client = MLClient.from_config(credential)
except Exception as ex:
    print(ex)
# Enter details of your AML workspace
subscription_id = ''
resource_group = ''
workspace_name = ''
ml_client = MLClient(credential, subscription_id, resource_group, workspace_name)
```

```
import mlflow
from mlflow.tracking.client import MlflowClient

# Obtain the tracking URL from MLClient
MLFLOW_TRACKING_URI = ml_client.workspaces.get(
    name=ml_client.workspace_name
).mlflow_tracking_uri

mlflow.set_tracking_uri(MLFLOW_TRACKING_URI)

# Specify the job name
job_name = ''

# Get the parent run
mlflow_parent_run = mlflow_client.get_run(job_name)
best_child_run_id = mlflow_parent_run.data.tags['automl_best_child_run_id']
# get the best child run
best_run = mlflow_client.get_run(best_child_run_id)
```

Download the `labels.json` file, which contains all the classes and labels in the training dataset.

```
local_dir = './automl_models'
if not os.path.exists(local_dir):
    os.mkdir(local_dir)

labels_file = mlflow_client.download_artifacts(
    best_run.info.run_id, 'train_artifacts/labels.json', local_dir
)
```

Download the `model.onnx` file.

```
onnx_model_path = mlflow_client.download_artifacts(
    best_run.info.run_id, 'train_artifacts/model.onnx', local_dir
)
```

In case of batch inferencing for Object Detection and Instance Segmentation using ONNX models, refer to the section on [model generation for batch scoring](#).

Model generation for batch scoring

By default, AutoML for Images supports batch scoring for classification. But object detection and instance segmentation ONNX models don't support batch inferencing. In case of batch inference for object detection and instance segmentation, use the following procedure to generate an ONNX model for the required batch size. Models generated for a specific batch size don't work for other batch sizes.

Download the conda environment file and create an environment object to be used with command job.

```
# Download conda file and define the environment

conda_file = mlflow_client.download_artifacts(
    best_run.info.run_id, "outputs/conda_env_v_1_0_0.yml", local_dir

from azure.ai.ml.entities import Environment
env = Environment(
    name="automl-images-env-onnx",
    description="environment for automl images ONNX batch model generation",
    image="mcr.microsoft.com/azureml/openmpi4.1.0-cuda11.1-cudnn8-ubuntu18.04",
    conda_file=conda_file,
)
```

Use the following model specific arguments to submit the script. For more details on arguments, refer to [model specific hyperparameters](#) and for supported object detection model names refer to the [supported model algorithm section](#).

To get the argument values needed to create the batch scoring model, refer to the scoring scripts generated under the outputs folder of the Automl training runs. Use the hyperparameter values available in the model settings variable inside the scoring file for the best child run.

- [Multi-class image classification](#)
- [Multi-label image classification](#)
- [Object detection with Faster R-CNN or RetinaNet](#)
- [Object detection with YOLO](#)
- [Instance segmentation](#)

For multi-class image classification, the generated ONNX model for the best child-run supports batch scoring by default. Therefore, no model specific arguments are needed for this task type and you can skip to the [Load the](#)

[labels and ONNX model files](#) section.

Download and keep the `ONNX_batch_model_generator_automl_for_images.py` file in the current directory to submit the script. Use the following command job to submit the script `ONNX_batch_model_generator_automl_for_images.py` available in the [azureml-examples GitHub repository](#), to generate an ONNX model of a specific batch size. In the following code, the trained model environment is used to submit this script to generate and save the ONNX model to the outputs directory.

- [Multi-class image classification](#)
- [Multi-label image classification](#)
- [Object detection with Faster R-CNN or RetinaNet](#)
- [Object detection with YOLO](#)
- [Instance segmentation](#)

For multi-class image classification, the generated ONNX model for the best child-run supports batch scoring by default. Therefore, no model specific arguments are needed for this task type and you can skip to the [Load the labels and ONNX model files](#) section.

Once the batch model is generated, either download it from **Outputs+logs > outputs** manually through UI, or use the following method:

```
batch_size = 8 # use the batch size used to generate the model
returned_job_run = mlflow_client.get_run(returned_job.name)

# Download run's artifacts/outputs
onnx_model_path = mlflow_client.download_artifacts(
    best_run.info.run_id, 'outputs/model_'+str(batch_size)+'.onnx', local_dir
)
```

After the model downloading step, you use the ONNX Runtime Python package to perform inferencing by using the `model.onnx` file. For demonstration purposes, this article uses the datasets from [How to prepare image datasets](#) for each vision task.

We've trained the models for all vision tasks with their respective datasets to demonstrate ONNX model inference.

Load the labels and ONNX model files

The following code snippet loads `labels.json`, where class names are ordered. That is, if the ONNX model predicts a label ID as 2, then it corresponds to the label name given at the third index in the `labels.json` file.

```
import json
import onnxruntime

labels_file = "automl_models/labels.json"
with open(labels_file) as f:
    classes = json.load(f)
print(classes)
try:
    session = onnxruntime.InferenceSession(onnx_model_path)
    print("ONNX model loaded...")
except Exception as e:
    print("Error loading ONNX file: ", str(e))
```

Get expected input and output details for an ONNX model

When you have the model, it's important to know some model-specific and task-specific details. These details include the number of inputs and number of outputs, expected input shape or format for preprocessing the image, and output shape so you know the model-specific or task-specific outputs.

```
sess_input = session.get_inputs()
sess_output = session.get_outputs()
print(f"No. of inputs : {len(sess_input)}, No. of outputs : {len(sess_output)}")

for idx, input_ in enumerate(range(len(sess_input))):
    input_name = sess_input[input_].name
    input_shape = sess_input[input_].shape
    input_type = sess_input[input_].type
    print(f"{idx} Input name : {input_name}, Input shape : {input_shape}, \
Input type : {input_type}")

for idx, output in enumerate(range(len(sess_output))):
    output_name = sess_output[output].name
    output_shape = sess_output[output].shape
    output_type = sess_output[output].type
    print(f" {idx} Output name : {output_name}, Output shape : {output_shape}, \
Output type : {output_type}")
```

Expected input and output formats for the ONNX model

Every ONNX model has a predefined set of input and output formats.

- [Multi-class image classification](#)
- [Multi-label image classification](#)
- [Object detection with Faster R-CNN or RetinaNet](#)
- [Object detection with YOLO](#)
- [Instance segmentation](#)

This example applies the model trained on the [fridgeObjects](#) dataset with 134 images and 4 classes/labels to explain ONNX model inference. For more information on training an image classification task, see the [multi-class image classification notebook](#).

Input format

The input is a preprocessed image.

INPUT NAME	INPUT SHAPE	INPUT TYPE	DESCRIPTION
input1	(batch_size, num_channels, height, width)	ndarray(float)	Input is a preprocessed image, with the shape (1, 3, 224, 224) for a batch size of 1, and a height and width of 224. These numbers correspond to the values used for crop_size in the training example.

Output format

The output is an array of logits for all the classes/labels.

OUTPUT NAME	OUTPUT SHAPE	OUTPUT TYPE	DESCRIPTION
-------------	--------------	-------------	-------------

OUTPUT NAME	OUTPUT SHAPE	OUTPUT TYPE	DESCRIPTION
output1	(batch_size, num_classes)	ndarray(float)	Model returns logits (without softmax). For instance, for batch size 1 and 4 classes, it returns (1, 4).

Preprocessing

- [Multi-class image classification](#)
- [Multi-label image classification](#)
- [Object detection with Faster R-CNN or RetinaNet](#)
- [Object detection with YOLO](#)
- [Instance segmentation](#)

Perform the following preprocessing steps for the ONNX model inference:

1. Convert the image to RGB.
2. Resize the image to `valid_resize_size` and `valid_resize_size` values that correspond to the values used in the transformation of the validation dataset during training. The default value for `valid_resize_size` is 256.
3. Center crop the image to `height_onnx_crop_size` and `width_onnx_crop_size`. It corresponds to `valid_crop_size` with the default value of 224.
4. Change `HxWxC` to `CxHxW`.
5. Convert to float type.
6. Normalize with ImageNet's `mean` = `[0.485, 0.456, 0.406]` and `std` = `[0.229, 0.224, 0.225]`.

If you chose different values for the [hyperparameters](#) `valid_resize_size` and `valid_crop_size` during training, then those values should be used.

Get the input shape needed for the ONNX model.

```
batch, channel, height_onnx_crop_size, width_onnx_crop_size = session.get_inputs()[0].shape
batch, channel, height_onnx_crop_size, width_onnx_crop_size
```

Without PyTorch

```

import glob
import numpy as np
from PIL import Image

def preprocess(image, resize_size, crop_size_onnx):
    """Perform pre-processing on raw input image

    :param image: raw input image
    :type image: PIL image
    :param resize_size: value to resize the image
    :type image: Int
    :param crop_size_onnx: expected height of an input image in onnx model
    :type crop_size_onnx: Int
    :return: pre-processed image in numpy format
    :rtype: ndarray 1xCxHxW
    """

    image = image.convert('RGB')
    # resize
    image = image.resize((resize_size, resize_size))
    # center crop
    left = (resize_size - crop_size_onnx)/2
    top = (resize_size - crop_size_onnx)/2
    right = (resize_size + crop_size_onnx)/2
    bottom = (resize_size + crop_size_onnx)/2
    image = image.crop((left, top, right, bottom))

    np_image = np.array(image)
    # HWC -> CHW
    np_image = np_image.transpose(2, 0, 1) # CxHxW
    # normalize the image
    mean_vec = np.array([0.485, 0.456, 0.406])
    std_vec = np.array([0.229, 0.224, 0.225])
    norm_img_data = np.zeros(np_image.shape).astype('float32')
    for i in range(np_image.shape[0]):
        norm_img_data[i,:,:] = (np_image[i,:,:]/255 - mean_vec[i])/std_vec[i]

    np_image = np.expand_dims(norm_img_data, axis=0) # 1xCxHxW
    return np_image

# following code loads only batch_size number of images for demonstrating ONNX inference
# make sure that the data directory has at least batch_size number of images

test_images_path = "automl_models_multi_cls/test_images_dir/*" # replace with path to images
# Select batch size needed
batch_size = 8
# you can modify resize_size based on your trained model
resize_size = 256
# height and width will be the same for classification
crop_size_onnx = height_onnx_crop_size

image_files = glob.glob(test_images_path)
img_processed_list = []
for i in range(batch_size):
    img = Image.open(image_files[i])
    img_processed_list.append(preprocess(img, resize_size, crop_size_onnx))

if len(img_processed_list) > 1:
    img_data = np.concatenate(img_processed_list)
elif len(img_processed_list) == 1:
    img_data = img_processed_list[0]
else:
    img_data = None

assert batch_size == img_data.shape[0]

```

With PyTorch

```
import glob
import torch
import numpy as np
from PIL import Image
from torchvision import transforms

def _make_3d_tensor(x) -> torch.Tensor:
    """This function is for images that have less channels.

    :param x: input tensor
    :type x: torch.Tensor
    :return: return a tensor with the correct number of channels
    :rtype: torch.Tensor
    """
    return x if x.shape[0] == 3 else x.expand((3, x.shape[1], x.shape[2]))

def preprocess(image, resize_size, crop_size_onnx):
    transform = transforms.Compose([
        transforms.Resize(resize_size),
        transforms.CenterCrop(crop_size_onnx),
        transforms.ToTensor(),
        transforms.Lambda(_make_3d_tensor),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]))]

    img_data = transform(image)
    img_data = img_data.numpy()
    img_data = np.expand_dims(img_data, axis=0)
    return img_data

# following code loads only batch_size number of images for demonstrating ONNX inference
# make sure that the data directory has at least batch_size number of images

test_images_path = "automl_models_multi_cls/test_images_dir/*" # replace with path to images
# Select batch size needed
batch_size = 8
# you can modify resize_size based on your trained model
resize_size = 256
# height and width will be the same for classification
crop_size_onnx = height_onnx_crop_size

image_files = glob.glob(test_images_path)
img_processed_list = []
for i in range(batch_size):
    img = Image.open(image_files[i])
    img_processed_list.append(preprocess(img, resize_size, crop_size_onnx))

if len(img_processed_list) > 1:
    img_data = np.concatenate(img_processed_list)
elif len(img_processed_list) == 1:
    img_data = img_processed_list[0]
else:
    img_data = None

assert batch_size == img_data.shape[0]
```

Inference with ONNX Runtime

Inferencing with ONNX Runtime differs for each computer vision task.

- [Multi-class image classification](#)
- [Multi-label image classification](#)
- [Object detection with Faster R-CNN or RetinaNet](#)

- [Object detection with YOLO](#)
- [Instance segmentation](#)

```
def get_predictions_from_ONNX(onnx_session, img_data):
    """Perform predictions with ONNX runtime

    :param onnx_session: onnx model session
    :type onnx_session: class InferenceSession
    :param img_data: pre-processed numpy image
    :type img_data: ndarray with shape 1xCxHxW
    :return: scores with shapes
             (1, No. of classes in training dataset)
    :rtype: numpy array
    """

    sess_input = onnx_session.get_inputs()
    sess_output = onnx_session.get_outputs()
    print(f"No. of inputs : {len(sess_input)}, No. of outputs : {len(sess_output)}")
    # predict with ONNX Runtime
    output_names = [output.name for output in sess_output]
    scores = onnx_session.run(output_names=output_names,\n                             input_feed={sess_input[0].name: img_data})

    return scores[0]

scores = get_predictions_from_ONNX(session, img_data)
```

Postprocessing

- [Multi-class image classification](#)
- [Multi-label image classification](#)
- [Object detection with Faster R-CNN or RetinaNet](#)
- [Object detection with YOLO](#)
- [Instance segmentation](#)

Apply `softmax()` over predicted values to get classification confidence scores (probabilities) for each class. Then the prediction will be the class with the highest probability.

Without PyTorch

```
def softmax(x):
    e_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return e_x / np.sum(e_x, axis=1, keepdims=True)

conf_scores = softmax(scores)
class_preds = np.argmax(conf_scores, axis=1)
print("predicted classes:", [(class_idx, classes[class_idx]) for class_idx in class_preds])
```

With PyTorch

```
conf_scores = torch.nn.functional.softmax(torch.from_numpy(scores), dim=1)
class_preds = torch.argmax(conf_scores, dim=1)
print("predicted classes:", [(class_idx.item(), classes[class_idx]) for class_idx in class_preds])
```

Visualize predictions

- Multi-class image classification
- Multi-label image classification
- Object detection with Faster R-CNN or RetinaNet
- Object detection with YOLO
- Instance segmentation

Visualize an input image with labels

```

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
%matplotlib inline

sample_image_index = 0 # change this for an image of interest from image_files list
IMAGE_SIZE = (18, 12)
plt.figure(figsize=IMAGE_SIZE)
img_np = mpimg.imread(image_files[sample_image_index])

img = Image.fromarray(img_np.astype('uint8'), 'RGB')
x, y = img.size

fig,ax = plt.subplots(1, figsize=(15, 15))
# Display the image
ax.imshow(img_np)

label = class_preds[sample_image_index]
if torch.is_tensor(label):
    label = label.item()

conf_score = conf_scores[sample_image_index]
if torch.is_tensor(conf_score):
    conf_score = np.max(conf_score.tolist())
else:
    conf_score = np.max(conf_score)

display_text = '{} ({})'.format(label, round(conf_score, 3))
print(display_text)

color = 'red'
plt.text(30, 30, display_text, color=color, fontsize=30)

plt.show()

```

Next steps

- Learn more about computer vision tasks in AutoML
- Troubleshoot AutoML experiments

Interpretability: Model explainability in automated ML (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

In this article, you learn how to get explanations for automated machine learning (automated ML) models in Azure Machine Learning using the Python SDK. Automated ML helps you understand feature importance of the models that are generated.

All SDK versions after 1.0.85 set `model_explainability=True` by default. In SDK version 1.0.85 and earlier versions users need to set `model_explainability=True` in the `AutoMLConfig` object in order to use model interpretability.

In this article, you learn how to:

- Perform interpretability during training for best model or any model.
- Enable visualizations to help you see patterns in data and explanations.
- Implement interpretability during inference or scoring.

Prerequisites

- Interpretability features. Run `pip install azureml-interpret` to get the necessary package.
- Knowledge of building automated ML experiments. For more information on how to use the Azure Machine Learning SDK, complete this [object detection model tutorial](#) or see how to [configure automated ML experiments](#).

Interpretability during training for the best model

Retrieve the explanation from the `best_run`, which includes explanations for both raw and engineered features.

NOTE

Interpretability, model explanation, is not available for the TCNForecaster model recommended by Auto ML forecasting experiments.

Download the engineered feature importances from the best run

You can use `ExplanationClient` to download the engineered feature explanations from the artifact store of the `best_run`.

```
from azureml.interpret import ExplanationClient

client = ExplanationClient.from_run(best_run)
engineered_explanations = client.download_model_explanation(raw=False)
print(engineered_explanations.get_feature_importance_dict())
```

Download the raw feature importances from the best run

You can use `ExplanationClient` to download the raw feature explanations from the artifact store of the `best_run`.

```
from azureml.interpret import ExplanationClient

client = ExplanationClient.from_run(best_run)
raw_explanations = client.download_model_explanation(raw=True)
print(raw_explanations.get_feature_importance_dict())
```

Interpretability during training for any model

When you compute model explanations and visualize them, you're not limited to an existing model explanation for an AutoML model. You can also get an explanation for your model with different test data. The steps in this section show you how to compute and visualize engineered feature importance based on your test data.

Retrieve any other AutoML model from training

```
automl_run, fitted_model = local_run.get_output(metric='accuracy')
```

Set up the model explanations

Use `automl_setup_model_explanations` to get the engineered and raw explanations. The `fitted_model` can generate the following items:

- Featured data from trained or test samples
- Engineered feature name lists
- Findable classes in your labeled column in classification scenarios

The `automl_explainer_setup_obj` contains all the structures from above list.

```
from azureml.train.automl.runtime.automl_explain_utilities import automl_setup_model_explanations

automl_explainer_setup_obj = automl_setup_model_explanations(fitted_model, X=X_train,
                                                             X_test=X_test, y=y_train,
                                                             task='classification')
```

Initialize the Mimic Explainer for feature importance

To generate an explanation for automated ML models, use the `MimicWrapper` class. You can initialize the `MimicWrapper` with these parameters:

- The explainer setup object
- Your workspace
- A surrogate model to explain the `fitted_model` automated ML model

The `MimicWrapper` also takes the `automl_run` object where the engineered explanations will be uploaded.

```
from azureml.interpret import MimicWrapper

# Initialize the Mimic Explainer
explainer = MimicWrapper(ws, automl_explainer_setup_obj.automl_estimator,
                         explainable_model=automl_explainer_setup_obj.surrogate_model,
                         init_dataset=automl_explainer_setup_obj.X_transform, run=automl_run,
                         features=automl_explainer_setup_obj.engineered_feature_names,
                         feature_maps=[automl_explainer_setup_obj.feature_map],
                         classes=automl_explainer_setup_obj.classes,
                         explainer_kwargs=automl_explainer_setup_obj.surrogate_model_params)
```

Use Mimic Explainer for computing and visualizing engineered feature importance

You can call the `explain()` method in MimicWrapper with the transformed test samples to get the feature importance for the generated engineered features. You can also sign in to [Azure Machine Learning studio](#) to view the explanations dashboard visualization of the feature importance values of the generated engineered features by automated ML featurizers.

```
engineered_explanations = explainer.explain(['local', 'global'],
eval_dataset=automl_explainer_setup_obj.X_test_transform)
print(engineered_explanations.get_feature_importance_dict())
```

For models trained with automated ML, you can get the best model using the `get_output()` method and compute explanations locally. You can visualize the explanation results with `ExplanationDashboard` from the `raiwidgets` package.

```
best_run, fitted_model = remote_run.get_output()

from azureml.train.automl.runtime.automl_explain_utilities import AutoMLEExplainerSetupClass,
automl_setup_model_explanations
automl_explainer_setup_obj = automl_setup_model_explanations(fitted_model, X=X_train,
                                                               X_test=X_test, y=y_train,
                                                               task='regression')

from interpret.ext.glassbox import LGBMExplainableModel
from azureml.interpret.mimic_wrapper import MimicWrapper

explainer = MimicWrapper(ws, automl_explainer_setup_obj.automl_estimator, LGBMExplainableModel,
                         init_dataset=automl_explainer_setup_obj.X_transform, run=best_run,
                         features=automl_explainer_setup_obj.engineered_feature_names,
                         feature_maps=[automl_explainer_setup_obj.feature_map],
                         classes=automl_explainer_setup_obj.classes)

pip install interpret-community[visualization]

engineered_explanations = explainer.explain(['local', 'global'],
eval_dataset=automl_explainer_setup_obj.X_test_transform)
print(engineered_explanations.get_feature_importance_dict()),
from raiwidgets import ExplanationDashboard
ExplanationDashboard(engineered_explanations, automl_explainer_setup_obj.automl_estimator,
datasetX=automl_explainer_setup_obj.X_test_transform)

raw_explanations = explainer.explain(['local', 'global'], get_raw=True,
                                      raw_feature_names=automl_explainer_setup_obj.raw_feature_names,
                                      eval_dataset=automl_explainer_setup_obj.X_test_transform)
print(raw_explanations.get_feature_importance_dict()),
from raiwidgets import ExplanationDashboard
ExplanationDashboard(raw_explanations, automl_explainer_setup_obj.automl_pipeline,
datasetX=automl_explainer_setup_obj.X_test_raw)
```

Use Mimic Explainer for computing and visualizing raw feature importance

You can call the `explain()` method in MimicWrapper with the transformed test samples to get the feature importance for the raw features. In the [Machine Learning studio](#), you can view the dashboard visualization of the feature importance values of the raw features.

```
raw_explanations = explainer.explain(['local', 'global'], get_raw=True,
                                      raw_feature_names=automl_explainer_setup_obj.raw_feature_names,
                                      eval_dataset=automl_explainer_setup_obj.X_test_transform,
                                      raw_eval_dataset=automl_explainer_setup_obj.X_test_raw)
print(raw_explanations.get_feature_importance_dict())
```

Interpretability during inference

In this section, you learn how to operationalize an automated ML model with the explainer that was used to compute the explanations in the previous section.

Register the model and the scoring explainer

Use the `TreeScoringExplainer` to create the scoring explainer that'll compute the engineered feature importance values at inference time. You initialize the scoring explainer with the `feature_map` that was computed previously.

Save the scoring explainer, and then register the model and the scoring explainer with the Model Management Service. Run the following code:

```
from azureml.interpret.scoring.scoring_explainer import TreeScoringExplainer, save

# Initialize the ScoringExplainer
scoring_explainer = TreeScoringExplainer(explainer.explainer, feature_maps=
[automl_explainer_setup_obj.feature_map])

# Pickle scoring explainer locally
save(scoring_explainer, exist_ok=True)

# Register trained automl model present in the 'outputs' folder in the artifacts
original_model = automl_run.register_model(model_name='automl_model',
                                             model_path='outputs/model.pkl')

# Register scoring explainer
automl_run.upload_file('scoring_explainer.pkl', 'scoring_explainer.pkl')
scoring_explainer_model = automl_run.register_model(model_name='scoring_explainer',
                                                     model_path='scoring_explainer.pkl')
```

Create the conda dependencies for setting up the service

Next, create the necessary environment dependencies in the container for the deployed model. Please note that `azureml-defaults` with version $\geq 1.0.45$ must be listed as a pip dependency, because it contains the functionality needed to host the model as a web service.

```
from azureml.core.conda_dependencies import CondaDependencies

azureml_pip_packages = [
    'azureml-interpret', 'azureml-train-automl', 'azureml-defaults'
]

myenv = CondaDependencies.create(conda_packages=['scikit-learn', 'pandas', 'numpy', 'py-xgboost<=0.80'],
                                 pip_packages=azureml_pip_packages,
                                 pin_sdk_version=True)

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())

with open("myenv.yml","r") as f:
    print(f.read())
```

Create the scoring script

Write a script that loads your model and produces predictions and explanations based on a new batch of data.

```

%%writefile score.py
import joblib
import pandas as pd
from azureml.core.model import Model
from azureml.train.automl.runtime.automl_explain_utilities import automl_setup_model_explanations


def init():
    global automl_model
    global scoring_explainer

    # Retrieve the path to the model file using the model name
    # Assume original model is named automl_model
    automl_model_path = Model.get_model_path('automl_model')
    scoring_explainer_path = Model.get_model_path('scoring_explainer')

    automl_model = joblib.load(automl_model_path)
    scoring_explainer = joblib.load(scoring_explainer_path)


def run(raw_data):
    data = pd.read_json(raw_data, orient='records')
    # Make prediction
    predictions = automl_model.predict(data)
    # Setup for inferencing explanations
    automl_explainer_setup_obj = automl_setup_model_explanations(automl_model,
                                                                X_test=data, task='classification')
    # Retrieve model explanations for engineered explanations
    engineered_local_importance_values =
    scoring_explainer.explain(automl_explainer_setup_obj.X_test_transform)
    # Retrieve model explanations for raw explanations
    raw_local_importance_values = scoring_explainer.explain(automl_explainer_setup_obj.X_test_transform,
    get_raw=True)
    # You can return any data type as long as it is JSON-serializable
    return {'predictions': predictions.tolist(),
            'engineered_local_importance_values': engineered_local_importance_values,
            'raw_local_importance_values': raw_local_importance_values}

```

Deploy the service

Deploy the service using the conda file and the scoring file from the previous steps.

```

from azureml.core.webservice import Webservice
from azureml.core.webservice import AciWebservice
from azureml.core.model import Model, InferenceConfig
from azureml.core.environment import Environment

aciconfig = AciWebservice.deploy_configuration(cpu_cores=1,
                                                memory_gb=1,
                                                tags={"data": "Bank Marketing",
                                                      "method" : "local_explanation"},
                                                description='Get local explanations for Bank marketing test
data')
myenv = Environment.from_conda_specification(name="myenv", file_path="myenv.yml")
inference_config = InferenceConfig(entry_script="score_local_explain.py", environment=myenv)

# Use configs and models generated above
service = Model.deploy(ws,
                       'model-scoring',
                       [scoring_explainer_model, original_model],
                       inference_config,
                       aciconfig)
service.wait_for_deployment(show_output=True)

```

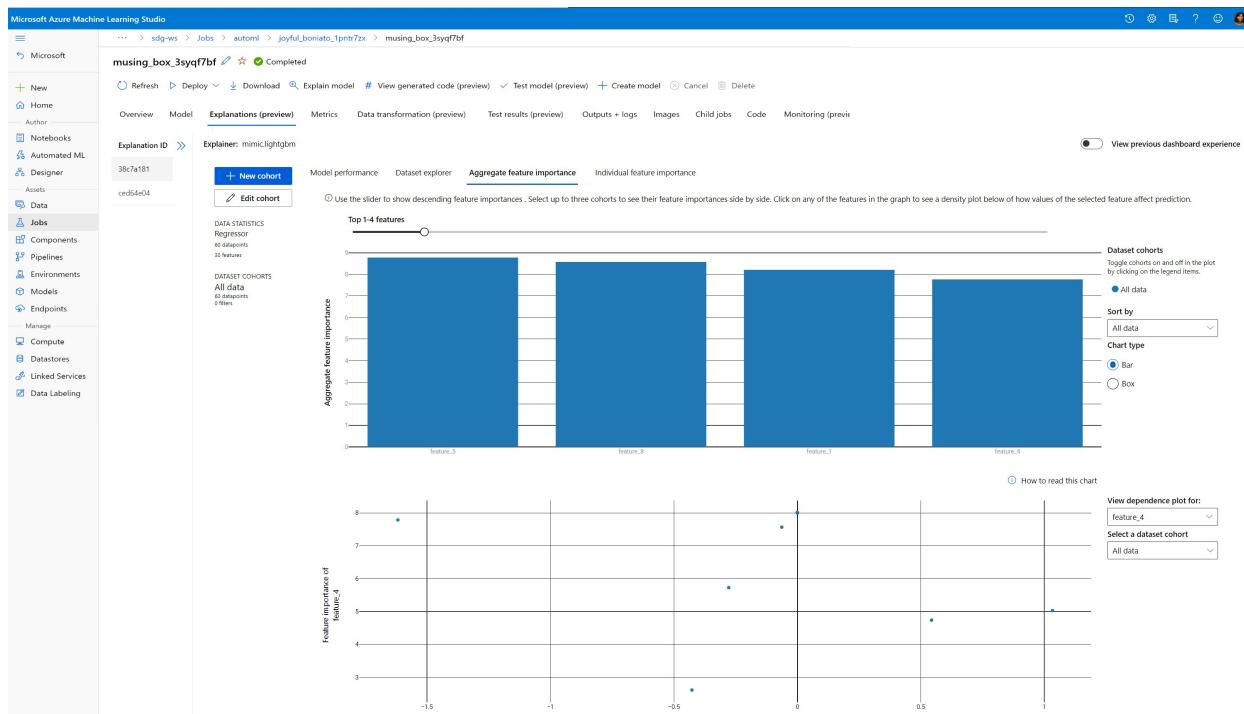
Inference with test data

Inference with some test data to see the predicted value from AutoML model, currently supported only in Azure Machine Learning SDK. View the feature importances contributing towards a predicted value.

```
if service.state == 'Healthy':
    # Serialize the first row of the test data into json
    X_test_json = X_test[:1].to_json(orient='records')
    print(X_test_json)
    # Call the service to get the predictions and the engineered explanations
    output = service.run(X_test_json)
    # Print the predicted value
    print(output['predictions'])
    # Print the engineered feature importances for the predicted value
    print(output['engineered_local_importance_values'])
    # Print the raw feature importances for the predicted value
    print('raw_local_importance_values:\n{}\n'.format(output['raw_local_importance_values']))
```

Visualize to discover patterns in data and explanations at training time

You can visualize the feature importance chart in your workspace in [Azure Machine Learning studio](#). After your AutoML run is complete, select **View model details** to view a specific run. Select the **Explanations** tab to see the visualizations in the explanation dashboard.



For more information on the explanation dashboard visualizations and specific plots, please refer to the [how-to doc on interpretability](#).

Next steps

For more information about how you can enable model explanations and feature importance in areas other than automated ML, see [more techniques for model interpretability](#).

Troubleshoot automated ML experiments in Python

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azureml v1](#)

In this guide, learn how to identify and resolve known issues in your automated machine learning experiments with the [Azure Machine Learning SDK](#).

Version dependencies

`AutoML` dependencies to newer package versions break compatibility. After SDK version 1.13.0, models aren't loaded in older SDKs due to incompatibility between the older versions pinned in previous `AutoML` packages, and the newer versions pinned today.

Expect errors such as:

- Module not found errors such as,

```
No module named 'sklearn.decomposition._truncated_svd'
```

- Import errors such as,

```
ImportError: cannot import name 'RollingOriginValidator'
```

- Attribute errors such as,

```
AttributeError: 'SimpleImputer' object has no attribute 'add_indicator'
```

Resolutions depend on your `AutoML` SDK training version:

- If your `AutoML` SDK training version is greater than 1.13.0, you need `pandas == 0.25.1` and `scikit-learn==0.22.1`.

- If there is a version mismatch, upgrade scikit-learn and/or pandas to correct version with the following,

```
pip install --upgrade pandas==0.25.1
pip install --upgrade scikit-learn==0.22.1
```

- If your `AutoML` SDK training version is less than or equal to 1.12.0, you need `pandas == 0.23.4` and `scikit-learn==0.20.3`.

- If there is a version mismatch, downgrade scikit-learn and/or pandas to correct version with the following,

```
pip install --upgrade pandas==0.23.4
pip install --upgrade scikit-learn==0.20.3
```

Setup

`AutoML` package changes since version 1.0.76 require the previous version to be uninstalled before updating to the new version.

- `ImportError: cannot import name AutoMLConfig`

If you encounter this error after upgrading from an SDK version before v1.0.76 to v1.0.76 or later, resolve the error by running: `pip uninstall azureml-train automl` and then `pip install azureml-train-automl`. The `automl_setup.cmd` script does this automatically.

- **automl_setup fails**

- On Windows, run `automl_setup` from an Anaconda Prompt. [Install Miniconda](#).
- Ensure that conda 64-bit version 4.4.10 or later is installed. You can check the bit with the `conda info` command. The `platform` should be `win-64` for Windows or `osx-64` for Mac. To check the version use the command `conda -v`. If you have a previous version installed, you can update it by using the command: `conda update conda`. To check 32-bit by running
- Ensure that conda is installed.
- Linux - `gcc: error trying to exec 'cc1plus'`
 1. If the `gcc: error trying to exec 'cc1plus': execvp: No such file or directory` error is encountered, install the GCC build tools for your Linux distribution. For example, on Ubuntu, use the command `sudo apt-get install build-essential`.
 2. Pass a new name as the first parameter to `automl_setup` to create a new conda environment. View existing conda environments using `conda env list` and remove them with `conda env remove -n <environmentname>`.

- **automl_setup_linux.sh fails:** If `automl_setup_linux.sh` fails on Ubuntu Linux with the error:

`unable to execute 'gcc': No such file or directory`

1. Make sure that outbound ports 53 and 80 are enabled. On an Azure virtual machine, you can do this from the Azure portal by selecting the VM and clicking on **Networking**.
2. Run the command: `sudo apt-get update`
3. Run the command: `sudo apt-get install build-essential --fix-missing`
4. Run `automl_setup_linux.sh` again

- **configuration.ipynb fails:**

- For local conda, first ensure that `automl_setup` has successfully run.
- Ensure that the `subscription_id` is correct. Find the `subscription_id` in the Azure portal by selecting All Service and then Subscriptions. The characters "<" and ">" should not be included in the `subscription_id` value. For example, `subscription_id = "12345678-90ab-1234-5678-1234567890abcd"` has the valid format.
- Ensure Contributor or Owner access to the subscription.
- Check that the region is one of the supported regions: `eastus2`, `eastus`, `westcentralus`, `southeastasia`, `westeurope`, `australiaeast`, `westus2`, `southcentralus`.
- Ensure access to the region using the Azure portal.

- **workspace.from_config fails:**

If the call `ws = Workspace.from_config()` fails:

1. Ensure that the `configuration.ipynb` notebook has run successfully.
2. If the notebook is being run from a folder that is not under the folder where the `configuration.ipynb` was run, copy the folder `aml_config` and the file `config.json` that it contains to the new folder. `Workspace.from_config` reads the `config.json` for the notebook folder or its parent folder.
3. If a new subscription, resource group, workspace, or region, is being used, make sure that you run the

`configuration.ipynb` notebook again. Changing config.json directly will only work if the workspace already exists in the specified resource group under the specified subscription.

4. If you want to change the region, change the workspace, resource group, or subscription.

`Workspace.create` will not create or update a workspace if it already exists, even if the region specified is different.

TensorFlow

As of version 1.5.0 of the SDK, automated machine learning does not install TensorFlow models by default. To install TensorFlow and use it with your automated ML experiments, install `tensorflow==1.12.0` via `CondaDependencies`.

```
from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies
run_config = RunConfiguration()
run_config.environment.python.conda_dependencies = CondaDependencies.create(conda_packages=['tensorflow==1.12.0'])
```

Numpy failures

- `import numpy` fails in Windows: Some Windows environments see an error loading numpy with the latest Python version 3.6.8. If you see this issue, try with Python version 3.6.7.
- `import numpy` fails: Check the TensorFlow version in the automated ml conda environment. Supported versions are < 1.13. Uninstall TensorFlow from the environment if version is >= 1.13.

You can check the version of TensorFlow and uninstall as follows:

1. Start a command shell, activate conda environment where automated ml packages are installed.
2. Enter `pip freeze` and look for `tensorflow`, if found, the version listed should be < 1.13
3. If the listed version is not a supported version, `pip uninstall tensorflow` in the command shell and enter `y` for confirmation.

`jwt.exceptions.DecodeError`

Exact error message:

```
jwt.exceptions.DecodeError: It is required that you pass in a value for the "algorithms" argument when calling decode()
```

For SDK versions <= 1.17.0, installation might result in an unsupported version of PyJWT. Check that the PyJWT version in the automated ml conda environment is a supported version. That is PyJWT version < 2.0.0.

You may check the version of PyJWT as follows:

1. Start a command shell and activate conda environment where automated ML packages are installed.
2. Enter `pip freeze` and look for `PyJWT`, if found, the version listed should be < 2.0.0

If the listed version is not a supported version:

1. Consider upgrading to the latest version of AutoML SDK: `pip install -U azureml-sdk[automl]`
2. If that is not viable, uninstall PyJWT from the environment and install the right version as follows:
 - a. `pip uninstall PyJWT` in the command shell and enter `y` for confirmation.

b. Install using `pip install 'PyJWT<2.0.0'`.

Data access

For automated ML jobs, you need to ensure the file datastore that connects to your AzureFile storage has the appropriate authentication credentials. Otherwise, the following message results. Learn how to [update your data access authentication credentials](#).

Error message:

```
Could not create a connection to the AzureFileService due to missing credentials. Either an Account Key or SAS token needs to be linked the default workspace blob store.
```

Data schema

When you try to create a new automated ML experiment via the **Edit and submit** button in the Azure Machine Learning studio, the data schema for the new experiment must match the schema of the data that was used in the original experiment. Otherwise, an error message similar to the following results. Learn more about how to [edit and submit experiments from the studio UI](#).

Error message non-vision experiments:

```
Schema mismatch error: (an) additional column(s): "Column1: String, Column2: String, Column3: String", (a) missing column(s)
```

Error message for vision datasets:

```
Schema mismatch error: (an) additional column(s): "dataType: String, dataSubtype: String, dateTIme: Date, category: String, subcategory: String, status: String, address: String, latitude: Decimal, longitude: Decimal, source: String, extendedProperties: String", (a) missing column(s): "image_url: Stream, image_details: DataRow, label: List" Vision dataset error(s): Vision dataset should have a target column with name 'label'. Vision dataset should have labelingProjectType tag with value as 'Object Identification (Bounding Box)'.
```

Databricks

See [How to configure an automated ML experiment with Databricks](#).

Forecasting R2 score is always zero

This issue arises if the training data provided has time series that contains the same value for the last `n_cv_splits` + `forecasting_horizon` data points.

If this pattern is expected in your time series, you can switch your primary metric to **normalized root mean squared error**.

Failed deployment

For versions <= 1.18.0 of the SDK, the base image created for deployment may fail with the following error:

```
ImportError: cannot import name cached_property from werkzeug .
```

The following steps can work around the issue:

1. Download the model package
2. Unzip the package
3. Deploy using the unzipped assets

Azure Functions application

Automated ML does not currently support Azure Functions applications.

Sample notebook failures

If a sample notebook fails with an error that property, method, or library does not exist:

- Ensure that the correct kernel has been selected in the Jupyter Notebook. The kernel is displayed in the top right of the notebook page. The default is *azure_automl*. The kernel is saved as part of the notebook. If you switch to a new conda environment, you need to select the new kernel in the notebook.
 - For Azure Notebooks, it should be Python 3.6.
 - For local conda environments, it should be the conda environment name that you specified in `automl_setup`.
- To ensure the notebook is for the SDK version that you are using,
 - Check the SDK version by executing `azureml.core.VERSION` in a Jupyter Notebook cell.
 - You can download previous version of the sample notebooks from GitHub with these steps:
 1. Select the `Branch` button
 2. Navigate to the `Tags` tab
 3. Select the version

Experiment throttling

If you have over 100 automated ML experiments, this may cause new automated ML experiments to have long run times.

VNet Firewall Setting Download Failure

If you are under virtual networks (VNets), you may run into model download failures when using AutoML NLP. This is because network traffic is blocked from downloading the models and tokenizers from Azure CDN. To unblock this, please allow list the below URLs in the "Application rules" setting of the VNet firewall policy:

- aka.ms
- <https://automlresources-prod.azureedge.net>

Please follow the instructions [here to configure the firewall settings](#).

Instructions for configuring workspace under vnet are available [here](#).

Next steps

- Learn more about [how to train a regression model with Automated machine learning](#) or [how to train using Automated machine learning on a remote resource](#).
- Learn more about [how and where to deploy a model](#).

Deploy and score a machine learning model by using an online endpoint

9/22/2022 • 16 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

Learn how to use an online endpoint to deploy your model, so you don't have to create and manage the underlying infrastructure. You'll begin by deploying a model on your local machine to debug any errors, and then you'll deploy and test it in Azure.

You'll also learn how to view the logs and monitor the service-level agreement (SLA). You start with a model and end up with a scalable HTTPS/REST endpoint that you can use for online and real-time scoring.

Managed online endpoints help to deploy your ML models in a turnkey manner. Managed online endpoints work with powerful CPU and GPU machines in Azure in a scalable, fully managed way. Managed online endpoints take care of serving, scaling, securing, and monitoring your models, freeing you from the overhead of setting up and managing the underlying infrastructure. The main example in this doc uses managed online endpoints for deployment. To use Kubernetes instead, see the notes in this document inline with the managed online endpoint discussion. For more information, see [What are Azure Machine Learning endpoints?](#).

Prerequisites

- To use Azure Machine Learning, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- Install and configure the Azure CLI and the `ml` extension to the Azure CLI. For more information, see [Install, set up, and use the CLI \(v2\)](#).
- You must have an Azure resource group, and you (or the service principal you use) must have Contributor access to it. A resource group is created in [Install, set up, and use the CLI \(v2\)](#).
- You must have an Azure Machine Learning workspace. A workspace is created in [Install, set up, and use the CLI \(v2\)](#).
- If you haven't already set the defaults for the Azure CLI, save your default settings. To avoid passing in the values for your subscription, workspace, and resource group multiple times, run this code:

```
az account set --subscription <subscription ID>
az configure --defaults workspace=<Azure Machine Learning workspace name> group=<resource group>
```

- Azure role-based access controls (Azure RBAC) are used to grant access to operations in Azure Machine Learning. To perform the steps in this article, your user account must be assigned the **owner** or **contributor** role for the Azure Machine Learning workspace, or a custom role allowing `Microsoft.MachineLearningServices/workspaces/onlineEndpoints/*`. For more information, see [Manage access to an Azure Machine Learning workspace](#).
- (Optional) To deploy locally, you must [install Docker Engine](#) on your local computer. We *highly recommend* this option, so it's easier to debug issues.

IMPORTANT

The examples in this document assume that you are using the Bash shell. For example, from a Linux system or [Windows Subsystem for Linux](#).

Prepare your system

To follow along with this article, first clone the samples repository (`azureml-examples`). Then, run the following code to go to the samples directory:

```
git clone https://github.com/Azure/azureml-examples  
cd azureml-examples  
cd cli
```

To set your endpoint name, choose one of the following commands, depending on your operating system (replace `<YOUR-ENDPOINT-NAME>` with a unique name).

For Unix, run this command:

```
export ENDPOINT_NAME=<YOUR-ENDPOINT-NAME>"
```

NOTE

Endpoint names must be unique within an Azure region. For example, in the Azure `westus2` region, there can be only one endpoint with the name `my-endpoint`.

Review the endpoint and deployment configurations

The following snippet shows the `endpoints/online/managed/sample/endpoint.yml` file:

```
$schema: https://azureschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json  
name: my-endpoint  
auth_mode: key
```

NOTE

For a full description of the YAML, see [Online endpoint YAML reference](#).

The reference for the endpoint YAML format is described in the following table. To learn how to specify these attributes, see the YAML example in [Prepare your system](#) or the [online endpoint YAML reference](#). For information about limits related to managed endpoints, see [Manage and increase quotas for resources with Azure Machine Learning](#).

KEY	DESCRIPTION
<code>\$schema</code>	(Optional) The YAML schema. To see all available options in the YAML file, you can view the schema in the preceding example in a browser.

KEY	DESCRIPTION
<code>name</code>	The name of the endpoint. It must be unique in the Azure region. Naming rules are defined under managed online endpoint limits .
<code>auth_mode</code>	Use <code>key</code> for key-based authentication. Use <code>aml_token</code> for Azure Machine Learning token-based authentication. <code>key</code> doesn't expire, but <code>aml_token</code> does expire. (Get the most recent token by using the <code>az ml online-endpoint get-credentials</code> command.)

The example contains all the files needed to deploy a model on an online endpoint. To deploy a model, you must have:

- Model files (or the name and version of a model that's already registered in your workspace). In the example, we have a scikit-learn model that does regression.
- The code that's required to score the model. In this case, we have a `score.py` file.
- An environment in which your model runs. As you'll see, the environment might be a Docker image with Conda dependencies, or it might be a Dockerfile.
- Settings to specify the instance type and scaling capacity.

The following snippet shows the `endpoints/online/managed/sample/blue-deployment.yml` file, with all the required inputs:

```
$schema: https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: blue
endpoint_name: my-endpoint
model:
  path: ../../model-1/model/
code_configuration:
  code: ../../model-1/onlinescorer/
  scoring_script: score.py
environment:
  conda_file: ../../model-1/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1
instance_type: Standard_DS2_v2
instance_count: 1
```

The table describes the attributes of a `deployment`:

KEY	DESCRIPTION
<code>name</code>	The name of the deployment.
<code>model</code>	In this example, we specify the model properties inline: <code>path</code> . Model files are automatically uploaded and registered with an autogenerated name. For related best practices, see the tip in the next section.
<code>code_configuration.code.path</code>	The directory on the local development environment that contains all the Python source code for scoring the model. You can use nested directories and packages.

KEY	DESCRIPTION
<code>code_configuration.scoring_script</code>	The Python file that's in the <code>code_configuration.code.path</code> scoring directory on the local development environment. This Python code must have an <code>init()</code> function and a <code>run()</code> function. The function <code>init()</code> will be called after the model is created or updated (you can use it to cache the model in memory, for example). The <code>run()</code> function is called at every invocation of the endpoint to do the actual scoring and prediction.
<code>environment</code>	Contains the details of the environment to host the model and code. In this example, we have inline definitions that include the <code>path</code> . We'll use <code>environment.docker.image</code> for the image. The <code>conda_file</code> dependencies will be installed on top of the image. For more information, see the tip in the next section.
<code>instance_type</code>	The VM SKU that will host your deployment instances. For more information, see Managed online endpoints supported VM SKUs .
<code>instance_count</code>	The number of instances in the deployment. Base the value on the workload you expect. For high availability, we recommend that you set <code>instance_count</code> to at least <code>3</code> . We reserve an extra 20% for performing upgrades. For more information, see managed online endpoint quotas .

During deployment, the local files such as the Python source for the scoring model, are uploaded from the development environment.

For more information about the YAML schema, see the [online endpoint YAML reference](#).

NOTE

To use Kubernetes instead of managed endpoints as a compute target:

1. Create and attach your Kubernetes cluster as a compute target to your Azure Machine Learning workspace by using [Azure Machine Learning studio](#).
2. Use the [endpoint YAML](#) to target Kubernetes instead of the managed endpoint YAML. You'll need to edit the YAML to change the value of `target` to the name of your registered compute target. You can use this [deployment.yaml](#) that has additional properties applicable to Kubernetes deployment.

All the commands that are used in this article (except the optional SLA monitoring and Azure Log Analytics integration) can be used either with managed endpoints or with Kubernetes endpoints.

Register your model and environment separately

In this example, we specify the `path` (where to upload files from) inline. The CLI automatically uploads the files and registers the model and environment. As a best practice for production, you should register the model and environment and specify the registered name and version separately in the YAML. Use the form

```
model: azureml:my-model:1 or environment: azureml:my-env:1.
```

For registration, you can extract the YAML definitions of `model` and `environment` into separate YAML files and use the commands `az ml model create` and `az ml environment create`. To learn more about these commands, run `az ml model create -h` and `az ml environment create -h`.

Use different CPU and GPU instance types

The preceding YAML uses a general-purpose type (`Standard_F2s_v2`) and a non-GPU Docker image (in the YAML, see the `image` attribute). For GPU compute, choose a GPU compute type SKU and a GPU Docker image.

For supported general-purpose and GPU instance types, see [Managed online endpoints supported VM SKUs](#). For a list of Azure Machine Learning CPU and GPU base images, see [Azure Machine Learning base images](#).

NOTE

To use Kubernetes instead of managed endpoints as a compute target, see [Introduction to Kubermentes compute target](#)

Use more than one model

Currently, you can specify only one model per deployment in the YAML. If you've more than one model, when you register the model, copy all the models as files or subdirectories into a folder that you use for registration. In your scoring script, use the environment variable `AZUREML_MODEL_DIR` to get the path to the model root folder. The underlying directory structure is retained.

Understand the scoring script

TIP

The format of the scoring script for online endpoints is the same format that's used in the preceding version of the CLI and in the Python SDK.

As noted earlier, the `code_configuration.scoring_script` must have an `init()` function and a `run()` function. This example uses the [score.py file](#). The `init()` function is called when the container is initialized or started. Initialization typically occurs shortly after the deployment is created or updated. Write logic here for global initialization operations like caching the model in memory (as we do in this example). The `run()` function is called for every invocation of the endpoint and should do the actual scoring and prediction. In the example, we extract the data from the JSON input, call the scikit-learn model's `predict()` method, and then return the result.

Deploy and debug locally by using local endpoints

To save time debugging, we *highly recommend* that you test-run your endpoint locally. For more, see [Debug online endpoints locally in Visual Studio Code](#).

NOTE

- To deploy locally, [Docker Engine](#) must be installed.
- Docker Engine must be running. Docker Engine typically starts when the computer starts. If it doesn't, you can [troubleshoot Docker Engine](#).

IMPORTANT

The goal of a local endpoint deployment is to validate and debug your code and configuration before you deploy to Azure. Local deployment has the following limitations:

- Local endpoints do *not* support traffic rules, authentication, or probe settings.
- Local endpoints support only one deployment per endpoint.

Deploy the model locally

First create the endpoint. Optionally, for a local endpoint, you can skip this step and directly create the

deployment (next step), which will, in turn, create the required metadata. This is useful for development and testing purposes.

```
az ml online-endpoint create --local -n $ENDPOINT_NAME -f endpoints/online/managed/sample/endpoint.yml
```

Now, create a deployment named `blue` under the endpoint.

```
az ml online-deployment create --local -n blue --endpoint $ENDPOINT_NAME -f endpoints/online/managed/sample/blue-deployment.yml
```

The `--local` flag directs the CLI to deploy the endpoint in the Docker environment.

TIP

Use Visual Studio Code to test and debug your endpoints locally. For more information, see [debug online endpoints locally in Visual Studio Code](#).

Verify the local deployment succeeded

Check the status to see whether the model was deployed without error:

```
az ml online-endpoint show -n $ENDPOINT_NAME --local
```

The output should appear similar to the following JSON. The `provisioning_state` is `Succeeded`.

```
{
  "auth_mode": "key",
  "location": "local",
  "name": "docs-endpoint",
  "properties": {},
  "provisioning_state": "Succeeded",
  "scoring_uri": "http://localhost:49158/score",
  "tags": {},
  "traffic": {}
}
```

The following table contains the possible values for `provisioning_state`:

STATE	DESCRIPTION
Creating	The resource is being created.
Updating	The resource is being updated.
Deleting	The resource is being deleted.
Succeeded	The create/update operation was successful.
Failed	The create/update/delete operation has failed.

Invoke the local endpoint to score data by using your model

Invoke the endpoint to score the model by using the convenience command `invoke` and passing query parameters that are stored in a JSON file:

```
az ml online-endpoint invoke --local --name $ENDPOINT_NAME --request-file endpoints/online/model-1/sample-request.json
```

If you want to use a REST client (like curl), you must have the scoring URI. To get the scoring URI, run

```
az ml online-endpoint show --local -n $ENDPOINT_NAME
```

In the returned data, find the `scoring_uri` attribute.

Sample curl based commands are available later in this doc.

Review the logs for output from the invoke operation

In the example `score.py` file, the `run()` method logs some output to the console. You can view this output by using the `get-logs` command again:

```
az ml online-deployment get-logs --local -n blue --endpoint $ENDPOINT_NAME
```

Deploy your online endpoint to Azure

Next, deploy your online endpoint to Azure.

Deploy to Azure

To create the endpoint in the cloud, run the following code:

```
az ml online-endpoint create --name $ENDPOINT_NAME -f endpoints/online/managed/sample/endpoint.yml
```

To create the deployment named `blue` under the endpoint, run the following code:

```
az ml online-deployment create --name blue --endpoint $ENDPOINT_NAME -f endpoints/online/managed/sample/blue-deployment.yml --all-traffic
```

This deployment might take up to 15 minutes, depending on whether the underlying environment or image is being built for the first time. Subsequent deployments that use the same environment will finish processing more quickly.

IMPORTANT

The `--all-traffic` flag in the above `az ml online-deployment create` allocates 100% of the traffic to the endpoint to the newly created deployment. Though this is helpful for development and testing purposes, for production, you might want to open traffic to the new deployment through an explicit command. For example,

```
az ml online-endpoint update -n $ENDPOINT_NAME --traffic "blue=100"
```

TIP

- If you prefer not to block your CLI console, you may add the flag `--no-wait` to the command. However, this will stop the interactive display of the deployment status.
- Use [Troubleshooting online endpoints deployment](#) to debug errors.

Check the status of the deployment

The `show` command contains information in `provisioning_status` for endpoint and deployment:

```
az ml online-endpoint show -n $ENDPOINT_NAME
```

You can list all the endpoints in the workspace in a table format by using the `list` command:

```
az ml online-endpoint list --output table
```

Check the status of the cloud deployment

Check the logs to see whether the model was deployed without error:

```
az ml online-deployment get-logs --name blue --endpoint $ENDPOINT_NAME
```

By default, logs are pulled from inference-server. To see the logs from storage-initializer (it mounts assets like model and code to the container), add the `--container storage-initializer` flag.

Invoke the endpoint to score data by using your model

You can use either the `invoke` command or a REST client of your choice to invoke the endpoint and score some data:

```
az ml online-endpoint invoke --name $ENDPOINT_NAME --request-file endpoints/online/model-1/sample-request.json
```

The following example shows how to get the key used to authenticate to the endpoint:

TIP

You can control which Azure Active Directory security principals can get the authentication key by assigning them to a custom role that allows `Microsoft.MachineLearningServices/workspaces/onlineEndpoints/token/action` and `Microsoft.MachineLearningServices/workspaces/onlineEndpoints/listkeys/action`. For more information, see [Manage access to an Azure Machine Learning workspace](#).

```
ENDPOINT_KEY=$(az ml online-endpoint get-credentials -n $ENDPOINT_NAME -o tsv --query primaryKey)
```

Next, use curl to score data.

```
SCORING_URI=$(az ml online-endpoint show -n $ENDPOINT_NAME -o tsv --query scoring_uri)

curl --request POST "$SCORING_URI" --header "Authorization: Bearer $ENDPOINT_KEY" --header 'Content-Type: application/json' --data @endpoints/online/model-1/sample-request.json
```

Notice we use `show` and `get-credentials` commands to get the authentication credentials. Also notice that we're using the `--query` flag to filter attributes to only what we need. To learn more about `--query`, see [Query Azure CLI command output](#).

To see the invocation logs, run `get-logs` again.

For information on authenticating using a token, see [Authenticate to online endpoints](#).

(Optional) Update the deployment

If you want to update the code, model, or environment, update the YAML file, and then run the `az ml online-endpoint update` command.

NOTE

If you update instance count and along with other model settings (code, model, or environment) in a single `update` command: first the scaling operation will be performed, then the other updates will be applied. In production environment is a good practice to perform these operations separately.

To understand how `update` works:

1. Open the file `online/model-1/onlinescorer/scor.py`.
2. Change the last line of the `init()` function: After `logging.info("Init complete")`, add
`logging.info("Updated successfully")`.
3. Save the file.
4. Run this command:

```
az ml online-deployment update -n blue --endpoint $ENDPOINT_NAME -f  
endpoints/online/managed/sample/blue-deployment.yml
```

NOTE

Updating by using YAML is declarative. That is, changes in the YAML are reflected in the underlying Azure Resource Manager resources (endpoints and deployments). A declarative approach facilitates **GitOps**: All changes to endpoints and deployments (even `instance_count`) go through the YAML. You can make updates without using the YAML by using the `--set` flag.

5. Because you modified the `init()` function (`init()` runs when the endpoint is created or updated), the message `Updated successfully` will be in the logs. Retrieve the logs by running:

```
az ml online-deployment get-logs --name blue --endpoint $ENDPOINT_NAME
```

The `update` command also works with local deployments. Use the same `az ml online-deployment update` command with the `--local` flag.

TIP

With the `update` command, you can use the `--set` parameter in the Azure CLI to override attributes in your YAML or to set specific attributes without passing the YAML file. Using `--set` for single attributes is especially valuable in development and test scenarios. For example, to scale up the `instance_count` value for the first deployment, you could use the `--set instance_count=2` flag. However, because the YAML isn't updated, this technique doesn't facilitate **GitOps**.

NOTE

The above is an example of inplace rolling update.

- For managed online endpoint, the same deployment is updated with the new configuration, with 20% nodes at a time, i.e. if the deployment has 10 nodes, 2 nodes at a time will be updated.
- For Kubernetes online endpoint, the system will iterately create a new deployment instance with the new configuration and delete the old one.
- For production usage, you might want to consider [blue-green deployment](#), which offers a safer alternative.

(Optional) Configure autoscaling

Autoscale automatically runs the right amount of resources to handle the load on your application. Managed online endpoints support autoscaling through integration with the Azure monitor autoscale feature. To configure autoscaling, see [How to autoscale online endpoints](#).

(Optional) Monitor SLA by using Azure Monitor

To view metrics and set alerts based on your SLA, complete the steps that are described in [Monitor online endpoints](#).

(Optional) Integrate with Log Analytics

The `get-logs` command provides only the last few hundred lines of logs from an automatically selected instance. However, Log Analytics provides a way to durably store and analyze logs. For more information on using logging, see [Monitor online endpoints](#)

How to configure emails in the studio (preview)

To start receiving emails when your job, online endpoint, or batch endpoint is complete or if there's an issue (failed, canceled), follow the preceding instructions.

1. In Azure ML studio, go to settings by selecting the gear icon.
2. Select the **Email notifications** tab.
3. Toggle to enable or disable email notifications for a specific event.



Settings

X

Language

Email Notifications

These settings apply across all workspaces you have access to

Job succeeded

Send me an email when my job succeeded



Enabled

Issue with my job

Send me an email when my job failed or cancelled



Enabled

Issue with my endpoint deployment

Send me an email when my inferencing endpoint deployment failed or cancelled



Enabled

Endpoint deployment succeeded

Send me an email when my inferencing endpoint deployment succeeded



Enabled

Delete the endpoint and the deployment

If you aren't going to use the deployment, you should delete it by running the following code (it deletes the endpoint and all the underlying deployments):

```
az ml online-endpoint delete --name $ENDPOINT_NAME --yes --no-wait
```

Next steps

To learn more, review these articles:

- [Deploy models with REST](#)
- [Create and use online endpoints in the studio](#)
- [Safe rollout for online endpoints](#)
- [How to autoscale managed online endpoints](#)
- [Use batch endpoints for batch scoring](#)
- [View costs for an Azure Machine Learning managed online endpoint](#)
- [Access Azure resources with a online endpoint and managed identity](#)
- [Troubleshoot online endpoints deployment](#)
- [Enable network isolation with managed online endpoints](#)

Deploy and score a machine learning model with managed online endpoint using Python SDK v2 (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you learn how to deploy your machine learning model to managed online endpoint and get predictions. You'll begin by deploying a model on your local machine to debug any errors, and then you'll deploy and test it in Azure.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Quickstart: Create workspace resources](#) article to create one.
- To install the Python SDK v2, use the following command:

```
pip install --pre azure-ai-ml
```

For more information, see [Install the Python SDK v2 for Azure Machine Learning \(preview\)](#).

- To deploy locally, you must install [Docker Engine](#) on your local computer. We highly recommend this option, so it's easier to debug issues.

Clone examples repository

To run the training examples, first clone the examples repository and change into the `sdk` directory:

```
git clone --depth 1 https://github.com/Azure/azureml-examples
cd azureml-examples/sdk
```

TIP

Use `--depth 1` to clone only the latest commit to the repository, which reduces time to complete the operation.

Connect to Azure Machine Learning workspace

The [workspace](#) is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. In this section, we'll connect to the workspace in which you'll perform deployment tasks.

1. Import the required libraries:

```
# import required libraries
from azure.ai.ml import MLClient
from azure.ai.ml.entities import (
    ManagedOnlineEndpoint,
    ManagedOnlineDeployment,
    Model,
    Environment,
    CodeConfiguration,
)
from azure.identity import DefaultAzureCredential
```

2. Configure workspace details and get a handle to the workspace:

To connect to a workspace, we need identifier parameters - a subscription, resource group and workspace name. We'll use these details in the `MLClient` from `azure.ai.ml` to get a handle to the required Azure Machine Learning workspace. This example uses the [default Azure authentication](#).

```
# enter details of your AzureML workspace
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
workspace = "<AZUREML_WORKSPACE_NAME>"
```

```
# get a handle to the workspace
ml_client = MLClient(
    DefaultAzureCredential(), subscription_id, resource_group, workspace
)
```

Create local endpoint and deployment

NOTE

To deploy locally, [Docker Engine](#) must be installed. Docker Engine must be running. Docker Engine typically starts when the computer starts. If it doesn't, you can [troubleshoot Docker Engine](#).

1. Create local endpoint:

The goal of a local endpoint deployment is to validate and debug your code and configuration before you deploy to Azure. Local deployment has the following limitations:

- Local endpoints don't support traffic rules, authentication, or probe settings.
- Local endpoints support only one deployment per endpoint.

```

# Creating a local endpoint
import datetime

local_endpoint_name = "local-" + datetime.datetime.now().strftime("%m%d%H%M%f")

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=local_endpoint_name, description="this is a sample local endpoint"
)

```

```
ml_client.online_endpoints.begin_create_or_update(endpoint, local=True)
```

2. Create local deployment:

The example contains all the files needed to deploy a model on an online endpoint. To deploy a model, you must have:

- Model files (or the name and version of a model that's already registered in your workspace). In the example, we have a scikit-learn model that does regression.
- The code that's required to score the model. In this case, we have a `score.py` file.
- An environment in which your model runs. As you'll see, the environment might be a Docker image with Conda dependencies, or it might be a Dockerfile.
- Settings to specify the instance type and scaling capacity.

Key aspects of deployment

- `name` - Name of the deployment.
- `endpoint_name` - Name of the endpoint to create the deployment under.
- `model` - The model to use for the deployment. This value can be either a reference to an existing versioned model in the workspace or an inline model specification.
- `environment` - The environment to use for the deployment. This value can be either a reference to an existing versioned environment in the workspace or an inline environment specification.
- `code_configuration` - the configuration for the source code and scoring script
 - `path` - Path to the source code directory for scoring the model
 - `scoring_script` - Relative path to the scoring file in the source code directory
- `instance_type` - The VM size to use for the deployment. For the list of supported sizes, see [Managed online endpoints SKU list](#).
- `instance_count` - The number of instances to use for the deployment

```

model = Model(path="../model-1/model/sklearn_regression_model.pkl")
env = Environment(
    conda_file="../model-1/environment/conda.yml",
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1",
)

blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=local_endpoint_name,
    model=model,
    environment=env,
    code_configuration=CodeConfiguration(
        code="../model-1/onlinescoring", scoring_script="score.py"
    ),
    instance_type="Standard_F2s_v2",
    instance_count=1,
)

```

```
ml_client.online_deployments.begin_create_or_update(  
    deployment=blue_deployment, local=True  
)
```

Verify the local deployment succeeded

1. Check the status to see whether the model was deployed without error:

```
ml_client.online_endpoints.get(name=local_endpoint_name, local=True)
```

2. Get logs:

```
ml_client.online_deployments.get_logs(  
    name="blue", endpoint_name=local_endpoint_name, local=True, lines=50  
)
```

Invoke the local endpoint

Invoke the endpoint to score the model by using the convenience command `invoke` and passing query parameters that are stored in a JSON file

```
ml_client.online_endpoints.invoke(  
    endpoint_name=local_endpoint_name,  
    request_file="../model-1/sample-request.json",  
    local=True,  
)
```

Deploy your online endpoint to Azure

Next, deploy your online endpoint to Azure.

1. Configure online endpoint:

TIP

- `endpoint_name` : The name of the endpoint. It must be unique in the Azure region. For more information on the naming rules, see [managed online endpoint limits](#).
- `auth_mode` : Use `key` for key-based authentication. Use `aml_token` for Azure Machine Learning token-based authentication. A `key` doesn't expire, but `aml_token` does expire. For more information on authenticating, see [Authenticate to an online endpoint](#).
- Optionally, you can add description, tags to your endpoint.

```

# Creating a unique endpoint name with current datetime to avoid conflicts
import datetime

online_endpoint_name = "endpoint-" + datetime.datetime.now().strftime("%m%d%H%M%f")

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is a sample online endpoint",
    auth_mode="key",
    tags={"foo": "bar"},
)

```

2. Create the endpoint:

Using the `MLClient` created earlier, we'll now create the Endpoint in the workspace. This command will start the endpoint creation and return a confirmation response while the endpoint creation continues.

```
ml_client.begin_create_or_update(endpoint)
```

3. Configure online deployment:

A deployment is a set of resources required for hosting the model that does the actual inferencing. We'll create a deployment for our endpoint using the `ManagedOnlineDeployment` class.

```

model = Model(path="../model-1/model/sklearn_regression_model.pkl")
env = Environment(
    conda_file="../model-1/environment/conda.yml",
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1",
)

blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    environment=env,
    code_configuration=CodeConfiguration(
        code="../model-1/onlinescoring", scoring_script="score.py"
    ),
    instance_type="Standard_F2s_v2",
    instance_count=1,
)

```

4. Create the deployment:

Using the `MLClient` created earlier, we'll now create the deployment in the workspace. This command will start the deployment creation and return a confirmation response while the deployment creation continues.

```
ml_client.begin_create_or_update(blue_deployment)
```

```

# blue deployment takes 100 traffic
endpoint.traffic = {"blue": 100}
ml_client.begin_create_or_update(endpoint)

```

Test the endpoint with sample data

Using the `MLClient` created earlier, we'll get a handle to the endpoint. The endpoint can be invoked using the `invoke` command with the following parameters:

- `endpoint_name` - Name of the endpoint
- `request_file` - File with request data
- `deployment_name` - Name of the specific deployment to test in an endpoint

We'll send a sample request using a `json` file.

```
# test the blue deployment with some sample data
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    deployment_name="blue",
    request_file="../model-1/sample-request.json",
)
```

Managing endpoints and deployments

1. Get details of the endpoint:

```
# Get the details for online endpoint
endpoint = ml_client.online_endpoints.get(name=online_endpoint_name)

# existing traffic details
print(endpoint.traffic)

# Get the scoring URI
print(endpoint.scoring_uri)
```

2. Get the logs for the new deployment:

Get the logs for the green deployment and verify as needed

```
ml_client.online_deployments.get_logs(
    name="blue", endpoint_name=online_endpoint_name, lines=50
)
```

Delete the endpoint

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Next steps

Try these next steps to learn how to use the Azure Machine Learning SDK (v2) for Python:

- [Managed online endpoint safe rollout](#)
- Explore online endpoint samples - <https://github.com/Azure/azureml-examples/tree/main/sdk/endpoints>

Safe rollout for online endpoints

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

You've an existing model deployed in production and you want to deploy a new version of the model. How do you roll out your new ML model without causing any disruption? A good answer is blue-green deployment, an approach in which a new version of a web service is introduced to production by rolling out the change to a small subset of users/requests before rolling it out completely. This article assumes you're using online endpoints; for more information, see [What are Azure Machine Learning endpoints?](#).

In this article, you'll learn to:

- Deploy a new online endpoint called "blue" that serves version 1 of the model
- Scale this deployment so that it can handle more requests
- Deploy version 2 of the model to an endpoint called "green" that accepts no live traffic
- Test the green deployment in isolation
- Send 10% of live traffic to the green deployment
- Fully cut-over all live traffic to the green deployment
- Delete the now-unused v1 blue deployment

Prerequisites

- To use Azure machine learning, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- You must install and configure the Azure CLI and ML extension. For more information, see [Install, set up, and use the CLI \(v2\)](#).
- You must have an Azure Resource group, in which you (or the service principal you use) need to have `contributor` access. You'll have such a resource group if you configured your ML extension per the above article.
- You must have an Azure Machine Learning workspace. You'll have such a workspace if you configured your ML extension per the above article.
- If you've not already set the defaults for Azure CLI, you should save your default settings. To avoid having to repeatedly pass in the values, run:

```
az account set --subscription <subscription id>
az configure --defaults workspace=<azureml workspace name> group=<resource group>
```

- An existing online endpoint and deployment. This article assumes that your deployment is as described in [Deploy and score a machine learning model with an online endpoint](#).
- If you haven't already set the environment variable `$ENDPOINT_NAME`, do so now:

```
export ENDPOINT_NAME=<YOUR_ENDPOINT_NAME>
```

- (Recommended) Clone the samples repository and switch to the repository's `cli/` directory:

```
git clone https://github.com/Azure/azureml-examples  
cd azureml-examples/cli
```

The commands in this tutorial are in the file `deploy-safe-rollout-online-endpoints.sh` and the YAML configuration files are in the `endpoints/online/managed/sample/` subdirectory.

Confirm your existing deployment is created

You can view the status of your existing endpoint and deployment by running:

```
az ml online-endpoint show --name $ENDPOINT_NAME  
  
az ml online-deployment show --name blue --endpoint $ENDPOINT_NAME
```

You should see the endpoint identified by `$ENDPOINT_NAME` and, a deployment called `blue`.

Scale your existing deployment to handle more traffic

In the deployment described in [Deploy and score a machine learning model with an online endpoint](#), you set the `instance_count` to the value `1` in the deployment yaml file. You can scale out using the `update` command:

```
az ml online-deployment update --name blue --endpoint $ENDPOINT_NAME --set instance_count=2
```

NOTE

Notice that in the above command we use `--set` to override the deployment configuration. Alternatively you can update the yaml file and pass it as an input to the `update` command using the `--file` input.

Deploy a new model, but send it no traffic yet

Create a new deployment named `green`:

```
az ml online-deployment create --name green --endpoint $ENDPOINT_NAME -f  
endpoints/online/managed/sample/green-deployment.yml
```

Since we haven't explicitly allocated any traffic to `green`, it will have zero traffic allocated to it. You can verify that using the command:

```
az ml online-endpoint show -n $ENDPOINT_NAME --query traffic
```

Test the new deployment

Though `green` has 0% of traffic allocated, you can invoke it directly by specifying the `--deployment` name:

```
az ml online-endpoint invoke --name $ENDPOINT_NAME --deployment green --request-file endpoints/online/model-  
2/sample-request.json
```

If you want to use a REST client to invoke the deployment directly without going through traffic rules, set the following HTTP header: `azureml-model-deployment: <deployment-name>`. The below code snippet uses `curl` to invoke the deployment directly. The code snippet should work in Unix/WSL environments:

```
# get the scoring uri
SCORING_URI=$(az ml online-endpoint show -n $ENDPOINT_NAME -o tsv --query scoring_uri)
# use curl to invoke the endpoint
curl --request POST "$SCORING_URI" --header "Authorization: Bearer $ENDPOINT_KEY" --header 'Content-Type: application/json' --header "azureml-model-deployment: green" --data @endpoints/online/model-2/sample-request.json
```

Test the deployment with mirrored traffic (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Once you've tested your `green` deployment, you can copy (or 'mirror') a percentage of the live traffic to it. Mirroring traffic doesn't change results returned to clients. Requests still flow 100% to the blue deployment. The mirrored percentage of the traffic is copied and submitted to the `green` deployment so you can gather metrics and logging without impacting your clients. Mirroring is useful when you want to validate a new deployment without impacting clients. For example, to check if latency is within acceptable bounds and that there are no HTTP errors.

WARNING

Mirroring traffic uses your [endpoint bandwidth quota](#) (default 5 MBPS). Your endpoint bandwidth will be throttled if you exceed the allocated quota. For information on monitoring bandwidth throttling, see [Monitor managed online endpoints](#).

The following command mirrors 10% of the traffic to the `green` deployment:

```
az ml online-endpoint update --name $ENDPOINT_NAME --mirror-traffic "green=10"
```

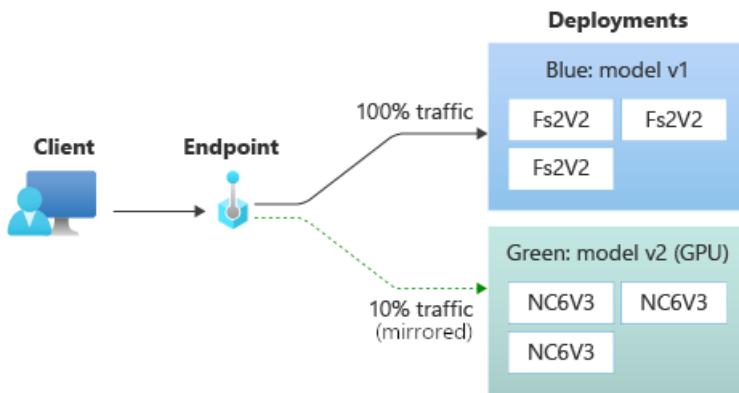
IMPORTANT

Mirroring has the following limitations:

- You can only mirror traffic to one deployment.
- Mirrored traffic is not currently supported with K8s.
- The maximum mirrored traffic you can configure is 50%. This limit is to reduce the impact on your endpoint bandwidth quota.

Also note the following behavior:

- A deployment can only be set to live or mirror traffic, not both.
- You can send traffic directly to the mirror deployment by specifying the deployment set for mirror traffic.
- You can send traffic directly to a live deployment by specifying the deployment set for live traffic, but in this case the traffic won't be mirrored to the mirror deployment. Mirror traffic is routed from traffic sent to endpoint without specifying the deployment.



After testing, you can set the mirror traffic to zero to disable mirroring:

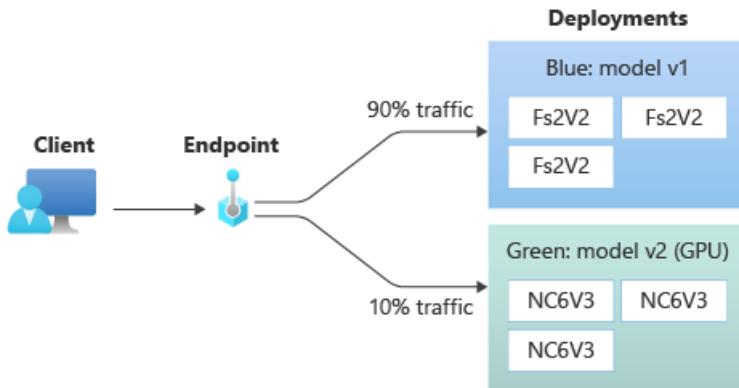
```
az ml online-endpoint update --name $ENDPOINT_NAME --mirror-traffic "green=0"
```

Test the new deployment with a small percentage of live traffic

Once you've tested your `green` deployment, allocate a small percentage of traffic to it:

```
az ml online-endpoint update --name $ENDPOINT_NAME --traffic "blue=90 green=10"
```

Now, your `green` deployment will receive 10% of requests.



Send all traffic to your new deployment

Once you're satisfied that your `green` deployment is fully satisfactory, switch all traffic to it.

```
az ml online-endpoint update --name $ENDPOINT_NAME --traffic "blue=0 green=100"
```

Remove the old deployment

```
az ml online-deployment delete --name blue --endpoint $ENDPOINT_NAME --yes --no-wait
```

Delete the endpoint and deployment

If you aren't going to use the deployment, you should delete it with:

```
az ml online-endpoint delete --name $ENDPOINT_NAME --yes --no-wait
```

Next steps

- [Deploy models with REST](#)
- [Create and use online endpoints in the studio](#)
- [Access Azure resources with a online endpoint and managed identity](#)
- [Monitor managed online endpoints](#)
- [Manage and increase quotas for resources with Azure Machine Learning](#)
- [View costs for an Azure Machine Learning managed online endpoint](#)
- [Managed online endpoints SKU list](#)
- [Troubleshooting online endpoints deployment and scoring](#)
- [Online endpoint YAML reference](#)

Safe rollout for managed online endpoints using Python SDK v2 (preview)

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO:  [Python SDK azure-ai-ml v2 \(preview\)](#)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you learn how to deploy a new version of the model without causing any disruption. With blue-green deployment or safe rollout, an approach in which a new version of a web service is introduced to production by rolling out the change to a small subset of users/requests before rolling it out completely. This article assumes you're using online endpoints; for more information, see [Azure Machine Learning endpoints](#).

In this article, you'll learn to:

- Deploy a new online endpoint called "blue" that serves version 1 of the model.
- Scale this deployment so that it can handle more requests.
- Deploy version 2 of the model to an endpoint called "green" that accepts no live traffic.
- Test the green deployment in isolation.
- Send 10% of live traffic to the green deployment.
- Fully cut-over all live traffic to the green deployment.
- Delete the now-unused v1 blue deployment.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- The [Azure Machine Learning SDK v2 for Python](#).
- You must have an Azure resource group, and you (or the service principal you use) must have Contributor access to it.
- You must have an Azure Machine Learning workspace.
- To deploy locally, you must install [Docker Engine](#) on your local computer. We highly recommend this option, so it's easier to debug issues.

Clone examples repository

To run the training examples, first clone the examples repository and change into the `sdk` directory:

```
git clone --depth 1 https://github.com/Azure/azureml-examples
cd azureml-examples/sdk
```

TIP

Use `--depth 1` to clone only the latest commit to the repository, which reduces time to complete the operation.

Connect to Azure Machine Learning workspace

The [workspace](#) is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. In this section, we'll connect to the workspace in which you'll perform deployment tasks.

1. Import the required libraries:

```
# import required libraries
from azure.ai.ml import MLClient
from azure.ai.ml.entities import (
    ManagedOnlineEndpoint,
    ManagedOnlineDeployment,
    Model,
    Environment,
    CodeConfiguration,
)
from azure.identity import DefaultAzureCredential
```

2. Configure workspace details and get a handle to the workspace:

To connect to a workspace, we need identifier parameters - a subscription, resource group and workspace name. We'll use these details in the `MLClient` from `azure.ai.ml` to get a handle to the required Azure Machine Learning workspace. This example uses the [default Azure authentication](#).

```
# enter details of your AzureML workspace
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
workspace = "<AZUREML_WORKSPACE_NAME>

# get a handle to the workspace
ml_client = MLClient(
    DefaultAzureCredential(), subscription_id, resource_group, workspace
)
```

Create online endpoint

Online endpoints are endpoints that are used for online (real-time) inferencing. Online endpoints contain deployments that are ready to receive data from clients and can send responses back in real time.

To create an online endpoint, we'll use `ManagedOnlineEndpoint`. This class allows user to configure the following key aspects:

- `name` - Name of the endpoint. Needs to be unique at the Azure region level
- `auth_mode` - The authentication method for the endpoint. Key-based authentication and Azure ML token-based authentication are supported. Key-based authentication doesn't expire but Azure ML token-based authentication does. Possible values are `key` or `aml_token`.
- `identity` - The managed identity configuration for accessing Azure resources for endpoint provisioning and inference.
 - `type` - The type of managed identity. Azure Machine Learning supports `system_assigned` or

- `user_assigned` identity.
- o `user_assigned_identities` - List (array) of fully qualified resource IDs of the user-assigned identities. This property is required if `identity.type` is `user_assigned`.
- `description` - Description of the endpoint.

1. Configure the endpoint:

```
# Creating a unique endpoint name with current datetime to avoid conflicts
import datetime

online_endpoint_name = "endpoint-" + datetime.datetime.now().strftime("%m%d%H%M%f")

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is a sample online endpoint",
    auth_mode="key",
    tags={"foo": "bar"},
)
```

2. Create the endpoint:

Using the `MLClient` created earlier, we'll now create the Endpoint in the workspace. This command will start the endpoint creation and return a confirmation response while the endpoint creation continues.

```
ml_client.begin_create_or_update(endpoint)
```

Create the 'blue' deployment

A deployment is a set of resources required for hosting the model that does the actual inferencing. We'll create a deployment for our endpoint using the `ManagedOnlineDeployment` class. This class allows user to configure the following key aspects.

Key aspects of deployment

- `name` - Name of the deployment.
- `endpoint_name` - Name of the endpoint to create the deployment under.
- `model` - The model to use for the deployment. This value can be either a reference to an existing versioned model in the workspace or an inline model specification.
- `environment` - The environment to use for the deployment. This value can be either a reference to an existing versioned environment in the workspace or an inline environment specification.
- `code_configuration` - the configuration for the source code and scoring script
 - o `path` - Path to the source code directory for scoring the model
 - o `scoring_script` - Relative path to the scoring file in the source code directory
- `instance_type` - The VM size to use for the deployment. For the list of supported sizes, see [Managed online endpoints SKU list](#).
- `instance_count` - The number of instances to use for the deployment

1. Configure blue deployment:

```

# create blue deployment
model = Model(path="..../model-1/model/sklearn_regression_model.pkl")
env = Environment(
    conda_file="..../model-1/environment/conda.yml",
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1",
)
blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    environment=env,
    code_configuration=CodeConfiguration(
        code="..../model-1/onlinescoring", scoring_script="score.py"
    ),
    instance_type="Standard_F2s_v2",
    instance_count=1,
)

```

2. Create the deployment:

Using the `MLClient` created earlier, we'll now create the deployment in the workspace. This command will start the deployment creation and return a confirmation response while the deployment creation continues.

```
ml_client.begin_create_or_update(blue_deployment)
```

```
# blue deployment takes 100 traffic
endpoint.traffic = {"blue": 100}
ml_client.begin_create_or_update(endpoint)
```

Test the endpoint with sample data

Using the `MLClient` created earlier, we'll get a handle to the endpoint. The endpoint can be invoked using the `invoke` command with the following parameters:

- `endpoint_name` - Name of the endpoint
- `request_file` - File with request data
- `deployment_name` - Name of the specific deployment to test in an endpoint

We'll send a sample request using a `json` file.

```
# test the blue deployment with some sample data
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    deployment_name="blue",
    request_file="..../model-1/sample-request.json",
)
```

Scale the deployment

Using the `MLClient` created earlier, we'll get a handle to the deployment. The deployment can be scaled by increasing or decreasing the `instance_count`.

```
# scale the deployment
blue_deployment = ml_client.online_deployments.get(
    name="blue", endpoint_name=online_endpoint_name
)
blue_deployment.instance_count = 2
ml_client.online_deployments.begin_create_or_update(blue_deployment)
```

Get endpoint details

```
# Get the details for online endpoint
endpoint = ml_client.online_endpoints.get(name=online_endpoint_name)

# existing traffic details
print(endpoint.traffic)

# Get the scoring URI
print(endpoint.scoring_uri)
```

Deploy a new model, but send no traffic yet

Create a new deployment named green:

```
# create green deployment
model2 = Model(path="../model-2/model/sklearn_regression_model.pkl")
env2 = Environment(
    conda_file="../model-2/environment/conda.yml",
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1",
)

green_deployment = ManagedOnlineDeployment(
    name="green",
    endpoint_name=online_endpoint_name,
    model=model2,
    environment=env2,
    code_configuration=CodeConfiguration(
        code="../model-2/onlinescoring", scoring_script="score.py"
    ),
    instance_type="Standard_F2s_v2",
    instance_count=1,
)
```

```
# use MLClient to create green deployment
ml_client.begin_create_or_update(green_deployment)
```

Test the new deployment

Though green has 0% of traffic allocated, you can still invoke the endpoint and deployment with [json](#) file.

```
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    deployment_name="green",
    request_file="../model-2/sample-request.json",
)
```

Test the deployment with mirrored traffic (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Once you've tested your **green** deployment, you can copy (or 'mirror') a percentage of the live traffic to it. Mirroring traffic doesn't change results returned to clients. Requests still flow 100% to the blue deployment. The mirrored percentage of the traffic is copied and submitted to the **green** deployment so you can gather metrics and logging without impacting your clients. Mirroring is useful when you want to validate a new deployment without impacting clients. For example, to check if latency is within acceptable bounds and that there are no HTTP errors.

WARNING

Mirroring traffic uses your [endpoint bandwidth quota](#) (default 5 MBPS). Your endpoint bandwidth will be throttled if you exceed the allocated quota. For information on monitoring bandwidth throttling, see [Monitor managed online endpoints](#).

The following command mirrors 10% of the traffic to the **green** deployment:

```
endpoint.mirror_traffic = {"green": 10}  
ml_client.begin_create_or_update(endpoint)
```

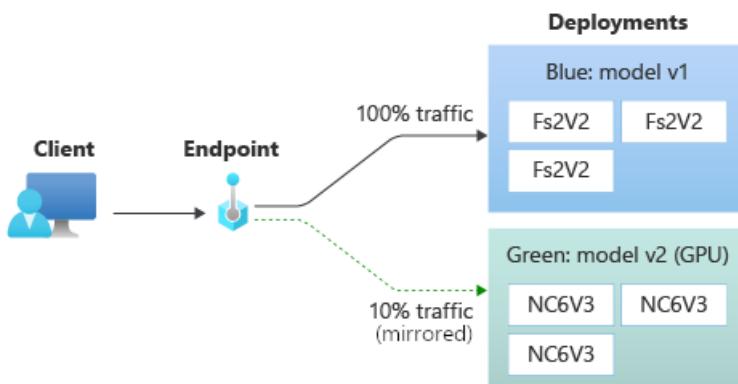
IMPORTANT

Mirroring has the following limitations:

- You can only mirror traffic to one deployment.
- Mirrored traffic is not currently supported with K8s.
- The maximum mirrored traffic you can configure is 50%. This limit is to reduce the impact on your endpoint bandwidth quota.

Also note the following behavior:

- A deployment can only be set to live or mirror traffic, not both.
- You can send traffic directly to the mirror deployment by specifying the deployment set for mirror traffic.
- You can send traffic directly to a live deployment by specifying the deployment set for live traffic, but in this case the traffic won't be mirrored to the mirror deployment. Mirror traffic is routed from traffic sent to endpoint without specifying the deployment.



After testing, you can set the mirror traffic to zero to disable mirroring:

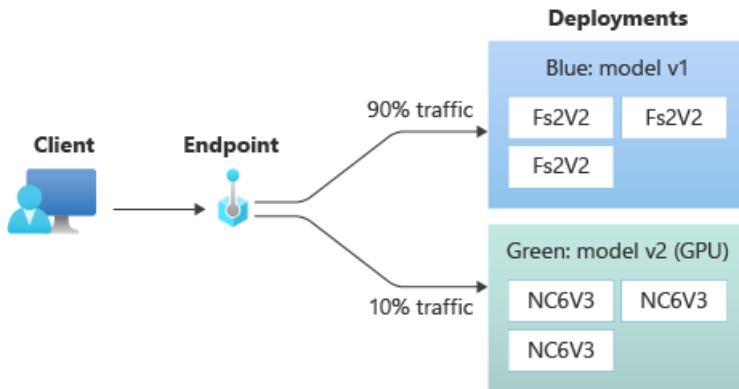
```
endpoint.mirror_traffic = {"green": 0}  
ml_client.begin_create_or_update(endpoint)
```

Test the new deployment with a small percentage of live traffic:

Once you've tested your green deployment, allocate a small percentage of traffic to it:

```
endpoint.traffic = {"blue": 90, "green": 10}  
ml_client.begin_create_or_update(endpoint)
```

Now, your green deployment will receive 10% of requests.



Send all traffic to your new deployment:

Once you're satisfied that your green deployment is fully satisfactory, switch all traffic to it.

```
endpoint.traffic = {"blue": 0, "green": 100}  
ml_client.begin_create_or_update(endpoint)
```

Remove the old deployment:

```
ml_client.online_deployments.delete(name="blue", endpoint_name=online_endpoint_name)
```

Delete endpoint

If you aren't going to use the deployment, you should delete it with:

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Next steps

- [Explore online endpoint samples](#)
- [Access Azure resources with a online endpoint and managed identity](#)
- [Monitor managed online endpoints](#)
- [Manage and increase quotas for resources with Azure Machine Learning](#)
- [View costs for an Azure Machine Learning managed online endpoint](#)
- [Managed online endpoints SKU list](#)

- Troubleshooting online endpoints deployment and scoring

Deploy MLflow models to online endpoints

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

In this article, learn how to deploy your [MLflow](#) model to an [online endpoint](#) for real-time inference. When you deploy your MLflow model to an online endpoint, it's a no-code-deployment so you don't have to provide a scoring script or an environment.

You only provide the typical MLflow model folder contents:

- MLmodel file
- `conda.yaml`
- model file(s)

For no-code-deployment, Azure Machine Learning

- Dynamically installs Python packages provided in the `conda.yaml` file, this means the dependencies are installed during container runtime.
 - The base container image/curated environment used for dynamic installation is
`mcr.microsoft.com/azureml/mlflow-ubuntu18.04-py37-cpu-inference` or
`AzureML-mlflow-ubuntu18.04-py37-cpu-inference`
- Provides a MLflow base image/curated environment that contains the following items:
 - `azureml-inference-server-http`
 - `mlflow-skinny`
 - `pandas`
 - The scoring script baked into the image.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- The [Azure CLI](#) and the `ml` extension to the Azure CLI. For more information, see [Install, set up, and use the CLI \(v2\)](#).

IMPORTANT

The CLI examples in this article assume that you are using the Bash (or compatible) shell. For example, from a Linux system or [Windows Subsystem for Linux](#).

- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Install, set up, and use the CLI \(v2\)](#) to create one.
- You must have a MLflow model. The examples in this article are based on the models from <https://github.com/Azure/azureml-examples/tree/main/cli/endpoints/online/mlflow>.
 - If you don't have an MLflow formatted model, you can [convert your custom ML model to MLflow format](#).

The information in this article is based on code samples contained in the [azureml-examples](#) repository. To run the commands locally without having to copy/paste YAML and other files, clone the repo and then change directories to the `cli` directory in the repo:

```
git clone https://github.com/Azure/azureml-examples --depth 1
cd azureml-examples
cd cli
```

If you haven't already set the defaults for the Azure CLI, save your default settings. To avoid passing in the values for your subscription, workspace, and resource group multiple times, use the following commands. Replace the following parameters with values for your specific configuration:

- Replace `<subscription>` with your Azure subscription ID.
- Replace `<workspace>` with your Azure Machine Learning workspace name.
- Replace `<resource-group>` with the Azure resource group that contains your workspace.
- Replace `<location>` with the Azure region that contains your workspace.

TIP

You can see what your current defaults are by using the `az configure -l` command.

```
az account set --subscription <subscription>
az configure --defaults workspace=<workspace> group=<resource-group> location=<location>
```

In this code snippet used in this article, the `<ENDPOINT_NAME>` environment variable contains the name of the endpoint to create and use. To set this, use the following command from the CLI. Replace `<YOUR_ENDPOINT_NAME>` with the name of your endpoint:

```
export ENDPOINT_NAME=<YOUR_ENDPOINT_NAME>"
```

Deploy using CLI (v2)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

This example shows how you can deploy an MLflow model to an online endpoint using CLI (v2).

IMPORTANT

For MLflow no-code-deployment, [testing via local endpoints](#) is currently not supported.

1. Create a YAML configuration file for your endpoint. The following example configures the name and authentication mode of the endpoint:

`create-endpoint.yaml`

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: my-endpoint
auth_mode: key
```

2. To create a new endpoint using the YAML configuration, use the following command:

```
az ml online-endpoint create --name $ENDPOINT_NAME -f endpoints/online/mlflow/create-endpoint.yaml
```

3. Create a YAML configuration file for the deployment. The following example configures a deployment of the `sklearn-diabetes` model to the endpoint created in the previous step:

IMPORTANT

For MLflow no-code-deployment (NCD) to work, setting `type` to `mlflow_model` is required, `type: mlflow_model`. For more information, see [CLI \(v2\) model YAML schema](#).

sklearn-deployment.yaml

```
$schema: https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: sklearn-deployment
endpoint_name: my-endpoint
model:
  name: mir-sample-sklearn-mlflow-model
  version: 1
  path: sklearn-diabetes/model
  type: mlflow_model
instance_type: Standard_DS2_v2
instance_count: 1
```

4. To create the deployment using the YAML configuration, use the following command:

```
az ml online-deployment create --name sklearn-deployment --endpoint $ENDPOINT_NAME -f
endpoints/online/mlflow/sklearn-deployment.yaml --all-traffic
```

Invoke the endpoint

Once your deployment completes, use the following command to make a scoring request to the deployed endpoint. The `sample-request-sklearn.json` file used in this command is located in the

`/cli/endpoints/online/mlflow` directory of the `azure-examples` repo:

```
az ml online-endpoint invoke --name $ENDPOINT_NAME --request-file endpoints/online/mlflow/sample-request-
sklearn.json
```

sample-request-sklearn.json

```
{"input_data": {  
    "columns": [  
        "age",  
        "sex",  
        "bmi",  
        "bp",  
        "s1",  
        "s2",  
        "s3",  
        "s4",  
        "s5",  
        "s6"  
    ],  
    "data": [  
        [ 1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0 ],  
        [ 10.0,2.0,9.0,8.0,7.0,6.0,5.0,4.0,3.0,2.0 ]  
    ],  
    "index": [0,1]  
}}
```

The response will be similar to the following text:

```
[  
 11633.100167144921,  
 8522.117402884991  
]
```

Delete endpoint

Once you're done with the endpoint, use the following command to delete it:

```
az ml online-endpoint delete --name $ENDPOINT_NAME --yes
```

Deploy using Azure Machine Learning studio

This example shows how you can deploy an MLflow model to an online endpoint using [Azure Machine Learning studio](#).

1. Models need to be registered in the Azure Machine Learning workspace to be deployed. Deployment of unregistered models is not supported. To create a model in Azure Machine Learning, open the Models page in Azure Machine Learning. Click **Register model** and select where your model is located. Fill out the required fields, and then select **Register**.

The screenshot shows the Microsoft Model List interface. On the left is a vertical toolbar with various icons. The main area displays a 'Model List' with one item named 'model-1'. A modal window titled 'Register a model' is open on the right, containing fields for Name, Description, Model framework, Framework name, Framework version, Model file or folder (with options for Upload file or Upload folder), Tags, and Add properties. At the bottom of the modal are 'Register' and 'Cancel' buttons.

Microsoft > larryml > Models

Model List

+ Register model Refresh

Search

Showing 1-1 of 1 models

Name

model-1

Name *

Description

Model framework *

Other

Framework name *

Framework version *

Model file or folder *

Upload file Upload folder

Browse

Tags

Name : Value

No tags

Add properties

Name : Value

No properties

Register Cancel

2. To create an endpoint deployment, use either the **endpoints** or **models** page :

- [Endpoints page](#)
- [Models page](#)

1. From the **Endpoints** page, Select **+Create**.

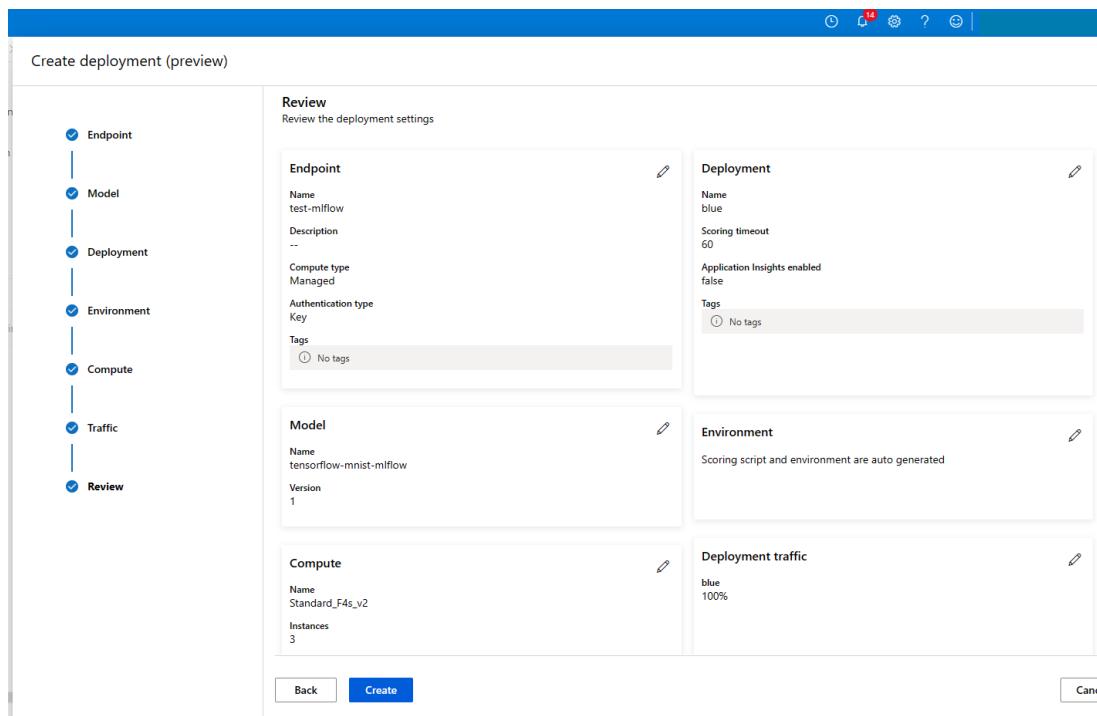
Microsoft Azure Machine Learning Studio

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The left sidebar contains various options like New, Home, Author, Notebooks, Automated ML, Designer, Assets, Data, Jobs, Components, Pipelines, Environments, Models, and Endpoints. The 'Endpoints' option is highlighted with a red box. The main content area shows the 'Endpoints' section with tabs for 'Real-time endpoints' and 'Batch endpoints'. A red box highlights the 'Create (preview)' button. Below it, there's a table showing two endpoints: 'triton-deployment-test' and 'custom-container-2 (deleting...)'. The table has columns for Name, Description, and a star icon.

2. Provide a name and authentication type for the endpoint, and then select **Next**.
3. When selecting a model, select the MLflow model registered previously. Select **Next** to continue.
4. When you select a model registered in MLflow format, in the Environment step of the wizard, you don't need a scoring script or an environment.

The screenshot shows the 'Create deployment (preview)' wizard. The left sidebar is identical to the previous screenshot. The main area shows the 'Create deployment (preview)' step. A red box highlights the 'Select environment' section, which includes a note: 'Select environment to use for your deployment' and 'For the selected model, the scoring script and environment are auto generated for you.' The 'Environment' checkbox is checked, while other options like 'Compute', 'Traffic', and 'Review' are unchecked.

5. Complete the wizard to deploy the model to the endpoint.



Deploy models after a training job

This section helps you understand how to deploy models to an online endpoint once you have completed your [training job](#). Models logged in a run are stored as artifacts. If you have used `mlflow.autolog()` in your training script, you will see model artifacts generated in the job's output. You can use `mlflow.autolog()` for several common ML frameworks to log model parameters, performance metrics, model artifacts, and even feature importance graphs.

For more information, see [Train models with CLI](#). Also see the [training job samples](#) in the GitHub repository.

1. Models need to be registered in the Azure Machine Learning workspace to be deployed. Deployment of unregistered models is not supported. You can register the model directly from the job's output using the Azure ML CLI (v2), the Azure ML SDK for Python (v2) or Azure Machine Learning studio.

TIP

To register the model, you will need to know the location where the model has been stored. If you are using `autolog` feature of MLflow, the path will depend on the type and framework of the model being used. We recommend to check the job's output to identify which is the name of this folder. You can look for the folder that contains a file named `MLModel`. If you are logging your models manually using `log_model`, then the path is the argument you pass to such method. As an example, if you log the model using `mlflow.sklearn.log_model(my_model, "classifier")`, then the path where the model is stored is `classifier`.

- [Azure Machine Learning studio](#)
- [Azure ML CLI \(v2\)](#)
- [Azure ML SDK for Python \(v2\)](#)

2. To deploy the registered model, you can use either studio or the Azure command-line interface. Use the model folder from the outputs for deployment:

- [Deploy using Azure Machine Learning studio.](#)
- [Deploy using Azure Machine Learning CLI \(v2\).](#)

Next steps

To learn more, review these articles:

- [Deploy models with REST](#)
- [Create and use online endpoints in the studio](#)
- [Safe rollout for online endpoints](#)
- [How to autoscale managed online endpoints](#)
- [Use batch endpoints for batch scoring](#)
- [View costs for an Azure Machine Learning managed online endpoint](#)
- [Access Azure resources with an online endpoint and managed identity](#)
- [Troubleshoot online endpoint deployment](#)

Deploy a TensorFlow model served with TF Serving using a custom container in an online endpoint

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Learn how to deploy a custom container as an online endpoint in Azure Machine Learning.

Custom container deployments can use web servers other than the default Python Flask server used by Azure Machine Learning. Users of these deployments can still take advantage of Azure Machine Learning's built-in monitoring, scaling, alerting, and authentication.

WARNING

Microsoft may not be able to help troubleshoot problems caused by a custom image. If you encounter problems, you may be asked to use the default image or one of the images Microsoft provides to see if the problem is specific to your image.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Quickstart: Create workspace resources](#) article to create one.
- The [Azure CLI](#) and the `ml` extension to the Azure CLI. For more information, see [Install, set up, and use the CLI \(v2\)](#).

IMPORTANT

The CLI examples in this article assume that you are using the Bash (or compatible) shell. For example, from a Linux system or [Windows Subsystem for Linux](#).

- To install the Python SDK v2 (preview), use the following command:

```
pip install --pre azure-ai-ml
```

For more information, see [Install the Python SDK v2 for Azure Machine Learning \(preview\)](#).

- You, or the service principal you use, must have [Contributor](#) access to the Azure Resource Group that contains your workspace. You'll have such a resource group if you configured your workspace using the quickstart article.
- To deploy locally, you must have [Docker engine](#) running locally. This step is **highly recommended**. It will help you debug issues.

Download source code

To follow along with this tutorial, download the source code below.

- [Azure CLI](#)
- [Python SDK](#)

```
git clone https://github.com/Azure/azureml-examples --depth 1
cd azureml-examples/cli
```

Initialize environment variables

Define environment variables:

```
BASE_PATH=endpoints/online/custom-container
AML_MODEL_NAME=tf-serving-mounted
MODEL_NAME=half_plus_two
MODEL_BASE_PATH=/var/azureml-app/azureml-models/$AML_MODEL_NAME/1
```

Download a TensorFlow model

Download and unzip a model that divides an input by two and adds 2 to the result:

```
wget https://aka.ms/half_plus_two-model -O $BASE_PATH/half_plus_two.tar.gz
tar -xvf $BASE_PATH/half_plus_two.tar.gz -C $BASE_PATH
```

Run a TF Serving image locally to test that it works

Use docker to run your image locally for testing:

```
docker run --rm -d -v $PWD/$BASE_PATH:$MODEL_BASE_PATH -p 8501:8501 \
-e MODEL_BASE_PATH=$MODEL_BASE_PATH -e MODEL_NAME=$MODEL_NAME \
--name="tf-serving-test" docker.io/tensorflow/serving:latest
sleep 10
```

Check that you can send liveness and scoring requests to the image

First, check that the container is "alive," meaning that the process inside the container is still running. You should get a 200 (OK) response.

```
curl -v http://localhost:8501/v1/models/$MODEL_NAME
```

Then, check that you can get predictions about unlabeled data:

```
curl --header "Content-Type: application/json" \  
--request POST \  
--data @$BASE_PATH/sample_request.json \  
http://localhost:8501/v1/models/$MODEL_NAME:predict
```

Stop the image

Now that you've tested locally, stop the image:

```
docker stop tf-serving-test
```

Deploy your online endpoint to Azure

Next, deploy your online endpoint to Azure.

- [Azure CLI](#)
- [Python SDK](#)

Create a YAML file for your endpoint and deployment

You can configure your cloud deployment using YAML. Take a look at the sample YAML for this example:

tf-serving-endpoint.yml

```
$schema: https://azurermldk2.blob.core.windows.net/latest/managedOnlineEndpoint.schema.json  
name: tf-serving-endpoint  
auth_mode: aml_token
```

tf-serving-deployment.yml

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineDeployment.schema.json  
name: tf-serving-deployment  
endpoint_name: tf-serving-endpoint  
model:  
  name: tf-serving-mounted  
  version: 1  
  path: ./half_plus_two  
environment_variables:  
  MODEL_BASE_PATH: /var/azureml-app/azureml-models/tf-serving-mounted/1  
  MODEL_NAME: half_plus_two  
environment:  
  #name: tf-serving  
  #version: 1  
  image: docker.io/tensorflow/serving:latest  
  inference_config:  
    liveness_route:  
      port: 8501  
      path: /v1/models/half_plus_two  
    readiness_route:  
      port: 8501  
      path: /v1/models/half_plus_two  
    scoring_route:  
      port: 8501  
      path: /v1/models/half_plus_two:predict  
instance_type: Standard_DS2_v2  
instance_count: 1
```

There are a few important concepts to notice in this YAML/Python parameter:

Readiness route vs. liveness route

An HTTP server defines paths for both *liveness* and *readiness*. A liveness route is used to check whether the server is running. A readiness route is used to check whether the server is ready to do work. In machine learning inference, a server could respond 200 OK to a liveness request before loading a model. The server could respond 200 OK to a readiness request only after the model has been loaded into memory.

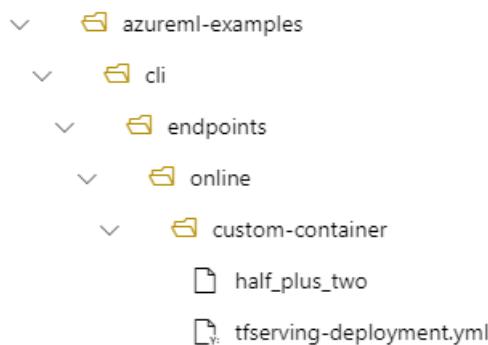
Review the [Kubernetes documentation](#) for more information about liveness and readiness probes.

Notice that this deployment uses the same path for both liveness and readiness, since TF Serving only defines a liveness route.

Locating the mounted model

When you deploy a model as an online endpoint, Azure Machine Learning *mounts* your model to your endpoint. Model mounting enables you to deploy new versions of the model without having to create a new Docker image. By default, a model registered with the name *foo* and version *1* would be located at the following path inside of your deployed container: `/var/azureml-app/azureml-models/foo/1`

For example, if you have a directory structure of `/azureml-examples/cli/endpoints/online/custom-container` on your local machine, where the model is named `half_plus_two`:

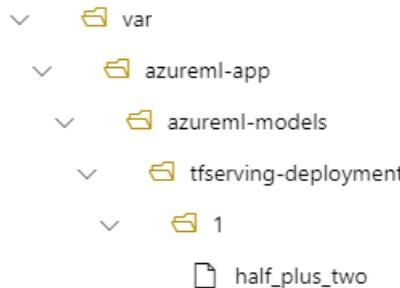


- [Azure CLI](#)
- [Python SDK](#)

and `tf-serving-deployment.yml` contains:

```
model:
  name: tf-serving-mounted
  version: 1
  path: ./half_plus_two
```

then your model will be located under `/var/azureml-app/azureml-models/tf-serving-deployment/1` in your deployment:



You can optionally configure your `model_mount_path`. It enables you to change the path where the model is mounted.

IMPORTANT

The `model_mount_path` must be a valid absolute path in Linux (the OS of the container image).

- [Azure CLI](#)
- [Python SDK](#)

For example, you can have `model_mount_path` parameter in your `tferving-deployment.yml`:

```
name: tferving-deployment
endpoint_name: tferving-endpoint
model:
  name: tferving-mounted
  version: 1
  path: ./half_plus_two
model_mount_path: /var/tferving-model-mount
....
```

then your model will be located at `/var/tferving-model-mount/tferving-deployment/1` in your deployment. Note that it is no longer under `azureml-app/azureml-models`, but under the mount path you specified:

```
└── var
    └── tferving-model-mount
        └── tferving-deployment
            └── 1
                └── half_plus_two
```

Create your endpoint and deployment

- [Azure CLI](#)
- [Python SDK](#)

Now that you've understood how the YAML was constructed, create your endpoint.

```
az ml online-endpoint create --name tferving-endpoint -f endpoints/online/custom-container/tfserving-endpoint.yml
```

Creating a deployment may take few minutes.

```
az ml online-deployment create --name tferving-deployment -f endpoints/online/custom-container/tfserving-deployment.yml --all-traffic
```

Invoke the endpoint

Once your deployment completes, see if you can make a scoring request to the deployed endpoint.

- [Azure CLI](#)
- [Python SDK](#)

```
RESPONSE=$(az ml online-endpoint invoke -n $ENDPOINT_NAME --request-file $BASE_PATH/sample_request.json)
```

Delete the endpoint

Now that you've successfully scored with your endpoint, you can delete it:

- [Azure CLI](#)
- [Python SDK](#)

```
az ml online-endpoint delete --name tf-serving-endpoint
```

Next steps

- [Safe rollout for online endpoints](#)
- [Troubleshooting online endpoints deployment](#)
- [Torch serve sample](#)

Create and use managed online endpoints in the studio

9/22/2022 • 5 minutes to read • [Edit Online](#)

Learn how to use the studio to create and manage your managed online endpoints in Azure Machine Learning. Use managed online endpoints to streamline production-scale deployments. For more information on managed online endpoints, see [What are endpoints](#).

In this article, you learn how to:

- Create a managed online endpoint
- View managed online endpoints
- Add a deployment to a managed online endpoint
- Update managed online endpoints
- Delete managed online endpoints and deployments

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create workspace resources](#).
- The examples repository - Clone the [AzureML Example repository](#). This article uses the assets in `/cli/endpoints/online`.

Create a managed online endpoint

Use the studio to create a managed online endpoint directly in your browser. When you create a managed online endpoint in the studio, you must define an initial deployment. You can't create an empty managed online endpoint.

1. Go to the [Azure Machine Learning studio](#).
2. In the left navigation bar, select the **Endpoints** page.
3. Select **+ Create**.

Name	Description	Created on	Created by	Updated on	Compute type
managed-endpoint-ex...		May 6, 2021 3:33 PM	User Name	May 6, 2021 3:33 PM	Managed
managed-endpoint		May 6, 2021 2:06 PM	User Name	May 6, 2021 3:28 PM	Managed
aks-endpoint		Apr 21, 2021 8:12 PM	User Name	Apr 27, 2021 9:51 AM	Kubernetes service
test-aks		Apr 27, 2021 7:54 AM	User Name	Apr 27, 2021 7:54 AM	Kubernetes service
test-aks-e		Apr 21, 2021 6:59 PM	User Name	Apr 21, 2021 6:59 PM	Kubernetes service

Create deployment (preview)

Create endpoint
An endpoint is used to deploy and score your models. [Learn more](#)

Endpoint name *

Description

Compute type [?](#)
 Managed

Authentication type
 Token Key

Endpoint tags [?](#)
 Name : Value [Add](#)
[No tags](#)

Endpoint
 Model
 Deployment
 Environment
 Compute
 Traffic
 Review

Register the model

A model registration is a logical entity in the workspace that may contain a single model file, or a directory containing multiple files. The steps in this article assume that you've registered the [model folder](#) that contains the model.

To register the example model using Azure Machine Learning studio, use the following steps:

1. Go to the [Azure Machine Learning studio](#).
2. In the left navigation bar, select the **Models** page.
3. Select **Register**, and then **From local files**.
4. Select **Unspecified type** for the **Model type**, then select **Browse**, and **Browse folder**.

Register model from local files

X

✖ Upload model

▼

Upload model

Upload the file or folder containing the model.

Model type * ⓘ

Unspecified type

▼

Select model file or folder *

Browse ▾

↑ Browse file

📁 Browse folder

Back

Next

Cancel

5. Select the `\azureml-examples\cli\endpoints\online\model-1\model` folder from the local copy of the repo you downloaded earlier. When prompted, select **Upload**. Once the upload completes, select **Next**.
6. Enter a friendly **Name** for the model. The steps in this article assume it's named `model-1`.
7. Select **Next**, and then **Register** to complete registration.

For more information on working with registered models, see [Register and work with models](#).

Follow the setup wizard to configure your managed online endpoint.

You can also create a managed online endpoint from the **Models** page in the studio. This is an easy way to add a model to an existing managed online deployment.

1. Go to the [Azure Machine Learning studio](#).
2. In the left navigation bar, select the **Models** page.
3. Select a model by checking the circle next to the model name.
4. Select **Deploy > Deploy to real-time endpoint**.

The screenshot shows the Azure Machine Learning studio interface. On the left, there is a navigation bar with icons for Home, Experiments, Datasets, Models, Metrics, Pipelines, and Compute. The 'Models' icon is highlighted. The main area is titled 'Model List'. At the top, there are buttons for Create, Refresh, Delete, Deploy (which is highlighted), Edit columns, and Reset view. A search bar is present. To the right of the search bar is a toggle switch for 'Show latest versions only'. Below the toolbar, a message says 'Success: Model 'model-1:1' created successfully.' A modal window is open over the list of models. It has a title 'Deploy to real-time endpoint' with the subtitle 'Deploy the model using the new real-time endpoint wizard'. The modal lists three options: 'Deploy to real-time endpoint', 'Deploy to batch endpoint', and 'Deploy to web service'. The first option is selected. The background shows a table with columns: Name, Job (Run ID), Created on, and Tags. There are three rows: 'model-1' (Run ID 1, Jun 17, 2022 9:19 AM), 'mnist-model' (Run ID 1, Jun 10, 2022 1:41 PM), and 'densenet-onnx-model' (Run ID 1, Jun 10, 2022 9:41 AM). Each row has a '...' button at the end.

- Enter an **Endpoint name** and select **Managed** as the compute type.
- Select **Next**, accepting defaults, until you're prompted for the environment. Here, select the following:
 - Select scoring file and dependencies:** Browse and select the `\azureml-examples\cli\endpoints\online\model-1\onlinescorer\score.py` file from the repo you downloaded earlier.
 - Choose an environment section:** Select the **Scikit-learn 0.24.1** curated environment.
- Select **Next**, accepting defaults, until you're prompted to create the deployment. Select the **Create** button.

View managed online endpoints

You can view your managed online endpoints in the **Endpoints** page. Use the endpoint details page to find critical information including the endpoint URI, status, testing tools, activity monitors, deployment logs, and sample consumption code:

- In the left navigation bar, select **Endpoints**.
- (Optional) Create a **Filter** on **Compute type** to show only **Managed** compute types.
- Select an endpoint name to view the endpoint detail page.

Attributes		Deployment summary	
Service ID	managed-endpoint-test	Deployments	
Description	this is my managed endpoint	blue	(70%)
Provisioning state	Succeeded	green	(30%)
Compute type	Managed		
Created by			
Created on	7/9/2021 1:17:41 PM		
Last updated on	7/9/2021 1:17:41 PM		
Authentication type	AMLToken		
Swagger URI	https://		
REST endpoint	https://		
Metrics	View metrics		

Deployment blue	
Name	blue
Traffic	70%
Scoring script	score.py
Provisioning state	Succeeded
SKU	Standard_F4s_v2
Instance count	3
Model ID	sklearn-model-new:1
Environment	AzureML-sklearn-0.24.1-ubuntu18.04-py37-cpu-inference:5
Logs	Deployment logs

Test

Use the **Test** tab in the endpoints details page to test your managed online deployment. Enter sample input and view the results.

- Select the **Test** tab in the endpoint's detail page.
- Use the dropdown to select the deployment you want to test.
- Enter sample input.
- Select **Test**.

Microsoft Azure Machine Learning

Home > Endpoints > managed-endpoint-example-1

managed-endpoint-example-1

Details Test Consume Monitoring Deployment logs

Deployment

test-blue

Input data to test real-time endpoint

Test

Test result

```
{"data": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]}
```

```
[5215.1981315798685, 3726.995485938578]
```

This screenshot shows the 'Test' tab for a managed endpoint named 'managed-endpoint-example-1'. The deployment selected is 'test-blue'. The input data is a JSON object with an array named 'data' containing two arrays of integers from 1 to 10. The resulting test result is two floating-point numbers: 5215.1981315798685 and 3726.995485938578.

Monitoring

Use the **Monitoring** tab to see high-level activity monitor graphs for your managed online endpoint.

To use the monitoring tab, you must select "Enable Application Insight diagnostic and data collection" when you create your endpoint.

Microsoft Azure Machine Learning

Home > Endpoints > managed-demo

managed-demo

Details Test Consume Monitoring Deployment logs

Time range Thu, May 6, 2021 1:50 PM - Fri, May 7, 2021 1:50 PM Auto refresh

Average Requests per minute 0

Average Request latency 10 ms

Requests per minute

Showing 3 out of 3 deployments

This screenshot shows the 'Monitoring' tab for a managed endpoint named 'managed-demo'. The time range is set from Thursday, May 6, 2021, 1:50 PM to Friday, May 7, 2021, 1:50 PM. The chart displays 'Requests per minute' with two distinct spikes reaching a value of 5 at approximately 15:00 and 18:00 on May 6, 2021. The baseline request rate is near zero throughout the rest of the period.

For more information on how viewing other monitors and alerts, see [How to monitor managed online endpoints](#).

Add a deployment to a managed online endpoint

You can add a deployment to your existing managed online endpoint.

From the **Endpoint details** page

1. Select **+ Add Deployment** button in the [endpoint details page](#).
2. Follow the instructions to complete the deployment.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a navigation bar with various icons. In the center, under 'Endpoints', it shows 'managed-endpoint-test'. Below this, there are tabs for 'Details' (which is selected), 'Test', 'Consume', 'Monitoring', and 'Deployment logs'. A red box highlights the '+ Add deployment' button. To the right, there are two main sections: 'Attributes' (listing Service ID, Description, Provisioning state, Compute type, Created by, and Created on) and 'Deployment summary' (showing Deployments: blue (70%) and green (30%)). Below these, there's a detailed view for 'Deployment blue' with fields for Name, Traffic, and Scoring script.

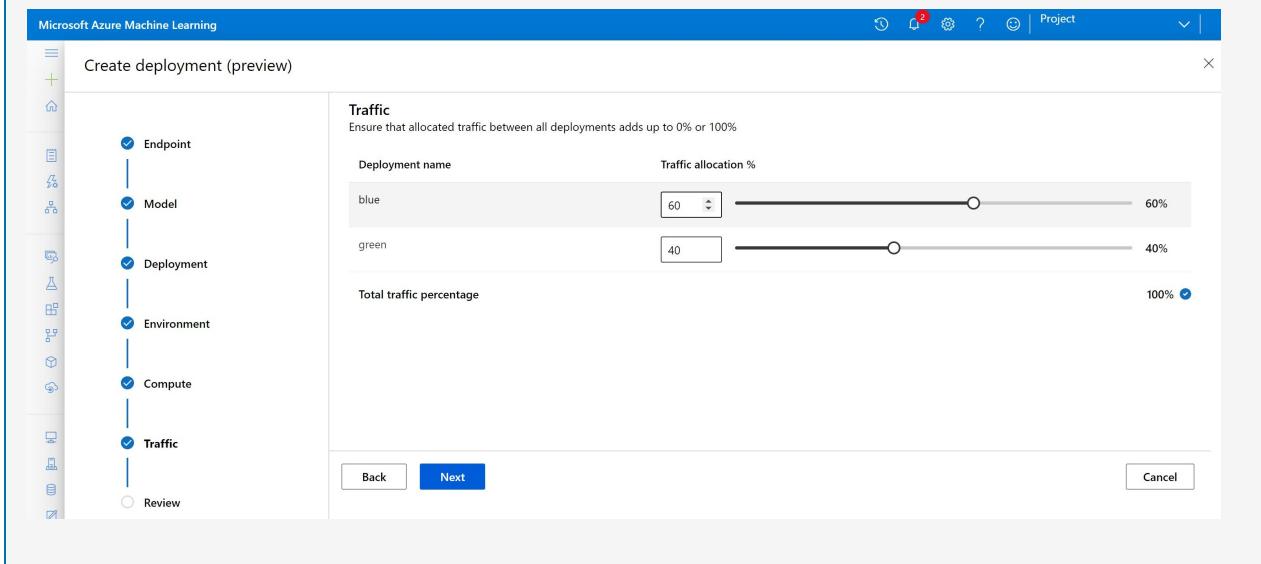
Alternatively, you can use the **Models** page to add a deployment:

1. In the left navigation bar, select the **Models** page.
2. Select a model by checking the circle next to the model name.
3. Select **Deploy > Deploy to real-time endpoint**.
4. Choose to deploy to an existing managed online endpoint.

The screenshot shows the 'Create deployment (preview)' page. On the left, there's a sidebar with options: Endpoint (selected), Model, Deployment, Environment, Compute, and Traffic. The main area is titled 'Select endpoint' with a sub-instruction: 'An endpoint is used to deploy and score your models. [Learn more](#)'. It shows two radio buttons: 'New' and 'Existing', with 'Existing' selected and highlighted with a red box. Below this, it says 'Showing 1-6 endpoints' and lists three endpoints: 'managed-endpoint-test2' (selected with a checkmark), 'managed-endpoint-test1', and 'managed-endpoint-vaidya'. At the bottom, there are 'Back', 'Next', and 'Cancel' buttons.

NOTE

You can adjust the traffic balance between deployments in an endpoint when adding a new deployment.



Update managed online endpoints

You can update deployment traffic percentage and instance count from Azure Machine Learning studio.

Update deployment traffic allocation

Use **deployment traffic allocation** to control the percentage of incoming requests going to each deployment in an endpoint.

1. In the endpoint details page, Select **Update traffic**.
2. Adjust your traffic and select **Update**.

TIP

The **Total traffic percentage** must sum to either 0% (to disable traffic) or 100% (to enable traffic).

Update deployment instance count

Use the following instructions to scale an individual deployment up or down by adjusting the number of instances:

1. In the endpoint details page. Find the card for the deployment you want to update.
2. Select the **edit icon** in the deployment detail card.
3. Update the instance count.
4. Select **Update**.

Delete managed online endpoints and deployments

Learn how to delete an entire managed online endpoint and its associated deployments. Or, delete an individual deployment from a managed online endpoint.

Delete a managed online endpoint

Deleting a managed online endpoint also deletes any deployments associated with it.

1. Go to the [Azure Machine Learning studio](#).
2. In the left navigation bar, select the **Endpoints** page.
3. Select an endpoint by checking the circle next to the model name.

4. Select **Delete**.

Alternatively, you can delete a managed online endpoint directly in the [endpoint details page](#).

Delete an individual deployment

Use the following steps to delete an individual deployment from a managed online endpoint. This does affect the other deployments in the managed online endpoint:

NOTE

You cannot delete a deployment that has allocated traffic. You must first [set traffic allocation](#) for the deployment to 0% before deleting it.

1. Go to the [Azure Machine Learning studio](#).
2. In the left navigation bar, select the **Endpoints** page.
3. Select your managed online endpoint.
4. In the endpoint details page, find the deployment you want to delete.
5. Select the **delete icon**.

Next steps

In this article, you learned how to use Azure Machine Learning managed online endpoints. See these next steps:

- [What are endpoints?](#)
- [How to deploy managed online endpoints with the Azure CLI](#)
- [Deploy models with REST](#)
- [How to monitor managed online endpoints](#)
- [Troubleshooting managed online endpoints deployment and scoring](#)
- [View costs for an Azure Machine Learning managed online endpoint](#)
- [Manage and increase quotas for resources with Azure Machine Learning](#)

High-performance serving with Triton Inference Server (Preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

Learn how to use [NVIDIA Triton Inference Server](#) in Azure Machine Learning with [Managed online endpoints](#).

Triton is multi-framework, open-source software that is optimized for inference. It supports popular machine learning frameworks like TensorFlow, ONNX Runtime, PyTorch, NVIDIA TensorRT, and more. It can be used for your CPU or GPU workloads.

In this article, you will learn how to deploy Triton and a model to a managed online endpoint. Information is provided on using both the CLI (command line) and Azure Machine Learning studio.

NOTE

- [NVIDIA Triton Inference Server](#) is an open-source third-party software that is integrated in Azure Machine Learning.
- While Azure Machine Learning online endpoints are generally available, *using Triton with an online endpoint deployment is still in preview*.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- The [Azure CLI](#) and the `ml` extension to the Azure CLI. For more information, see [Install, set up, and use the CLI \(v2\)](#).

IMPORTANT

The CLI examples in this article assume that you are using the Bash (or compatible) shell. For example, from a Linux system or [Windows Subsystem for Linux](#).

- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Install, set up, and use the CLI \(v2\)](#) to create one.
- A working Python 3.8 (or higher) environment.
- Access to NCv3-series VMs for your Azure subscription.

IMPORTANT

You may need to request a quota increase for your subscription before you can use this series of VMs. For more information, see [NCv3-series](#).

The information in this article is based on code samples contained in the [azureml-examples](#) repository. To run the commands locally without having to copy/paste YAML and other files, clone the repo and then change

directories to the `cli` directory in the repo:

```
git clone https://github.com/Azure/azureml-examples --depth 1
cd azureml-examples
cd cli
```

If you haven't already set the defaults for the Azure CLI, save your default settings. To avoid passing in the values for your subscription, workspace, and resource group multiple times, use the following commands. Replace the following parameters with values for your specific configuration:

- Replace `<subscription>` with your Azure subscription ID.
- Replace `<workspace>` with your Azure Machine Learning workspace name.
- Replace `<resource-group>` with the Azure resource group that contains your workspace.
- Replace `<location>` with the Azure region that contains your workspace.

TIP

You can see what your current defaults are by using the `az configure -l` command.

```
az account set --subscription <subscription>
az configure --defaults workspace=<workspace> group=<resource-group> location=<location>
```

NVIDIA Triton Inference Server requires a specific model repository structure, where there is a directory for each model and subdirectories for the model version. The contents of each model version subdirectory is determined by the type of the model and the requirements of the backend that supports the model. To see all the model repository structure https://github.com/triton-inference-server/server/blob/main/docs/user_guide/model_repository.md#model-files

The information in this document is based on using a model stored in ONNX format, so the directory structure of the model repository is `<model-repository>/<model-name>/1/model.onnx`. Specifically, this model performs image identification.

Deploy using CLI (v2)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

This section shows how you can deploy Triton to managed online endpoint using the Azure CLI with the Machine Learning extension (v2).

IMPORTANT

For Triton no-code-deployment, [testing via local endpoints](#) is currently not supported.

1. To avoid typing in a path for multiple commands, use the following command to set a `BASE_PATH` environment variable. This variable points to the directory where the model and associated YAML configuration files are located:

```
BASE_PATH=endpoints/online/triton/single-model
```

2. Use the following command to set the name of the endpoint that will be created. In this example, a random name is created for the endpoint:

```
export ENDPOINT_NAME=triton-single-endpt-`echo $RANDOM`
```

3. Install Python requirements using the following commands:

```
pip install numpy
pip install tritonclient[http]
pip install pillow
pip install gevent
```

4. Create a YAML configuration file for your endpoint. The following example configures the name and authentication mode of the endpoint. The one used in the following commands is located at

`/cli/endpoints/online/triton/single-model/create-managed-endpoint.yml` in the `azureml-examples` repo you cloned earlier:

`create-managed-endpoint.yaml`

```
$schema: https://azuremlschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: my-endpoint
auth_mode: aml_token
```

5. To create a new endpoint using the YAML configuration, use the following command:

```
az ml online-endpoint create -n $ENDPOINT_NAME -f $BASE_PATH/create-managed-endpoint.yaml
```

6. Create a YAML configuration file for the deployment. The following example configures a deployment named **blue** to the endpoint created in the previous step. The one used in the following commands is located at `/cli/endpoints/online/triton/single-model/create-managed-deployment.yml` in the `azureml-examples` repo you cloned earlier:

IMPORTANT

For Triton no-code-deployment (NCD) to work, setting `type` to `triton_model` is required, `type: triton_model`. For more information, see [CLI \(v2\) model YAML schema](#).

This deployment uses a Standard_NC6s_v3 VM. You may need to request a quota increase for your subscription before you can use this VM. For more information, see [NCv3-series](#).

```
$schema: https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: blue
endpoint_name: my-endpoint
model:
  name: sample-densenet-onnx-model
  version: 1
  path: ./models
  type: triton_model
  instance_count: 1
  instance_type: Standard_NC6s_v3
```

7. To create the deployment using the YAML configuration, use the following command:

```
az ml online-deployment create --name blue --endpoint $ENDPOINT_NAME -f $BASE_PATH/create-managed-deployment.yaml --all-traffic
```

Invoke your endpoint

Once your deployment completes, use the following command to make a scoring request to the deployed endpoint.

TIP

The file `/cli/endpoints/online/triton/single-model/triton_densenet_scoring.py` in the `azureml-examples` repo is used for scoring. The image passed to the endpoint needs pre-processing to meet the size, type, and format requirements, and post-processing to show the predicted label. The `triton_densenet_scoring.py` uses the `tritonclient.http` library to communicate with the Triton inference server.

1. To get the endpoint scoring uri, use the following command:

```
scoring_uri=$(az ml online-endpoint show -n $ENDPOINT_NAME --query scoring_uri -o tsv)
scoring_uri=${scoring_uri%/*}
```

2. To get an authentication token, use the following command:

```
auth_token=$(az ml online-endpoint get-credentials -n $ENDPOINT_NAME --query accessToken -o tsv)
```

3. To score data with the endpoint, use the following command. It submits the image of a peacock (<https://aka.ms/peacock-pic>) to the endpoint:

```
python $BASE_PATH/triton_densenet_scoring.py --base_url=$scoring_uri --token=$auth_token --image_path
$BASE_PATH/data/peacock.jpg
```

The response from the script is similar to the following text:

```
Is server ready - True
Is model ready - True
/azureml-examples/cli/endpoints/online/triton/single-model/densenet_labels.txt
84 : PEACOCK
```

Delete your endpoint and model

Once you're done with the endpoint, use the following command to delete it:

```
az ml online-endpoint delete -n $ENDPOINT_NAME --yes
```

Use the following command to delete your model:

```
az ml model delete --name $MODEL_NAME --version $MODEL_VERSION
```

Deploy using Azure Machine Learning studio

This section shows how you can deploy Triton to managed online endpoint using [Azure Machine Learning studio](#).

1. Register your model in Triton format using the following YAML and CLI command. The YAML uses a densenet-onnx model from <https://github.com/Azure/azureml-examples/tree/main/cli/endpoints/online/triton/single-model>

create-triton-model.yaml

```
name: densenet-onnx-model
version: 1
path: ./models
type: triton_model
description: Registering my Triton format model.
```

```
az ml model create -f create-triton-model.yaml
```

The following screenshot shows how your registered model will look on the **Models** page of Azure Machine Learning studio.

The screenshot shows the Azure Machine Learning studio interface. On the left, there is a navigation sidebar with various options like Microsoft, New, Home, Author, Notebooks, Automated ML, Designer, Assets, Data, Jobs, Components, Pipelines, Environments, Models (which is highlighted with a red box), Endpoints, Manage, Compute, and Datastores. The main content area shows the 'Models' page. At the top, there is a breadcrumb navigation: Microsoft > mymlworkspace > Models > densenet-onnx-model:1. Below the breadcrumb, the model name 'densenet-onnx-model:1' is displayed with a star icon. There are tabs for Details, Versions, Artifacts, Endpoints, Data, Explanations (preview), Fairness (preview), and Responsible AI (preview). A 'Refresh' button, a 'Deploy' dropdown, and a 'Create Responsible AI dashboard (preview)' button are also present. A yellow callout box contains the text: 'Responsible AI dashboard currently only supports MLflow format models with scikit-learn flavor. Learn more about model conversion to MLflow.' In the 'Attributes' section, the model's Name is listed as 'densenet-onnx-model', Version as '1', Created on as 'Jun 10, 2022 9:41 AM', Created by as 'me', Type as 'TRITON' (which is highlighted with a red box), and Job (Run ID) as '---'. In the 'Tags' section, it says 'No tags'. In the 'Description' section, the text 'Registering my Triton format model.' is shown.

- From **studio**, select your workspace and then use either the **endpoints** or **models** page to create the endpoint deployment:

- [Endpoints page](#)
- [Models page](#)

- From the **Endpoints** page, select **Create**.

Microsoft > mymlworkspace > Endpoints

Endpoints

Real-time endpoints Batch endpoints



Azure Machine Learning endpoints allow you to deploy machine learning models

Real-time endpoints are endpoints that are used for real-time inferencing. Real-time endpoints contain deployments that are ready to receive data from clients and can send responses back in real time. [Learn more](#)

[Create](#)

[View Azure Machine Learning tutorials](#)
[Create and use managed endpoints in the studio](#)

2. Provide a name and authentication type for the endpoint, and then select **Next**.
3. When selecting a model, select the Triton model registered previously. Select **Next** to continue.
4. When you select a model registered in Triton format, in the Environment step of the wizard, you don't need scoring script and environment.

Create deployment

Endpoint

Model

Deployment

Environment

Compute

Traffic

Review

Environment
Select environment to use for your deployment

For the selected model, the scoring script and environment are auto generated for you.

[Back](#) [Next](#) [Cancel](#)

5. Complete the wizard to deploy the model to the endpoint.

Create deployment

X

The screenshot shows the 'Create deployment' interface in the Azure Machine Learning studio. On the left, a vertical navigation bar lists the steps: Endpoint, Model, Deployment, Environment, Compute, Traffic, and Review. The 'Review' step is currently selected. The main area is titled 'Review' and contains a sub-section 'Review the deployment settings'. Below this, there are five configuration cards:

- Endpoint**: Name: triton-deployment, Description: --, Compute type: Managed, Authentication type: Key, Tags: No tags.
- Deployment**: Name: default, Scoring timeout: 60, Application Insights enabled: false, Tags: No tags.
- Model**: Name: densenet-onnx-model, Version: 1. This card is highlighted with a red border.
- Environment**: Scoring script and environment are auto generated.
- Deployment traffic**: default, 100%.
- Compute**: Name: Standard_F4s_v2, Instances: 3.

At the bottom are three buttons: 'Back', 'Create' (highlighted in blue), and 'Cancel'.

Next steps

To learn more, review these articles:

- [Deploy models with REST](#)
- [Create and use managed online endpoints in the studio](#)
- [Safe rollout for online endpoints](#)
- [How to autoscale managed online endpoints](#)
- [View costs for an Azure Machine Learning managed online endpoint](#)
- [Access Azure resources with a managed online endpoint and managed identity](#)
- [Troubleshoot managed online endpoints deployment](#)

Deploy models with REST

9/22/2022 • 5 minutes to read • [Edit Online](#)

Learn how to use the Azure Machine Learning REST API to deploy models.

The REST API uses standard HTTP verbs to create, retrieve, update, and delete resources. The REST API works with any language or tool that can make HTTP requests. REST's straightforward structure makes it a good choice in scripting environments and for MLOps automation.

In this article, you learn how to use the new REST APIs to:

- Create machine learning assets
- Create a basic training job
- Create a hyperparameter tuning sweep job

Prerequisites

- An **Azure subscription** for which you have administrative rights. If you don't have such a subscription, try the [free or paid personal subscription](#).
- An [Azure Machine Learning workspace](#).
- A service principal in your workspace. Administrative REST requests use [service principal authentication](#).
- A service principal authentication token. Follow the steps in [Retrieve a service principal authentication token](#) to retrieve this token.
- The **curl** utility. The **curl** program is available in the [Windows Subsystem for Linux](#) or any UNIX distribution. In PowerShell, **curl** is an alias for **Invoke-WebRequest** and `curl -d "key=val" -X POST uri` becomes `Invoke-WebRequest -Body "key=val" -Method POST -Uri uri`.

Set endpoint name

NOTE

Endpoint names need to be unique at the Azure region level. For example, there can be only one endpoint with the name `my-endpoint` in `westus2`.

```
export ENDPOINT_NAME=endpt-rest-`echo $RANDOM`
```

Azure Machine Learning online endpoints

Online endpoints allow you to deploy your model without having to create and manage the underlying infrastructure as well as Kubernetes clusters. In this article, you'll create an online endpoint and deployment, and validate it by invoking it. But first you'll have to register the assets needed for deployment, including model, code, and environment.

There are many ways to create an Azure Machine Learning online endpoints [including the Azure CLI](#), and visually with [the studio](#). The following example an online endpoint with the REST API.

Create machine learning assets

First, set up your Azure Machine Learning assets to configure your job.

In the following REST API calls, we use `SUBSCRIPTION_ID`, `RESOURCE_GROUP`, `LOCATION`, and `WORKSPACE` as placeholders. Replace the placeholders with your own values.

Administrative REST requests a [service principal authentication token](#). Replace `TOKEN` with your own value. You can retrieve this token with the following command:

```
response=$(curl -H "Content-Length: 0" --location --request POST  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Micros  
oft.MachineLearningServices/workspaces/$WORKSPACE/onlineEndpoints/$ENDPOINT_NAME/token?api-  
version=$API_VERSION" \  
--header "Authorization: Bearer $TOKEN")  
accessToken=$(echo $response | jq -r '.accessToken')
```

The service provider uses the `api-version` argument to ensure compatibility. The `api-version` argument varies from service to service. Set the API version as a variable to accommodate future versions:

```
API_VERSION="2022-05-01"
```

Get storage account details

To register the model and code, first they need to be uploaded to a storage account. The details of the storage account are available in the data store. In this example, you get the default datastore and Azure Storage account for your workspace. Query your workspace with a GET request to get a JSON file with the information.

You can use the tool `jq` to parse the JSON result and get the required values. You can also use the Azure portal to find the same information:

```
# Get values for storage account  
response=$(curl --location --request GET  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Micros  
oft.MachineLearningServices/workspaces/$WORKSPACE/datastores?api-version=$API_VERSION&isDefault=true" \  
--header "Authorization: Bearer $TOKEN")  
AZUREML_DEFAULT_DATASTORE=$(echo $response | jq -r '.value[0].name')  
AZUREML_DEFAULT_CONTAINER=$(echo $response | jq -r '.value[0].properties.containerName')  
export AZURE_STORAGE_ACCOUNT=$(echo $response | jq -r '.value[0].properties.accountName')
```

Upload & register code

Now that you have the datastore, you can upload the scoring script. Use the Azure Storage CLI to upload a blob into your default container:

```
az storage blob upload-batch -d $AZUREML_DEFAULT_CONTAINER/score -s endpoints/online/model-1/onlinescorer
```

TIP

You can also use other methods to upload, such as the Azure portal or [Azure Storage Explorer](#).

Once you upload your code, you can specify your code with a PUT request and refer to the datastore with

`datastoreId`:

```

curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/codes/score-sklearn/versions/1?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\": {
    \"codeUri\": \"https://$AZURE_STORAGE_ACCOUNT.blob.core.windows.net/$AZUREML_DEFAULT_CONTAINER/score\"
  }
}"

```

Upload and register model

Similar to the code, Upload the model files:

```
az storage blob upload-batch -d $AZUREML_DEFAULT_CONTAINER/model -s endpoints/online/model-1/model
```

Now, register the model:

```

curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/models/sklearn/versions/1?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\": {
    \"modelUri\": \"$azureml://subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/workspaces/$WORKSPACE/datastores/$AZUREML_DEFAULT_DATASTORE/paths/model\"
  }
}"

```

Create environment

The deployment needs to run in an environment that has the required dependencies. Create the environment with a PUT request. Use a docker image from Microsoft Container Registry. You can configure the docker image with `Docker` and add conda dependencies with `condaFile`.

In the following snippet, the contents of a Conda environment (YAML file) has been read into an environment variable:

```

ENV_VERSION=$RANDOM
curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/environments/sklearn-env/versions/$ENV_VERSION?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\":{
    \"condaFile\": \"$CONDA_FILE\",
    \"image\": \"mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1\"
  }
}"

```

Create endpoint

Create the online endpoint:

```

response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/onlineEndpoints/$ENDPOINT_NAME?api-version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN" \
--data-raw "{
    \"identity\": {
        \"type\": \"systemAssigned\",
    },
    \"properties\": {
        \"authMode\": \"AMLToken\",
    },
    \"location\": \"$LOCATION\"
}")

```

Create deployment

Create a deployment under the endpoint:

```

response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/onlineEndpoints/$ENDPOINT_NAME/deployments/blue?api-
version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN" \
--data-raw "{
    \"location\": \"$LOCATION\",
    \"sku\": {
        \"capacity\": 1,
        \"name\": \"Standard_DS2_v2\"
    },
    \"properties\": {
        \"endpointComputeType\": \"Managed\",
        \"scaleSettings\": {
            \"scaleType\": \"Default\"
        },
        \"model\": {
            \"subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/
workspaces/$WORKSPACE/models/sklearn/versions/1\",
                \"codeConfiguration\": {
                    \"codeId\": \"\",
                    \"environmentId\": \"\",
                    \"scoringScript\": \"score.py\"
                }
            }
        }
    }
}")

```

Invoke the endpoint to score data with your model

We need the scoring uri and access token to invoke the endpoint. First get the scoring uri:

```

response=$(curl --location --request GET
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/onlineEndpoints/$ENDPOINT_NAME?api-version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN")

scoringUri=$(echo $response | jq -r '.properties' | jq -r '.scoringUri')

```

Get the endpoint access token:

```
response=$(curl -H "Content-Length: 0" --location --request POST  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/onlineEndpoints/$ENDPOINT_NAME/token?api-version=$API_VERSION" \  
--header "Authorization: Bearer $TOKEN")  
accessToken=$(echo $response | jq -r '.accessToken')
```

Now, invoke the endpoint using curl:

```
curl --location --request POST $scoringUri \  
--header "Authorization: Bearer $accessToken" \  
--header "Content-Type: application/json" \  
--data-raw @endpoints/online/model-1/sample-request.json
```

Check the logs

Check the deployment logs:

```
curl --location --request POST  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/onlineEndpoints/$ENDPOINT_NAME/deployments/blue/getLogs?api-version=$API_VERSION" \  
--header "Authorization: Bearer $TOKEN" \  
--header "Content-Type: application/json" \  
--data-raw "{ \"tail\": 100 }"
```

Delete the endpoint

If you aren't going use the deployment, you should delete it with the below command (it deletes the endpoint and all the underlying deployments):

```
curl --location --request DELETE  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/onlineEndpoints/$ENDPOINT_NAME?api-version=$API_VERSION" \  
--header "Content-Type: application/json" \  
--header "Authorization: Bearer $TOKEN" || true
```

Next steps

- Learn how to deploy your model using the [Azure CLI](#).
- Learn how to deploy your model using [studio](#).
- Learn to [Troubleshoot online endpoints deployment and scoring](#)
- Learn how to [Access Azure resources with a online endpoint and managed identity](#)
- Learn how to [monitor online endpoints](#).
- Learn [safe rollout for online endpoints](#).
- [View costs for an Azure Machine Learning managed online endpoint](#).
- [Managed online endpoints SKU list](#).
- Learn about limits on managed online endpoints in [Manage and increase quotas for resources with Azure Machine Learning](#).

How to deploy an AutoML model to an online endpoint

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you'll learn how to deploy an AutoML-trained machine learning model to an online (real-time inference) endpoint. Automated machine learning, also referred to as automated ML or AutoML, is the process of automating the time-consuming, iterative tasks of developing a machine learning model. For more, see [What is automated machine learning \(AutoML\)?](#).

In this article you'll know how to deploy AutoML trained machine learning model to online endpoints using:

- Azure Machine Learning studio
- Azure Machine Learning CLI v2
- Azure Machine Learning Python SDK v2

Prerequisites

An AutoML-trained machine learning model. For more, see [Tutorial: Train a classification model with no-code AutoML in the Azure Machine Learning studio](#) or [Tutorial: Forecast demand with automated machine learning](#).

Deploy from Azure Machine Learning studio and no code

Deploying an AutoML-trained model from the Automated ML page is a no-code experience. That is, you don't need to prepare a scoring script and environment, both are auto generated.

1. Go to the Automated ML page in the studio
2. Select your experiment and run
3. Choose the Models tab
4. Select the model you want to deploy
5. Once you select a model, the Deploy button will light up with a drop-down menu
6. Select *Deploy to real-time endpoint* option

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The left sidebar has a tree view with 'Automated ML' selected, indicated by a yellow circle labeled 1. The top navigation bar shows the project path: Microsoft > sdg-ws > Automated ML > automl > joyful_boniato_1ptr7zx. The main content area displays the 'joyful_boniato_1ptr7zx' project details. The 'Models' tab is active, indicated by a yellow circle labeled 3. Below it, the 'Deploy' button is highlighted with a red box and a yellow circle labeled 5. A dropdown menu for 'Deploy' is open, showing 'Deploy to real-time endpoint' and 'Deploy to web service'. The 'Deploy to real-time endpoint' option is highlighted with a red box and a yellow circle labeled 6. The table below lists four models:

Algorithm name	Explained	Normalized ro...	Sampling	Created on
TruncatedSVDWrapper, LassoLars	0.17391	100.00 %		May 13, 2022 11:19 AM
RobustScaler, LassoLars	0.17400	100.00 %		May 13, 2022 12:07 PM
TruncatedSVDWrapper, LassoLars	0.17406	100.00 %		May 13, 2022 11:19 AM
RobustScaler, ElasticNet	0.17407	100.00 %		May 13, 2022 11:19 AM
StandardScalerWrapper, LassoLars	0.17409	100.00 %		May 13, 2022 11:19 AM

The system will generate the Model and Environment needed for the deployment.

The screenshot shows the 'Create deployment (preview)' wizard. The left sidebar shows a navigation tree with 'Endpoint', 'Model' (selected, indicated by a yellow circle labeled 4), 'Deployment', 'Environment', 'Compute', 'Traffic', and 'Review'. The main panel is titled 'Select model' and contains the following table:

Name	Version	Framework
AutoMLBacBadd3925	1	AutoML

At the bottom of the panel are 'Back', 'Next', and 'Cancel' buttons.

The screenshot shows the 'Create deployment (preview)' wizard. The left sidebar shows a navigation tree with 'Endpoint', 'Model', 'Deployment' (selected, indicated by a yellow circle labeled 5), 'Environment', 'Compute', 'Traffic', and 'Review'. The main panel is titled 'Select environment' and contains the following message:

Select environment
Select environment to use for your deployment.
For the selected model, the scoring script and environment are auto-generated for you.

At the bottom of the panel are 'Back', 'Next', and 'Cancel' buttons.

7. Complete the wizard to deploy the model to an online endpoint

Deploy manually from the studio or command line

If you wish to have more control over the deployment, you can download the training artifacts and deploy them.

To download the components you'll need for deployment:

1. Go to your Automated ML experiment and run in your machine learning workspace
2. Choose the Models tab
3. Select the model you wish to use. Once you select a model, the *Download* button will become enabled
4. Choose *Download*

Algorithm name	Explained	Normalized ro...	Sampling	Size
VotingEnsemble	View explanation	0.01142	100.00 %	A
MinMaxScaler, RandomForest		0.01696	100.00 %	A
MinMaxScaler, ExtremeRandomTrees		0.01737	100.00 %	A
StandardScalerWrapper, ElasticNet		0.02054	100.00 %	A
MaxAbsScaler, ElasticNet		0.02054	100.00 %	A
StandardScalerWrapper, ElasticNet		0.02054	100.00 %	A
MinMaxScaler, ElasticNet		0.02054	100.00 %	A

You'll receive a zip file containing:

- A conda environment specification file named `conda_env_<VERSION>.yml`

- A Python scoring file named `scoring_file_<VERSION>.py`
- The model itself, in a Python `.pkl` file named `model.pkl`

To deploy using these files, you can use either the studio or the Azure CLI.

- [Studio](#)
- [Azure CLI](#)
- [Python SDK](#)

1. Go to the Models page in Azure Machine Learning studio
2. Select + Register Model option
3. Register the model you downloaded from Automated ML run
4. Go to Environments page, select Custom environment, and select + Create option to create an environment for your deployment. Use the downloaded conda yaml to create a custom environment
5. Select the model, and from the Deploy drop-down option, select Deploy to real-time endpoint
6. Complete all the steps in wizard to create an online endpoint and deployment

Next steps

- [Troubleshooting online endpoints deployment](#)
- [Safe rollout for online endpoints](#)

Key and token-based authentication for online endpoints

9/22/2022 • 2 minutes to read • [Edit Online](#)

When consuming an online endpoint from a client, you can use either a *key* or a *token*. Keys don't expire, tokens do.

Configure the endpoint authentication

You can set the authentication type when you create an online endpoint. Set the `auth_mode` to `key` or `aml_token` depending on which one you want to use. The default value is `key`.

When deploying using CLI v2, set this value in the [online endpoint YAML file](#). For more information, see [How to deploy an online endpoint](#).

When deploying using the Python SDK v2 (preview), use the [OnlineEndpoint](#) class.

Get the key or token

Access to retrieve the key or token for an online endpoint is restricted by Azure role-based access controls (Azure RBAC). To retrieve the authentication key or token, your security principal (user identity or service principal) must be assigned one of the following roles:

- Owner
- Contributor
- A custom role that allows `Microsoft.MachineLearningServices/workspaces/onlineEndpoints/token/action` and `Microsoft.MachineLearningServices/workspaces/onlineEndpoints/listkeys/action`.

For more information on using Azure RBAC with Azure Machine Learning, see [Manage access to Azure Machine Learning](#).

To get the key, use `az ml online-endpoint get-credentials`. This command returns a JSON document that contains the key or token. **Keys** will be returned in the `primaryKey` and `secondaryKey` fields. **Tokens** will be returned in the `accessToken` field. Additionally, the `expiryTimeUtc` and `refreshAfterTimeUtc` fields contain the token expiration and refresh times. The following example shows how to use the `--query` parameter to return only the primary key:

```
ENDPOINT_KEY=$(az ml online-endpoint get-credentials -n $ENDPOINT_NAME -o tsv --query primaryKey)
```

Score data using the token

When calling the online endpoint for scoring, pass the key or token in the authorization header. The following example shows how to use the curl utility to call the online endpoint using a key (if using a token, replace `$ENDPOINT_KEY` with the token value):

```
SCORING_URI=$(az ml online-endpoint show -n $ENDPOINT_NAME -o tsv --query scoring_uri)

curl --request POST "$SCORING_URI" --header "Authorization: Bearer $ENDPOINT_KEY" --header 'Content-Type: application/json' --data @endpoints/online/model-1/sample-request.json
```

Next steps

- Deploy a machine learning model using an online endpoint
- Enable network isolation for managed online endpoints

Use network isolation with managed online endpoints (preview)

9/22/2022 • 14 minutes to read • [Edit Online](#)

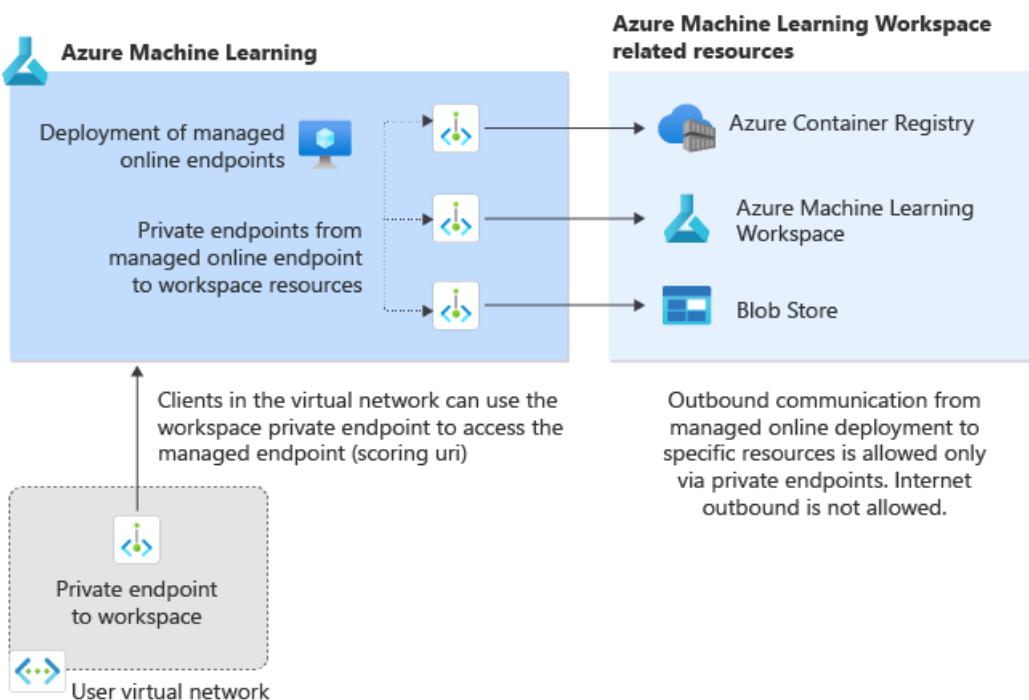
When deploying a machine learning model to a managed online endpoint, you can secure communication with the online endpoint by using [private endpoints](#). Using a private endpoint with online endpoints is currently a preview feature.

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

You can secure the inbound scoring requests from clients to an *online endpoint*. You can also secure the outbound communications between a *deployment* and the Azure resources used by the deployment. Security for inbound and outbound communication is configured separately. For more information on endpoints and deployments, see [What are endpoints and deployments](#).

The following diagram shows how communications flow through private endpoints to the managed online endpoint. Incoming scoring requests from clients are received through the workspace private endpoint from your virtual network. Outbound communication with services is handled through private endpoints to those service instances from the deployment:



Prerequisites

- To use Azure machine learning, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.

- You must install and configure the Azure CLI and ML extension. For more information, see [Install, set up, and use the CLI \(v2\)](#).
- You must have an Azure Resource Group, in which you (or the service principal you use) need to have **Contributor** access. You'll have such a resource group if you configured your ML extension per the above article.
- You must have an Azure Machine Learning workspace, and the workspace must use a private endpoint. If you don't have one, the steps in this article create an example workspace, VNet, and VM. For more information, see [Configure a private endpoint for Azure Machine Learning workspace](#).

The workspace can be configured to allow or disallow public network access. If you plan on using managed online endpoint deployments that use **public outbound**, then you must also [configure the workspace to allow public access](#).

Outbound communication from managed online endpoint deployment is to the *workspace API*. When the endpoint is configured to use **public outbound**, then the workspace must be able to accept that public communication (allow public access).

- When the workspace is configured with a private endpoint, the Azure Container Registry for the workspace must be configured for **Premium** tier. For more information, see [Azure Container Registry service tiers](#).
- The Azure Container Registry and Azure Storage Account must be in the same Azure Resource Group as the workspace.

IMPORTANT

The end-to-end example in this article comes from the files in the [azureml-examples](#) GitHub repository. To clone the samples repository and switch to the repository's `cli/` directory, use the following commands:

```
git clone https://github.com/Azure/azureml-examples
cd azureml-examples/cli
```

Limitations

- The `v1_legacy_mode` flag must be disabled (false) on your Azure Machine Learning workspace. If this flag is enabled, you won't be able to create a managed online endpoint. For more information, see [Network isolation with v2 API](#).
- If your Azure Machine Learning workspace has a private endpoint that was created before May 24, 2022, you must recreate the workspace's private endpoint before configuring your online endpoints to use a private endpoint. For more information on creating a private endpoint for your workspace, see [How to configure a private endpoint for Azure Machine Learning workspace](#).
- Secure outbound communication creates three private endpoints per deployment. One to Azure Blob storage, one to Azure Container Registry, and one to your workspace.
- Azure Log Analytics and Application Insights aren't supported when using network isolation with a deployment. To see the logs for the deployment, use the `az ml online-deployment get_logs` command instead.
- You can configure public access to a **managed online endpoint** (*inbound* and *outbound*). You can also configure [public access to an Azure Machine Learning workspace](#).

Outbound communication from managed online endpoint deployment is to the *workspace API*. When the

endpoint is configured to use **public outbound**, then the workspace must be able to accept that public communication (allow public access).

NOTE

Requests to create, update, or retrieve the authentication keys are sent to the Azure Resource Manager over the public network.

Inbound (scoring)

To secure scoring requests to the online endpoint to your virtual network, set the `public_network_access` flag for the endpoint to `disabled`:

```
az ml online-endpoint create -f endpoint.yml --set public_network_access=disabled
```

When `public_network_access` is `disabled`, inbound scoring requests are received using the [private endpoint of the Azure Machine Learning workspace](#) and the endpoint can't be reached from public networks.

Outbound (resource access)

To restrict communication between a deployment and the Azure resources used by the deployment, set the `egress_public_network_access` flag to `disabled`. Use this flag to ensure that the download of the model, code, and images needed by your deployment are secured with a private endpoint.

The following are the resources that the deployment communicates with over the private endpoint:

- The Azure Machine Learning workspace.
- The Azure Storage blob that is the default storage for the workspace.
- The Azure Container Registry for the workspace.

When you configure the `egress_public_network_access` to `disabled`, a new private endpoint is created per deployment, per service. For example, if you set the flag to `disabled` for three deployments to an online endpoint, nine private endpoints are created. Each deployment would have three private endpoints that are used to communicate with the workspace, blob, and container registry.

```
az ml online-deployment create -f deployment.yml --set egress_public_network_access=disabled
```

Scenarios

The following table lists the supported configurations when configuring inbound and outbound communications for an online endpoint:

CONFIGURATION	INBOUND (ENDPOINT PROPERTY)	OUTBOUND (DEPLOYMENT PROPERTY)	SUPPORTED?
secure inbound with secure outbound	<code>public_network_access</code> is disabled	<code>egress_public_network_access</code> is disabled	Yes
secure inbound with public outbound	<code>public_network_access</code> is disabled The workspace must also allow public access.	<code>egress_public_network_access</code> is enabled	Yes

CONFIGURATION	INBOUND (ENDPOINT PROPERTY)	OUTBOUND (DEPLOYMENT PROPERTY)	SUPPORTED?
public inbound with secure outbound	<code>public_network_access</code> is enabled	<code>egress_public_network_access</code> is disabled	Yes
public inbound with public outbound	<code>public_network_access</code> is enabled The workspace must also allow public access.	<code>egress_public_network_access</code> is enabled	Yes

IMPORTANT

Outbound communication from managed online endpoint deployment is to the *workspace API*. When the endpoint is configured to use **public outbound**, then the workspace must be able to accept that public communication (allow public access).

End-to-end example

Use the information in this section to create an example configuration that uses private endpoints to secure online endpoints.

TIP

In this example, an Azure Virtual Machine is created inside the VNet. You connect to the VM using SSH, and run the deployment from the VM. This configuration is used to simplify the steps in this example, and does not represent a typical secure configuration. For example, in a production environment you would most likely use a VPN client or Azure ExpressRoute to directly connect clients to the virtual network.

Create workspace and secured resources

The steps in this section use an Azure Resource Manager template to create the following Azure resources:

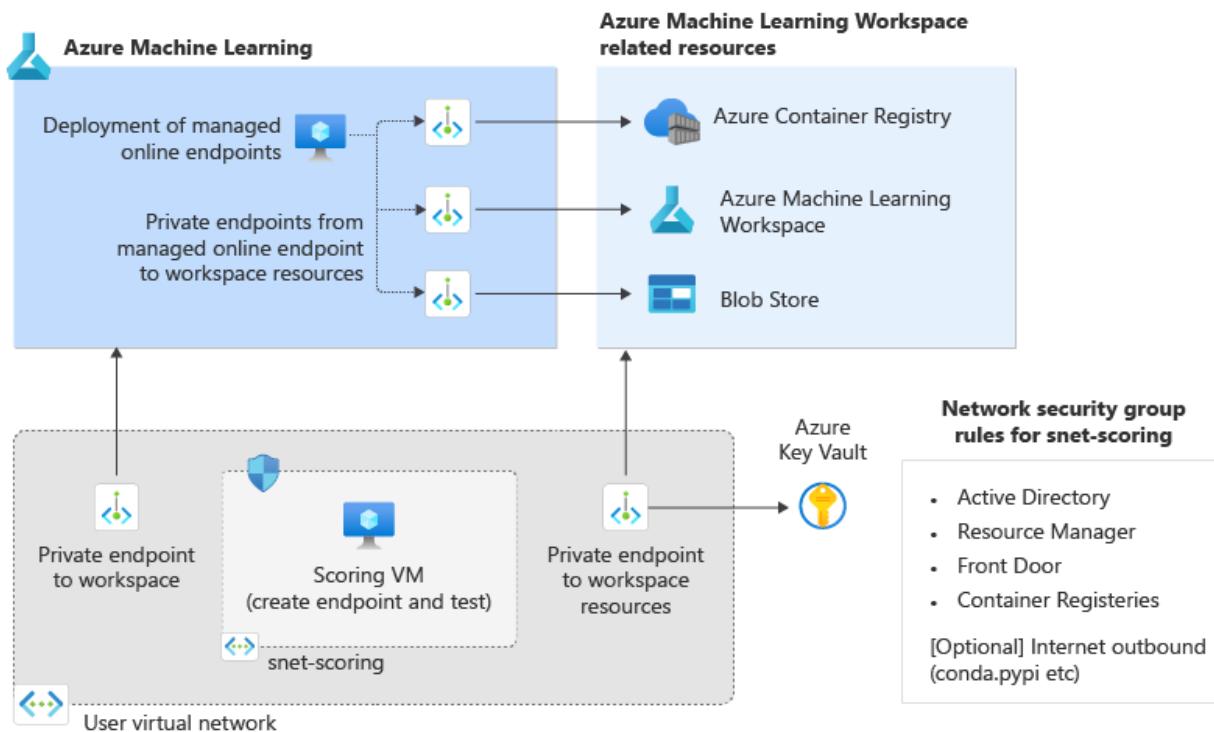
- Azure Virtual Network
- Azure Machine Learning workspace
- Azure Container Registry
- Azure Key Vault
- Azure Storage account (blob & file storage)

Public access is disabled for all the services. While the Azure Machine Learning workspace is secured behind a vnet, it's configured to allow public network access. For more information, see [CLI 2.0 secure communications](#). A scoring subnet is created, along with outbound rules that allow communication with the following Azure services:

- Azure Active Directory
- Azure Resource Manager
- Azure Front Door
- Microsoft Container Registries

The following diagram shows the different components created in this architecture:

The following diagram shows the overall architecture of this example:



To create the resources, use the following Azure CLI commands. Replace <UNIQUE_SUFFIX> with a unique suffix for the resources that are created.

```
# SUFFIX will be used as resource name suffix in created workspace and related resources
export SUFFIX=<UNIQUE_SUFFIX>"
```

```
az deployment group create --template-file endpoints/online/managed/vnet/setup_ws/main.bicep --parameters
suffix=$SUFFIX
# Note: if you get an error that appinsights is not available in your current location, use optional
parameter to the above script: appinsightsLocation=<location> (e.g. westus2)
```

Create the virtual machine jump box

To create an Azure Virtual Machine that can be used to connect to the VNet, use the following command. Replace <your-new-password> with the password you want to use when connecting to this VM:

```
# create vm
az vm create --name test-vm --vnet-name vnet-$SUFFIX --subnet snet-scoring --image UbuntuLTS --admin-
username azureuser --admin-password <your-new-password>
```

IMPORTANT

The VM created by these commands has a public endpoint that you can connect to over the public network.

The response from this command is similar to the following JSON document:

```
{  
    "fqdns": "",  
    "id": "/subscriptions/<GUID>/resourceGroups/<my-resource-group>/providers/Microsoft.Compute/virtualMachines/test-vm",  
    "location": "westus",  
    "macAddress": "00-0D-3A-ED-D8-E8",  
    "powerState": "VM running",  
    "privateIpAddress": "192.168.0.12",  
    "publicIpAddress": "20.114.122.77",  
    "resourceGroup": "<my-resource-group>",  
    "zones": ""  
}
```

Use the following command to connect to the VM using SSH. Replace `publicIpAddress` with the value of the public IP address in the response from the previous command:

```
ssh azureuse@publicIpAddress
```

When prompted, enter the password you used when creating the VM.

Configure the VM

1. Use the following commands from the SSH session to install the CLI and Docker:

```
# setup docker  
sudo apt-get update -y && sudo apt install docker.io -y && sudo snap install docker && docker --version && sudo usermod -aG docker $USER  
# setup az cli and ml extension  
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash && az extension add --upgrade -n ml -y
```

2. To create the environment variables used by this example, run the following commands. Replace `<YOUR_SUBSCRIPTION_ID>` with your Azure subscription ID. Replace `<YOUR_RESOURCE_GROUP>` with the resource group that contains your workspace. Replace `<SUFFIX_USED_IN_SETUP>` with the suffix you provided earlier. Replace `<LOCATION>` with the location of your Azure workspace. Replace `<YOUR_ENDPOINT_NAME>` with the name to use for the endpoint.

TIP

Use the tabs to select whether you want to perform a deployment using an MLflow model or generic ML model.

- [Generic model](#)
- [MLflow model](#)

```

export SUBSCRIPTION=<YOUR_SUBSCRIPTION_ID>
export RESOURCE_GROUP=<YOUR_RESOURCE_GROUP>
export LOCATION=<LOCATION>

# SUFFIX that was used when creating the workspace resources. Alternatively the resource names can be
looked up from the resource group after the vnet setup script has completed.
export SUFFIX=<SUFFIX_USED_IN_SETUP>

# SUFFIX used during the initial setup. Alternatively the resource names can be looked up from the
resource group after the setup script has completed.
export WORKSPACE=mlw-$SUFFIX
export ACR_NAME=cr$SUFFIX

# provide a unique name for the endpoint
export ENDPOINT_NAME=<YOUR_ENDPOINT_NAME>

# name of the image that will be built for this sample and pushed into acr - no need to change this
export IMAGE_NAME="img"

# Yaml files that will be used to create endpoint and deployment. These are relative to azureml-
examples/cli/ directory. Do not change these
export ENDPOINT_FILE_PATH="endpoints/online/managed/vnet/sample/endpoint.yml"
export DEPLOYMENT_FILE_PATH="endpoints/online/managed/vnet/sample/blue-deployment-vnet.yml"
export SAMPLE_REQUEST_PATH="endpoints/online/managed/vnet/sample/sample-request.json"
export ENV_DIR_PATH="endpoints/online/managed/vnet/sample/environment"

```

- To sign in to the Azure CLI in the VM environment, use the following command:

```
az login
```

- To configure the defaults for the CLI, use the following commands:

```

# configure cli defaults
az account set --subscription $SUBSCRIPTION
az configure --defaults group=$RESOURCE_GROUP workspace=$WORKSPACE location=$LOCATION

```

- To clone the example files for the deployment, use the following command:

```
sudo mkdir -p /home/samples; sudo git clone -b main --depth 1 https://github.com/Azure/azureml-examples.git /home/samples/azureml-examples
```

- To build a custom docker image to use with the deployment, use the following commands:

```

# Navigate to the samples
cd /home/samples/azureml-examples/cli/$ENV_DIR_PATH
# login to acr. Optionally, to avoid using sudo, complete the docker post install steps:
https://docs.docker.com/engine/install/linux-postinstall/
sudo az acr login -n $ACR_NAME
# Build the docker image with the sample docker file
sudo docker build -t $ACR_NAME.azurecr.io/repo/$IMAGE_NAME:v1 .
# push the image to the ACR
sudo docker push $ACR_NAME.azurecr.io/repo/$IMAGE_NAME:v1
# check if the image exists in acr
az acr repository show -n $ACR_NAME --repository repo/$IMAGE_NAME

```

TIP

In this example, we build the Docker image before pushing it to Azure Container Registry. Alternatively, you can build the image in your vnet by using an Azure Machine Learning compute cluster and environments. For more information, see [Secure Azure Machine Learning workspace](#).

Create a secured managed online endpoint

1. To create a managed online endpoint that is secured using a private endpoint for inbound and outbound communication, use the following commands:

TIP

You can test or debug the Docker image locally by using the `--local` flag when creating the deployment. For more information, see the [Deploy and debug locally](#) article.

```
# navigate to the cli directory in the azureml-examples repo
cd /home/samples/azureml-examples/cli/

# create endpoint
az ml online-endpoint create --name $ENDPOINT_NAME -f $ENDPOINT_FILE_PATH --set
public_network_access="disabled"
# create deployment in managed vnet
az ml online-deployment create --name blue --endpoint $ENDPOINT_NAME -f $DEPLOYMENT_FILE_PATH --all-
traffic --set environment.image="$ACR_NAME.azurecr.io/repo/$IMAGE_NAME:v1"
egress_public_network_access="disabled"
```

2. To make a scoring request with the endpoint, use the following commands:

```
# Try scoring using the CLI
az ml online-endpoint invoke --name $ENDPOINT_NAME --request-file $SAMPLE_REQUEST_PATH

# Try scoring using curl
ENDPOINT_KEY=$(az ml online-endpoint get-credentials -n $ENDPOINT_NAME -o tsv --query primaryKey)
SCORING_URI=$(az ml online-endpoint show -n $ENDPOINT_NAME -o tsv --query scoring_uri)
curl --request POST "$SCORING_URI" --header "Authorization: Bearer $ENDPOINT_KEY" --header 'Content-
Type: application/json' --data @$SAMPLE_REQUEST_PATH
```

Cleanup

To delete the endpoint, use the following command:

```
az ml online-endpoint delete --name $ENDPOINT_NAME --yes --no-wait
```

To delete the VM, use the following command:

```
az vm delete -n $VM_NAME -y --no-wait
```

To delete all the resources created in this article, use the following command. Replace `<resource-group-name>` with the name of the resource group used in this example:

```
az group delete --resource-group <resource-group-name>
```

Troubleshooting

Online endpoint creation fails with a V1LegacyMode == true message

The Azure Machine Learning workspace can be configured for `v1_legacy_mode`, which disables v2 APIs.

Managed online endpoints are a feature of the v2 API platform, and won't work if `v1_legacy_mode` is enabled for the workspace.

IMPORTANT

Check with your network security team before disabling `v1_legacy_mode`. It may have been enabled by your network security team for a reason.

For information on how to disable `v1_legacy_mode`, see [Network isolation with v2](#).

Online endpoint creation with key-based authentication fails

Use the following command to list the network rules of the Azure Key Vault for your workspace. Replace `<keyvault-name>` with the name of your key vault:

```
az keyvault network-rule list -n <keyvault-name>
```

The response for this command is similar to the following JSON document:

```
{
  "bypass": "AzureServices",
  "defaultAction": "Deny",
  "ipRules": [],
  "virtualNetworkRules": []
}
```

If the value of `bypass` isn't `AzureServices`, use the guidance in the [Configure key vault network settings](#) to set it to `AzureServices`.

Online deployments fail with an image download error

1. Check if the `egress-public-network-access` flag is **disabled** for the deployment. If this flag is enabled, and the visibility of the container registry is private, then this failure is expected.
2. Use the following command to check the status of the private endpoint connection. Replace `<registry-name>` with the name of the Azure Container Registry for your workspace:

```
az acr private-endpoint-connection list -r <registry-name> --query "[? privateLinkServiceConnectionState.description=='Egress for Microsoft.MachineLearningServices/workspaces/onlineEndpoints'].{Name:name, status:privateLinkServiceConnectionState.status}"
```

In the response document, verify that the `status` field is set to `Approved`. If it isn't approved, use the following command to approve it. Replace `<private-endpoint-name>` with the name returned from the previous command:

```
az network private-endpoint-connection approve -n <private-endpoint-name>
```

Scoring endpoint can't be resolved

1. Verify that the client issuing the scoring request is a virtual network that can access the Azure Machine

Learning workspace.

2. Use the `nslookup` command on the endpoint hostname to retrieve the IP address information:

```
nslookup endpointname.westcentralus.inference.ml.azure.com
```

The response contains an **address**. This address should be in the range provided by the virtual network.

3. If the host name isn't resolved by the `nslookup` command, check if an A record exists in the private DNS zone for the virtual network. To check the records, use the following command:

```
az network private-dns record-set list -z privatelink.api.azureml.ms -o tsv --query [].name
```

The results should contain an entry that is similar to `*.<GUID>.inference.<region>`.

4. If no inference value is returned, delete the private endpoint for the workspace and then recreate it. For more information, see [How to configure a private endpoint](#).

Online deployments can't be scored

1. Use the following command to see if the deployment was successfully deployed:

```
az ml online-deployment show -e <endpointname> -n <deploymentname> --query  
'{name:name,state:provisioning_state}'
```

If the deployment completed successfully, the value of `state` will be `Succeeded`.

2. If the deployment was successful, use the following command to check that traffic is assigned to the deployment. Replace `<endpointname>` with the name of your endpoint:

```
az ml online-endpoint show -n <endpointname> --query traffic
```

TIP

This step isn't needed if you are using the `azureml-model-deployment` header in your request to target this deployment.

The response from this command should list percentage of traffic assigned to deployments.

3. If the traffic assignments (or deployment header) are set correctly, use the following command to get the logs for the endpoint. Replace `<endpointname>` with the name of the endpoint, and `<deploymentname>` with the deployment:

```
az ml online-deployment get-logs -e <endpointname> -n <deploymentname>
```

Look through the logs to see if there's a problem running the scoring code when you submit a request to the deployment.

Next steps

- [Safe rollout for online endpoints](#)
- [How to autoscale managed online endpoints](#)
- [View costs for an Azure Machine Learning managed online endpoint](#)

- [Access Azure resources with a online endpoint and managed identity](#)
- [Troubleshoot online endpoints deployment](#)

Access Azure resources from an online endpoint with a managed identity

9/22/2022 • 12 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

Learn how to access Azure resources from your scoring script with an online endpoint and either a system-assigned managed identity or a user-assigned managed identity.

Managed endpoints allow Azure Machine Learning to manage the burden of provisioning your compute resource and deploying your machine learning model. Typically your model needs to access Azure resources such as the Azure Container Registry or your blob storage for inferencing; with a managed identity you can access these resources without needing to manage credentials in your code. [Learn more about managed identities](#).

This guide assumes you don't have a managed identity, a storage account or an online endpoint. If you already have these components, skip to the [give access permission to the managed identity](#) section.

Prerequisites

- To use Azure Machine Learning, you must have an Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- Install and configure the Azure CLI and ML (v2) extension. For more information, see [Install, set up, and use the 2.0 CLI](#).
- An Azure Resource group, in which you (or the service principal you use) need to have `User Access Administrator` and `Contributor` access. You'll have such a resource group if you configured your ML extension per the above article.
- An Azure Machine Learning workspace. You'll have a workspace if you configured your ML extension per the above article.
- A trained machine learning model ready for scoring and deployment. If you are following along with the sample, a model is provided.
- If you haven't already set the defaults for the Azure CLI, save your default settings. To avoid passing in the values for your subscription, workspace, and resource group multiple times, run this code:

```
az account set --subscription <subscription ID>
az configure --defaults workspace=<Azure Machine Learning workspace name> group=<resource group>
```

- To follow along with the sample, clone the samples repository

```
git clone https://github.com/Azure/azureml-examples --depth 1
cd azureml-examples/cli
```

Limitations

- The identity for an endpoint is immutable. During endpoint creation, you can associate it with a system-

assigned identity (default) or a user-assigned identity. You can't change the identity after the endpoint has been created.

Define configuration YAML file for deployment

To deploy an online endpoint with the CLI, you need to define the configuration in a YAML file. For more information on the YAML schema, see [online endpoint YAML reference](#) document.

The YAML files in the following examples are used to create online endpoints.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

The following YAML example is located at `endpoints/online/managed/managed-identities/1-sai-create-endpoint`.

The file,

- Defines the name by which you want to refer to the endpoint, `my-sai-endpoint`.
- Specifies the type of authorization to use to access the endpoint, `auth-mode: key`.

```
$schema: https://azureschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: my-sai-endpoint
auth_mode: key
```

This YAML example, `2-sai-deployment.yml`,

- Specifies that the type of endpoint you want to create is an `online` endpoint.
- Indicates that the endpoint has an associated deployment called `blue`.
- Configures the details of the deployment such as, which model to deploy and which environment and scoring script to use.

```
$schema: https://azureschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: blue
model:
  path: ../../model-1/model/
code_configuration:
  code: ../../model-1/onlinescoring/
  scoring_script: score_managedidentity.py
environment:
  conda_file: ../../model-1/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1
instance_type: Standard_DS2_v2
instance_count: 1
environment_variables:
  STORAGE_ACCOUNT_NAME: "storage_place_holder"
  STORAGE_CONTAINER_NAME: "container_place_holder"
  FILE_NAME: "file_place_holder"
```

Configure variables for deployment

Configure the variable names for the workspace, workspace location, and the endpoint you want to create for use with your deployment.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

The following code exports these values as environment variables in your endpoint:

```
export WORKSPACE=<WORKSPACE_NAME>
export LOCATION=<WORKSPACE_LOCATION>
export ENDPOINT_NAME=<ENDPOINT_NAME>
```

Next, specify what you want to name your blob storage account, blob container, and file. These variable names are defined here, and are referred to in `az storage account create` and `az storage container create` commands in the next section.

The following code exports those values as environment variables:

```
export STORAGE_ACCOUNT_NAME=<BLOB_STORAGE_TO_ACCESS>
export STORAGE_CONTAINER_NAME=<CONTAINER_TO_ACCESS>
export FILE_NAME=<FILE_TO_ACCESS>
```

After these variables are exported, create a text file locally. When the endpoint is deployed, the scoring script will access this text file using the system-assigned managed identity that's generated upon endpoint creation.

Create the managed identity

To access Azure resources, create a system-assigned or user-assigned managed identity for your online endpoint.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

When you [create an online endpoint](#), a system-assigned managed identity is automatically generated for you, so no need to create a separate one.

Create storage account and container

For this example, create a blob storage account and blob container, and then upload the previously created text file to the blob container. This is the storage account and blob container that you'll give the online endpoint and managed identity access to.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

First, create a storage account.

```
az storage account create --name $STORAGE_ACCOUNT_NAME --location $LOCATION
```

Next, create the blob container in the storage account.

```
az storage container create --account-name $STORAGE_ACCOUNT_NAME --name $STORAGE_CONTAINER_NAME
```

Then, upload your text file to the blob container.

```
az storage blob upload --account-name $STORAGE_ACCOUNT_NAME --container-name $STORAGE_CONTAINER_NAME --name $FILE_NAME --file endpoints/online/managed/managed-identities/hello.txt
```

Create an online endpoint

The following code creates an online endpoint without specifying a deployment.

WARNING

The identity for an endpoint is immutable. During endpoint creation, you can associate it with a system-assigned identity (default) or a user-assigned identity. You can't change the identity after the endpoint has been created.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

When you create an online endpoint, a system-assigned managed identity is created for the endpoint by default.

```
az ml online-endpoint create --name $ENDPOINT_NAME -f endpoints/online/managed/managed-identities/1-sai-create-endpoint.yml
```

Check the status of the endpoint with the following.

```
az ml online-endpoint show --name $ENDPOINT_NAME
```

If you encounter any issues, see [Troubleshooting online endpoints deployment and scoring](#).

Give access permission to the managed identity

IMPORTANT

Online endpoints require Azure Container Registry pull permission, AcrPull permission, to the container registry and Storage Blob Data Reader permission to the default datastore of the workspace.

You can allow the online endpoint permission to access your storage via its system-assigned managed identity or give permission to the user-assigned managed identity to access the storage account created in the previous section.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

Retrieve the system-assigned managed identity that was created for your endpoint.

```
system_identity=`az ml online-endpoint show --name $ENDPOINT_NAME --query "identity.principal_id" -o tsv`
```

From here, you can give the system-assigned managed identity permission to access your storage.

```
az role assignment create --assignee-object-id $system_identity --assignee-principal-type ServicePrincipal --role "Storage Blob Data Reader" --scope $storage_id
```

Scoring script to access Azure resource

Refer to the following script to understand how to use your identity token to access Azure resources, in this scenario, the storage account created in previous sections.

```
import os
import logging
```

```

import json
import numpy
import joblib
import requests

def get_token():
    access_token = None
    msi_endpoint = os.environ.get("MSI_ENDPOINT", None)
    msi_secret = os.environ.get("MSI_SECRET", None)

    # If UAI_CLIENT_ID is provided then assume that endpoint was created with user assigned identity,
    # # otherwise system assigned identity deployment.
    client_id = os.environ.get("UAI_CLIENT_ID", None)
    if client_id is not None:
        token_url = (
            msi_endpoint + f"?clientid={client_id}&resource=https://storage.azure.com/"
        )
    else:
        token_url = msi_endpoint + f"?resource=https://storage.azure.com/"

    logging.info("Trying to get identity token...")
    headers = {"secret": msi_secret, "Metadata": "true"}
    resp = requests.get(token_url, headers=headers)
    resp.raise_for_status()
    access_token = resp.json()["access_token"]
    logging.info("Retrieved token successfully.")
    return access_token

def access_blob_storage():
    logging.info("Trying to access blob storage...")
    storage_account = os.environ.get("STORAGE_ACCOUNT_NAME")
    storage_container = os.environ.get("STORAGE_CONTAINER_NAME")
    file_name = os.environ.get("FILE_NAME")
    logging.info(
        f"storage_account: {storage_account}, container: {storage_container}, filename: {file_name}"
    )
    token = get_token()

    blob_url = f"https://{storage_account}.blob.core.windows.net/{storage_container}/{file_name}?api-version=2019-04-01"
    auth_headers = {
        "Authorization": f"Bearer {token}",
        "x-ms-blob-type": "BlockBlob",
        "x-ms-version": "2019-02-02",
    }
    resp = requests.get(blob_url, headers=auth_headers)
    resp.raise_for_status()
    logging.info(f"Blob contains: {resp.text}")

def init():
    global model
    # AZUREML_MODEL_DIR is an environment variable created during deployment.
    # It is the path to the model folder (./azureml-models/$MODEL_NAME/$VERSION)
    # For multiple models, it points to the folder containing all deployed models (./azureml-models)
    # Please provide your model's folder name if there is one
    model_path = os.path.join(
        os.getenv("AZUREML_MODEL_DIR"), "model/sklearn_regression_model.pkl"
    )
    # deserialize the model file back into a sklearn model
    model = joblib.load(model_path)
    logging.info("Model loaded")

    # Access Azure resource (Blob storage) using system assigned identity token
    access_blob_storage()

    logging.info("Init complete")

```

```
# note you can pass in multiple rows for scoring
def run(raw_data):
    logging.info("Request received")
    data = json.loads(raw_data)["data"]
    data = numpy.array(data)
    result = model.predict(data)
    logging.info("Request processed")
    return result.tolist()
```

Create a deployment with your configuration

Create a deployment that's associated with the online endpoint. [Learn more about deploying to online endpoints.](#)

WARNING

This deployment can take approximately 8-14 minutes depending on whether the underlying environment/image is being built for the first time. Subsequent deployments using the same environment will go quicker.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

```
az ml online-deployment create --endpoint-name $ENDPOINT_NAME --all-traffic --name blue --file
endpoints/online/managed/managed-identities/2-sai-deployment.yml --set
environment_variables.STORAGE_ACCOUNT_NAME=$STORAGE_ACCOUNT_NAME
environment_variables.STORAGE_CONTAINER_NAME=$STORAGE_CONTAINER_NAME
environment_variables.FILE_NAME=$FILE_NAME
```

NOTE

The value of the `--name` argument may override the `name` key inside the YAML file.

Check the status of the deployment.

```
az ml online-deployment show --endpoint-name $ENDPOINT_NAME --name blue
```

To refine the above query to only return specific data, see [Query Azure CLI command output](#).

NOTE

The init method in the scoring script reads the file from your storage account using the system assigned managed identity token.

To check the init method output, see the deployment log with the following code.

```
# Check deployment logs to confirm blob storage file contents read operation success.
az ml online-deployment get-logs --endpoint-name $ENDPOINT_NAME --name blue
```

When your deployment completes, the model, the environment, and the endpoint are registered to your Azure Machine Learning workspace.

Confirm your endpoint deployed successfully

Once your online endpoint is deployed, confirm its operation. Details of inferencing vary from model to model.

For this guide, the JSON query parameters look like:

```
{"data": [
    [1,2,3,4,5,6,7,8,9,10],
    [10,9,8,7,6,5,4,3,2,1]
]}
```

To call your endpoint, run:

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

```
az ml online-endpoint invoke --name $ENDPOINT_NAME --request-file endpoints/online/model-1/sample-request.json
```

Delete the endpoint and storage account

If you don't plan to continue using the deployed online endpoint and storage, delete them to reduce costs. When you delete the endpoint, all of its associated deployments are deleted as well.

- [System-assigned managed identity](#)
- [User-assigned managed identity](#)

```
az ml online-endpoint delete --name $ENDPOINT_NAME --yes
```

```
az storage account delete --name $STORAGE_ACCOUNT_NAME --yes
```

Next steps

- [Deploy and score a machine learning model by using a online endpoint.](#)
- For more on deployment, see [Safe rollout for online endpoints](#).
- For more information on using the CLI, see [Use the CLI extension for Azure Machine Learning](#).
- To see which compute resources you can use, see [Managed online endpoints SKU list](#).
- For more on costs, see [View costs for an Azure Machine Learning managed online endpoint](#).
- For information on monitoring endpoints, see [Monitor managed online endpoints](#).
- For limitations for managed endpoints, see [Manage and increase quotas for resources with Azure Machine Learning](#).

Autoscale an online endpoint

9/22/2022 • 8 minutes to read • [Edit Online](#)

Autoscale automatically runs the right amount of resources to handle the load on your application. [Online endpoints](#) supports autoscaling through integration with the Azure Monitor autoscale feature.

Azure Monitor autoscaling supports a rich set of rules. You can configure metrics-based scaling (for instance, CPU utilization >70%), schedule-based scaling (for example, scaling rules for peak business hours), or a combination. For more information, see [Overview of autoscale in Microsoft Azure](#).



Today, you can manage autoscaling using either the Azure CLI, REST, ARM, or the browser-based Azure portal. Other Azure ML SDKs, such as the Python SDK, will add support over time.

Prerequisites

- A deployed endpoint. [Deploy and score a machine learning model by using an online endpoint](#).

Define an autoscale profile

To enable autoscale for an endpoint, you first define an autoscale profile. This profile defines the default, minimum, and maximum scale set capacity. The following example sets the default and minimum capacity as two VM instances, and the maximum capacity as five:

- [Azure CLI](#)
- [Python](#)
- [Studio](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

The following snippet sets the endpoint and deployment names:

```
# set your existing endpoint name
ENDPOINT_NAME=your-endpoint-name
DEPLOYMENT_NAME=blue
```

Next, get the Azure Resource Manager ID of the deployment and endpoint:

```
# ARM id of the deployment
DEPLOYMENT_RESOURCE_ID=$(az ml online-deployment show -e $ENDPOINT_NAME -n $DEPLOYMENT_NAME -o tsv --query
"id")
# ARM id of the deployment. todo: change to --query "id"
ENDPOINT_RESOURCE_ID=$(az ml online-endpoint show -n $ENDPOINT_NAME -o tsv --query
"properties.\\"azurerm.onlineendpointid\\\"")
# set a unique name for autoscale settings for this deployment. The below will append a random number to
make the name unique.
AUTOSCALE_SETTINGS_NAME=autoscale-$ENDPOINT_NAME-$DEPLOYMENT_NAME-`echo $RANDOM`
```

The following snippet creates the autoscale profile:

```
az monitor autoscale create \
--name $AUTOSCALE_SETTINGS_NAME \
--resource $DEPLOYMENT_RESOURCE_ID \
--min-count 2 --max-count 5 --count 2
```

NOTE

For more, see the [reference page for autoscale](#)

Create a rule to scale out using metrics

A common scaling out rule is one that increases the number of VM instances when the average CPU load is high. The following example will allocate two more nodes (up to the maximum) if the CPU average a load of greater than 70% for five minutes::

- [Azure CLI](#)
- [Python](#)
- [Studio](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az monitor autoscale rule create \
--autoscale-name $AUTOSCALE_SETTINGS_NAME \
--condition "CpuUtilizationPercentage > 70 avg 5m" \
--scale out 2
```

The rule is part of the `my-scale-settings` profile (`autoscale-name` matches the `name` of the profile). The value of its `condition` argument says the rule should trigger when "The average CPU consumption among the VM instances exceeds 70% for five minutes." When that condition is satisfied, two more VM instances are allocated.

NOTE

For more information on the CLI syntax, see [az monitor autoscale](#).

Create a rule to scale in using metrics

When load is light, a scaling in rule can reduce the number of VM instances. The following example will release a single node, down to a minimum of 2, if the CPU load is less than 30% for 5 minutes:

- [Azure CLI](#)
- [Python](#)
- [Studio](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az monitor autoscale rule create \
--autoscale-name $AUTOSCALE_SETTINGS_NAME \
--condition "CpuUtilizationPercentage < 25 avg 5m" \
--scale in 1
```

Create a scaling rule based on endpoint metrics

The previous rules applied to the deployment. Now, add a rule that applies to the endpoint. In this example, if the request latency is greater than an average of 70 milliseconds for 5 minutes, allocate another node.

- [Azure CLI](#)
- [Python](#)
- [Studio](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

```
az monitor autoscale rule create \
--autoscale-name $AUTOSCALE_SETTINGS_NAME \
--condition "RequestLatency > 70 avg 5m" \
--scale out 1 \
--resource $ENDPOINT_RESOURCE_ID
```

Create scaling rules based on a schedule

You can also create rules that apply only on certain days or at certain times. In this example, the node count is set to 2 on the weekend.

- [Azure CLI](#)
- [Python](#)
- [Studio](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

```
az monitor autoscale profile create \
--name weekend-profile \
--autoscale-name $AUTOSCALE_SETTINGS_NAME \
--min-count 2 --count 2 --max-count 2 \
--recurrence week sat sun --timezone "Pacific Standard Time"
```

Delete resources

If you are not going to use your deployments, delete them:

- [Azure CLI](#)
- [Python](#)
- [Studio](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

```
# delete the autoscaling profile
az monitor autoscale delete -n "$AUTOSCALE_SETTINGS_NAME"

# delete the endpoint
az ml online-endpoint delete --name $ENDPOINT_NAME --yes --no-wait
```

Next steps

To learn more about autoscale with Azure Monitor, see the following articles:

- Understand autoscale settings
- Overview of common autoscale patterns
- Best practices for autoscale
- Troubleshooting Azure autoscale

Managed online endpoints SKU list

9/22/2022 • 2 minutes to read • [Edit Online](#)

This table shows the VM SKUs that are supported for Azure Machine Learning managed online endpoints.

- The `instance_type` attribute used for deployment must be specified in the form "Standard_F4s_v2". The table below lists instance names, for example, F2s v2. These names should be put in the specified form (`Standard_{name}`) for Azure CLI or Azure Resource Manager templates (ARM templates) requests to create and update deployments.
- For more information on configuration details such as CPU and RAM, see [Azure Machine Learning Pricing](#).

IMPORTANT

If you use a Windows-based image for your deployment, we recommend using a VM SKU that provides a minimum of 4 cores.

SIZE	GENERAL PURPOSE	COMPUTE OPTIMIZED	MEMORY OPTIMIZED	GPU
V.Small	DS1 v2 DS2 v2	F2s v2	E2s v3	NC4as_T4_v3
Small	DS3 v2	F4s v2	E4s v3	NC6s v2 NC6s v3 NC8as_T4_v3
Medium	DS4 v2	F8s v2	E8s v3	NC12s v2 NC12s v3 NC16as_T4_v3
Large	DS5 v2	F16s v2	E16s v3	NC24s v2 NC24s v3 NC64as_T4_v3
X-Large	-	F32s v2 F48s v2 F64s v2 F72s v2	E32s v3 E48s v3 E64s v3	-

View costs for an Azure Machine Learning managed online endpoint (preview)

9/22/2022 • 2 minutes to read • [Edit Online](#)

Learn how to view costs for a managed online endpoint (preview). Costs for your endpoints will accrue to the associated workspace. You can see costs for a specific endpoint using tags.

IMPORTANT

This article only applies to viewing costs for Azure Machine Learning managed online endpoints (preview). Managed online endpoints are different from other resources since they must use tags to track costs. For more information on viewing the costs of other Azure resources, see [Quickstart: Explore and analyze costs with cost analysis](#).

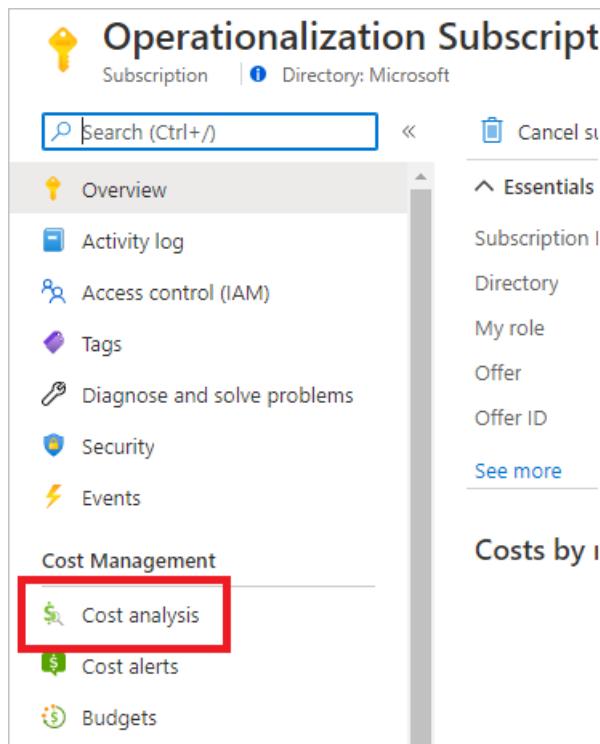
Prerequisites

- Deploy an Azure Machine Learning managed online endpoint (preview).
- Have at least [Billing Reader](#) access on the subscription where the endpoint is deployed

View costs

Navigate to the [Cost Analysis](#) page for your subscription:

1. In the [Azure portal](#), Select [Cost Analysis](#) for your subscription.



Create a filter to scope data to your Azure Machine learning workspace resource:

1. At the top navigation bar, select **Add filter**.
2. In the first filter dropdown, select **Resource** for the filter type.

3. In the second filter dropdown, select your Azure Machine Learning workspace.

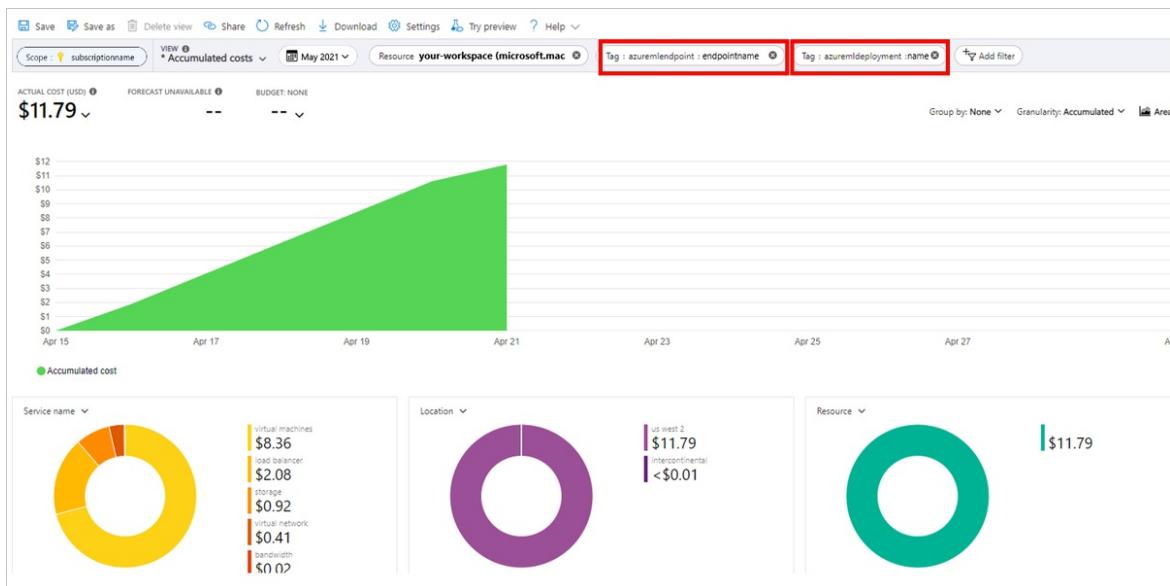
The screenshot shows the Azure Cost Management portal interface. At the top, there are various navigation links like Save, Save as, Delete view, Share, Refresh, Download, Cost by resource, Settings, Try preview, and Help. Below these are dropdown menus for Scope (set to 'subscriptionname'), View (set to '* Accumulated costs'), Date (set to 'May 2021'), and Resource (which has a dropdown arrow and the value 'your-workspace (microsoft.mac)'). A red box highlights the 'Resource' dropdown and its value.

Create a tag filter to show your managed online endpoint and/or managed online deployment:

1. Select **Add filter** > **Tag** > **azuremlendpoint**: "<your endpoint name>"
2. Select **Add filter** > **Tag** > **azuremldeployment**: "<your deployment name>".

NOTE

Dollar values in this image are fictitious and do not reflect actual costs.



Next steps

- [What are endpoints?](#)
- [Learn how to monitor your managed online endpoint.](#)
- [How to deploy managed online endpoints with the Azure CLI](#)
- [How to deploy managed online endpoints with the studio](#)

Monitor online endpoints

9/22/2022 • 6 minutes to read • [Edit Online](#)

In this article, you learn how to monitor [Azure Machine Learning online endpoints](#). Use Application Insights to view metrics and create alerts to stay up to date with your online endpoints.

In this article you learn how to:

- View metrics for your online endpoint
- Create a dashboard for your metrics
- Create a metric alert

Prerequisites

- Deploy an Azure Machine Learning online endpoint.
- You must have at least [Reader access](#) on the endpoint.

Metrics

Use the following steps to view metrics for an online endpoint or deployment:

1. Go to the [Azure portal](#).
2. Navigate to the online endpoint or deployment resource.

online endpoints and deployments are Azure Resource Manager (ARM) resources that can be found by going to their owning resource group. Look for the resource types **Machine Learning online endpoint** and **Machine Learning online deployment**.

3. In the left-hand column, select **Metrics**.

Available metrics

Depending on the resource that you select, the metrics that you see will be different. Metrics are scoped differently for online endpoints and online deployments.

Metrics at endpoint scope

- Request Latency
- Request Latency P50 (Request latency at the 50th percentile)
- Request Latency P90 (Request latency at the 90th percentile)
- Request Latency P95 (Request latency at the 95th percentile)
- Requests per minute
- New connections per second
- Active connection count
- Network bytes

Split on the following dimensions:

- Deployment
- Status Code
- Status Code Class

Bandwidth throttling

Bandwidth will be throttled if the limits are exceeded for *managed* online endpoints (see managed online endpoints section in [Manage and increase quotas for resources with Azure Machine Learning](#)). To determine if requests are throttled:

- Monitor the "Network bytes" metric
- The response trailers will have the fields: `ms-azureml-bandwidth-request-delay-ms` and `ms-azureml-bandwidth-response-delay-ms`. The values of the fields are the delays, in milliseconds, of the bandwidth throttling.

Metrics at deployment scope

- CPU Utilization Percentage
- Deployment Capacity (the number of instances of the requested instance type)
- Disk Utilization
- GPU Memory Utilization (only applicable to GPU instances)
- GPU Utilization (only applicable to GPU instances)
- Memory Utilization Percentage

Split on the following dimension:

- `Instanceld`

Create a dashboard

You can create custom dashboards to visualize data from multiple sources in the Azure portal, including the metrics for your online endpoint. For more information, see [Create custom KPI dashboards using Application Insights](#).

Create an alert

You can also create custom alerts to notify you of important status updates to your online endpoint:

1. At the top right of the metrics page, select **New alert rule**.



2. Select a condition name to specify when your alert should be triggered.

The screenshot shows the Azure portal interface for creating an alert rule. On the left, the 'Create alert rule' page is displayed, showing a condition where 'Whenever the avg requests per minute is greater than <logic undefined>'. On the right, the 'Configure signal logic' dialog box is open, detailing the alert logic: 'Operator: Greater than', 'Aggregation type: Average', 'Threshold value: 10', and 'Unit: Count'. The dialog also shows a preview condition: 'Whenever the average requests per minute is greater than 10'.

3. Select **Add action groups > Create action groups** to specify what should happen when your alert is triggered.
4. Choose **Create alert rule** to finish creating your alert.

Logs

There are three logs that can be enabled for online endpoints:

- **AMLOnlineEndpointTrafficLog (preview):** You could choose to enable traffic logs if you want to check the information of your request. Below are some cases:
 - If the response isn't 200, check the value of the column "ResponseCodeReason" to see what happened. Also check the reason in the "HTTPS status codes" section of the [Troubleshoot online endpoints](#) article.
 - You could check the response code and response reason of your model from the column "ModelStatusCode" and "ModelStatusReason".
 - You want to check the duration of the request like total duration, the request/response duration, and the delay caused by the network throttling. You could check it from the logs to see the breakdown latency.
 - If you want to check how many requests or failed requests recently. You could also enable the logs.
- **AMLOnlineEndpointConsoleLog:** Contains logs that the containers output to the console. Below are some cases:
 - If the container fails to start, the console log may be useful for debugging.
 - Monitor container behavior and make sure that all requests are correctly handled.
 - Write request IDs in the console log. Joining the request ID, the AMLOnlineEndpointConsoleLog, and AMLOnlineEndpointTrafficLog in the Log Analytics workspace, you can trace a request from the network entry point of an online endpoint to the container.
 - You may also use this log for performance analysis in determining the time required by the model to process each request.
- **AMLOnlineEndpointEventLog (preview):** Contains event information regarding the container's life cycle. Currently, we provide information on the following types of events:

NAME	MESSAGE
BackOff	Back-off restarting failed container
Pulled	Container image "<IMAGE_NAME>" already present on machine
Killing	Container inference-server failed liveness probe, will be restarted
Created	Created container image-fetcher
Created	Created container inference-server
Created	Created container model-mount
Unhealthy	Liveness probe failed: <FAILURE_CONTENT>
Unhealthy	Readiness probe failed: <FAILURE_CONTENT>
Started	Started container image-fetcher
Started	Started container inference-server
Started	Started container model-mount
Killing	Stopping container inference-server
Killing	Stopping container model-mount

How to enable/disable logs

IMPORTANT

Logging uses Azure Log Analytics. If you do not currently have a Log Analytics workspace, you can create one using the steps in [Create a Log Analytics workspace in the Azure portal](#).

1. In the [Azure portal](#), go to the resource group that contains your endpoint and then select the endpoint.
2. From the **Monitoring** section on the left of the page, select **Diagnostic settings** and then **Add settings**.
3. Select the log categories to enable, select **Send to Log Analytics workspace**, and then select the Log Analytics workspace to use. Finally, enter a **Diagnostic setting name** and select **Save**.

Diagnostic setting

X

Save Discard Delete Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name *

endpoint-logging

Logs

Category groups

 allLogs

Categories

 AmlOnlineEndpointConsoleLog AmlOnlineEndpointTrafficLog AmlOnlineEndpointEventLog

Metrics

 Traffic

Destination details

 Send to Log Analytics workspace

Subscription

ML-docs

Log Analytics workspace

myloganalytics-abcdef01-2345-6789-0abc-def0...

 Archive to a storage account Stream to an event hub Send to partner solution

IMPORTANT

It may take up to an hour for the connection to the Log Analytics workspace to be enabled. Wait an hour before continuing with the next steps.

4. Submit scoring requests to the endpoint. This activity should create entries in the logs.
5. From either the online endpoint properties or the Log Analytics workspace, select **Logs** from the left of the screen.
6. Close the **Queries** dialog that automatically opens, and then double-click the **AmlOnlineEndpointConsoleLog**. If you don't see it, use the **Search** field.

myendpoint (mymlworkspace/myendpoint) | Logs

New Query 1*

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Events

Settings

- Identity
- Properties
- Locks

Deployments

Deployments

Monitoring

- Alerts
- Metrics
- Diagnostic settings

Logs

AmlOnlineEndpointConsoleLog

Filter Group by: Resource type

Collapse all

Favorites

You can add favorites by clicking on the icon

Other

AmlOnlineEndpointConsoleLog

Queries History

No queries history

You haven't run any queries yet. To start, go to Queries on the side pane or type a query in the query editor.

7. Select Run.

myendpoint (mymlworkspace/myendpoint) | Logs

New Query 1*

Run

Time range : Last 24 hours

Save Share New alert rule Export

1 AmlOnlineEndpointConsoleLog

TimeGenerated [UTC]	InstanceId	DeploymentName	ContainerName
> 6/27/2022, 2:19:29.862 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:29.862 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:29.872 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:29.873 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:29.875 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:38.136 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:38.136 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:38.137 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:38.137 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:38.137 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:41.792 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:41.792 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:41.792 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:19:41.793 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:20:13.914 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:20:13.914 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:20:13.915 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:20:13.915 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server
> 6/27/2022, 2:20:13.915 PM	bf52b0ba5e2542169ee4ed71437a9d53000000	blue	inference-server

Example queries

You can find example queries on the **Queries** tab while viewing logs. Search for **Online endpoint** to find example queries.

New Query 1*

mymlworkspace/myend... Select scope <>

Tables Queries Functions ...

online

Filter Group by: Topic

Collapse all

Favorites

You can add favorites by clicking on the icon

Workloads

- Online endpoint console logs
- Online endpoint failed requests

Log column details

The following tables provide details on the data stored in each log:

AMLOnlineEndpointTrafficLog (preview)

FIELD NAME	DESCRIPTION
Method	The requested method from client.
Path	The requested path from client.
SubscriptionId	The machine learning subscription ID of the online endpoint.
WorkspaceId	The machine learning workspace ID of the online endpoint.
EndpointName	The name of the online endpoint.
DeploymentName	The name of the online deployment.
Protocol	The protocol of the request.
ResponseCode	The final response code returned to the client.
ResponseCodeReason	The final response code reason returned to the client.
ModelStatusCode	The response status code from model.
ModelStatusReason	The response status reason from model.
RequestPayloadSize	The total bytes received from the client.
ResponsePayloadSize	The total bytes sent back to the client.

FIELD NAME	DESCRIPTION
UserAgent	The user-agent header of the request.
XRequestId	The request ID generated by Azure Machine Learning for internal tracing.
XMSClientRequestId	The tracking ID generated by the client.
TotalDurationMs	Duration in milliseconds from the request start time to the last response byte sent back to the client. If the client disconnected, it measures from the start time to client disconnect time.
RequestDurationMs	Duration in milliseconds from the request start time to the last byte of the request received from the client.
ResponseDurationMs	Duration in milliseconds from the request start time to the first response byte read from the model.
RequestThrottlingDelayMs	Delay in milliseconds in request data transfer due to network throttling.
ResponseThrottlingDelayMs	Delay in milliseconds in response data transfer due to network throttling.

AMLOnlineEndpointConsoleLog

FIELD NAME	DESCRIPTION
TimeGenerated	The timestamp (UTC) of when the log was generated.
OperationName	The operation associated with log record.
Instanceld	The ID of the instance that generated this log record.
DeploymentName	The name of the deployment associated with the log record.
ContainerName	The name of the container where the log was generated.
Message	The content of the log.

AMLOnlineEndpointEventLog (preview)

FIELD NAME	DESCRIPTION
TimeGenerated	The timestamp (UTC) of when the log was generated.
OperationName	The operation associated with log record.
Instanceld	The ID of the instance that generated this log record.
DeploymentName	The name of the deployment associated with the log record.

FIELD NAME	DESCRIPTION
Name	The name of the event.
Message	The content of the event.

Next steps

- Learn how to [view costs for your deployed endpoint](#).
- Read more about [metrics explorer](#).

Debug online endpoints locally in Visual Studio Code (preview)

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#)

APPLIES TO: [Python SDK azure-ai-ml v2 \(preview\)](#)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Learn how to use the Visual Studio Code (VS Code) debugger to test and debug online endpoints locally before deploying them to Azure.

Azure Machine Learning local endpoints help you test and debug your scoring script, environment configuration, code configuration, and machine learning model locally.

Online endpoint local debugging

Debugging endpoints locally before deploying them to the cloud can help you catch errors in your code and configuration earlier. You have different options for debugging endpoints locally with VS Code.

- [Azure Machine Learning inference HTTP server \(Preview\)](#)
- Local endpoint

This guide focuses on local endpoints.

The following table provides an overview of scenarios to help you choose what works best for you.

SCENARIO	INFERENCE HTTP SERVER	LOCAL ENDPOINT
Update local Python environment, without Docker image rebuild	Yes	No
Update scoring script	Yes	Yes
Update deployment configurations (deployment, environment, code, model)	No	Yes
VS Code Debugger integration	Yes	Yes

Prerequisites

- [Azure CLI](#)
- [Python](#)

This guide assumes you have the following items installed locally on your PC.

- [Docker](#)
- [VS Code](#)
- [Azure CLI](#)
- [Azure CLI m1 extension \(v2\)](#)

For more information, see the guide on [how to prepare your system to deploy managed online endpoints](#).

The examples in this article are based on code samples contained in the [azureml-examples](#) repository. To run the commands locally without having to copy/paste YAML and other files, clone the repo and then change directories to the `cli` directory in the repo:

```
git clone https://github.com/Azure/azureml-examples --depth 1
cd azureml-examples
cd cli
```

If you haven't already set the defaults for the Azure CLI, save your default settings. To avoid passing in the values for your subscription, workspace, and resource group multiple times, use the following commands. Replace the following parameters with values for your specific configuration:

- Replace `<subscription>` with your Azure subscription ID.
- Replace `<workspace>` with your Azure Machine Learning workspace name.
- Replace `<resource-group>` with the Azure resource group that contains your workspace.
- Replace `<location>` with the Azure region that contains your workspace.

TIP

You can see what your current defaults are by using the `az configure -l` command.

```
az account set --subscription <subscription>
az configure --defaults workspace=<workspace> group=<resource-group> location=<location>
```

Launch development container

- [Azure CLI](#)
- [Python](#)

Azure Machine Learning local endpoints use Docker and VS Code development containers (dev container) to build and configure a local debugging environment. With dev containers, you can take advantage of VS Code features from inside a Docker container. For more information on dev containers, see [Create a development container](#).

To debug online endpoints locally in VS Code, use the `--vscode-debug` flag when creating or updating an Azure Machine Learning online deployment. The following command uses a deployment example from the examples repo:

```
az ml online-deployment create --file endpoints/online/managed/sample/blue-deployment.yml --local --vscode-debug
```

IMPORTANT

On Windows Subsystem for Linux (WSL), you'll need to update your PATH environment variable to include the path to the VS Code executable or use WSL interop. For more information, see [Windows interoperability with Linux](#).

A Docker image is built locally. Any environment configuration or model file errors are surfaced at this stage of the process.

NOTE

The first time you launch a new or updated dev container it can take several minutes.

Once the image successfully builds, your dev container opens in a VS Code window.

You'll use a few VS Code extensions to debug your deployments in the dev container. Azure Machine Learning automatically installs these extensions in your dev container.

- Inference Debug
- [Pylance](#)
- [Jupyter](#)
- [Python](#)

IMPORTANT

Before starting your debug session, make sure that the VS Code extensions have finished installing in your dev container.

Start debug session

Once your environment is set up, use the VS Code debugger to test and debug your deployment locally.

1. Open your scoring script in Visual Studio Code.

TIP

The score.py script used by the endpoint deployed earlier is located at `azureml-samples/cli/endpoints/online/managed/sample/score.py` in the repository you cloned. However, the steps in this guide work with any scoring script.

2. Set a breakpoint anywhere in your scoring script.

- To debug startup behavior, place your breakpoint(s) inside the `init` function.
- To debug scoring behavior, place your breakpoint(s) inside the `run` function.

3. Select the VS Code Job view.

4. In the Run and Debug dropdown, select **Azure ML: Debug Local Endpoint** to start debugging your endpoint locally.

In the **Breakpoints** section of the Run view, check that:

- **Raised Exceptions** is **unchecked**
- **Uncaught Exceptions** is **checked**

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** score.py - onlinescorer [Dev Container] - Visual Studio Code.
- Run and Debug Bar:** RUN AND DEBUG dropdown set to "Azure ML: Debug".
- Variables Panel:** Shows "Azure ML: Debug Local Endpoint" selected in the dropdown. Other options include "Node.js..." and "Add Configuration...".
- Breakpoints Panel:** Shows two breakpoints set on lines 15 and 25 of the "score.py" file.
- Code Editor:** Displays the "score.py" script with syntax highlighting. Breakpoints are marked with red dots on lines 15 and 25.
- Bottom Status Bar:** Dev Container, Python 3.6.1 64-bit ('inf-conda-env': conda), Ln 20, Col 1, Spaces: 4, UTF-8, LF, Python.

5. Select the play icon next to the Run and Debug dropdown to start your debugging session.

At this point, any breakpoints in your `init` function are caught. Use the debug actions to step through your code. For more information on debug actions, see the [debug actions guide](#).

For more information on the VS Code debugger, see [Debugging in VS Code](#)

Debug your endpoint

- [Azure CLI](#)
- [Python](#)

Now that your application is running in the debugger, try making a prediction to debug your scoring script.

Use the `ml` extension `invoke` command to make a request to your local endpoint.

```
az ml online-endpoint invoke --name <ENDPOINT-NAME> --request-file <REQUEST-FILE> --local
```

In this case, `<REQUEST-FILE>` is a JSON file that contains input data samples for the model to make predictions on similar to the following JSON:

```
{"data": [
    [1,2,3,4,5,6,7,8,9,10],
    [10,9,8,7,6,5,4,3,2,1]
]}
```

TIP

The scoring URI is the address where your endpoint listens for requests. Use the `ml` extension to get the scoring URI.

```
az ml online-endpoint show --name <ENDPOINT-NAME> --local
```

The output should look similar to the following:

```
{
  "auth_mode": "aml_token",
  "location": "local",
  "name": "my-new-endpoint",
  "properties": {},
  "provisioning_state": "Succeeded",
  "scoring_uri": "http://localhost:5001/score",
  "tags": {},
  "traffic": {},
  "type": "online"
}
```

The scoring URI can be found in the `scoring_uri` property.

At this point, any breakpoints in your `run` function are caught. Use the debug actions to step through your code. For more information on debug actions, see the [debug actions guide](#).

Edit your endpoint

- [Azure CLI](#)
- [Python](#)

As you debug and troubleshoot your application, there are scenarios where you need to update your scoring script and configurations.

To apply changes to your code:

1. Update your code
2. Restart your debug session using the `Developer: Reload Window` command in the command palette. For more information, see the [command palette documentation](#).

NOTE

Since the directory containing your code and endpoint assets is mounted onto the dev container, any changes you make in the dev container are synced with your local file system.

For more extensive changes involving updates to your environment and endpoint configuration, use the `ml` extension `update` command. Doing so will trigger a full image rebuild with your changes.

```
az ml online-deployment update --file <DEPLOYMENT-YAML-SPECIFICATION-FILE> --local --vscode-debug
```

Once the updated image is built and your development container launches, use the VS Code debugger to test and troubleshoot your updated endpoint.

Next steps

- [Deploy and score a machine learning model by using a managed online endpoint \(preview\)](#)

- Troubleshooting managed online endpoints deployment and scoring (preview)

Troubleshooting online endpoints deployment and scoring

9/22/2022 • 18 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Learn how to resolve common issues in the deployment and scoring of Azure Machine Learning online endpoints.

This document is structured in the way you should approach troubleshooting:

1. Use [local deployment](#) to test and debug your models locally before deploying in the cloud.
2. Use [container logs](#) to help debug issues.
3. Understand [common deployment errors](#) that might arise and how to fix them.

The section [HTTP status codes](#) explains how invocation and prediction errors map to HTTP status codes when scoring endpoints with REST requests.

Prerequisites

- An [Azure subscription](#). Try the [free or paid version of Azure Machine Learning](#).
- The [Azure CLI](#).
- For Azure Machine Learning CLI v2, see [Install, set up, and use the CLI \(v2\)](#).
- For Azure Machine Learning Python SDK v2, see [Install the Azure Machine Learning SDK v2 for Python](#).

Deploy locally

Local deployment is deploying a model to a local Docker environment. Local deployment is useful for testing and debugging before deployment to the cloud.

TIP

Use Visual Studio Code to test and debug your endpoints locally. For more information, see [debug online endpoints locally in Visual Studio Code](#).

Local deployment supports creation, update, and deletion of a local endpoint. It also allows you to invoke and get logs from the endpoint.

- [Azure CLI](#)
- [Python SDK](#)

To use local deployment, add `--local` to the appropriate CLI command:

```
az ml online-deployment create --endpoint-name <endpoint-name> -n <deployment-name> -f <spec_file.yaml> --local
```

As a part of local deployment the following steps take place:

- Docker either builds a new container image or pulls an existing image from the local Docker cache. An existing image is used if there's one that matches the environment part of the specification file.
- Docker starts a new container with mounted local artifacts such as model and code files.

For more, see [Deploy locally in Deploy and score a machine learning model with a managed online endpoint](#).

Conda installation

Generally, issues with mlflow deployment stem from issues with the installation of the user environment specified in the `conda.yaml` file.

To debug conda installation problems, try the following:

1. Check the logs for conda installation. If the container crashed or taking too long to start up, it is likely that conda environment update has failed to resolve correctly.
2. Install the mlflow conda file locally with the command
`conda env create -n userenv -f <CONDA_ENV_FILENAME>`
3. If there are errors locally, try resolving the conda environment and creating a functional one before redeploying.
4. If the container crashes even if it resolves locally, the SKU size used for deployment may be too small.
 - a. Conda package installation occurs at runtime, so if the SKU size is too small to accommodate all of the packages detailed in the `conda.yaml` environment file, then the container may crash.
 - b. A Standard_F4s_v2 VM is a good starting SKU size, but larger ones may be needed depending on which dependencies are specified in the conda file.

Get container logs

You can't get direct access to the VM where the model is deployed. However, you can get logs from some of the containers that are running on the VM. The amount of information depends on the provisioning status of the deployment. If the specified container is up and running you'll see its console output, otherwise you'll get a message to try again later.

- [Azure CLI](#)
- [Python SDK](#)

To see log output from container, use the following CLI command:

```
az ml online-deployment get-logs -e <endpoint-name> -n <deployment-name> -l 100
```

or

```
az ml online-deployment get-logs --endpoint-name <endpoint-name> --name <deployment-name> --lines 100
```

Add `--resource-group` and `--workspace-name` to the commands above if you have not already set these parameters via `az configure`.

To see information about how to set these parameters, and if current values are already set, run:

```
az ml online-deployment get-logs -h
```

By default the logs are pulled from the inference server. Logs include the console log from the inference server, which contains print/log statements from your `score.py` code.

NOTE

If you use Python logging, ensure you use the correct logging level order for the messages to be published to logs. For example, INFO.

You can also get logs from the storage initializer container by passing `--container storage-initializer`. These logs contain information on whether code and model data were successfully downloaded to the container.

Add `--help` and/or `--debug` to commands to see more information.

Request tracing

There are three supported tracing headers:

- `x-request-id` is reserved for server tracing. We override this header to ensure it's a valid GUID.

NOTE

When you create a support ticket for a failed request, attach the failed request ID to expedite investigation.

- `x-ms-client-request-id` is available for client tracing scenarios. We sanitize this header to remove non-alphanumeric symbols. This header is truncated to 72 characters.

Common deployment errors

Below is a list of common deployment errors that are reported as part of the deployment operation status.

- [ImageBuildFailure](#)
- [OutOfQuota](#)
- [OutOfCapacity](#)
- [BadArgument](#)
- [ResourceNotReady](#)
- [ResourceNotFound](#)
- [OperationCanceled](#)
- [InternalServerError](#)

ERROR: ImageBuildFailure

This error is returned when the environment (docker image) is being built. You can check the build log for more information on the failure(s). The build log is located in the default storage for your Azure Machine Learning workspace. The exact location is returned as part of the error. For example, 'The build log is available in the workspace blob store "storage-account-name" under the path "/azureml/ImageLogs/your-image-id/build.log"'. In this case, "azureml" is the name of the blob container in the storage account.

If no obvious error is found in the build log, and the last line is `Installing pip dependencies: ...working...`, then the error may be caused by a dependency. Pinning version dependencies in your conda file could fix this problem.

We also recommend using a [local deployment](#) to test and debug your models locally before deploying in the cloud.

ERROR: OutOfQuota

Below is a list of common resources that might run out of quota when using Azure services:

- [CPU](#)
- [Disk](#)
- [Memory](#)
- [Role assignments](#)
- [Endpoints](#)
- [Kubernetes](#)
- [Other](#)

CPU Quota

Before deploying a model, you need to have enough compute quota. This quota defines how much virtual cores are available per subscription, per workspace, per SKU, and per region. Each deployment subtracts from available quota and adds it back after deletion, based on type of the SKU.

A possible mitigation is to check if there are unused deployments that can be deleted. Or you can submit a [request for a quota increase](#).

Disk quota

This issue happens when the size of the model is larger than the available disk space and the model is not able to be downloaded. Try a SKU with more disk space.

- Try a [Managed online endpoints SKU list](#) with more disk space
- Try reducing image and model size

Memory quota

This issue happens when the memory footprint of the model is larger than the available memory. Try a [Managed online endpoints SKU list](#) with more memory.

Role assignment quota

Try to delete some unused role assignments in this subscription. You can check all role assignments in the Azure portal in the Access Control menu.

Endpoint quota

Try to delete some unused endpoints in this subscription.

Kubernetes quota

The requested CPU or memory couldn't be satisfied. Adjust your request or the cluster.

Other quota

To run the `score.py` provided as part of the deployment, Azure creates a container that includes all the resources that the `score.py` needs, and runs the scoring script on that container.

If your container could not start, this means scoring could not happen. It might be that the container is requesting more resources than what `instance_type` can support. If so, consider updating the `instance_type` of the online deployment.

To get the exact reason for an error, run:

- [Azure CLI](#)
- [Python SDK](#)

```
az ml online-deployment get-logs -e <endpoint-name> -n <deployment-name> -l 100
```

ERROR: OutOfCapacity

The specified VM Size failed to provision due to a lack of Azure Machine Learning capacity. Retry later or try deploying to a different region.

ERROR: BadArgument

Below is a list of reasons you might run into this error:

- Resource request was greater than limits
- Startup task failed due to authorization error
- Startup task failed due to incorrect role assignments on resource
- Unable to download user container image
- Unable to download user model
- `azureml-fe` for Kubernetes online endpoint is not ready

Resource requests greater than limits

Requests for resources must be less than or equal to limits. If you don't set limits, we set default values when you attach your compute to an Azure Machine Learning workspace. You can check limits in the Azure portal or by using the `az ml compute show` command.

Authorization error

After you provisioned the compute resource, during deployment creation, Azure tries to pull the user container image from the workspace private Azure Container Registry (ACR) and mount the user model and code artifacts into the user container from the workspace storage account.

First, check if there's a permissions issue accessing ACR.

To pull blobs, Azure uses [managed identities](#) to access the storage account.

- If you created the associated endpoint with SystemAssigned, Azure role-based access control (RBAC) permission is automatically granted, and no further permissions are needed.
- If you created the associated endpoint with UserAssigned, the user's managed identity must have Storage blob data reader permission on the workspace storage account.

Unable to download user container image

It's possible that the user container couldn't be found. Check [container logs](#) to get more details.

Make sure container image is available in workspace ACR.

For example, if image is `testacr.azurecr.io/azureml/azureml_92a029f831ce58d2ed011c3c42d35acb:latest` check the repository with

```
az acr repository show-tags -n testacr --repository azureml/azureml_92a029f831ce58d2ed011c3c42d35acb --orderby time_desc --output table
```

Unable to download user model

It is possible that the user model can't be found. Check [container logs](#) to get more details.

Make sure the model is registered to the same workspace as the deployment. Use the `show` command or equivalent Python method to show details for a model in a workspace.

- For example:

- [Azure CLI](#)
- [Python SDK](#)

```
az ml model show --name <model-name> --version <version>
```

WARNING

You must specify either version or label to get the model information.

You can also check if the blobs are present in the workspace storage account.

- For example, if the blob is

```
https://foobar.blob.core.windows.net/210212154504-1517266419/WebUpload/210212154504-1517266419/GaussianNB.pkl
```

, you can use this command to check if it exists:

```
az storage blob exists --account-name foobar --container-name 210212154504-1517266419 --name WebUpload/210212154504-1517266419/GaussianNB.pkl --subscription <sub-name>
```

- If the blob is present, you can use this command to obtain the logs from the storage initializer:

- [Azure CLI](#)
- [Python SDK](#)

```
az ml online-deployment get-logs --endpoint-name <endpoint-name> --name <deployment-name> --container storage-initializer
```

azureml-fe not ready

The front-end component (azureml-fe) that routes incoming inference requests to deployed services automatically scales as needed. It's installed during your k8s-extension installation.

This component should be healthy on cluster, at least one healthy replica. You will get this error message if it's not available when you trigger kubernetes online endpoint and deployment creation/update request.

Please check the pod status and logs to fix this issue, you can also try to update the k8s-extension installed on the cluster.

ERROR: ResourceNotReady

To run the `score.py` provided as part of the deployment, Azure creates a container that includes all the resources that the `score.py` needs, and runs the scoring script on that container. The error in this scenario is that this container is crashing when running, which means scoring can't happen. This error happens when:

- There's an error in `score.py`. Use `get-logs` to help diagnose common problems:
 - A package that was imported but isn't in the conda environment.
 - A syntax error.
 - A failure in the `init()` method.
- If `get-logs` isn't producing any logs, it usually means that the container has failed to start. To debug this issue, try [deploying locally](#) instead.
- Readiness or liveness probes aren't set up correctly.
- There's an error in the environment setup of the container, such as a missing dependency.
- When you face `TypeError: register() takes 3 positional arguments but 4 were given` error, the error may be caused by the dependency between flask v2 and `azureml-inference-server-http`. See [FAQs for inference HTTP server](#) for more details.

ERROR: ResourceNotFound

Below is a list of reasons you might run into this error:

- [Azure Resource Manager cannot find a required resource](#)
- [Azure Container Registry is private or otherwise inaccessible](#)

Resource Manager cannot find a resource

This error occurs when Azure Resource Manager can't find a required resource. For example, you'll receive this error if a storage account was referred to but can't be found at the path on which it was specified. Be sure to double check resources that might have been supplied by exact path or the spelling of their names.

For more information, see [Resolve Resource Not Found Errors](#).

Container registry authorization error

This error occurs when an image belonging to a private or otherwise inaccessible container registry was supplied for deployment. At this time, our APIs cannot accept private registry credentials.

To mitigate this error, either ensure that the container registry is **not private** or follow the following steps:

1. Grant your private registry's `acrPull` role to the system identity of your online endpoint.
2. In your environment definition, specify the address of your private image as well as the additional instruction to not modify (build) the image.

If the mitigation is successful, the image will not require any building and the final image address will simply be the given image address. At deployment time, your online endpoint's system identity will pull the image from the private registry.

For more diagnostic information, see [How To Use the Workspace Diagnostic API](#).

ERROR: OperationCanceled

Below is a list of reasons you might run into this error:

- [Operation was canceled by another operation that has a higher priority](#)
- [Operation was canceled due to a previous operation waiting for lock confirmation](#)

Operation canceled by another higher priority operation

Azure operations have a certain priority level and are executed from highest to lowest. This error happens when your operation happened to be overridden by another operation that has a higher priority.

Retrying the operation might allow it to be performed without cancellation.

Operation canceled waiting for lock confirmation

Azure operations have a brief waiting period after being submitted during which they retrieve a lock to ensure that we don't run into race conditions. This error happens when the operation you submitted is the same as another operation that is currently still waiting for confirmation that it has received the lock to proceed. It may indicate that you've submitted a very similar request too soon after the initial request.

Retrying the operation after waiting a few seconds up to a minute may allow it to be performed without cancellation.

ERROR: InternalServerError

Although we do our best to provide a stable and reliable service, sometimes things don't go according to plan. If you get this error, it means that something isn't right on our side, and we need to fix it. Submit a [customer support ticket](#) with all related information and we'll address the issue.

Autoscaling issues

If you're having trouble with autoscaling, see [Troubleshooting Azure autoscale](#).

Bandwidth limit issues

Managed online endpoints have bandwidth limits for each endpoint. You find the limit configuration in [Manage and increase quotas for resources with Azure Machine Learning](#) here. If your bandwidth usage exceeds the limit, your request will be delayed. To monitor the bandwidth delay:

- Use metric "Network bytes" to understand the current bandwidth usage. For more information, see [Monitor managed online endpoints](#).
- There are two response trailers will be returned if the bandwidth limit enforced:
 - `ms-azureml-bandwidth-request-delay-ms` : delay time in milliseconds it took for the request stream transfer.
 - `ms-azureml-bandwidth-response-delay-ms` : delay time in milliseconds it took for the response stream transfer.

HTTP status codes

When you access online endpoints with REST requests, the returned status codes adhere to the standards for [HTTP status codes](#). Below are details about how endpoint invocation and prediction errors map to HTTP status codes.

STATUS CODE	REASON PHRASE	WHY THIS CODE MIGHT GET RETURNED
200	OK	Your model executed successfully, within your latency bound.
401	Unauthorized	You don't have permission to do the requested action, such as score, or your token is expired.
404	Not found	Your URL isn't correct.
408	Request timeout	The model execution took longer than the timeout supplied in <code>request_timeout_ms</code> under <code>request_settings</code> of your model deployment config.
424	Model Error	If your model container returns a non-200 response, Azure returns a 424. Check the <code>Model Status Code</code> dimension under the <code>Requests Per Minute</code> metric on your endpoint's Azure Monitor Metric Explorer . Or check response headers <code>ms-azureml-model-error-statuscode</code> and <code>ms-azureml-model-error-reason</code> for more information.
429	Rate-limiting	The number of requests per second reached the limit of managed online endpoints.

STATUS CODE	REASON PHRASE	WHY THIS CODE MIGHT GET RETURNED
429	Too many pending requests	<p>Your model is getting more requests than it can handle. We allow <code>2 * max_concurrent_requests_per_instance * instance_count</code> requests in parallel at any time. Additional requests are rejected. You can confirm these settings in your model deployment config under <code>request_settings</code> and <code>scale_settings</code>. If you're using auto-scaling, your model is getting requests faster than the system can scale up. With auto-scaling, you can try to resend requests with exponential backoff. Doing so can give the system time to adjust.</p>
500	Internal server error	Azure ML-provisioned infrastructure is failing.

Common network isolation issues

Online endpoint creation fails with a V1LegacyMode == true message

The Azure Machine Learning workspace can be configured for `v1_legacy_mode`, which disables v2 APIs.

Managed online endpoints are a feature of the v2 API platform, and won't work if `v1_legacy_mode` is enabled for the workspace.

IMPORTANT

Check with your network security team before disabling `v1_legacy_mode`. It may have been enabled by your network security team for a reason.

For information on how to disable `v1_legacy_mode`, see [Network isolation with v2](#).

Online endpoint creation with key-based authentication fails

Use the following command to list the network rules of the Azure Key Vault for your workspace. Replace `<keyvault-name>` with the name of your key vault:

```
az keyvault network-rule list -n <keyvault-name>
```

The response for this command is similar to the following JSON document:

```
{
  "bypass": "AzureServices",
  "defaultAction": "Deny",
  "ipRules": [],
  "virtualNetworkRules": []
}
```

If the value of `bypass` isn't `AzureServices`, use the guidance in the [Configure key vault network settings](#) to set it to `AzureServices`.

Online deployments fail with an image download error

1. Check if the `egress-public-network-access` flag is **disabled** for the deployment. If this flag is enabled, and the visibility of the container registry is private, then this failure is expected.
2. Use the following command to check the status of the private endpoint connection. Replace `<registry-name>` with the name of the Azure Container Registry for your workspace:

```
az acr private-endpoint-connection list -r <registry-name> --query "[? privateLinkServiceConnectionState.description=='Egress for Microsoft.MachineLearningServices/workspaces/onlineEndpoints'].{Name:name, status:privateLinkServiceConnectionState.status}"
```

In the response document, verify that the `status` field is set to `Approved`. If it isn't approved, use the following command to approve it. Replace `<private-endpoint-name>` with the name returned from the previous command:

```
az network private-endpoint-connection approve -n <private-endpoint-name>
```

Scoring endpoint can't be resolved

1. Verify that the client issuing the scoring request is a virtual network that can access the Azure Machine Learning workspace.
2. Use the `nslookup` command on the endpoint hostname to retrieve the IP address information:

```
nslookup endpointname.westcentralus.inference.ml.azure.com
```

The response contains an **address**. This address should be in the range provided by the virtual network.

3. If the host name isn't resolved by the `nslookup` command, check if an A record exists in the private DNS zone for the virtual network. To check the records, use the following command:

```
az network private-dns record-set list -z privatelink.api.azureml.ms -o tsv --query [].name
```

The results should contain an entry that is similar to `*.<GUID>.inference.<region>`.

4. If no inference value is returned, delete the private endpoint for the workspace and then recreate it. For more information, see [How to configure a private endpoint](#).

Online deployments can't be scored

1. Use the following command to see if the deployment was successfully deployed:

```
az ml online-deployment show -e <endpointname> -n <deploymentname> --query '{name:name,state:provisioning_state}'
```

If the deployment completed successfully, the value of `state` will be `Succeeded`.

2. If the deployment was successful, use the following command to check that traffic is assigned to the deployment. Replace `<endpointname>` with the name of your endpoint:

```
az ml online-endpoint show -n <endpointname> --query traffic
```

TIP

This step isn't needed if you are using the `azureml-model-deployment` header in your request to target this deployment.

The response from this command should list percentage of traffic assigned to deployments.

3. If the traffic assignments (or deployment header) are set correctly, use the following command to get the logs for the endpoint. Replace `<endpointname>` with the name of the endpoint, and `<deploymentname>` with the deployment:

```
az ml online-deployment get-logs -e <endpointname> -n <deploymentname>
```

Look through the logs to see if there's a problem running the scoring code when you submit a request to the deployment.

Next steps

- [Deploy and score a machine learning model with a managed online endpoint](#)
- [Safe rollout for online endpoints](#)
- [Online endpoint YAML reference](#)

Use batch endpoints for batch scoring

9/22/2022 • 18 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

Learn how to use batch endpoints to do batch scoring. Batch endpoints simplify the process of hosting your models for batch scoring, so you can focus on machine learning, not infrastructure. For more information, see [What are Azure Machine Learning endpoints?](#).

In this article, you learn to do the following tasks:

- Create a batch endpoint and a default batch deployment
- Start a batch scoring job using Azure CLI
- Monitor batch scoring job execution progress and check scoring results
- Deploy a new MLflow model with auto generated code and environment to an existing endpoint without impacting the existing flow
- Test the new deployment and set it as the default deployment
- Delete the not in-use endpoint and deployment

Prerequisites

- You must have an Azure subscription to use Azure Machine Learning. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- Install the Azure CLI and the `ml` extension. Follow the installation steps in [Install, set up, and use the CLI \(v2\)](#).
- Create an Azure resource group if you don't have one, and you (or the service principal you use) must have `Contributor` permission. For resource group creation, see [Install, set up, and use the CLI \(v2\)](#).
- Create an Azure Machine Learning workspace if you don't have one. For workspace creation, see [Install, set up, and use the CLI \(v2\)](#).
- Configure your default workspace and resource group for the Azure CLI. Machine Learning CLI commands require the `--workspace/-w` and `--resource-group/-g` parameters. Configure the defaults can avoid passing in the values multiple times. You can override these on the command line. Run the following code to set up your defaults. For more information, see [Install, set up, and use the CLI \(v2\)](#).

```
az account set -s "<subscription ID>"  
az configure --defaults group="<resource group>" workspace="<workspace name>" location="<location>"
```

Clone the example repository

Run the following commands to clone the [AzureML Example repository](#) and go to the `cli` directory. This article uses the assets in `/cli/endpoints/batch`, and the end to end working example is `/cli/batch-score.sh`.

```
git clone https://github.com/Azure/azureml-examples  
cd azureml-examples/cli
```

Set your endpoint name. Replace `YOUR_ENDPOINT_NAME` with a unique name within an Azure region.

For Unix, run this command:

```
export ENDPOINT_NAME=<YOUR_ENDPOINT_NAME>"
```

For Windows, run this command:

```
set ENDPOINT_NAME=<YOUR_ENDPOINT_NAME>"
```

NOTE

Batch endpoint names need to be unique within an Azure region. For example, there can be only one batch endpoint with the name mybatchendpoint in westus2.

Create compute

Batch endpoint runs only on cloud computing resources, not locally. The cloud computing resource is a reusable virtual computer cluster. Run the following code to create an Azure Machine Learning compute cluster. The following examples in this article use the compute created here named `batch-cluster`. Adjust as needed and reference your compute using `azureml:<your-compute-name>`.

```
az ml compute create -n batch-cluster --type amlcompute --min-instances 0 --max-instances 5
```

NOTE

You are not charged for compute at this point as the cluster will remain at 0 nodes until a batch endpoint is invoked and a batch scoring job is submitted. Learn more about [manage and optimize cost for AmlCompute](#).

Understand batch endpoints and batch deployments

A batch endpoint is an HTTPS endpoint that clients can call to trigger a batch scoring job. A batch scoring job is a job that scores multiple inputs (for more, see [What are batch endpoints?](#)). A batch deployment is a set of compute resources hosting the model that does the actual batch scoring. One batch endpoint can have multiple batch deployments.

TIP

One of the batch deployments will serve as the default deployment for the endpoint. The default deployment will be used to do the actual batch scoring when the endpoint is invoked. Learn more about [batch endpoints and batch deployment](#).

The following YAML file defines a batch endpoint, which you can include in the CLI command for [batch endpoint creation](#). In the repository, this file is located at `/cli/endpoints/batch/batch-endpoint.yml`.

```
$schema: https://azuremlschemas.azureedge.net/latest/batchEndpoint.schema.json
name: mybatchedp
description: my sample batch endpoint
auth_mode: aad_token
```

The following table describes the key properties of the endpoint YAML. For the full batch endpoint YAML schema, see [CLI \(v2\) batch endpoint YAML schema](#).

KEY	DESCRIPTION
<code>\$schema</code>	[Optional] The YAML schema. You can view the schema in the above example in a browser to see all available options for a batch endpoint YAML file.
<code>name</code>	The name of the batch endpoint. Needs to be unique at the Azure region level.
<code>auth_mode</code>	The authentication method for the batch endpoint. Currently only Azure Active Directory token-based authentication (<code>aad_token</code>) is supported.
<code>defaults.deployment_name</code>	The name of the deployment that will serve as the default deployment for the endpoint.

To create a batch deployment, you need all the following items:

- Model files, or a registered model in your workspace referenced using `azureml:<model-name>:<model-version>`.
- The code to score the model.
- The environment in which the model runs. It can be a Docker image with Conda dependencies, or an environment already registered in your workspace referenced using `azureml:<environment-name>:<environment-version>`.
- The pre-created compute referenced using `azureml:<compute-name>` and resource settings.

For more information about how to reference an Azure ML entity, see [Referencing an Azure ML entity](#).

The example repository contains all the required files. The following YAML file defines a batch deployment with all the required inputs and optional settings. You can include this file in your CLI command to [create your batch deployment](#). In the repository, this file is located at `/cli/endpoints/batch/nonmlflow-deployment.yml`.

```
$schema: https://azurermschemas.azureedge.net/latest/batchDeployment.schema.json
name: nonmlflowdp
endpoint_name: mybatchedp
model:
  path: ./mnist/model/
code_configuration:
  code: ./mnist/code/
  scoring_script: digit_identification.py
environment:
  conda_file: ./mnist/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:latest
compute: azureml:batch-cluster
resources:
  instance_count: 1
  max_concurrency_per_instance: 2
  mini_batch_size: 10
  output_action: append_row
  output_file_name: predictions.csv
retry_settings:
  max_retries: 3
  timeout: 30
error_threshold: -1
logging_level: info
```

For the full batch deployment YAML schema, see [CLI \(v2\) batch deployment YAML schema](#).

KEY	DESCRIPTION
<code>\$schema</code>	[Optional] The YAML schema. You can view the schema in the above example in a browser to see all available options for a batch deployment YAML file.
<code>name</code>	The name of the deployment.
<code>endpoint_name</code>	The name of the endpoint to create the deployment under.
<code>model</code>	The model to be used for batch scoring. The example defines a model inline using <code>path</code> . Model files will be automatically uploaded and registered with an autogenerated name and version. Follow the Model schema for more options. As a best practice for production scenarios, you should create the model separately and reference it here. To reference an existing model, use the <code>azureml:<model-name>:<model-version></code> syntax.
<code>code_configuration.code.path</code>	The local directory that contains all the Python source code to score the model.
<code>code_configuration.scoring_script</code>	The Python file in the above directory. This file must have an <code>init()</code> function and a <code>run()</code> function. Use the <code>init()</code> function for any costly or common preparation (for example, load the model in memory). <code>init()</code> will be called only once at beginning of process. Use <code>run(mini_batch)</code> to score each entry; the value of <code>mini_batch</code> is a list of file paths. The <code>run()</code> function should return a pandas DataFrame or an array. Each returned element indicates one successful run of input element in the <code>mini_batch</code> . For more information on how to author scoring script, see Understanding the scoring script .
<code>environment</code>	The environment to score the model. The example defines an environment inline using <code>conda_file</code> and <code>image</code> . The <code>conda_file</code> dependencies will be installed on top of the <code>image</code> . The environment will be automatically registered with an autogenerated name and version. Follow the Environment schema for more options. As a best practice for production scenarios, you should create the environment separately and reference it here. To reference an existing environment, use the <code>azureml:<environment-name>:<environment-version></code> syntax.
<code>compute</code>	The compute to run batch scoring. The example uses the <code>batch-cluster</code> created at the beginning and reference it using <code>azureml:<compute-name></code> syntax.
<code>resources.instance_count</code>	The number of instances to be used for each batch scoring job.
<code>max_concurrency_per_instance</code>	[Optional] The maximum number of parallel <code>scoring_script</code> runs per instance.

KEY	DESCRIPTION
<code>mini_batch_size</code>	[Optional] The number of files the <code>scoring_script</code> can process in one <code>run()</code> call.
<code>output_action</code>	[Optional] How the output should be organized in the output file. <code>append_row</code> will merge all <code>run()</code> returned output results into one single file named <code>output_file_name</code> . <code>summary_only</code> won't merge the output results and only calculate <code>error_threshold</code> .
<code>output_file_name</code>	[Optional] The name of the batch scoring output file for <code>append_row</code> <code>output_action</code> .
<code>retry_settings.max_retries</code>	[Optional] The number of max tries for a failed <code>scoring_script</code> <code>run()</code> .
<code>retry_settings.timeout</code>	[Optional] The timeout in seconds for a <code>scoring_script</code> <code>run()</code> for scoring a mini batch.
<code>error_threshold</code>	[Optional] The number of input file scoring failures that should be ignored. If the error count for the entire input goes above this value, the batch scoring job will be terminated. The example uses <code>-1</code> , which indicates that any number of failures is allowed without terminating the batch scoring job.
<code>logging_level</code>	[Optional] Log verbosity. Values in increasing verbosity are: WARNING, INFO, and DEBUG.

Understanding the scoring script

As mentioned earlier, the `code_configuration.scoring_script` must contain two functions:

- `init()` : Use this function for any costly or common preparation. For example, use it to load the model into a global object. This function will be called once at the beginning of the process.
- `run(mini_batch)` : This function will be called for each `mini_batch` and do the actual scoring.
 - `mini_batch` : The `mini_batch` value is a list of file paths.
 - `response` : The `run()` method should return a pandas DataFrame or an array. Each returned output element indicates one successful run of an input element in the input `mini_batch`. Make sure that enough data is included in your `run()` response to correlate the input with the output. The resulting DataFrame or array is populated according to this scoring script. It's up to you how much or how little information you'd like to output to correlate output values with the input value, for example, the array can represent a list of tuples containing both the model's output and input. There's no requirement on the cardinality of the results. All elements in the result DataFrame or array will be written to the output file as-is (given that the `output_action` isn't `summary_only`).

The example uses `/cli/endpoints/batch/mnist/code/digit_identification.py`. The model is loaded in `init()` from `AZUREML_MODEL_DIR`, which is the path to the model folder created during deployment. `run(mini_batch)` iterates each file in `mini_batch`, does the actual model scoring and then returns output results.

Deploy with batch endpoints and run batch scoring

Now, let's deploy the model with batch endpoints and run batch scoring.

Create a batch endpoint

The simplest way to create a batch endpoint is to run the following code providing only a `--name`.

```
az ml batch-endpoint create --name $ENDPOINT_NAME
```

You can also create a batch endpoint using a YAML file. Add `--file` parameter in above command and specify the YAML file path.

Create a batch deployment

Run the following code to create a batch deployment named `nonmlflowdp` under the batch endpoint and set it as the default deployment.

```
az ml batch-deployment create --name nonmlflowdp --endpoint-name $ENDPOINT_NAME --file endpoints/batch/nonmlflow-deployment.yml --set-default
```

TIP

The `--set-default` parameter sets the newly created deployment as the default deployment of the endpoint. It's a convenient way to create a new default deployment of the endpoint, especially for the first deployment creation. As a best practice for production scenarios, you may want to create a new deployment without setting it as default, verify it, and update the default deployment later. For more information, see the [Deploy a new model](#) section.

Check batch endpoint and deployment details

Use `show` to check endpoint and deployment details.

To check a batch deployment, run the following code:

```
az ml batch-deployment show --name nonmlflowdp --endpoint-name $ENDPOINT_NAME
```

To check a batch endpoint, run the following code. As the newly created deployment is set as the default deployment, you should see `nonmlflowdp` in `defaults.deployment_name` from the response.

```
az ml batch-endpoint show --name $ENDPOINT_NAME
```

Invoke the batch endpoint to start a batch scoring job

Invoke a batch endpoint triggers a batch scoring job. A job `name` will be returned from the invoke response and can be used to track the batch scoring progress. The batch scoring job runs for a period of time. It splits the entire inputs into multiple `mini_batch` and processes in parallel on the compute cluster. One `scoring_script` `run()` takes one `mini_batch` and processes it by a process on an instance. The batch scoring job outputs will be stored in cloud storage, either in the workspace's default blob storage, or the storage you specified.

Invoke the batch endpoint with different input options

You can either use CLI or REST to `invoke` the endpoint. For REST experience, see [Use batch endpoints with REST](#)

There are several options to specify the data inputs in CLI `invoke`.

- **Option 1-1: Data in the cloud**

Use `--input` and `--input-type` to specify a file or folder on an Azure Machine Learning registered datastore or a publicly accessible path. When you're specifying a single file, use `--input-type uri_file`, and when you're specifying a folder, use `--input-type uri_folder`.

When the file or folder is on Azure ML registered datastore, the syntax for the URI is `azureml://datastores/<datastore-name>/paths/<path-on-datastore>/` for folder, and `azureml://datastores/<datastore-name>/paths/<path-on-datastore>/<file-name>` for a specific file. When the file or folder is on a publicly accessible path, the syntax for the URI is `https://<public-path>/` for folder, `https://<public-path>/<file-name>` for a specific file.

For more information about data URI, see [Azure Machine Learning data reference URI](#).

The example uses publicly available data in a folder from

`https://pipelinedata.blob.core.windows.net/sampleddata/mnist`, which contains thousands of hand-written digits. Name of the batch scoring job will be returned from the invoke response. Run the following code to invoke the batch endpoint using this data. `--query name` is added to only return the job name from the invoke response, and it will be used later to [Monitor batch scoring job execution progress](#) and [Check batch scoring results](#). Remove `--query name -o tsv` if you want to see the full invoke response. For more information on the `--query` parameter, see [Query Azure CLI command output](#).

```
JOB_NAME=$(az ml batch-endpoint invoke --name $ENDPOINT_NAME --input  
https://pipelinedata.blob.core.windows.net/sampleddata/mnist --input-type uri_folder --query name -o  
tsv)
```

- **Option 1-2: Registered data asset**

Use `--input` to pass in an Azure Machine Learning registered V2 data asset (with the type of either `uri_file` or `url_folder`). You don't need to specify `--input-type` in this option. The syntax for this option is `azureml:<dataset-name>:<dataset-version>`.

```
az ml batch-endpoint invoke --name $ENDPOINT_NAME --input azureml:<dataset-name>:<dataset-version>
```

- **Option 2: Data stored locally**

Use `--input` to pass in data files stored locally. You don't need to specify `--input-type` in this option. The data files will be automatically uploaded as a folder to Azure ML datastore, and passed to the batch scoring job.

```
az ml batch-endpoint invoke --name $ENDPOINT_NAME --input <local-path>
```

NOTE

- If you are using existing V1 FileDataset for batch endpoint, we recommend migrating them to V2 data assets and refer to them directly when invoking batch endpoints. Currently only data assets of type `uri_folder` or `uri_file` are supported. Batch endpoints created with GA CLIV2 (2.4.0 and newer) or GA REST API (2022-05-01 and newer) will not support V1 Dataset.
- You can also extract the URI or path on datastore extracted from V1 FileDataset by using `az ml dataset show` command with `--query` parameter and use that information for invoke.
- While Batch endpoints created with earlier APIs will continue to support V1 FileDataset, we will be adding further V2 data assets support with the latest API versions for even more usability and flexibility. For more information on V2 data assets, see [Work with data using SDK v2 \(preview\)](#). For more information on the new V2 experience, see [What is v2](#).

Configure the output location and overwrite settings

The batch scoring results are by default stored in the workspace's default blob store within a folder named by job name (a system-generated GUID). You can configure where to store the scoring outputs when you invoke

the batch endpoint. Use `--output-path` to configure any folder in an Azure Machine Learning registered datastore. The syntax for the `--output-path` is the same as `--input` when you're specifying a folder, that is, `azureml://datastores/<datastore-name>/paths/<path-on-datastore>/`. The prefix `folder:` isn't required anymore. Use `--set output_file_name=<your-file-name>` to configure a new output file name if you prefer having one output file containing all scoring results (specified `output_action=append_row` in your deployment YAML).

IMPORTANT

You must use a unique output location. If the output file exists, the batch scoring job will fail.

Some settings can be overwritten when invoke to make best use of the compute resources and to improve performance:

- Use `--instance-count` to overwrite `instance_count`. For example, for larger volume of data inputs, you may want to use more instances to speed up the end to end batch scoring.
- Use `--mini-batch-size` to overwrite `mini_batch_size`. The number of mini batches is decided by total input file counts and `mini_batch_size`. Smaller `mini_batch_size` generates more mini batches. Mini batches can be run in parallel, but there might be extra scheduling and invocation overhead.
- Use `--set` to overwrite other settings including `max_retries`, `timeout`, and `error_threshold`. These settings might impact the end to end batch scoring time for different workloads.

To specify the output location and overwrite settings when invoke, run the following code. The example stores the outputs in a folder with the same name as the endpoint in the workspace's default blob storage, and also uses a random file name to ensure the output location uniqueness. The code should work in Unix. Replace with your own unique folder and file name.

```
export OUTPUT_FILE_NAME=predictions_`echo $RANDOM`.csv
JOB_NAME=$(az ml batch-endpoint invoke --name $ENDPOINT_NAME --input
https://pipelinedata.blob.core.windows.net/sampledata/mnist --input-type uri_folder --output-path
azureml://datastores/workspaceblobstore/paths/$ENDPOINT_NAME --set output_file_name=$OUTPUT_FILE_NAME --
mini-batch-size 20 --instance-count 5 --query name -o tsv)
```

Monitor batch scoring job execution progress

Batch scoring jobs usually take some time to process the entire set of inputs.

You can use CLI `job show` to view the job. Run the following code to check job status from the previous endpoint invoke. To learn more about job commands, run `az ml job -h`.

```
STATUS=$(az ml job show -n $JOB_NAME --query status -o tsv)
echo $STATUS
if [[ $STATUS == "Completed" ]]
then
    echo "Job completed"
elif [[ $STATUS == "Failed" ]]
then
    echo "Job failed"
    exit 1
else
    echo "Job status not failed or completed"
    exit 2
fi
```

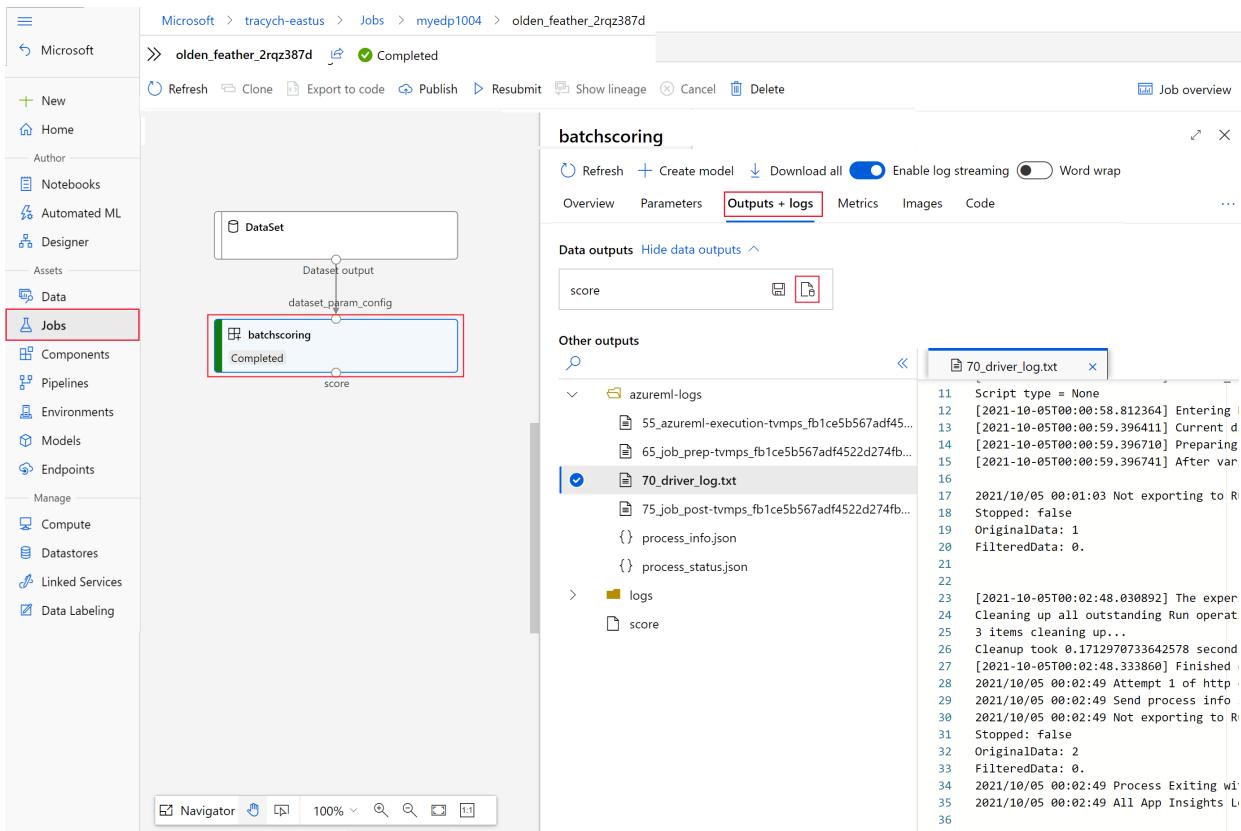
Check batch scoring results

Follow the below steps to view the scoring results in Azure Storage Explorer when the job is completed:

- Run the following code to open batch scoring job in Azure Machine Learning studio. The job studio link is also included in the response of `invoke`, as the value of `interactionEndpoints.Studio.endpoint`.

```
az ml job show -n $JOB_NAME --web
```

- In the graph of the job, select the `batchscoring` step.
- Select the **Outputs + logs** tab and then select **Show data outputs**.
- From **Data outputs**, select the icon to open **Storage Explorer**.



The scoring results in Storage Explorer are similar to the following sample page:

The screenshot shows the Azure Blob storage interface. On the left, the navigation sidebar is open, with the **Container** section highlighted. In the center, a blob named "predictions.csv" is listed under the "Overview" tab. The blob's content is displayed as a table with two columns: "Name" and "Score". The "Score" column contains numerical values ranging from 0 to 9. The table has 32 rows, starting with row 1 and ending with row 32. The last row shows a value of 5. The "Preview" button at the bottom right of the table can be used to view the data.

Name	Score
1	170.png: 4
2	171.png: 7
3	172.png: 2
4	173.png: 3
5	174.png: 2
6	175.png: 7
7	176.png: 1
8	177.png: 8
9	178.png: 1
10	179.png: 8
11	18.png: 3
12	180.png: 1
13	181.png: 8
14	182.png: 5
15	183.png: 0
16	184.png: 8
17	185.png: 9
18	186.png: 2
19	187.png: 5
20	188.png: 0
21	206.png: 9
22	207.png: 5
23	208.png: 2
24	209.png: 9
25	211.png: 6
26	210.png: 4
27	211.png: 5
28	212.png: 9
29	213.png: 3
30	214.png: 9
31	288.png: 1
32	289.png: 5
33	290.png: 5

Deploy a new model

Once you have a batch endpoint, you can continue to refine your model and add new deployments.

Create a new batch deployment hosting an MLflow model

To create a new batch deployment under the existing batch endpoint but not set it as the default deployment, run the following code:

```
az ml batch-deployment create --name mlflowdp --endpoint-name $ENDPOINT_NAME --file endpoints/batch/mlflow-deployment.yml
```

Notice that `--set-default` isn't used. If you `show` the batch endpoint again, you should see no change of the `defaults.deployment_name`.

The example uses a model (`/cli/endpoints/batch/autolog_nyc_taxi`) trained and tracked with MLflow. `scoring_script` and `environment` can be auto generated using model's metadata, no need to specify in the YAML file. For more about MLflow, see [Train and track ML models with MLflow and Azure Machine Learning](#).

Below is the YAML file the example uses to deploy an MLflow model, which only contains the minimum required properties. The source file in repository is `/cli/endpoints/batch/mlflow-deployment.yml`.

```
$schema: https://azurermschemas.azureedge.net/latest/batchDeployment.schema.json
name: mlflowdp
endpoint_name: mybatchedp
model:
  path: ./autolog_nyc_taxi
compute: azurerm1:batch-cluster
```

NOTE

`scoring_script` and `environment` auto generation only supports Python Function model flavor and column-based model signature.

Test a non-default batch deployment

To test the new non-default deployment, run the following code. The example uses a different model that accepts a publicly available csv file from

```
https://pipelinedata.blob.core.windows.net/sampleddata/nytaxi/taxi-tip-data.csv.
```

```

JOB_NAME=$(az ml batch-endpoint invoke --name $ENDPOINT_NAME --deployment-name mlflowdp --input
https://pipelinedata.blob.core.windows.net/sampleddata/nytaxi/taxi-tip-data.csv --input-type uri_file --query
name -o tsv)

echo "Show job detail"
az ml job show -n $JOB_NAME --web

echo "Stream job logs to console"
az ml job stream -n $JOB_NAME

STATUS=$(az ml job show -n $JOB_NAME --query status -o tsv)
echo $STATUS
if [[ $STATUS == "Completed" ]]
then
    echo "Job completed"
elif [[ $STATUS == "Failed" ]]
then
    echo "Job failed"
    exit 1
else
    echo "Job status not failed or completed"
    exit 2
fi

```

Notice `--deployment-name` is used to specify the new deployment name. This parameter allows you to `invoke` a non-default deployment, and it will not update the default deployment of the batch endpoint.

Update the default batch deployment

To update the default batch deployment of the endpoint, run the following code:

```
az ml batch-endpoint update --name $ENDPOINT_NAME --set defaults.deployment_name=mlflowdp
```

Now, if you `show` the batch endpoint again, you should see `defaults.deployment_name` is set to `mlflowdp`. You can `invoke` the batch endpoint directly without the `--deployment-name` parameter.

(Optional) Update the deployment

If you want to update the deployment (for example, update code, model, environment, or settings), update the YAML file, and then run `az ml batch-deployment update`. You can also update without the YAML file by using `--set`. Check `az ml batch-deployment update -h` for more information.

Delete the batch endpoint and the deployment

If you aren't going to use the old batch deployment, you should delete it by running the following code. `--yes` is used to confirm the deletion.

```
az ml batch-deployment delete --name nonmlflowdp --endpoint-name $ENDPOINT_NAME --yes
```

Run the following code to delete the batch endpoint and all the underlying deployments. Batch scoring jobs won't be deleted.

```
az ml batch-endpoint delete --name $ENDPOINT_NAME --yes
```

Next steps

- [Batch endpoints in studio](#)

- Deploy models with REST for batch scoring
- Troubleshooting batch endpoints

Use batch endpoints for batch scoring using Python SDK v2 (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Learn how to use batch endpoints to do batch scoring using Python SDK v2. Batch endpoints simplify the process of hosting your models for batch scoring, so you can focus on machine learning, not infrastructure. For more information, see [What are Azure Machine Learning endpoints?](#).

In this article, you'll learn to:

- Connect to your Azure machine learning workspace from the Python SDK v2.
- Create a batch endpoint from Python SDK v2.
- Create deployments on that endpoint from Python SDK v2.
- Test a deployment with a sample request.

Prerequisites

- A basic understanding of Machine Learning.
- An Azure account with an active subscription. [Create an account for free](#).
- An Azure ML workspace with computer cluster to run your batch scoring job.
- The [Azure Machine Learning SDK v2 for Python](#).

Clone examples repository

To run the examples, first clone the examples repository and change into the `sdk` directory:

```
git clone --depth 1 https://github.com/Azure/azureml-examples
cd azureml-examples/sdk
```

TIP

Use `--depth 1` to clone only the latest commit to the repository, which reduces time to complete the operation.

Connect to Azure Machine Learning workspace

The [workspace](#) is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. In this section, we'll connect to the workspace in which the job will be run.

1. Import the required libraries:

```

# import required libraries
from azure.ai.ml import MLClient, Input
from azure.ai.ml.entities import (
    AmlCompute,
    BatchEndpoint,
    BatchDeployment,
    Model,
    Environment,
    BatchRetrySettings,
)
from azure.ai.ml.entities._assets import Dataset
from azure.identity import DefaultAzureCredential
from azure.ai.ml.constants import BatchDeploymentOutputAction

```

2. Configure workspace details and get a handle to the workspace:

To connect to a workspace, we need identifier parameters - a subscription, resource group and workspace name. We'll use these details in the `MLClient` from `azure.ai.ml` to get a handle to the required Azure Machine Learning workspace. This example uses the [default Azure authentication](#).

```

# enter details of your AzureML workspace
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
workspace = "<AZUREML_WORKSPACE_NAME>"

```

```

# get a handle to the workspace
ml_client = MLClient(
    DefaultAzureCredential(), subscription_id, resource_group, workspace
)

```

Create batch endpoint

Batch endpoints are endpoints that are used for batch inferencing on large volumes of data over a period of time. Batch endpoints receive pointers to data and run jobs asynchronously to process the data in parallel on compute clusters. Batch endpoints store outputs to a data store for further analysis.

To create an online endpoint, we'll use `BatchEndpoint`. This class allows user to configure the following key aspects:

- `name` - Name of the endpoint. Needs to be unique at the Azure region level
- `auth_mode` - The authentication method for the endpoint. Currently only Azure Active Directory (Azure AD) token-based (`aad_token`) authentication is supported.
- `identity` - The managed identity configuration for accessing Azure resources for endpoint provisioning and inference.
- `defaults` - Default settings for the endpoint.
 - `deployment_name` - Name of the deployment that will serve as the default deployment for the endpoint.
- `description` - Description of the endpoint.

1. Configure the endpoint:

```

# Creating a unique endpoint name with current datetime to avoid conflicts
import datetime

batch_endpoint_name = "my-batch-endpoint-" + datetime.datetime.now().strftime(
    "%Y%m%d%H%M"
)

# create a batch endpoint
endpoint = BatchEndpoint(
    name=batch_endpoint_name,
    description="this is a sample batch endpoint",
    tags={"foo": "bar"},
)

```

2. Create the endpoint:

Using the `MLClient` created earlier, we'll now create the Endpoint in the workspace. This command will start the endpoint creation and return a confirmation response while the endpoint creation continues.

```
ml_client.begin_create_or_update(endpoint)
```

Create batch compute

Batch endpoint runs only on cloud computing resources, not locally. The cloud computing resource is a reusable virtual computer cluster. Run the following code to create an Azure Machine Learning compute cluster. The following examples in this article use the compute created here named `cpu-cluster`.

```

compute_name = "cpu-cluster"
compute_cluster = AmlCompute(name=compute_name, description="amlcompute", min_instances=0, max_instances=5)
ml_client.begin_create_or_update(compute_cluster)

```

Create a deployment

A deployment is a set of resources required for hosting the model that does the actual inferencing. We'll create a deployment for our endpoint using the `BatchDeployment` class. This class allows user to configure the following key aspects.

- `name` - Name of the deployment.
- `endpoint_name` - Name of the endpoint to create the deployment under.
- `model` - The model to use for the deployment. This value can be either a reference to an existing versioned model in the workspace or an inline model specification.
- `environment` - The environment to use for the deployment. This value can be either a reference to an existing versioned environment in the workspace or an inline environment specification.
- `code_path` - Path to the source code directory for scoring the model
- `scoring_script` - Relative path to the scoring file in the source code directory
- `compute` - Name of the compute target to execute the batch scoring jobs on
- `instance_count` - The number of nodes to use for each batch scoring job.
- `max_concurrency_per_instance` - The maximum number of parallel scoring_script runs per instance.
- `mini_batch_size` - The number of files the code_configuration.scoring_script can process in one `run()` call.
- `retry_settings` - Retry settings for scoring each mini batch.
 - `max_retries` - The maximum number of retries for a failed or timed-out mini batch (default is 3)
 - `timeout` - The timeout in seconds for scoring a mini batch (default is 30)

- `output_action` - Indicates how the output should be organized in the output file. Allowed values are `append_row` or `summary_only`. Default is `append_row`
- `output_file_name` - Name of the batch scoring output file. Default is `predictions.csv`
- `environment_variables` - Dictionary of environment variable name-value pairs to set for each batch scoring job.
- `logging_level` - The log verbosity level. Allowed values are `warning`, `info`, `debug`. Default is `info`.

1. Configure the deployment:

```
# create a batch deployment
model = Model(path="./mnist/model/")
env = Environment(
    conda_file="./mnist/environment/conda.yml",
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:latest",
)
deployment = BatchDeployment(
    name="non-mlflow-deployment",
    description="this is a sample non-mlflow deployment",
    endpoint_name=batch_endpoint_name,
    model=model,
    code_path="./mnist/code/",
    scoring_script="digit_identification.py",
    environment=env,
    compute=compute_name,
    instance_count=2,
    max_concurrency_per_instance=2,
    mini_batch_size=10,
    output_action=BatchDeploymentOutputAction.APPEND_ROW,
    output_file_name="predictions.csv",
    retry_settings=BatchRetrySettings(max_retries=3, timeout=30),
    logging_level="info",
)
```

2. Create the deployment:

Using the `MLClient` created earlier, we'll now create the deployment in the workspace. This command will start the deployment creation and return a confirmation response while the deployment creation continues.

```
ml_client.begin_create_or_update(deployment)
```

Test the endpoint with sample data

Using the `MLClient` created earlier, we'll get a handle to the endpoint. The endpoint can be invoked using the `invoke` command with the following parameters:

- `endpoint_name` - Name of the endpoint
- `input` - Path where input data is present
- `deployment_name` - Name of the specific deployment to test in an endpoint

1. Invoke the endpoint:

```
# create a dataset from the folderpath
input = Input(path="https://pipelinedata.blob.core.windows.net/sampleddata/mnist")

# invoke the endpoint for batch scoring job
job = ml_client.batch_endpoints.invoke(
    endpoint_name=batch_endpoint_name,
    input=input,
    deployment_name="non-mlflow-deployment", # name is required as default deployment is not set
    params_override=[{"mini_batch_size": "20"}, {"compute.instance_count": "4"}],
)
```

2. Get the details of the invoked job:

Let us get details and logs of the invoked job

```
# get the details of the job
job_name = job.name
batch_job = ml_client.jobs.get(name=job_name)
print(batch_job.status)
# stream the job logs
ml_client.jobs.stream(name=job_name)
```

Clean up resources

Delete endpoint

```
ml_client.batch_endpoints.begin_delete(name=batch_endpoint_name)
```

Delete compute: optional, as you may choose to reuse your compute cluster with later deployments.

```
ml_client.compute.begin_delete(name=compute_name)
```

Next steps

If you encounter problems using batch endpoints, see [Troubleshooting batch endpoints](#).

How to use batch endpoints in Azure Machine Learning studio

9/22/2022 • 4 minutes to read • [Edit Online](#)

In this article, you learn how to use batch endpoints to do batch scoring in [Azure Machine Learning studio](#). For more, see [What are Azure Machine Learning endpoints?](#).

In this article, you learn about:

- Create a batch endpoint with a no-code experience for MLflow model
- Check batch endpoint details
- Start a batch scoring job
- Overview of batch endpoint features in Azure machine learning studio

IMPORTANT

When working on a private link-enabled workspaces, batch endpoints can be created and managed using Azure Machine Learning studio. However, they can't be invoked from the UI. Please use the [Azure ML CLI v](#) instead for job creation. For more details about how to use it see [Invoke the batch endpoint to start a batch scoring job](#).

Prerequisites

- An Azure subscription - If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- The example repository - Clone the [AzureML Example repository](#). This article uses the assets in `/cli/endpoints/batch`.
- A compute target where you can run batch scoring workflows. For more information on creating a compute target, see [Create compute targets in Azure Machine Learning studio](#).
- Register machine learning model.

Create a batch endpoint

There are two ways to create Batch Endpoints in Azure Machine Learning studio:

- From the **Endpoints** page, select **Batch Endpoints** and then select **+ Create**.

Microsoft Azure Machine Learning

Home > Endpoints

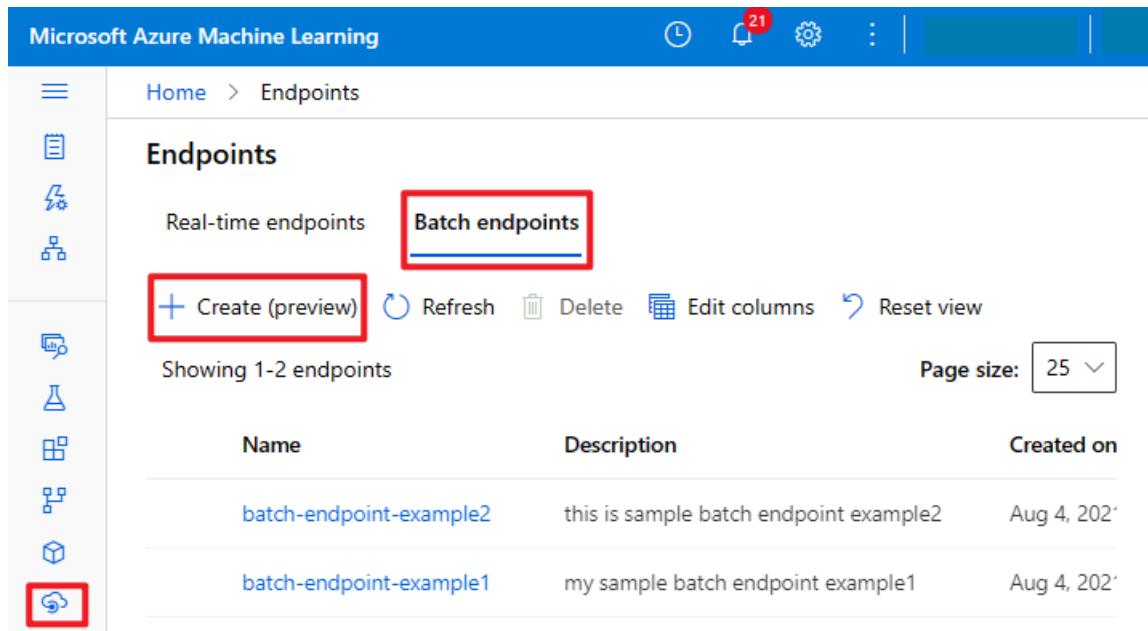
Endpoints

Real-time endpoints **Batch endpoints**

Create (preview) Refresh Delete Edit columns Reset view

Showing 1-2 endpoints Page size: 25

Name	Description	Created on
batch-endpoint-example2	this is sample batch endpoint example2	Aug 4, 202
batch-endpoint-example1	my sample batch endpoint example1	Aug 4, 202



OR

- From the **Models** page, select the model you want to deploy and then select **Deploy to batch endpoint**.

Microsoft Azure Machine Learning

Home > Models

Model List

Register model Refresh Delete Deploy ▾ Edit columns Reset view ...

Search

Showing 1-5 of 5 models

Deploy to real-time endpoint (preview)
Deploy the model using the new real-time endpoint wizard

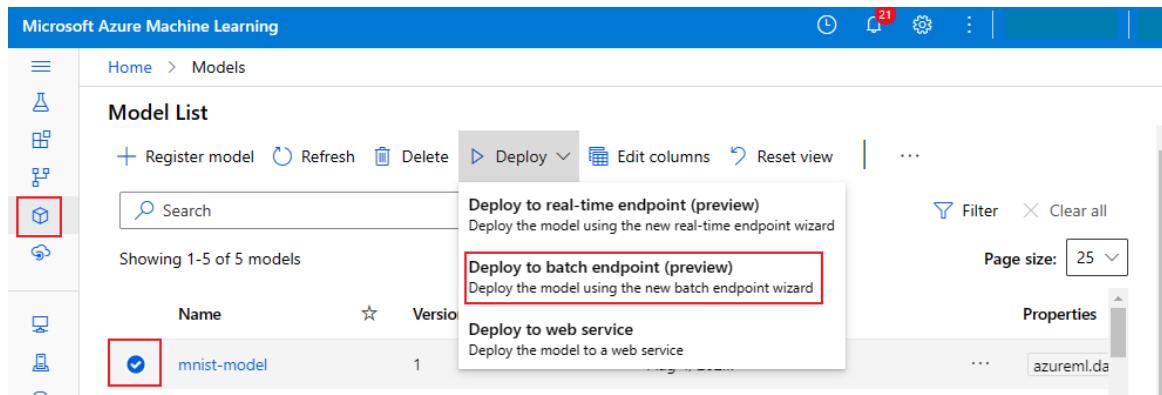
Deploy to batch endpoint (preview)
Deploy the model using the new batch endpoint wizard

Deploy to web service
Deploy the model to a web service

Page size: 25

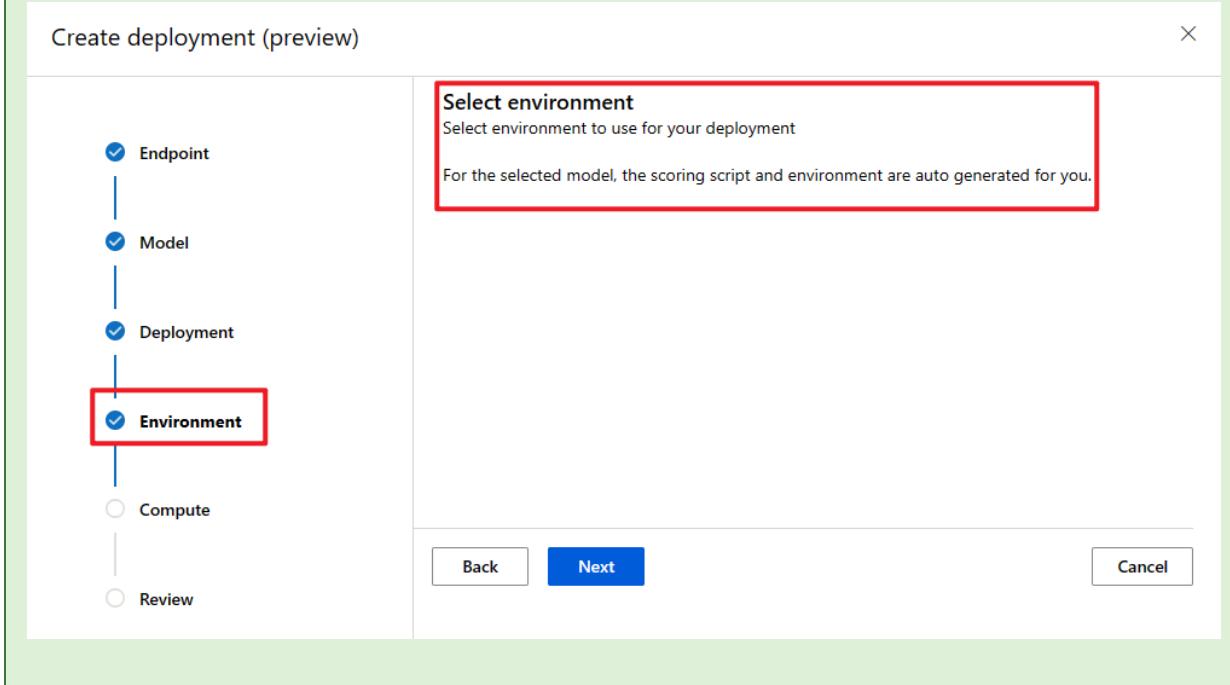
Properties

Name	Version
mnist-model	1

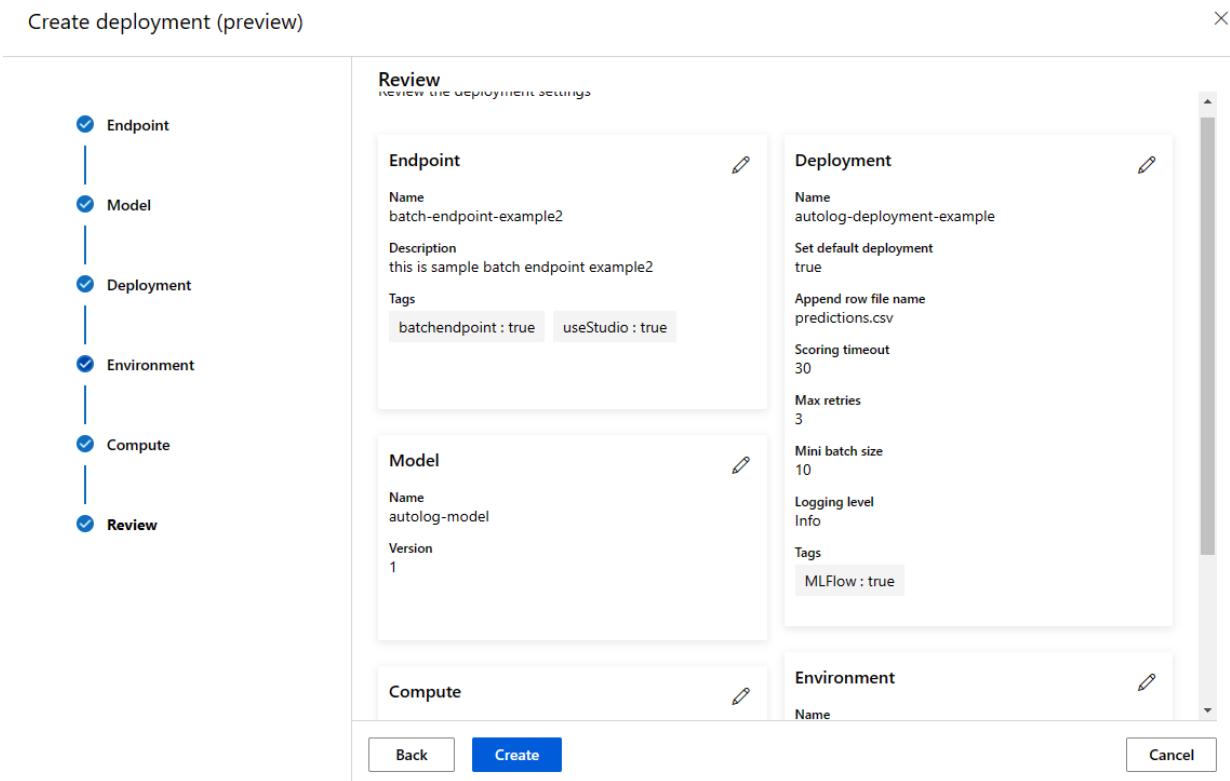


TIP

If you're using an MLflow model, you can use no-code batch endpoint creation. That is, you don't need to prepare a scoring script and environment, both can be auto generated. For more, see [Train and track ML models with MLflow and Azure Machine Learning](#).



Complete all the steps in the wizard to create a batch endpoint and deployment.



Check batch endpoint details

After a batch endpoint is created, select it from the **Endpoints** page to view the details.

Microsoft Azure Machine Learning

Home > Endpoints > batch-endpoint-example2

batch-endpoint-example2

Details Runs

+ Add deployment Refresh Update default deployment + Create job Delete

Attributes

Service ID
batch-endpoint-example2

Description
this is sample batch endpoint example2

Created by [redacted]

Created on
8/4/2021 2:49:42 PM

Last updated on
8/9/2021 1:46:48 PM

Authentication type
AADToken

REST endpoint
<https://batch-endpoint-example2>

Tags

batchendpoint true

useStudio true

Deployment summary

Deployments

mnist-deployment-example autolog-deployment-example Default

Deployment mnist-deployment-example

Name
mnist-deployment-example

Model ID
mnist-model:1

Compute
aml-compute

Environment
mnist-env:1

Instance count
1

Mini batch size
10

Error threshold
-1

Scoring timeout
30s

Max retries
3

Tags

No tags

Start a batch scoring job

A batch scoring workload runs as an offline job. By default, batch scoring stores the scoring outputs in blob storage. You can also configure the outputs location and overwrite some of the settings to get the best performance.

1. Select **+ Create job**:

batch-endpoint-example2

Details Runs

+ Add deployment Refresh Update default deployment + Create job Delete

Attributes

Service ID
batch-endpoint-example2

Description
this is sample batch endpoint example2

Deployment summary

Deployments

mnist-deployment-example autolog-deployment-example Default

2. You can update the default deployment while submitting a job from the drop-down:

Create a batch scoring job

X

The screenshot shows the 'Job settings' step of a wizard. On the left, a vertical navigation bar has three items: 'Job settings' (selected, indicated by a blue checkmark), 'Select data source', and 'Configure output location'. The main area is titled 'Job settings: Customize settings for the batch job'. It contains a 'Deployment' dropdown set to 'autolog-deployment-example', an 'Override deployment settings' toggle switch (unchecked), and three input fields: 'Output file name' (set to 'predictions.csv'), 'Mini batch size' (set to '10'), and 'Scoring timeout (seconds)' (set to '30'). At the bottom are 'Back', 'Next', and 'Cancel' buttons.

Overwrite settings

Some settings can be overwritten when you start a batch scoring job. For example, you might overwrite settings to make better use of the compute resource, or to improve performance. To override settings, select **Override deployment settings** and provide the settings. For more information, see [Use batch endpoints](#).

This screenshot is identical to the one above, but the 'Override deployment settings' toggle switch is now checked (indicated by a blue circle). A red box highlights this switch to draw attention to the change. The other settings remain the same: 'Output file name' is 'predictions.csv', 'Mini batch size' is '10', and 'Scoring timeout (seconds)' is '30'.

Start a batch scoring job with different input options

You have two options to specify the data inputs in Azure machine learning studio:

- Use a registered dataset:

NOTE

During Preview, only FileDataset is supported.

Create a batch scoring job

X

Job settings

Select data source

Configure output location

Select data source

Data source type *

Dataset Datastore

Data asset name	Dataset type
<input type="radio"/> taxi-tip-data	File
<input type="radio"/> sample-mnist dataset	File

Back Next Cancel

OR

- Use a **datastore**:

You can specify AzureML registered datastore or if your data is publicly available, specify the public path.

Create a batch scoring job

X

Job settings

Select data source

Configure output location

Select data source

Data source type *

Dataset Datastore

Datastore

workspaceblobstore

Path

Supported file types include: delimited (i.e. csv, tsv), Parquet, JSON Lines, and plain text.

https://pipelinedata.blob.core.windows.net/sampleddata/nytaxi/taxi-tip-data.csv

Browse

Back Next Cancel

Configure the output location

By default, the batch scoring results are stored in the default blob store for the workspace. Results are in a folder named after the job name (a system-generated GUID).

To change where the results are stored, providing a blob store and output path when you start a job.

IMPORTANT

You must use a unique output location. If the output file exists, the batch scoring job will fail.

Create a batch scoring job

The screenshot shows the 'Create a batch scoring job' wizard at the 'Configure output location' step. On the left, a vertical navigation bar lists three steps: 'Job settings' (selected), 'Select data source', and 'Configure output location' (highlighted in blue). The main area contains a section titled 'Configure output location' with the sub-instruction 'Optionally configure location settings for scoring output'. It includes a 'Blob datastore' dropdown labeled 'Select or search by name' and an 'Output path' input field. At the bottom are 'Back', 'Create' (highlighted in blue), and 'Cancel' buttons.

Summary of all submitted jobs

To see a summary of all the submitted jobs for an endpoint, select the endpoint and then select the **Runs** tab.

batch-endpoint-example2

The screenshot shows the 'Runs' tab for the endpoint 'batch-endpoint-example2'. The tab has options for 'Create job', 'Refresh', 'Edit columns', and 'Reset view'. Below are filters for 'Search', 'Filter', and 'Clear all', and a 'Page size' dropdown set to 25. It displays two running jobs:

Display name	Status	Created on
ivory_arm_rpf76lg8	Running	Jun 13, 2022 10:08 AM
gentle_net_86p4kyvq	Running	Jun 13, 2022 10:07 AM

Check batch scoring results

To learn how to view the scoring results, see [Use batch endpoints](#).

Add a deployment to an existing batch endpoint

In Azure machine learning studio, there are two ways to add a deployment to an existing batch endpoint:

- From the **Endpoints** page, select the batch endpoint to add a new deployment to. Select **+ Add deployment**, and complete the wizard to add a new deployment.

batch-endpoint-example2

Details Runs

+ Add deployment

Refresh

Update default deployment

+ Create job

...

Attributes

Service ID

batch-endpoint-example2

Description

this is sample batch endpoint example2

Created by

OR

- From the **Models** page, select the model you want to deploy. Then select **Deploy to batch endpoint** option from the drop-down. In the wizard, on the **Endpoint** screen, select **Existing**. Complete the wizard to add the new deployment.

Create deployment (preview)

X

Endpoint

Model

Deployment

Environment

Compute

Review

Create endpoint

An endpoint is used to deploy and score your models. [Learn more](#)

Endpoint

New Existing

Showing 1-2 endpoints

Name	Description	Created on
batch-endpoint-example2	this is sample batch endpoint example2	Aug 4, 2021 2...
batch-endpoint-example1	my sample batch endpoint example1	Aug 4, 2021 1...

Back

Next

Cancel

Update the default deployment

If an endpoint has multiple deployments, one of the deployments is the *default*. The default deployment receives 100% of the traffic to the endpoint. To change the default deployment, use the following steps:

- Select the endpoint from the **Endpoints** page.
- Select **Update default deployment**. From the **Details** tab, select the deployment you want to set as default and then select **Update**.

Microsoft Azure Machine Learning

Home > Endpoints > batch-endpoint-example2

batch-endpoint-example2

Details Runs

+ Add deployment Refresh Update default deployment +

Attributes

Service ID
batch-endpoint-example2

Update default deployment

Select which deployment will be the default deployment for batch jobs

Select default deployment *

autolog-deployment-example

autolog-deployment-example

mnist-deployment-example

Update Cancel

Delete batch endpoint and deployments

To delete an **endpoint**, select the endpoint from the **Endpoints** page and then select delete.

WARNING

Deleting an endpoint also deletes all deployments to that endpoint.

To delete a **deployment**, select the endpoint from the **Endpoints** page, select the deployment, and then select delete.

Next steps

In this article, you learned how to create and call batch endpoints. See these other articles to learn more about Azure Machine Learning:

- [Troubleshooting batch endpoints](#)
- [Deploy and score a machine learning model with a managed online endpoint](#)

Deploy models with REST for batch scoring

9/22/2022 • 11 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current)

Learn how to use the Azure Machine Learning REST API to deploy models for batch scoring.

The REST API uses standard HTTP verbs to create, retrieve, update, and delete resources. The REST API works with any language or tool that can make HTTP requests. REST's straightforward structure makes it a good choice in scripting environments and for MLOps automation.

In this article, you learn how to use the new REST APIs to:

- Create machine learning assets
- Create a batch endpoint and a batch deployment
- Invoke a batch endpoint to start a batch scoring job

Prerequisites

- An **Azure subscription** for which you have administrative rights. If you don't have such a subscription, try the [free or paid personal subscription](#).
- An [Azure Machine Learning workspace](#).
- A service principal in your workspace. Administrative REST requests use [service principal authentication](#).
- A service principal authentication token. Follow the steps in [Retrieve a service principal authentication token](#) to retrieve this token.
- The `curl` utility. The `curl` program is available in the [Windows Subsystem for Linux](#) or any UNIX distribution. In PowerShell, `curl` is an alias for `Invoke-WebRequest` and `curl -d "key=val" -X POST uri` becomes `Invoke-WebRequest -Body "key=val" -Method POST -Uri uri`.
- The `jq` JSON processor.

IMPORTANT

The code snippets in this article assume that you are using the Bash shell.

The code snippets are pulled from the `/cli/batch-score-rest.sh` file in the [AzureML Example repository](#).

Set endpoint name

NOTE

Batch endpoint names need to be unique at the Azure region level. For example, there can be only one batch endpoint with the name `mybatchendpoint` in `westus2`.

```
export ENDPOINT_NAME=endpt-`echo $RANDOM`
```

Azure Machine Learning batch endpoints

[Batch endpoints](#) simplify the process of hosting your models for batch scoring, so you can focus on machine learning, not infrastructure. In this article, you'll create a batch endpoint and deployment, and invoking it to start a batch scoring job. But first you'll have to register the assets needed for deployment, including model, code, and environment.

There are many ways to create an Azure Machine Learning batch endpoint, including [the Azure CLI](#), and visually with [the studio](#). The following example creates a batch endpoint and a batch deployment with the REST API.

Create machine learning assets

First, set up your Azure Machine Learning assets to configure your job.

In the following REST API calls, we use `SUBSCRIPTION_ID`, `RESOURCE_GROUP`, `LOCATION`, and `WORKSPACE` as

placeholders. Replace the placeholders with your own values.

Administrative REST requests a [service principal authentication token](#). Replace `TOKEN` with your own value. You can retrieve this token with the following command:

```
TOKEN=$(az account get-access-token --query accessToken -o tsv)
```

The service provider uses the `api-version` argument to ensure compatibility. The `api-version` argument varies from service to service. Set the API version as a variable to accommodate future versions:

```
API_VERSION="2022-05-01"
```

Create compute

Batch scoring runs only on cloud computing resources, not locally. The cloud computing resource is a reusable virtual computer cluster where you can run batch scoring workflows.

Create a compute cluster:

```
response=$(curl --location --request PUT  
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/computes/batch-cluster?api-version=$API_VERSION" \  
--header "Authorization: Bearer $TOKEN" \  
--header "Content-Type: application/json" \  
--data-raw "{  
    \"properties\":{  
        \"computeType\": \"AmlCompute\",  
        \"properties\": {  
            \"osType\": \"Linux\",  
            \"vmSize\": \"STANDARD_D2_V2\",  
            \"scaleSettings\": {  
                \"maxNodeCount\": 5,  
                \"minNodeCount\": 0  
            },  
            \"remoteLoginPortPublicAccess\": \"NotSpecified\"  
        },  
        \"location\": \"$LOCATION\"  
    }"}  
")
```

TIP

If you want to use an existing compute instead, you must specify the full Azure Resource Manager ID when [creating the batch deployment](#). The full ID uses the format

```
/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/compute-name>
```

.

Get storage account details

To register the model and code, first they need to be uploaded to a storage account. The details of the storage account are available in the data store. In this example, you get the default datastore and Azure Storage account for your workspace. Query your workspace with a GET request to get a JSON file with the information.

You can use the tool `jq` to parse the JSON result and get the required values. You can also use the Azure portal to find the same information:

```

# Get values for storage account
response=$(curl --location --request GET
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/datastores?api-version=$API_VERSION&isDefault=true" \
--header "Authorization: Bearer $TOKEN")
DATASTORE_PATH=$(echo $response | jq -r '.value[0].id')
BLOB_URI_PROTOCOL=$(echo $response | jq -r '.value[0].properties.protocol')
BLOB_URI_ENDPOINT=$(echo $response | jq -r '.value[0].properties.endpoint')
AZUREML_DEFAULT_DATASTORE=$(echo $response | jq -r '.value[0].name')
AZUREML_DEFAULT_CONTAINER=$(echo $response | jq -r '.value[0].properties.containerName')
AZURE_STORAGE_ACCOUNT=$(echo $response | jq -r '.value[0].properties.accountName')
export AZURE_STORAGE_ACCOUNT $($echo $AZURE_STORAGE_ACCOUNT)
STORAGE_RESPONSE=$(echo az storage account show-connection-string --name $AZURE_STORAGE_ACCOUNT)
AZURE_STORAGE_CONNECTION_STRING=$($STORAGE_RESPONSE | jq -r '.connectionString')

BLOB_URI_ROOT="$BLOB_URI_PROTOCOL://$AZURE_STORAGE_ACCOUNT.blob.$BLOB_URI_ENDPOINT/$AZUREML_DEFAULT_CONTAINER"

```

Upload & register code

Now that you have the datastore, you can upload the scoring script. For more information about how to author the scoring script, see [Understanding the scoring script](#). Use the Azure Storage CLI to upload a blob into your default container:

```
az storage blob upload-batch -d $AZUREML_DEFAULT_CONTAINER/score -s endpoints/batch/mnist/code/ --connection-string $AZURE_STORAGE_CONNECTION_STRING
```

TIP

You can also use other methods to upload, such as the Azure portal or [Azure Storage Explorer](#).

Once you upload your code, you can specify your code with a PUT request:

```

response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/codes/score-mnist/versions/1?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\": {
    \"description\": \"Score code\",
    \"codeUri\": \"$BLOB_URI_ROOT/score\"
  }
}")

```

Upload and register model

Similar to the code, upload the model files:

```
az storage blob upload-batch -d $AZUREML_DEFAULT_CONTAINER/model -s endpoints/batch/mnist/model --connection-string $AZURE_STORAGE_CONNECTION_STRING
```

Now, register the model:

```

response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/models/mnist/versions/1?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\": {
    \"modelUri\": \"azureml://subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/workspaces/$WORKSPACE/datastores/$AZUREML_DEFAULT_DATASTORE/paths/model\"
  }
}")

```

Create environment

The deployment needs to run in an environment that has the required dependencies. Create the environment with a PUT request. Use a docker image from Microsoft Container Registry. You can configure the docker image

with `image` and add conda dependencies with `condaFile`.

Run the following code to read the `condaFile` defined in json. The source file is at `/cli/endpoints/batch/mnist/environment/conda.json` in the example repository:

```
CONDA_FILE=$(cat endpoints/batch/mnist/environment/conda.json | sed 's/"\\\"/g')
```

Now, run the following snippet to create an environment:

```
ENV_VERSION=$RANDOM
response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/environments/mnist-env/versions/$ENV_VERSION?api-version=$API_VERSION" \
--header "Authorization: Bearer $TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
  \"properties\": {
    \"condaFile\": $(echo \"$CONDA_FILE\"), 
    \"image\": \"mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:latest\"
  }
}")
```

Deploy with batch endpoints

Next, create a batch endpoint, a batch deployment, and set the default deployment for the endpoint.

Create batch endpoint

Create the batch endpoint:

```
response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/batchEndpoints/$ENDPOINT_NAME?api-version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN" \
--data-raw "{
  \"properties\": {
    \"authMode\": \"aadToken\",
    \"location\": \"$LOCATION\"
  }
}")
```

Create batch deployment

Create a batch deployment under the endpoint:

```

DEPLOYMENT_NAME="nomlflowedp"
response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/batchEndpoints/$ENDPOINT_NAME/deployments/$DEPLOYMENT_NAME
?api-version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN" \
--data-raw "{"
    "\"location\": \"$LOCATION\",
    \"properties\": {
        \"model\": {
            \"referenceType\": \"Id\",
            \"assetId\": \"$/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/models/mnist/versions/1\"
        },
        \"codeConfiguration\": {
            \"codeId\": \"$/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/codes/score-mnist/versions/1\",
            \"scoringScript\": \"digit_identification.py\"
        },
        \"environmentId\": \"$/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/environments/mnist-env/versions/$ENV_VERSION\",
        \"compute\": \"$/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/computes/batch-cluster\",
        \"resources\": {
            \"instanceCount\": 1
        },
        \"maxConcurrencyPerInstance\": \"4\",
        \"retrySettings\": {
            \"maxRetries\": 3,
            \"timeout\": \"PT30S\"
        },
        \"errorThreshold\": \"10\",
        \"loggingLevel\": \"info\",
        \"miniBatchSize\": \"5\",
    }
}
")

```

Set the default batch deployment under the endpoint

There's only one default batch deployment under one endpoint, which will be used when invoke to run batch scoring job.

```

response=$(curl --location --request PUT
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/batchEndpoints/$ENDPOINT_NAME?api-version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN" \
--data-raw "{"
    "\"properties\": {
        \"authMode\": \"aadToken\",
        \"defaults\": {
            \"deploymentName\": \"$DEPLOYMENT_NAME\"
        }
    },
    \"location\": \"$LOCATION\"
}
")

operation_id=$(echo $response | jq -r '.properties' | jq -r '.properties' | jq -r '.AzureAsyncOperationUri')
wait_for_completion $operation_id $TOKEN

```

Run batch scoring

Invoking a batch endpoint triggers a batch scoring job. A job `id` is returned in the response, and can be used to track the batch scoring progress. In the following snippets, `jq` is used to get the job `id`.

Invoke the batch endpoint to start a batch scoring job

Getting the Scoring URI and access token

Get the scoring uri and access token to invoke the batch endpoint. First get the scoring uri:

```

response=$(curl --location --request GET
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/batchEndpoints/$ENDPOINT_NAME?api-version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN")

SCORING_URI=$(echo $response | jq -r '.properties.scoringUri')

```

Get the batch endpoint access token:

```
SCORING_TOKEN=$(az account get-access-token --resource https://ml.azure.com --query accessToken -o tsv)
```

Invoke the batch endpoint with different input options

It's time to invoke the batch endpoint to start a batch scoring job. If your data is a folder (potentially with multiple files) publicly available from the web, you can use the following snippet:

```

response=$(curl --location --request POST $SCORING_URI \
--header "Authorization: Bearer $SCORING_TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    \"properties\": {
        \"InputData\": {
            \"mnistinput\": {
                \"JobInputType\" : \"UriFolder\",
                \"Uri\":  \"https://pipelinedata.blob.core.windows.net/sampledata/mnist\"
            }
        }
    }
}")

JOB_ID=$(echo $response | jq -r '.id')
JOB_ID_SUFFIX=$(echo ${JOB_ID##*/})

```

Now, let's look at other options for invoking the batch endpoint. When it comes to input data, there are multiple scenarios you can choose from, depending on the input type (whether you are specifying a folder or a single file), and the URI type (whether you are using a path on Azure Machine Learning registered datastore, a reference to Azure Machine Learning registered V2 data asset, or a public URI).

- An `InputData` property has `JobInputType` and `Uri` keys. When you are specifying a single file, use `'JobInputType": "UriFile"`, and when you are specifying a folder, use `'JobInputType": "UriFolder"`.
- When the file or folder is on Azure ML registered datastore, the syntax for the `Uri` is
 - `azureml://datastores/<datastore-name>/paths/<path-on-datastore>` for folder, and
 - `azureml://datastores/<datastore-name>/paths/<path-on-datastore>/<file-name>` for a specific file. You can also use the longer form to represent the same path, such as
`azureml://subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/workspaces/<workspace-name>/datastores/<datastore-name>/paths/<path-on-datastore>/`
- When the file or folder is registered as V2 data asset as `uri_folder` or `uri_file`, the syntax for the `uri` is
 - `\"azureml://locations/<location-name>/workspaces/<workspace-name>/data/<data-name>/versions/<data-version>"`
 (Asset ID form)
 - `\"/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>/data/<data-name>/versions/<data-version>\\"`
 (ARM ID form)
- When the file or folder is a publicly accessible path, the syntax for the URI is `https://<public-path>` for folder, `https://<public-path>/<file-name>` for a specific file.

NOTE

For more information about data URI, see [Azure Machine Learning data reference URI](#).

Below are some examples using different types of input data.

- If your data is a folder on the Azure ML registered datastore, you can either:

- Use the short form to represent the URI:

```
response=$(curl --location --request POST $SCORING_URI \
--header "Authorization: Bearer $SCORING_TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    \"properties\": {
        \"InputData\": {
            \"mnistInput\": {
                \"JobInputType\" : \"UriFolder\",
                \"Uri\": \"azureml://datastores/workspaceblobstore/paths/$ENDPOINT_NAME/mnist\"
            }
        }
    }
}")

JOB_ID=$(echo $response | jq -r '.id')
JOB_ID_SUFFIX=$(echo ${JOB_ID##/*})
```

- Or use the long form for the same URL:

```
response=$(curl --location --request POST $SCORING_URI \
--header "Authorization: Bearer $SCORING_TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    \"properties\": {
        \"InputData\": {
            \"mnistinput\": {
                \"JobInputType\" : \"UriFolder\",
                \"Uri\":
                \"azureml://subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/workspaces/$WORKSPACE/data-
stores/workspaceblobstore/paths/$ENDPOINT_NAME/mnist\"
            }
        }
    }
}")

JOB_ID=$(echo $response | jq -r '.id')
JOB_ID_SUFFIX=$(echo ${JOB_ID##/*})
```

- If you want to manage your data as Azure ML registered V2 data asset as `uri_folder`, you can follow the two steps below:

1. Create the V2 data asset:

```
DATA_NAME="mnist"
DATA_VERSION=$RANDOM

response=$(curl --location --request PUT
https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/
Microsoft.MachineLearningServices/workspaces/$WORKSPACE/data/$DATA_NAME/versions/$DATA_VERSION?api-
version=$API_VERSION \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN" \
--data-raw "{
    \"properties\": {
        \"dataType\": \"uri_folder\",
        \"dataUri\": \"https://pipelinedata.blob.core.windows.net/sampleddata/mnist\",
        \"description\": \"Mnist data asset\"
    }
}")
```

2. Reference the data asset in the batch scoring job:

```

response=$(curl --location --request POST $SCORING_URI \
--header "Authorization: Bearer $SCORING_TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    \"properties\": {
        \"InputData\": {
            \"mnistInput\": {
                \"JobInputType\" : \"UriFolder\",
                \"Uri\": \"$azureml://locations/$LOCATION_NAME/workspaces/$WORKSPACE_NAME/data/$DATA_NAME/versions/$DATA_VERSION/\"
            }
        }
    }
}")

JOB_ID=$(echo $response | jq -r '.id')
JOB_ID_SUFFIX=$(echo ${JOB_ID##*/})

```

- If your data is a single file publicly available from the web, you can use the following snippet:

```

response=$(curl --location --request POST $SCORING_URI \
--header "Authorization: Bearer $SCORING_TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    \"properties\": {
        \"InputData\": {
            \"mnistInput\": {
                \"JobInputType\" : \"UriFile\",
                \"Uri\": \"https://pipelinedata.blob.core.windows.net/sampleddata/mnist/0.png\"
            }
        }
    }
}")

JOB_ID=$(echo $response | jq -r '.id')
JOB_ID_SUFFIX=$(echo ${JOB_ID##*/})

```

NOTE

We strongly recommend using the latest REST API version for batch scoring.

- If you want to use local data, you can upload it to Azure Machine Learning registered datastore and use REST API for Cloud data.
- If you are using existing V1 FileDataset for batch endpoint, we recommend migrating them to V2 data assets and refer to them directly when invoking batch endpoints. Currently only data assets of type `uri_folder` or `uri_file` are supported. Batch endpoints created with GA CLIV2 (2.4.0 and newer) or GA REST API (2022-05-01 and newer) will not support V1 Dataset.
- You can also extract the URI or path on datastore extracted from V1 FileDataset by using `az ml dataset show` command with `--query` parameter and use that information for invoke.
- While Batch endpoints created with earlier APIs will continue to support V1 FileDataset, we will be adding further V2 data assets support with the latest API versions for even more usability and flexibility. For more information on V2 data assets, see [Work with data using SDK v2 \(preview\)](#). For more information on the new V2 experience, see [What is v2.](#)

Configure the output location and overwrite settings

The batch scoring results are by default stored in the workspace's default blob store within a folder named by job name (a system-generated GUID). You can configure where to store the scoring outputs when you invoke the batch endpoint. Use `OutputData` to configure the output file path on an Azure Machine Learning registered datastore. `OutputData` has `JobOutputType` and `Uri` keys. `UriFile` is the only supported value for `JobOutputType`. The syntax for `Uri` is the same as that of `InputData`, i.e., `azureml://datastores/<datastore-name>/paths/<path-on-datastore>/<file-name>`.

Following is the example snippet for configuring the output location for the batch scoring results.

```

response=$(curl --location --request POST $SCORING_URI \
--header "Authorization: Bearer $SCORING_TOKEN" \
--header "Content-Type: application/json" \
--data-raw "{
    \"properties\": {
        \"InputData\": {
            \"mnistInput\": {
                \"JobInputType\" : \"UriFolder\",
                \"Uri\": \"azureml://datastores/workspaceblobstore/paths/$ENDPOINT_NAME/mnist\"
            }
        },
        \"OutputData\": {
            \"mnistOutput\": {
                \"JobOutputType\": \"UriFile\",
                \"Uri\": \"$azurerm://datastores/workspaceblobstore/paths/$ENDPOINT_NAME/mnistOutput/$OUTPUT_FILE_NAME\"
            }
        }
    }
}")

JOB_ID=$(echo $response | jq -r '.id')
JOB_ID_SUFFIX=$(echo ${JOB_ID##*/})

```

IMPORTANT

You must use a unique output location. If the output file exists, the batch scoring job will fail.

Check the batch scoring job

Batch scoring jobs usually take some time to process the entire set of inputs. Monitor the job status and check the results after it's completed:

TIP

The example invokes the default deployment of the batch endpoint. To invoke a non-default deployment, use the `azurerm-model-deployment` HTTP header and set the value to the deployment name. For example, using a parameter of `--header "azurerm-model-deployment: $DEPLOYMENT_NAME"` with curl.

```
wait_for_completion $SCORING_URI/$JOB_ID_SUFFIX $SCORING_TOKEN
```

Check batch scoring results

For information on checking the results, see [Check batch scoring results](#).

Delete the batch endpoint

If you aren't going use the batch endpoint, you should delete it with the below command (it deletes the batch endpoint and all the underlying deployments):

```

curl --location --request DELETE
"https://management.azure.com/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP/providers/Microsoft.MachineLearningServices/workspaces/$WORKSPACE/batchEndpoints/$ENDPOINT_NAME?api-version=$API_VERSION" \
--header "Content-Type: application/json" \
--header "Authorization: Bearer $TOKEN" || true

```

Next steps

- Learn how to deploy your model for batch scoring [using the Azure CLI](#).
- Learn how to deploy your model for batch scoring [using studio](#).
- Learn to [Troubleshoot batch endpoints](#)

Troubleshooting batch endpoints

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

Learn how to troubleshoot and solve, or work around, common errors you may come across when using [batch endpoints](#) for batch scoring.

The following table contains common problems and solutions you may see during batch endpoint development and consumption.

PROBLEM	POSSIBLE SOLUTION
Code configuration or Environment is missing.	Ensure you provide the scoring script and an environment definition if you're using a non-MLflow model. No-code deployment is supported for the MLflow model only. For more, see Track ML models with MLflow and Azure Machine Learning
Unsupported input data.	Batch endpoint accepts input data in three forms: 1) registered data 2) data in the cloud 3) data in local. Ensure you're using the right format. For more, see Use batch endpoints for batch scoring
Output already exists.	If you configure your own output location, ensure you provide a new output for each endpoint invocation.

Understanding logs of a batch scoring job

Get logs

After you invoke a batch endpoint using the Azure CLI or REST, the batch scoring job will run asynchronously. There are two options to get the logs for a batch scoring job.

Option 1: Stream logs to local console

You can run the following command to stream system-generated logs to your console. Only logs in the `azureml-logs` folder will be streamed.

```
az ml job stream -name <job_name>
```

Option 2: View logs in studio

To get the link to the run in studio, run:

```
az ml job show --name <job_name> --query interaction_endpoints.Studio.endpoint -o tsv
```

1. Open the job in studio using the value returned by the above command.
2. Choose **batchscoring**
3. Open the **Outputs + logs** tab
4. Choose the log(s) you wish to review

Understand log structure

There are two top-level log folders, `azureml-logs` and `logs`.

The file `~/azureml-logs/70_driver_log.txt` contains information from the controller that launches the scoring script.

Because of the distributed nature of batch scoring jobs, there are logs from several different sources. However, two combined files are created that provide high-level information:

- `~/logs/job_progress_overview.txt`: This file provides high-level information about the number of mini-batches (also known as tasks) created so far and the number of mini-batches processed so far. As the mini-batches end, the log records the results of the job. If the job failed, it will show the error message and where to start the troubleshooting.
- `~/logs/sys/master_role.txt`: This file provides the principal node (also known as the orchestrator) view of the running job. This log provides information on task creation, progress monitoring, the job result.

For a concise understanding of errors in your script there is:

- `~/logs/user/error.txt`: This file will try to summarize the errors in your script.

For more information on errors in your script, there is:

- `~/logs/user/error/`: This file contains full stack traces of exceptions thrown while loading and running the entry script.

When you need a full understanding of how each node executed the score script, look at the individual process logs for each node. The process logs can be found in the `sys/node` folder, grouped by worker nodes:

- `~/logs/sys/node/<ip_address>/<process_name>.txt`: This file provides detailed info about each mini-batch as it's picked up or completed by a worker. For each mini-batch, this file includes:
 - The IP address and the PID of the worker process.
 - The total number of items, the number of successfully processed items, and the number of failed items.
 - The start time, duration, process time, and run method time.

You can also view the results of periodic checks of the resource usage for each node. The log files and setup files are in this folder:

- `~/logs/perf`: Set `--resource_monitor_interval` to change the checking interval in seconds. The default interval is `600`, which is approximately 10 minutes. To stop the monitoring, set the value to `0`. Each `<ip_address>` folder includes:
 - `os/`: Information about all running processes in the node. One check runs an operating system command and saves the result to a file. On Linux, the command is `ps`.
 - `%Y%m%d%H`: The sub folder name is the time to hour.
 - `processes_%M`: The file ends with the minute of the checking time.
 - `node_disk_usage.csv`: Detailed disk usage of the node.
 - `node_resource_usage.csv`: Resource usage overview of the node.
 - `processes_resource_usage.csv`: Resource usage overview of each process.

How to log in scoring script

You can use Python logging in your scoring script. Logs are stored in

`logs/user/stdout/<node_id>/processNNN.stdout.txt`.

```
import argparse
import logging

# Get logging_level
arg_parser = argparse.ArgumentParser(description="Argument parser.")
arg_parser.add_argument("--logging_level", type=str, help="logging level")
args, unknown_args = arg_parser.parse_known_args()
print(args.logging_level)

# Initialize Python logger
logger = logging.getLogger(__name__)
logger.setLevel(args.logging_level.upper())
logger.info("Info log statement")
logger.debug("Debug log statement")
```

Deploy MLflow models

9/22/2022 • 13 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

In this article, learn how to deploy your [MLflow](#) model to Azure ML for both real-time and batch inference. Azure ML supports no-code deployment of models created and logged with MLflow. This means that you don't have to provide a scoring script or an environment.

For no-code-deployment, Azure Machine Learning

- Dynamically installs Python packages provided in the `conda.yaml` file, this means the dependencies are installed during container runtime.
- Provides a MLflow base image/curated environment that contains the following items:
 - `azureml-inference-server-http`
 - `mlflow-skinny`
 - The scoring script baked into the image.

IMPORTANT

If you are used to deploying models using scoring scripts and custom environments and you are looking to know how to achieve the same functionality using MLflow models, we recommend reading [Using MLflow models for no-code deployment](#).

NOTE

For information about inputs format and limitation in online endpoints, view [Considerations when deploying to real-time inference](#). For more information about the supported file types in batch endpoints, view [Considerations when deploying to batch inference](#).

Deployment tools

There are three workflows for deploying MLflow models to Azure Machine Learning:

- [Deploy using the MLflow plugin](#)
- [Deploy using Azure ML CLI/SDK \(v2\)](#)
- [Deploy using Azure Machine Learning studio](#)

Each workflow has different capabilities, particularly around which type of compute they can target. The following table shows them:

SCENARIO	MLFLOW SDK	AZURE ML CLI/SDK V2	AZURE ML STUDIO
Deploy MLflow models to managed online endpoints	✓	✓	✓
Deploy MLflow models to managed batch endpoints		✓	✓

SCENARIO	MLFLOW SDK	AZURE ML CLI/SDK V2	AZURE ML STUDIO
Deploy MLflow models to ACI/AKS	✓		
Deploy MLflow models to ACI/AKS (with a scoring script)			✓ ¹

NOTE

- 1 No-code deployment is not supported when deploying to ACI/AKS from Azure ML studio. We recommend switching to our [managed online endpoints](#) instead.

Which option to use?

If you are familiar with MLflow or your platform support MLflow natively (like Azure Databricks) and you wish to continue using the same set of methods, use the `azureml-mlflow` plugin. On the other hand, if you are more familiar with the [Azure ML CLI v2](#), you want to automate deployments using automation pipelines, or you want to keep deployments configuration in a git repository; we recommend you to use the [Azure ML CLI v2](#). If you want to quickly deploy and test models trained with MLflow, you can use [Azure Machine Learning studio](#) UI deployment.

Deploy using the MLflow plugin

The MLflow plugin `azureml-mlflow` can deploy models to Azure ML, either to Azure Kubernetes Service (AKS), Azure Container Instances (ACI) and Managed Endpoints for real-time serving.

WARNING

Deploying to managed batch endpoints is not supported in the MLflow plugin at the moment.

Prerequisites

- Install the `azureml-mlflow` package.
- If you are running outside an Azure ML compute, configure the MLflow tracking URI or MLflow's registry URI to point to the workspace you are working on. For more information about how to Set up tracking environment, see [Track runs using MLflow with Azure Machine Learning](#) for more details.

Steps

1. Ensure your model is registered in Azure Machine Learning registry. Deployment of unregistered models is not supported in Azure Machine Learning. You can register a new model using the MLflow SDK:

- [From a training job](#)
- [From a local model](#)

```
mlflow.register_model(f"runs:{run_id}/{artifact_path}", "sample-sklearn-mlflow-model")
```

2. Deployments can be generated using both the Python SDK for MLflow or MLflow CLI. In both cases, a JSON configuration file can be indicated with the details of the deployment you want to achieve. If not indicated, then a default deployment is done using Azure Container Instances (ACI) and a minimal configuration.

- [Managed endpoints](#)
- [AKS](#)
- [ACI](#)

```
{
    "instance_type": "Standard_DS2_v2",
    "instance_count": 1,
}
```

NOTE

The full specification of this configuration can be found at [Managed online deployment schema \(v2\)](#).

3. Save the deployment configuration to a file:

- [Managed endpoints](#)
- [AKS](#)
- [ACI](#)

```
import json

deploy_config = {
    "instance_type": "Standard_DS2_v2",
    "instance_count": 1,
}

deployment_config_path = "deployment_config.json"
with open(deployment_config_path, "w") as outfile:
    outfile.write(json.dumps(deploy_config))
```

4. Create a deployment client using the Azure Machine Learning Tracking URI.

```
from mlflow.deployments import get_deploy_client

# Set the tracking uri in the deployment client.
client = get_deploy_client("<azureml-mlflow-tracking-url>")
```

5. Run the deployment

```
model_name = "mymodel"
model_version = 1

# define the model path and the name is the service name
# if model is not registered, it gets registered automatically and a name is autogenerated using the
"name" parameter below
client.create_deployment(
    model_uri=f"models:/{{model_name}}/{{model_version}}",
    config={ "deploy-config-file": deployment_config_path },
    name="mymodel-deployment",
)
```

Deploy using Azure ML CLI/SDK (v2)

You can use Azure ML CLI/SDK v2 to deploy models trained and logged with MLflow to [managed endpoints](#)

([Online/batch](#)). Deployment of MLflow models support no-code-deployment, so you don't have to provide a scoring script or an environment, but you can if needed.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- The [Azure CLI](#) and the `m1` extension to the Azure CLI. For more information, see [Install, set up, and use the CLI \(v2\)](#).

IMPORTANT

The CLI examples in this article assume that you are using the Bash (or compatible) shell. For example, from a Linux system or [Windows Subsystem for Linux](#).

- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Install, set up, and use the CLI \(v2\)](#) to create one.
- You must have a MLflow model. If your model is not in MLflow format and you want to use this feature, you can [convert your custom ML model to MLflow format](#).

Steps

This example shows how you can deploy an MLflow model to an online endpoint using CLI (v2).

IMPORTANT

For MLflow no-code-deployment, [testing via local endpoints](#) is currently not supported.

1. Connect to Azure Machine Learning workspace

- [Azure ML CLI \(v2\)](#)
- [Azure ML SDK for Python \(v2\)](#)

```
az account set --subscription <subscription>
az configure --defaults workspace=<workspace> group=<resource-group> location=<location>
```

2. The following example configures the name and authentication mode of the endpoint:

- [Azure ML CLI \(v2\)](#)
- [Azure ML SDK for Python \(v2\)](#)

Create a YAML configuration file for your endpoint:

`create-endpoint.yaml`

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: my-endpoint
auth_mode: key
```

3. Execute the endpoint creation. This operation will create the endpoint in the Azure Machine Learning workspace:

- [Azure ML CLI \(v2\)](#)
- [Azure ML SDK for Python \(v2\)](#)

To create a new endpoint using the YAML configuration, use the following command:

```
az ml online-endpoint create --name $ENDPOINT_NAME -f endpoints/online/mlflow/create-endpoint.yaml
```

4. Before going further, we need to register the model we want to deploy. Deployment of unregistered models is not supported in Azure Machine Learning.

- [Azure ML CLI \(v2\)](#)
- [Azure ML SDK for Python \(v2\)](#)

We first need to register the model we want to deploy. Deployment of unregistered models is not supported in Azure Machine Learning.

From a training job

In this example, the model is registered from a job previously run. Assuming that your model was registered with an instruction similar like this:

```
mlflow.sklearn.log_model(scikit_model, "model")
```

To register the model from a previous run we would need the job name/run ID in question. For simplicity, let's assume that we are looking to register the model trained in the last run submitted to the workspace:

```
JOB_NAME=$(az ml job list --query "[0].name" | tr -d '')
```

Then, let's register the model in the registry.

```
az ml model create --name "mir-sample-sklearn-mlflow-model" \
    --type "mlflow_model" \
    --path "azureml://jobs/$JOB_NAME/outputs/artifacts/model"
```

From a local model

If your model is located in the local file system or compute, then you can register it as follows:

```
az ml model create --name "mir-sample-sklearn-mlflow-model" \
    --type "mlflow_model" \
    --path "sklearn-diabetes/model"
```

5. Once the endpoint is created, we need to create a deployment on it. Remember that endpoints can contain one or multiple deployments and traffic can be configured for each of them. In this example, we are going to create only one deployment to serve all the traffic, named `sklearn-deployment`.

- [Azure ML CLI \(v2\)](#)
- [Azure ML SDK for Python \(v2\)](#)

Create the deployment `YAML` file:

`sklearn-deployment.yaml`

```
$schema: https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: sklearn-deployment
endpoint_name: my-endpoint
model: azureml:mir-sample-sklearn-mlflow-model@latest
instance_type: Standard_DS2_v2
instance_count: 1
```

6. Create the deployment and assign all the traffic to it.

- [Azure ML CLI \(v2\)](#)
- [Azure ML SDK for Python \(v2\)](#)

```
az ml online-deployment create --name sklearn-deployment --endpoint $ENDPOINT_NAME -f
endpoints/online/mlflow/sklearn-deployment.yaml --all-traffic
```

7. Once the deployment is completed, the service is ready to receive requests. If you are not sure about how to submit requests to the service, see [Creating requests](#).

Deploy using Azure Machine Learning studio

You can use [Azure Machine Learning studio](#) to deploy models to Managed Online Endpoints.

IMPORTANT

Although deploying to ACI or AKS with [Azure Machine Learning studio](#) is possible, no-code deployment feature is not available for these compute targets. We recommend the use of [managed online endpoints](#) as it provides a superior set of features.

1. Ensure your model is registered in the Azure Machine Learning registry. Deployment of unregistered models is not supported in Azure Machine Learning. You can register models from files in the local file system or from the output of a job:

- [From a training job](#)
- [From a local model](#)

You can register the model directly from the job's output using Azure Machine Learning studio. To do so, navigate to the **Outputs + logs** tab in the run where your model was trained and select the option **Create model**.

Properties

- Status: Completed
- Created on: Jun 13, 2022 12:36 PM
- Start time: Jun 13, 2022 12:36 PM
- Duration: 2m 50.16s
- Compute duration: 2m 50.16s
- Compute target: None
- Name: 936a3c79-c655-48e9-8a0a-c3b70132e60d
- Script name: --
- Created by: Facundo Santiago
- Experiment: heart-condition-classifier
- Arguments: None
- Registered models: None

Outputs

- Output name: mlflow_log_model
- Model: azureml_936a3c79-c655-48e9-8a0a-c3b70132e60d_output_mlflow_log_model:1

Tags

- mlflow.source.name : /anaconda/envs/carpicer/lib/python3.8/site-packages/ipykernel_launcher.py
- mlflow.source.type : LOCAL mlflow.user : Facundo Santiago

Params

- base_score : None booster : None colsample_bylevel : None
- colsample_bynode : None colsample_bytree : None custom_metric : None
- early_stopping_rounds : None eval_metric : logloss gamma : None
- gpu_id : None grow_policy : None interaction_constraints : None
- learning_rate : None max_bin : None max_cat_to_onehot : None
- max_delta_step : None max_depth : None max_leaves : None
- maximize : None min_child_weight : None monotone_constraints : None
- n_jobs : None num_boost_round : 100 num_parallel_tree : None
- objective : binary:logistic predictor : None random_state : None

2. From **studio**, select your workspace and then use either the **endpoints** page to create the endpoint deployment:

a. From the **Endpoints** page, Select **+ Create**.

Name	Star	Description
triton-deployment-test		
custom-container-2 (deleting...)		

b. Provide a name and authentication type for the endpoint, and then select **Next**.

c. When selecting a model, select the MLflow model registered previously. Select **Next** to continue.

d. When you select a model registered in MLflow format, in the Environment step of the wizard, you don't need a scoring script or an environment.

e. Complete the wizard to deploy the model to the endpoint.

Considerations when deploying to real time inference

The following input's types are supported in Azure ML when deploying models with no-code deployment. Take a look at *Notes* in the bottom of the table for additional considerations.

INPUT TYPE	SUPPORT IN MLFLOW MODELS (SERVE)	SUPPORT IN AZURE ML
JSON-serialized pandas DataFrames in the split orientation	✓	✓
JSON-serialized pandas DataFrames in the records orientation	✓	1
CSV-serialized pandas DataFrames	✓	2

INPUT TYPE	SUPPORT IN MLFLOW MODELS (SERVE)	SUPPORT IN AZURE ML
Tensor input format as JSON-serialized lists (tensors) and dictionary of lists (named tensors)		✓
Tensor input formatted as in TF Serving's API	✓	

NOTE

- ¹ We suggest you to use split orientation instead. Records orientation doesn't guarantee column ordering preservation.
- ² We suggest you to explore batch inference for processing files.

Regardless of the input type used, Azure Machine Learning requires inputs to be provided in a JSON payload, within a dictionary key `input_data`. Note that such key is not required when serving models using the command `mlflow models serve` and hence payloads can't be used interchangeably.

Creating requests

Your inputs should be submitted inside a JSON payload containing a dictionary with key `input_data`.

Payload example for a JSON-serialized pandas DataFrames in the split orientation

```
{
  "input_data": {
    "columns": [
      "age", "sex", "trestbps", "chol", "fbs", "restecg", "thalach", "exang", "oldpeak", "slope",
      "ca", "thal"
    ],
    "index": [1],
    "data": [
      [1, 1, 145, 233, 1, 2, 150, 0, 2.3, 3, 0, 2]
    ]
  }
}
```

Payload example for a tensor input

```
{
  "input_data": [
    [1, 1, 0, 233, 1, 2, 150, 0, 2.3, 3, 0, 2],
    [1, 1, 0, 233, 1, 2, 150, 0, 2.3, 3, 0, 2],
    [1, 1, 0, 233, 1, 2, 150, 0, 2.3, 3, 0, 2],
    [1, 1, 145, 233, 1, 2, 150, 0, 2.3, 3, 0, 2]
  ]
}
```

Payload example for a named-tensor input

```
{
  "input_data": {
    "tokens": [
      [0, 655, 85, 5, 23, 84, 23, 52, 856, 5, 23, 1]
    ],
    "mask": [
      [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
    ]
  }
}
```

Limitations

The following limitations apply to real time inference deployments:

NOTE

Consider the following limitations when deploying MLflow models to Azure Machine Learning:

- Spark flavor is not supported at the moment for deployment.
- Data type `mlflow.types.DataType.Binary` is not supported as column type in signatures. For models that work with images, we suggest you to use or (a) tensors inputs using the [TensorSpec input type](#), or (b) `Base64` encoding schemes with a `mlflow.types.DataType.String` column type, which is commonly used when there is a need to encode binary data that needs be stored and transferred over media.
- Signatures with tensors with unspecified shapes `(-1)` is only supported at the batch size by the moment. For instance, a signature with shape `(-1, -1, -1, 3)` is not supported but `(-1, 300, 300, 3)` is.

Considerations when deploying to batch inference

Azure Machine Learning supports no-code deployment for batch inference in [managed endpoints](#). This represents a convenient way to deploy models that require processing of big amounts of data in a batch-fashion.

How work is distributed on workers

Work is distributed at the file level, for both structured and unstructured data. As a consequence, only [file datasets](#) or [URI folders](#) are supported for this feature. Each worker processes batches of `Mini batch size` files at a time. Further parallelism can be achieved if `Max concurrency per instance` is increased.

WARNING

Nested folder structures are not explored during inference. If you are partitioning your data using folders, make sure to flatten the structure beforehand.

File's types support

The following data types are supported for batch inference.

FILE EXTENSION	TYPE RETURNED AS MODEL'S INPUT	SIGNATURE REQUIREMENT
<code>.csv</code>	<code>pd.DataFrame</code>	<code>ColSpec</code> . If not provided, columns typing is not enforced.
<code>.png</code> , <code>.jpg</code> , <code>.jpeg</code> , <code>.tiff</code> , <code>.bmp</code> , <code>.gif</code>	<code>np.ndarray</code>	<code>TensorSpec</code> . Input is reshaped to match tensors shape if available. If no signature is available, tensors of type <code>np.uint8</code> are inferred.

FILE EXTENSION	TYPE RETURNED AS MODEL'S INPUT	SIGNATURE REQUIREMENT
----------------	--------------------------------	-----------------------

Next steps

To learn more, review these articles:

- [Deploy models with REST \(preview\)](#)
- [Create and use online endpoints in the studio](#)
- [Safe rollout for online endpoints](#)
- [How to autoscale managed online endpoints](#)
- [Use batch endpoints for batch scoring](#)
- [View costs for an Azure Machine Learning managed online endpoint \(preview\)](#)
- [Access Azure resources with an online endpoint and managed identity \(preview\)](#)
- [Troubleshoot online endpoint deployment](#)

Logging MLflow models

9/22/2022 • 10 minutes to read • [Edit Online](#)

The following article explains how to start logging your trained models (or artifacts) as MLflow models. It explores the different methods to customize the way MLflow packages your models and hence how it runs them.

Why logging models instead of artifacts?

If you are not familiar with MLflow, you may not be aware of the difference between logging artifacts or files vs. logging MLflow models. We recommend reading the article [From artifacts to models in MLflow](#) for an introduction to the topic.

A model in MLflow is also an artifact, but with a specific structure that serves as a contract between the person that created the model and the person that intends to use it. Such contract helps build the bridge about the artifacts themselves and what they mean.

Logging models has the following advantages:

- You don't need to provide a scoring script nor an environment for deployment.
- Swagger is enabled in endpoints automatically and the **Test** feature can be used in Azure ML studio.
- Models can be used as pipelines inputs directly.
- You can use the Responsible AI dashboard.

There are different ways to start using the model's concept in Azure Machine Learning with MLflow, as explained in the following sections:

Logging models using autolog

One of the simplest ways to start using this approach is by using MLflow autolog functionality. Autolog allows MLflow to instruct the framework associated to with the framework you are using to log all the metrics, parameters, artifacts and models that the framework considers relevant. By default, most models will be log if autolog is enabled. Some flavors may decide not to do that in specific situations. For instance, the flavor PySpark won't log models if they exceed a certain size.

You can turn on autologging by using either `mlflow.autolog()` or `mlflow.<flavor>.autolog()`. The following example uses `autolog()` for logging a classifier model trained with XGBoost:

```
import mlflow
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

mlflow.autolog()

model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

TIP

If you are using Machine Learning pipelines, like for instance [Scikit-Learn pipelines](#), use the `autolog` functionality of that flavor for logging models. Models are automatically logged when the `fit()` method is called on the pipeline object. The notebook [Training and tracking an XGBoost classifier with MLflow](#) demonstrates how to log a model with preprocessing using pipelines.

Logging models with a custom signature, environment or samples

You can log models manually using the method `mlflow.<flavor>.log_model` in MLflow. Such workflow has the advantages of retaining control of different aspects of how the model is logged.

Use this method when:

- You want to indicate pip packages or a conda environment different from the ones that are automatically detected.
- You want to include input examples.
- You want to include specific artifacts into the package that will be needed.
- Your signature is not correctly inferred by `autolog`. This is specifically important when you deal with inputs that are tensors where the signature needs specific shapes.
- Somehow the default behavior of autolog doesn't fill your purpose.

The following example code logs a model for an XGBoost classifier:

```
import mlflow
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from mlflow.models import infer_signature
from mlflow.utils.environment import _mlflow_conda_env

mlflow.autolog(log_models=False)

model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

# Signature
signature = infer_signature(X_test, y_test)

# Conda environment
custom_env = _mlflow_conda_env(
    additional_conda_deps=None,
    additional_pip_deps=["xgboost==1.5.2"],
    additional_conda_channels=None,
)

# Sample
input_example = X_train.sample(n=1)

# Log the model manually
mlflow.xgboost.log_model(model,
                        artifact_path="classifier",
                        conda_env=custom_env,
                        signature=signature,
                        input_example=input_example)
```

NOTE

- `log_models=False` is configured in `autolog`. This prevents MLflow to automatically log the model, as it is done manually later.
- `infer_signature` is a convenient method to try to infer the signature directly from inputs and outputs.
- `mlflow.utils.environment._mlflow_conda_env` is a private method in MLflow SDK and it may change in the future. This example uses it just for sake of simplicity, but it must be used with caution or generate the YAML definition manually as a Python dictionary.

Logging models with a different behavior in the predict method

When you log a model using either `mlflow.autolog` or using `mlflow.<flavor>.log_model`, the flavor used for the model decides how inference should be executed and what gets returned by the model. MLflow doesn't enforce any specific behavior in how the `predict` generate results. There are scenarios where you probably want to do some pre-processing or post-processing before and after your model is executed.

A solution to this scenario is to implement machine learning pipelines that moves from inputs to outputs directly. Although this is possible (and sometimes encourageable for performance considerations), it may be challenging to achieve. For those cases, you probably want to [customize how your model does inference using a custom models](#) as explained in the following section.

Logging custom models

MLflow provides support for a variety of [machine learning frameworks](#) including FastAI, MXNet Gluon, PyTorch, TensorFlow, XGBoost, CatBoost, h2o, Keras, LightGBM, MLeap, ONNX, Prophet, spaCy, Spark MLLib, Scikit-Learn, and statsmodels. However, they may be times where you need to change how a flavor works, log a model not natively supported by MLflow or even log a model that uses multiple elements from different frameworks. For those cases, you may need to create a custom model flavor.

For this type of models, MLflow introduces a flavor called `pyfunc` (standing from Python function). Basically this flavor allows you to log any object you want as a model, as long as it satisfies two conditions:

- You implement the method `predict` (at least).
- The Python object inherits from `mlflow.pyfunc.PythonModel`.

TIP

Serializable models that implements the Scikit-learn API can use the Scikit-learn flavor to log the model, regardless of whether the model was built with Scikit-learn. If your model can be persisted in Pickle format and the object has methods `predict()` and `predict_proba()` (at least), then you can use `mlflow.sklearn.log_model()` to log it inside a MLflow run.

- [Using a model wrapper](#)
- [Using artifacts](#)
- [Using a model loader](#)

The simplest way of creating your custom model's flavor is by creating a wrapper around your existing model object. MLflow will serialize it and package it for you. Python objects are serializable when the object can be stored in the file system as a file (generally in Pickle format). During runtime, the object can be materialized from such file and all the values, properties and methods available when it was saved will be restored.

Use this method when:

- Your model can be serialized in Pickle format.
- You want to retain the models state as it was just after training.
- You want to customize the way the `predict` function works.

The following sample wraps a model created with XGBoost to make it behaves in a different way to the default implementation of the XGBoost flavor (it returns the probabilities instead of the classes):

```
from mlflow.pyfunc import PythonModel, PythonModelContext

class ModelWrapper(PythonModel):
    def __init__(self, model):
        self._model = model

    def predict(self, context: PythonModelContext, data):
        # You don't have to keep the semantic meaning of `predict`. You can use here model.recommend(),
        model.forecast(), etc
        return self._model.predict_proba(data)

    # You can even add extra functions if you need to. Since the model is serialized,
    # all of them will be available when you load your model back.
    def predict_batch(self, data):
        pass
```

Then, a custom model can be logged in the run like this:

```
import mlflow
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from mlflow.models import infer_signature

mlflow.xgboost.autolog(log_models=False)

model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)
y_probs = model.predict_proba(X_test)

accuracy = accuracy_score(y_test, y_probs.argmax(axis=1))
mlflow.log_metric("accuracy", accuracy)

signature = infer_signature(X_test, y_probs)
mlflow.pyfunc.log_model("classifier",
                       python_model=ModelWrapper(model),
                       signature=signature)
```

TIP

Note how the `infer_signature` method now uses `y_probs` to infer the signature. Our target column has the target class, but our model now returns the two probabilities for each class.

Next steps

- [Deploy MLflow models](#)

Convert custom ML models to MLflow formatted models

9/22/2022 • 3 minutes to read • [Edit Online](#)

In this article, learn how to convert your custom ML model into MLflow format. [MLflow](#) is an open-source library for managing the lifecycle of your machine learning experiments. In some cases, you might use a machine learning framework without its built-in MLflow model flavor support. Due to this lack of built-in MLflow model flavor, you cannot log or register the model with MLflow model fluent APIs. To resolve this, you can convert your model to an MLflow format where you can leverage the following benefits of Azure Machine Learning and MLflow models.

With Azure Machine Learning, MLflow models get the added benefits of,

- No code deployment
- Portability as an open source standard format
- Ability to deploy both locally and on cloud

MLflow provides support for a variety of [machine learning frameworks](#) (scikit-learn, Keras, Pytorch, and more); however, it might not cover every use case. For example, you may want to create an MLflow model with a framework that MLflow does not natively support or you may want to change the way your model does pre-processing or post-processing when running jobs. To know more about MLflow models read [From artifacts to models in MLflow](#).

If you didn't train your model with MLFlow and want to use Azure Machine Learning's MLflow no-code deployment offering, you need to convert your custom model to MLFlow. Learn more about [custom python models and MLflow](#).

Prerequisites

Only the `mlflow` package installed is needed to convert your custom models to an MLflow format.

Create a Python wrapper for your model

Before you can convert your model to an MLflow supported format, you need to first create a python wrapper for your model. The following code demonstrates how to create a Python wrapper for an `sklearn` model.

```

# Load training and test datasets
from sys import version_info
import sklearn
from sklearn import datasets
from sklearn.model_selection import train_test_split

import mlflow.pyfunc
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

PYTHON_VERSION = "{major}.{minor}.{micro}".format(major=version_info.major,
                                                minor=version_info.minor,
                                                micro=version_info.micro)

# Train and save an SKLearn model
sklearn_model_path = "model.pkl"

artifacts = {
    "sklearn_model": sklearn_model_path
}

# create wrapper
class SKLearnWrapper(mlflow.pyfunc.PythonModel):

    def load_context(self, context):
        import pickle
        self.sklearn_model = pickle.load(open(context.artifacts["sklearn_model"], 'rb'))

    def predict(self, model, data):
        return self.sklearn_model.predict(data)

```

Create a Conda environment

Next, you need to create Conda environment for the new MLflow Model that contains all necessary dependencies. If not indicated, the environment is inferred from the current installation. If not, it can be specified.

```

import cloudpickle
conda_env = {
    'channels': ['defaults'],
    'dependencies': [
        'python={}'.format(PYTHON_VERSION),
        'pip',
        {
            'pip': [
                'mlflow',
                'scikit-learn=={}'.format(sklearn.__version__),
                'cloudpickle=={}'.format(cloudpickle.__version__),
            ],
        },
    ],
    'name': 'sklearn_env'
}

```

Load the MLFlow formatted model and test predictions

Once your environment is ready, you can pass the SKlearnWrapper, the Conda environment, and your newly created artifacts dictionary to the `mlflow.pyfunc.save_model()` method. Doing so saves the model to your disk.

```
mlflow_pyfunc_model_path = "sklearn_mlflow_pyfunc7"
mlflow.pyfunc.save_model(path=mlflow_pyfunc_model_path, python_model=SKLearnWrapper(), conda_env=conda_env,
artifacts=artifacts)
```

To ensure your newly saved MLflow formatted model didn't change during the save, you can load your model and print out a test prediction to compare your original model.

The following code prints a test prediction from the mlflow formatted model and a test prediction from the sklearn model that's saved to your disk for comparison.

```
loaded_model = mlflow.pyfunc.load_model(mlflow_pyfunc_model_path)

input_data = "<insert test data>"
# Evaluate the model
import pandas as pd
test_predictions = loaded_model.predict(input_data)
print(test_predictions)

# load the model from disk
import pickle
loaded_model = pickle.load(open(sklearn_model_path, 'rb'))
result = loaded_model.predict(input_data)
print(result)
```

Register the MLflow formatted model

Once you've confirmed that your model saved correctly, you can create a test run, so you can register and save your MLflow formatted model to your model registry.

```
mlflow.start_run()

mlflow.pyfunc.log_model(artifact_path=mlflow_pyfunc_model_path,
                       loader_module=None,
                       data_path=None,
                       code_path=None,
                       python_model=SKLearnWrapper(),
                       registered_model_name="Custom_mlflow_model",
                       conda_env=conda_env,
                       artifacts=artifacts)

mlflow.end_run()
```

IMPORTANT

In some cases, you might use a machine learning framework without its built-in MLflow model flavor support. For instance, the `vaderSentiment` library is a standard natural language processing (NLP) library used for sentiment analysis. Since it lacks a built-in MLflow model flavor, you cannot log or register the model with MLflow model fluent APIs. See an example on [how to save, log and register a model that doesn't have a supported built-in MLflow model flavor](#).

Next steps

- [No-code deployment for MLflow models](#)
- Learn more about [MLflow and Azure Machine Learning](#)

Azure Machine Learning inference HTTP server (preview)

9/22/2022 • 5 minutes to read • [Edit Online](#)

The Azure Machine Learning inference HTTP server ([preview](#)) is a Python package that allows you to easily validate your entry script (`score.py`) in a local development environment. If there's a problem with the scoring script, the server will return an error. It will also return the location where the error occurred.

The server can also be used when creating validation gates in a continuous integration and deployment pipeline. For example, start the server with the candidate script and run the test suite against the local endpoint.

Prerequisites

- Requires: Python >=3.7

Installation

NOTE

To avoid package conflicts, install the server in a virtual environment.

To install the `azureml-inference-server-http` package, run the following command in your cmd/terminal:

```
python -m pip install azureml-inference-server-http
```

Use the server

1. Create a directory to hold your files:

```
mkdir server_quickstart  
cd server_quickstart
```

2. To avoid package conflicts, create a virtual environment and activate it:

```
virtualenv myenv  
source myenv/bin/activate
```

3. Install the `azureml-inference-server-http` package from the [pypi](#) feed:

```
python -m pip install azureml-inference-server-http
```

4. Create your entry script (`score.py`). The following example creates a basic entry script:

```

echo '
import time

def init():
    time.sleep(1)

def run(input_data):
    return {"message": "Hello, World!"}
' > score.py

```

- Start the server and set `score.py` as the entry script:

```
azmlinfsrv --entry_script score.py
```

NOTE

The server is hosted on 0.0.0.0, which means it will listen to all IP addresses of the hosting machine.

- Send a scoring request to the server using `curl`:

```
curl -p 127.0.0.1:5001/score
```

The server should respond like this.

```
{"message": "Hello, World!"}
```

Now you can modify the scoring script and test your changes by running the server again.

Server Routes

The server is listening on port 5001 at these routes.

NAME	ROUTE
Liveness Probe	127.0.0.1:5001/
Score	127.0.0.1:5001/score

Server parameters

The following table contains the parameters accepted by the server:

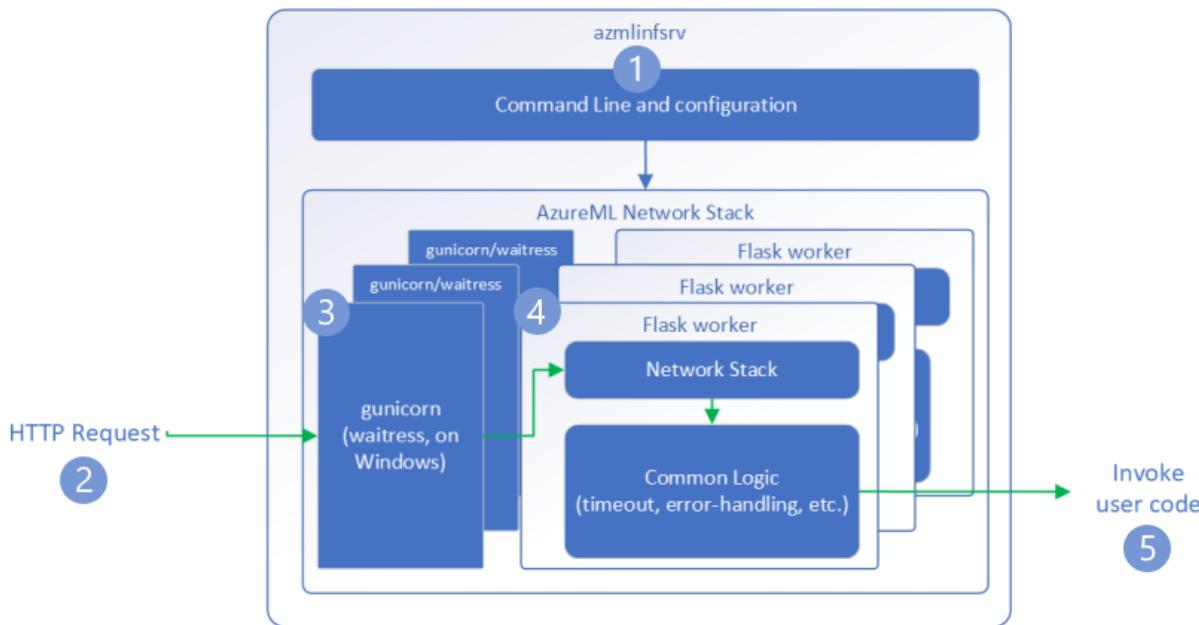
PARAMETER	REQUIRED	DEFAULT	DESCRIPTION
entry_script	True	N/A	The relative or absolute path to the scoring script.
model_dir	False	N/A	The relative or absolute path to the directory holding the model used for inferencing.

PARAMETER	REQUIRED	DEFAULT	DESCRIPTION
port	False	5001	The serving port of the server.
worker_count	False	1	The number of worker threads that will process concurrent requests.
appinsights_instrumentation_key	False	N/A	The instrumentation key to the application insights where the logs will be published.

Request flow

The following steps explain how the Azure Machine Learning inference HTTP server works handles incoming requests:

1. A Python CLI wrapper sits around the server's network stack and is used to start the server.
2. A client sends a request to the server.
3. When a request is received, it goes through the [WSGI](#) server and is then dispatched to one of the workers.
 - [Gunicorn](#) is used on [Linux](#).
 - [Waitress](#) is used on [Windows](#).
4. The requests are then handled by a [Flask](#) app, which loads the entry script & any dependencies.
5. Finally, the request is sent to your entry script. The entry script then makes an inference call to the loaded model and returns a response.



How to integrate with Visual Studio Code

There are two ways to use Visual Studio Code (VSCode) and [Python Extension](#) to debug with [azureml-inference-server-http](#) package.

1. User starts the AzureML Inference Server in a command line and use VSCode + Python Extension to attach to the process.
2. User sets up the `launch.json` in the VSCode and starts the AzureML Inference Server within VSCode.

launch.json

```
{  
    "name": "Debug score.py",  
    "type": "python",  
    "request": "launch",  
    "module": "azureml_inference_server_http.amlserver",  
    "args": [  
        "--entry_script",  
        "score.py"  
    ]  
}
```

In both ways, user can set breakpoint and debug step by step.

Frequently asked questions

1. I encountered the following error during server startup:

```
TypeError: register() takes 3 positional arguments but 4 were given  
  
File "/var/azureml-server/ml_blueprint.py", line 251, in register  
  
    super(MLBlueprint, self).register(app, options, first_registration)  
  
TypeError: register() takes 3 positional arguments but 4 were given
```

You have **Flask 2** installed in your python environment but are running a version of `azureml-inference-server-http` that doesn't support Flask 2. Support for Flask 2 is added in `azureml-inference-server-http>=0.7.0`, which is also in `azureml-defaults>=1.44`.

1. If you're not using this package in an AzureML docker image, use the latest version of `azureml-inference-server-http` or `azureml-defaults`.
2. If you're using this package with an AzureML docker image, make sure you're using an image built in or after July, 2022. The image version is available in the container logs. You should be able to find a log similar to below:

```
2022-08-22T17:05:02,147738763+00:00 | gunicorn/run | AzureML Container Runtime Information  
2022-08-22T17:05:02,161963207+00:00 | gunicorn/run | #####  
2022-08-22T17:05:02,168970479+00:00 | gunicorn/run |  
2022-08-22T17:05:02,174364834+00:00 | gunicorn/run |  
2022-08-22T17:05:02,187280665+00:00 | gunicorn/run | AzureML image information: openmpi4.1.0-  
ubuntu20.04, Materializaton Build:20220708.v2  
2022-08-22T17:05:02,188930082+00:00 | gunicorn/run |  
2022-08-22T17:05:02,190557998+00:00 | gunicorn/run |
```

The build date of the image appears after "Materialization Build", which in the above example is `20220708`, or July 8, 2022. This image is compatible with Flask 2. If you don't see a banner like this in your container log, your image is out-of-date, and should be updated. If you're using a cuda image, and are unable to find a newer image, check if your image is deprecated in [AzureML-Containers](#). If it is, you should be able to find replacements.

If this is an online endpoint, you can also find the logs under "Deployment logs" in the [online endpoint page in Azure Machine Learning studio](#). If you deploy with SDK v1 and don't explicitly specify an image in your deployment configuration, it will default to using a version of `openmpi4.1.0-ubuntu20.04` that

matches your local SDK toolset, which may not be the latest version of the image. For example, SDK 1.43 will default to using `openmpi4.1.0-ubuntu20.04:20220616`, which is incompatible. Make sure you use the latest SDK for your deployment.

If for some reason you're unable to update the image, you can temporarily avoid the issue by pinning `azureml-defaults==1.43` or `azureml-inference-server-http~0.4.13`, which will install the older version server with `Flask 1.0.x`.

See also [Troubleshooting online endpoints deployment](#).

2. I encountered an `ImportError` or `ModuleNotFoundError` on modules `opencensus`, `jinja2`, `MarkupSafe`, or `click` during startup like the following message:

```
ImportError: cannot import name 'Markup' from 'jinja2'
```

Older versions (<= 0.4.10) of the server didn't pin Flask's dependency to compatible versions. This problem is fixed in the latest version of the server.

3. Do I need to reload the server when changing the score script?

After changing your scoring script (`score.py`), stop the server with `ctrl + c`. Then restart it with `azmlinfsrv --entry_script score.py`.

4. Which OS is supported?

The Azure Machine Learning inference server runs on Windows & Linux based operating systems.

Next steps

- For more information on creating an entry script and deploying models, see [How to deploy a model using Azure Machine Learning](#).
- Learn about [Prebuilt docker images for inference](#)

Work with models in Azure Machine Learning

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO: [Azure CLI ml extension v2 \(current\)](#) [Python SDK azure-ai-ml v2 \(preview\)](#)

Azure Machine Learning allows you to work with different types of models. In this article, you learn about using Azure Machine Learning to work with different model types, such as custom, MLflow, and Triton. You also learn how to register a model from different locations, and how to use the Azure Machine Learning SDK, the user interface (UI), and the Azure Machine Learning CLI to manage your models.

TIP

If you have model assets created that use the SDK/CLI v1, you can still use those with SDK/CLI v2. Full backward compatibility is provided. All models registered with the V1 SDK are assigned the type `custom`.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace.
- The Azure Machine Learning [SDK v2 for Python](#).
- The Azure Machine Learning [CLI v2](#).

Supported paths

When you provide a model you want to register, you'll need to specify a `path` parameter that points to the data or job location. Below is a table that shows the different data locations supported in Azure Machine Learning and examples for the `path` parameter:

LOCATION	EXAMPLES
A path on your local computer	<code>mlflow-model/model.pkl</code>
A path on an AzureML Datastore	<code>azureml://datastores/<datastore-name>/paths/<path_on_datastore></code>
A path from an AzureML job	<code>azureml://jobs/<job-name>/outputs/<output-name>/paths/<path-to-model-relative-to-the-named-output-location></code>
A path from an MLflow job	<code>runs:/<run-id>/<path-to-model-relative-to-the-root-of-the-artifact-location></code>

Supported modes

When you run a job with model inputs/outputs, you can specify the `mode` - for example, whether you would like the model to be read-only mounted or downloaded to the compute target. The table below shows the possible modes for different type/mode/input/output combinations:

TYPE	INPUT/OUTPUT	DIRECT	DOWNLOAD	RO_MOUNT
custom file	Input	✓		
custom folder	Input	✓	✓	✓
mlflow	Input		✓	✓
custom file	Output	✓	✓	✓
custom folder	Output	✓	✓	✓
mlflow	Output	✓	✓	✓

Create a model in the model registry

Model registration allows you to store and version your models in the Azure cloud, in your workspace. The model registry helps you organize and keep track of your trained models.

The code snippets in this section cover how to:

- Register your model as an asset in Machine Learning by using the CLI.
- Register your model as an asset in Machine Learning by using the SDK.
- Register your model as an asset in Machine Learning by using the UI.

These snippets use `custom` and `mlflow`.

- `custom` is a type that refers to a model file or folder trained with a custom standard not currently supported by Azure ML.
- `mlflow` is a type that refers to a model trained with `mlflow`. MLflow trained models are in a folder that contains the `MLmodel` file, the `model` file, the `conda dependencies` file, and the `requirements.txt` file.

Register your model as an asset in Machine Learning by using the CLI

Use the following tabs to select where your model is located.

- Local model
- Datastore
- Job output

```
$schema: https://azureschemas.azureedge.net/latest/model.schema.json
name: local-file-example
path: mlflow-model/model.pkl
description: Model created from local file.
```

```
az ml model create -f <file-name>.yml
```

For a complete example, see the [model YAML](#).

Register your model as an asset in Machine Learning by using the SDK

Use the following tabs to select where your model is located.

- Local model

- [Datastore](#)
- [Job output](#)

```
from azure.ai.ml.entities import Model
from azure.ai.ml.constants import ModelType

file_model = Model(
    path="mlflow-model/model.pkl",
    type=ModelType.CUSTOM,
    name="local-file-example",
    description="Model created from local file."
)
ml_client.models.create_or_update(file_model)
```

Register your model as an asset in Machine Learning by using the UI

To create a model in Machine Learning, from the UI, open the **Models** page. Select **Register model**, and select where your model is located. Fill out the required fields, and then select **Register**.

Register a model

Name *****

Description

Model framework *****

Framework name *****

Framework version *****

Model file or folder *****

Tags

Add properties

Register Cancel

Next steps

- [Install and set up Python SDK v2 \(preview\)](#)
- [No-code deployment for MLflow models](#)
- Learn more about [MLflow and Azure Machine Learning](#)

Manage models registries in Azure Machine Learning with MLflow

9/22/2022 • 6 minutes to read • [Edit Online](#)

Azure Machine Learning supports MLflow for model management. This represents a convenient way to support the entire model lifecycle for users familiar with the MLflow client. The following article describes the different capabilities and how it compares with other options.

Support matrix for managing models with MLflow

The MLflow client exposes several methods to retrieve and manage models. The following table shows which of those methods are currently supported in MLflow when connected to Azure ML. It also compares it with other models management capabilities in Azure ML.

FEATURE	MLFLOW	AZURE ML WITH MLFLOW	AZURE ML CLIV2	AZURE ML STUDIO
Registering models in MLflow format	✓	✓	✓	✓
Registering models not in MLflow format			✓	✓
Registering models from runs outputs/artifacts	✓	✓ ¹	✓ ²	✓
Registering models from runs outputs/artifacts in a different tracking server/workspace	✓			
Listing registered models	✓	✓	✓	✓
Retrieving details of registered model's versions	✓	✓	✓	✓
Editing registered model's versions description	✓	✓	✓	✓
Editing registered model's versions tags	✓	✓	✓	✓
Renaming registered models	✓	3	3	3

FEATURE	MLFLOW	AZURE ML WITH MLFLOW	AZURE ML CLIV2	AZURE ML STUDIO
Deleting a registered model (container)	✓	3	3	3
Deleting a registered model's version	✓	✓	✓	✓
Manage MLflow model stages	✓	✓		
Search registered models by name	✓	✓	✓	✓ ⁴
Search registered models using string comparators <code>LIKE</code> and <code>ILIKE</code>	✓			✓ ⁴
Search registered models by tag				✓ ⁴

NOTE

- ¹ Use URIs with format `runs:/<run-id>/<path>`.
- ² Use URIs with format `azureml://jobs/<job-id>/outputs/artifacts/<path>`.
- ³ Registered models are immutable objects in Azure ML.
- ⁴ Use search box in Azure ML Studio. Partial match supported.

Prerequisites

- Install the `azureml-mlflow` package.
- If you are running outside an Azure ML compute, configure the MLflow tracking URI or MLflow's registry URI to point to the workspace you are working on. For more information about how to Set up tracking environment, see [Track runs using MLflow with Azure Machine Learning](#) for more details.

Registering new models in the registry

Creating models from an existing run

If you have an MLflow model logged inside of a run and you want to register it in a registry, you can do that by using the run ID and the path where the model was logged. See [Manage experiments and runs with MLflow](#) to know how to query this information if you don't have it.

```
mlflow.register_model(f"runs:{run_id}/{artifact_path}", model_name)
```

NOTE

Models can only be registered to the registry in the same workspace where the run was tracked. Cross-workspace operations are not supported by the moment in Azure Machine Learning.

Creating models from assets

If you have a folder with an MLModel MLflow model, then you can register it directly. There's no need for the

model to be always in the context of a run. To do that you can use the URI schema `file://path/to/model` to register MLflow models stored in the local file system. Let's create a simple model using `Scikit-Learn` and save it in MLflow format in the local storage:

```
from sklearn import linear_model  
  
reg = linear_model.LinearRegression()  
reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])  
  
mlflow.sklearn.save_model(reg, "./regressor")
```

TIP

The method `save_model()` works in the same way as `log_model()`. While `log_model()` saves the model inside on an active run, `save_model()` uses the local file system for saving the model.

You can now register the model from the local path:

```
import os  
  
model_local_path = os.path.abspath("./regressor")  
mlflow.register_model(f"file://{model_local_path}", "local-model-test")
```

NOTE

Notice how the model URI schema `file:/` requires absolute paths.

Querying model registries

Querying all the models in the registry

You can query all the registered models in the registry using the MLflow client with the method `list_registered_models`. The MLflow client is required to do all these operations.

```
using mlflow  
  
client = mlflow.tracking.MlflowClient()
```

The following sample prints all the model's names:

```
for model in client.list_registered_models():  
    print(f"{model.name}")
```

Getting specific versions of the model

The command above will retrieve the model object which contains all the model versions. However, if you want to get the last registered model version of a given model, you can use `get_registered_model`:

```
client.get_registered_model(model_name)
```

If you need a specific version of the model, you can indicate so:

```
client.get_model_version(model_name, version=2)
```

Loading models from registry

You can load models directly from the registry to restore the models objects that were logged. Use the functions `mlflow.<flavor>.load_model()` or `mlflow.pyfunc.load_model()` indicating the URI of the model you want to load using the following syntax:

- `models:/<model-name>/latest`, to load the last version of the model.
- `models:/<model-name>/<version-number>`, to load a specific version of the model.
- `models:/<model-name>/<stage-name>`, to load a specific version in a given stage for a model. View [Model stages](#) for details.

TIP

For learning about the difference between `mlflow.<flavor>.load_model()` and `mlflow.pyfunc.load_model()`, view [Loading MLflow models back](#) article.

Model stages

MLflow supports model's stages to manage model's lifecycle. Model's version can transition from one stage to another. Stages are assigned to a model's version (instead of models) which means that a given model can have multiple versions on different stages.

IMPORTANT

Stages can only be accessed using the MLflow SDK. They don't show up in the [Azure ML Studio portal](#) and can't be retrieved using neither Azure ML SDK, Azure ML CLI, or Azure ML REST API. Creating deployment from a given model's stage is not supported by the moment.

Querying model stages

You can use the MLflow client to check all the possible stages a model can be:

```
client.get_model_version_stages(model_name, version="latest")
```

You can see what model's version is on each stage by getting the model from the registry. The following example gets the model's version currently in the stage `Staging`.

```
client.get_latest_versions(model_name, stages=["Staging"])
```

NOTE

Multiple versions can be in the same stage at the same time in MLflow, however, this method returns the latest version (greater version) among all of them.

WARNING

Stage names are case sensitive.

Transitioning models

Transitioning a model's version to a particular stage can be done using the MLflow client.

```
client.transition_model_version_stage(model_name, version=3, stage="Staging")
```

By default, if there were an existing model version in that particular stage, it will remain there. Hence, it won't be replaced as multiple model's versions can be in the same stage at the same time. Alternatively, you can indicate `archive_existing_versions=True` to tell MLflow to move the existing model's version to the stage `Archived`.

```
client.transition_model_version_stage(  
    model_name, version=3, stage="Staging", archive_existing_versions=True  
)
```

Loading models from stages

You can load a model in a particular stage directly from Python using the `load_model` function and the following URI format. Notice that for this method to success, you need to have all the libraries and dependencies already installed in the environment you're working at.

```
model = mlflow.pyfunc.load_model(f"models:{model_name}/Staging")
```

Editing and deleting models

Editing registered models is supported in both MLflow and Azure ML, however, there are some differences between them that are important to notice:

WARNING

Renaming models is not supported in Azure Machine Learning as model objects are immutable.

Editing models

You can edit model's description and tags from a model using MLflow:

```
client.update_model_version(model_name, version=1, description="My classifier description")
```

To edit tags, you have to use the method `set_model_version_tag` and `remove_model_version_tag`:

```
client.set_model_version_tag(model_name, version="1", key="type", value="classification")
```

Removing a tag:

```
client.delete_model_version_tag(model_name, version="1", key="type")
```

Deleting a model's version

You can delete any model version in the registry using the MLflow client, as demonstrated in the following example:

```
client.delete_model_version(model_name, version="2")
```

NOTE

Azure Machine Learning doesn't support deleting the entire model container. To achieve the same thing, you will need to delete all the model versions from a given model.

Use GitHub Actions with Azure Machine Learning

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

Get started with [GitHub Actions](#) to train a model on Azure Machine Learning.

This article will teach you how to create a GitHub Actions workflow that builds and deploys a machine learning model to [Azure Machine Learning](#). You'll train a [scikit-learn](#) linear regression model on the NYC Taxi dataset.

GitHub Actions uses a workflow YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace. If you don't have one, use the steps in the [Quickstart: Create workspace resources](#) article to create one.
- To install the Python SDK v2, use the following command:

```
pip install --pre azure-ai-ml
```

For more information, see [Install the Python SDK v2 for Azure Machine Learning \(preview\)](#).

- A GitHub account. If you don't have one, sign up for [free](#).

Step 1. Get the code

Fork the following repo at GitHub:

```
https://github.com/azure/azureml-examples
```

Step 2. Authenticate with Azure

You'll need to first define how to authenticate with Azure. You can use a [service principal](#) or [OpenID Connect](#).

Generate deployment credentials

- [Service principal](#)
- [OpenID Connect](#)

Create a [service principal](#) with the `az ad sp create-for-rbac` command in the [Azure CLI](#). Run this command with [Azure Cloud Shell](#) in the Azure portal or by selecting the [Try it](#) button.

```
az ad sp create-for-rbac --name "myML" --role contributor \
    --scopes /subscriptions/<subscription-id>/resourceGroups/<group-name> \
    --sdk-auth
```

In the example above, replace the placeholders with your subscription ID, resource group name, and app name. The output is a JSON object with the role assignment credentials that provide access to your App Service app similar to below. Copy this JSON object for later.

```
{
  "clientId": "<GUID>",
  "clientSecret": "<GUID>",
  "subscriptionId": "<GUID>",
  "tenantId": "<GUID>",
  (...)
```

Create secrets

- [Service principal](#)
- [OpenID Connect](#)

1. In [GitHub](#), browse your repository, select **Settings > Secrets > Actions**. Select **New repository secret**.
2. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZ_CREDS`.

Step 3. Update `setup.sh` to connect to your Azure Machine Learning workspace

You'll need to update the CLI setup file variables to match your workspace.

1. In your cloned repository, go to `azureml-examples/cli/`.
2. Edit `setup.sh` and update these variables in the file.

VARIABLE	DESCRIPTION
GROUP	Name of resource group
LOCATION	Location of your workspace (example: <code>eastus2</code>)
WORKSPACE	Name of Azure ML workspace

Step 4. Update `pipeline.yml` with your compute cluster name

You'll use a `pipeline.yml` file to deploy your Azure ML pipeline. This is a machine learning pipeline and not a DevOps pipeline. You only need to make this update if you're using a name other than `cpu-cluster` for your computer cluster name.

1. In your cloned repository, go to `azureml-examples/cli/jobs/pipelines/nyc-taxi/pipeline.yml`.
2. Each time you see `compute: azureml:cpu-cluster`, update the value of `cpu-cluster` with your compute cluster name. For example, if your cluster is named `my-cluster`, your new value would be `azureml:my-cluster`. There are five updates.

Step 5: Run your GitHub Actions workflow

Your workflow authenticates with Azure, sets up the Azure Machine Learning CLI, and uses the CLI to train a model in Azure Machine Learning.

- [Service principal](#)
- [OpenID Connect](#)

Your workflow file is made up of a trigger section and jobs:

- A trigger starts the workflow in the `on` section. The workflow runs by default on a cron schedule and when a pull request is made from matching branches and paths. Learn more about [events that trigger workflows](#).
- In the jobs section of the workflow, you checkout code and log into Azure with your service principal secret.
- The jobs section also includes a setup action that installs and sets up the [Machine Learning CLI \(v2\)](#). Once the CLI is installed, the run job action runs your Azure Machine Learning `pipeline.yml` file to train a model with NYC taxi data.

Enable your workflow

1. In your cloned repository, open `.github/workflows/cli-jobs-pipelines-nyc-taxi-pipeline.yml` and verify that your workflow looks like this.

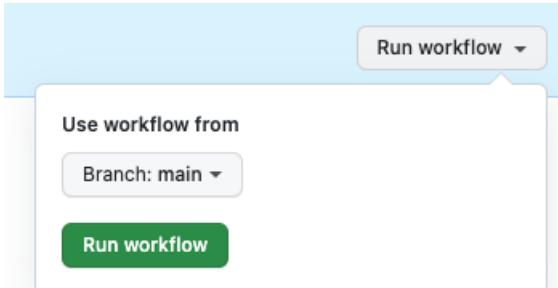
```
name: cli-jobs-pipelines-nyc-taxi-pipeline
on:
  workflow_dispatch:
  schedule:
    - cron: "0 0/4 * * *"
  pull_request:
    branches:
      - main
      - sdk-preview
    paths:
      - cli/jobs/pipelines/nyc-taxi/**
      - .github/workflows/cli-jobs-pipelines-nyc-taxi-pipeline.yml
      - cli/run-pipeline-jobs.sh
      - cli/setup.sh
jobs:
  build:
    build:
      runs-on: ubuntu-latest
      steps:
        - name: check out repo
          uses: actions/checkout@v2
        - name: azure login
          uses: azure/login@v1
          with:
            creds: ${{secrets.AZ_CREDS}}
        - name: setup
          run: bash setup.sh
          working-directory: cli
          continue-on-error: true
        - name: run job
          run: bash -x ../../run-job.sh pipeline.yml
          working-directory: cli/jobs/pipelines/nyc-taxi
```

2. Select **View runs**.
3. Enable workflows by selecting **I understand my workflows, go ahead and enable them**.
4. Select the **cli-jobs-pipelines-nyc-taxi-pipeline workflow** and choose to **Enable workflow**.

⚠️ This scheduled workflow is disabled because scheduled workflows are disabled by default in forks.

[Enable workflow](#)

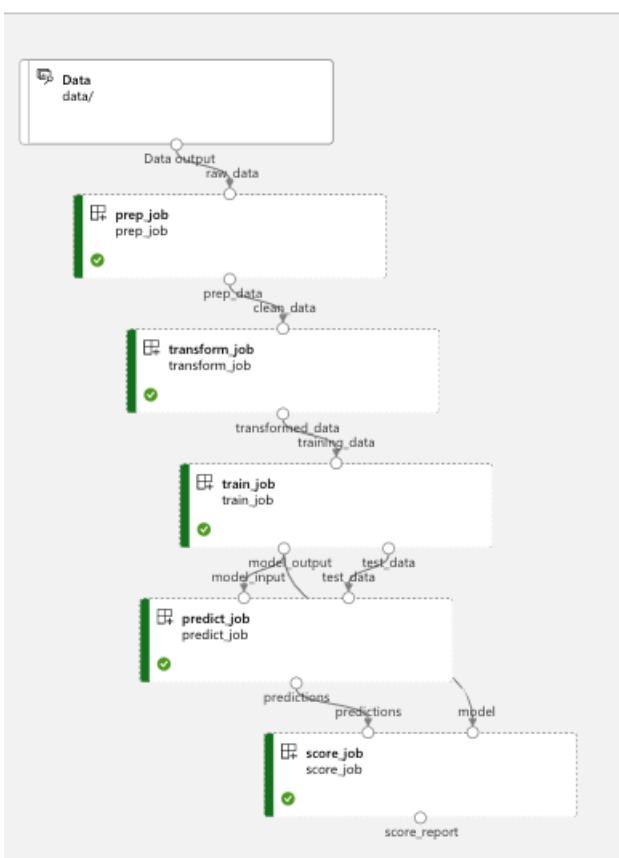
5. Select **Run workflow** and choose the option to **Run workflow now**.



Step 6: Verify your workflow run

1. Open your completed workflow run and verify that the build job ran successfully. You'll see a green checkmark next to the job.
2. Open Azure Machine Learning studio and navigate to the **nyc-taxi-pipeline-example**. Verify that each part of your job (prep, transform, train, predict, score) completed and that you see a green checkmark.

nyc-taxi-pipeline-example Completed



Clean up resources

When your resource group and repository are no longer needed, clean up the resources you deployed by deleting the resource group and your GitHub repository.

Next steps

[Create production ML pipelines with Python SDK](#)

Trigger applications, processes, or CI/CD workflows based on Azure Machine Learning events (preview)

9/22/2022 • 8 minutes to read • [Edit Online](#)

In this article, you learn how to set up event-driven applications, processes, or CI/CD workflows based on Azure Machine Learning events, such as failure notification emails or ML pipeline runs, when certain conditions are detected by [Azure Event Grid](#).

Azure Machine Learning manages the entire lifecycle of machine learning process, including model training, model deployment, and monitoring. You can use Event Grid to react to Azure Machine Learning events, such as the completion of training runs, the registration and deployment of models, and the detection of data drift, by using modern serverless architectures. You can then subscribe and consume events such as run status changed, run completion, model registration, model deployment, and data drift detection within a workspace.

When to use Event Grid for event driven actions:

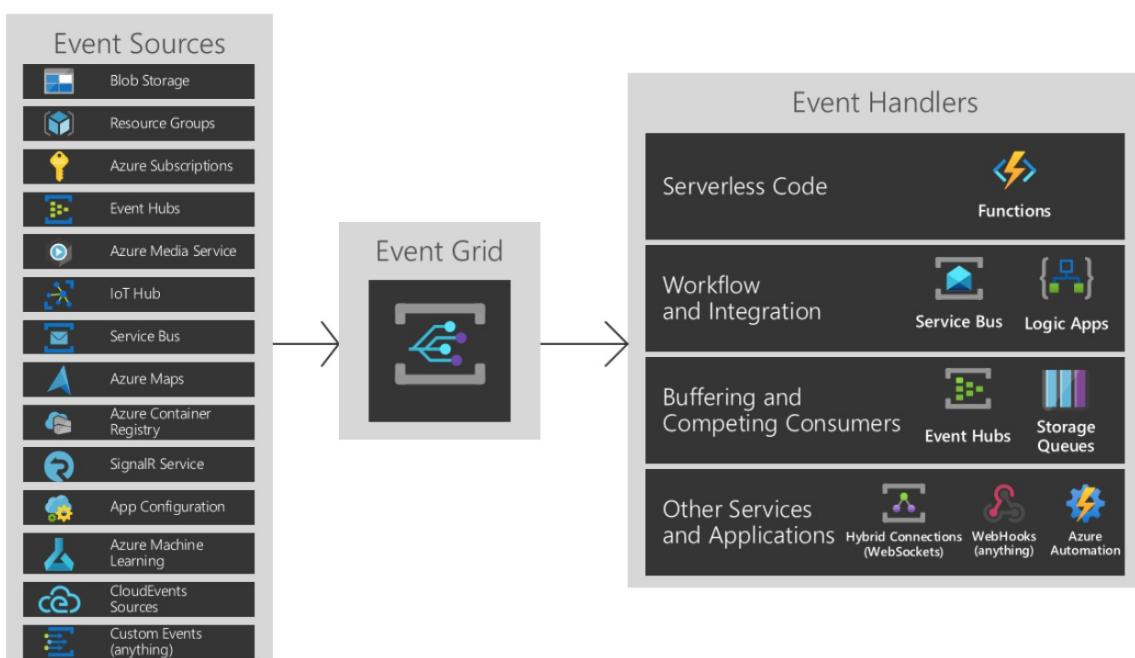
- Send emails on run failure and run completion
- Use an Azure function after a model is registered
- Streaming events from Azure Machine Learning to various of endpoints
- Trigger an ML pipeline when drift is detected

Prerequisites

To use Event Grid, you need contributor or owner access to the Azure Machine Learning workspace you will create events for.

The event model & types

Azure Event Grid reads events from sources, such as Azure Machine Learning and other Azure services. These events are then sent to event handlers such as Azure Event Hubs, Azure Functions, Logic Apps, and others. The following diagram shows how Event Grid connects sources and handlers, but is not a comprehensive list of supported integrations.



For more information on event sources and event handlers, see [What is Event Grid?](#)

Event types for Azure Machine Learning

Azure Machine Learning provides events in the various points of machine learning lifecycle:

EVENT TYPE	DESCRIPTION
<code>Microsoft.MachineLearningServices.RunCompleted</code>	Raised when a machine learning experiment run is completed
<code>Microsoft.MachineLearningServices.ModelRegistered</code>	Raised when a machine learning model is registered in the workspace
<code>Microsoft.MachineLearningServices.ModelDeployed</code>	Raised when a deployment of inference service with one or more models is completed
<code>Microsoft.MachineLearningServices.DatasetDriftDetected</code>	Raised when a data drift detection job for two datasets is completed
<code>Microsoft.MachineLearningServices.RunStatusChanged</code>	Raised when a run status is changed

Filter & subscribe to events

These events are published through Azure Event Grid. Using Azure portal, PowerShell or Azure CLI, customers can easily subscribe to events by [specifying one or more event types, and filtering conditions](#).

When setting up your events, you can apply filters to only trigger on specific event data. In the example below, for run status changed events, you can filter by run types. The event only triggers when the criteria is met. Refer to the [Azure Machine Learning event grid schema](#) to learn about event data you can filter by.

Subscriptions for Azure Machine Learning events are protected by Azure role-based access control (Azure RBAC). Only [contributor or owner](#) of a workspace can create, update, and delete event subscriptions. Filters can be applied to event subscriptions either during the [creation](#) of the event subscription or at a later time.

1. Go to the Azure portal, select a new subscription or an existing one.
2. Select the Events entry from the left navigation area, and then select **+ Event subscription**.
3. Select the filters tab and scroll down to Advanced filters. For the **Key** and **Value**, provide the property types you want to filter by. Here you can see the event will only trigger when the run type is a pipeline run or pipeline step run.

Create Event Subscription

Event Grid

Basics **Filters** Additional Features Delivery Properties Advanced Editor

SUBJECT FILTERS

Apply filters to the subject of each event. Only events with matching subjects get delivered. [Learn more](#)

Enable subject filtering

ADVANCED FILTERS

Filter on attributes of each event. Only events that match all filters get delivered. Up to 25 filters can be specified. All string comparisons are case-insensitive. [Learn more](#)

Valid keys for currently selected event schema:

- id, topic, subject, eventtype, dataversion
- Custom properties inside the data payload, using "." as the nesting separator. (e.g. data, data.key, data.key1.key2)

Key	Operator	Value
data.runType	String is in	azureml.PipelineRun <input type="button" value="X"/> azuremlStepRun <input type="button" value="X"/>
<input type="button" value="Add new value (up to 25)"/>		

[Add new filter](#)

Enable advanced filtering on arrays ⓘ

- **Filter by event type:** An event subscription can specify one or more Azure Machine Learning event types.
- **Filter by event subject:** Azure Event Grid supports subject filters based on **begins with** and **ends with** matches, so that events with a matching subject are delivered to the subscriber. Different machine learning events have different subject format.

EVENT TYPE	SUBJECT FORMAT	SAMPLE SUBJECT
Microsoft.MachineLearningServices.R	experiments/{ExperimentId}/runs/{RunId}	experiments/b1d7966c-f73a-4c68-b846-992ace89551f/runs/my_exp1_1554835758_38dbaa94
Microsoft.MachineLearningServices.M	models/{modelName}:{modelVersion}	models/sklearn_regression_model:3
Microsoft.MachineLearningServices.M	endpoints/{serviceId}	endpoints/my_sklearn_aks
Microsoft.MachineLearningServices.D	datadrift/{data.DataDriftId}/run/{RunId}	datadrift/4e694bf5-712e-4e40-b06a-d2a2755212d4/run/my_driftrun1_1550564444_fbbcd
Microsoft.MachineLearningServices.R	experiments/{ExperimentId}/runs/{RunId}	experiments/b1d7966c-f73a-4c68-b846-992ace89551f/runs/my_exp1_1554835758_38dbaa94

- **Advanced filtering:** Azure Event Grid also supports advanced filtering based on published event schema. Azure Machine Learning event schema details can be found in [Azure Event Grid event schema for Azure Machine Learning](#). Some sample advanced filterings you can perform include:

For `Microsoft.MachineLearningServices.ModelRegistered` event, to filter model's tag value:

```
--advanced-filter data.ModelTags.key1 StringIn ('value1')
```

To learn more about how to apply filters, see [Filter events for Event Grid](#).

Consume Machine Learning events

Applications that handle Machine Learning events should follow a few recommended practices:

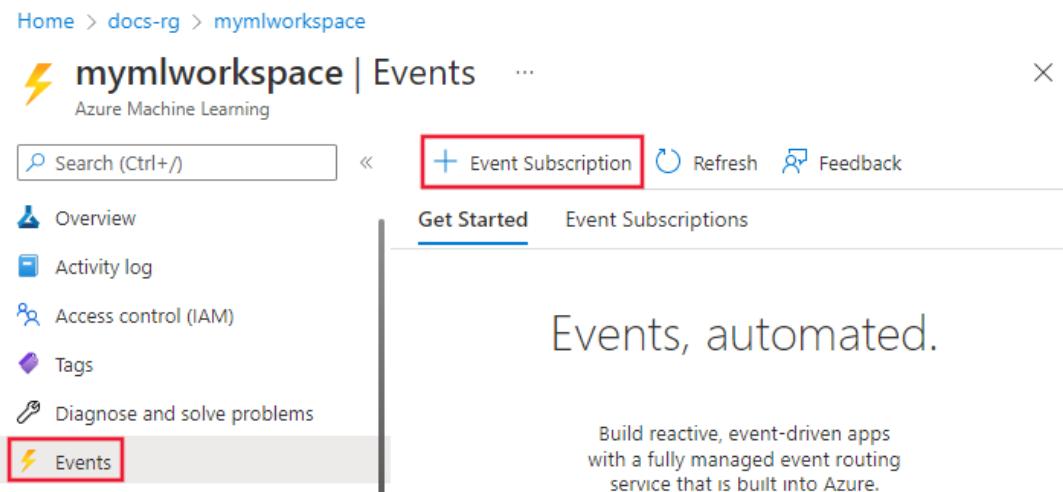
- As multiple subscriptions can be configured to route events to the same event handler, it is important not to assume events are from a particular source, but to check the topic of the message to ensure that it comes from the machine learning workspace you are expecting.
- Similarly, check that the eventType is one you are prepared to process, and do not assume that all events you receive will be the types you expect.
- As messages can arrive out of order and after some delay, use the etag fields to understand if your information about objects is still up-to-date. Also, use the sequencer fields to understand the order of events on any particular object.
- Ignore fields you don't understand. This practice will help keep you resilient to new features that might be added in the future.
- Failed or cancelled Azure Machine Learning operations will not trigger an event. For example, if a model deployment fails Microsoft.MachineLearningServices.ModelDeployed won't be triggered. Consider such failure mode when design your applications. You can always use Azure Machine Learning SDK, CLI or portal to check the status of an operation and understand the detailed failure reasons.

Azure Event Grid allows customers to build de-coupled message handlers, which can be triggered by Azure Machine Learning events. Some notable examples of message handlers are:

- Azure Functions
- Azure Logic Apps
- Azure Event Hubs
- Azure Data Factory Pipeline
- Generic webhooks, which may be hosted on the Azure platform or elsewhere

Set up in Azure portal

1. Open the [Azure portal](#) and go to your Azure Machine Learning workspace.
2. From the left bar, select **Events** and then select **Event Subscriptions**.



3. Select the event type to consume. For example, the following screenshot has selected **Model registered**, **Model deployed**, **Run completed**, and **Dataset drift detected**:



Create Event Subscription

...

Event Grid

Basic

Filters

Additional Features

Delivery Properties

Advanced Editor

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

EVENT SUBSCRIPTION DETAILS

Name *

Event Schema

Event Grid Schema



TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type

Azure Machine Learning

Source Resource

mymlworkspace

System Topic Name *

EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types *

5 selected

Model registered
 Model deployed
 Run completed
 Dataset drift detected
 Run status changed

Create

4. Select the endpoint to publish the event to. In the following screenshot, **Event hub** is the selected endpoint:

Once you have confirmed your selection, click **Create**. After configuration, these events will be pushed to your endpoint.

Set up with the CLI

You can either install the latest [Azure CLI](#), or use the Azure Cloud Shell that is provided as part of your Azure subscription.

To install the Event Grid extension, use the following command from the CLI:

```
az add extension --name eventgrid
```

The following example demonstrates how to select an Azure subscription and creates a new event subscription for Azure Machine Learning:

```
# Select the Azure subscription that contains the workspace
az account set --subscription "<name or ID of the subscription>"

# Subscribe to the machine learning workspace. This example uses EventHub as a destination.
az eventgrid event-subscription create --name {eventGridFilterName} \
--source-resource-id
/subscriptions/{subId}/resourceGroups/{RG}/providers/Microsoft.MachineLearningServices/workspaces/{wsName} \
--endpoint-type eventhub \
--endpoint
/subscriptions/{SubID}/resourceGroups/TestRG/providers/Microsoft.EventHub/namespaces/n1/eventhubs/EH1 \
--included-event-types Microsoft.MachineLearningServices.ModelRegistered \
--subject-begins-with "models/mymodelname"
```

Examples

Example: Send email alerts

Use [Azure Logic Apps](#) to configure emails for all your events. Customize with conditions and specify recipients

to enable collaboration and awareness across teams working together.

1. In the Azure portal, go to your Azure Machine Learning workspace and select the events tab from the left bar. From here, select Logic apps.

Home > docs-rg > mymlworkspace

mymlworkspace | Events ... X

Azure Machine Learning

Search (Ctrl+ /) Event Subscription Refresh Feedback

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Events

Logic Apps

Use events as a trigger for executing a Logic App, starting Azure-wide workflows and automation.

2. Sign into the Logic App UI and select Machine Learning service as the topic type.

When a resource event occurs

* Subscription: {topicSubscriptionId}

* Resource Type: Microsoft.MachineLearningServices.Workspaces

* Resource Name:

Event Type Item - 1

+ Add new item

Add new parameter

Microsoft.MachineLearningServices.Workspaces

MicrosoftMaps.Accounts

MicrosoftMedia.MediaServices

MicrosoftResources.ResourceGroups

MicrosoftResources.Subscriptions

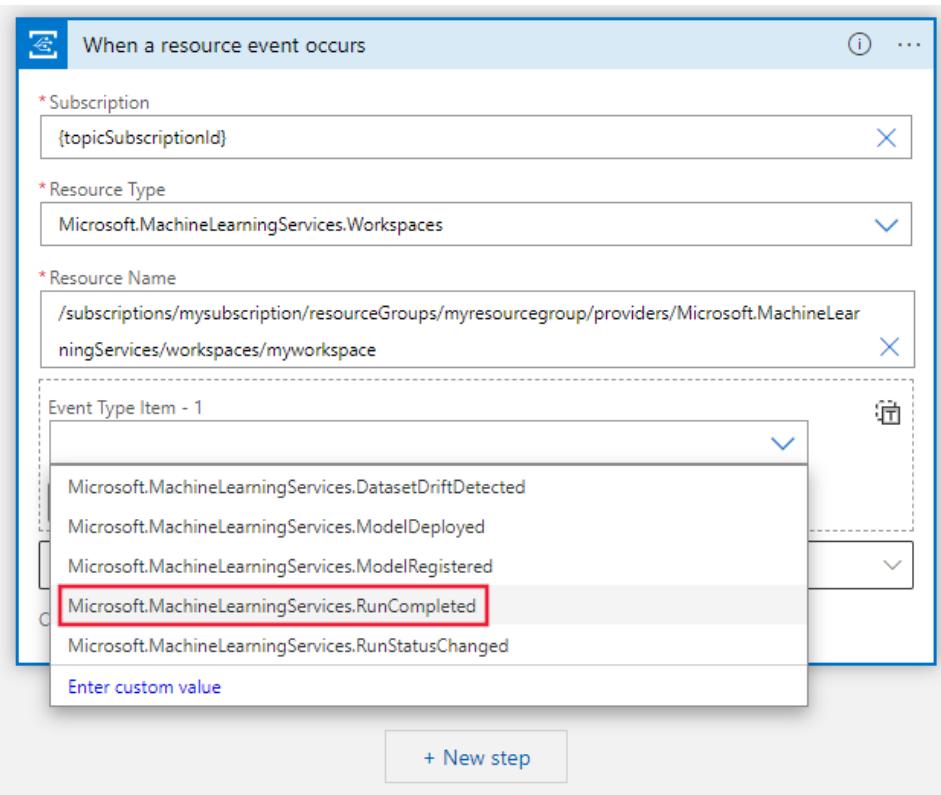
MicrosoftServiceBus.Namespaces

MicrosoftSignalRService.SignalR

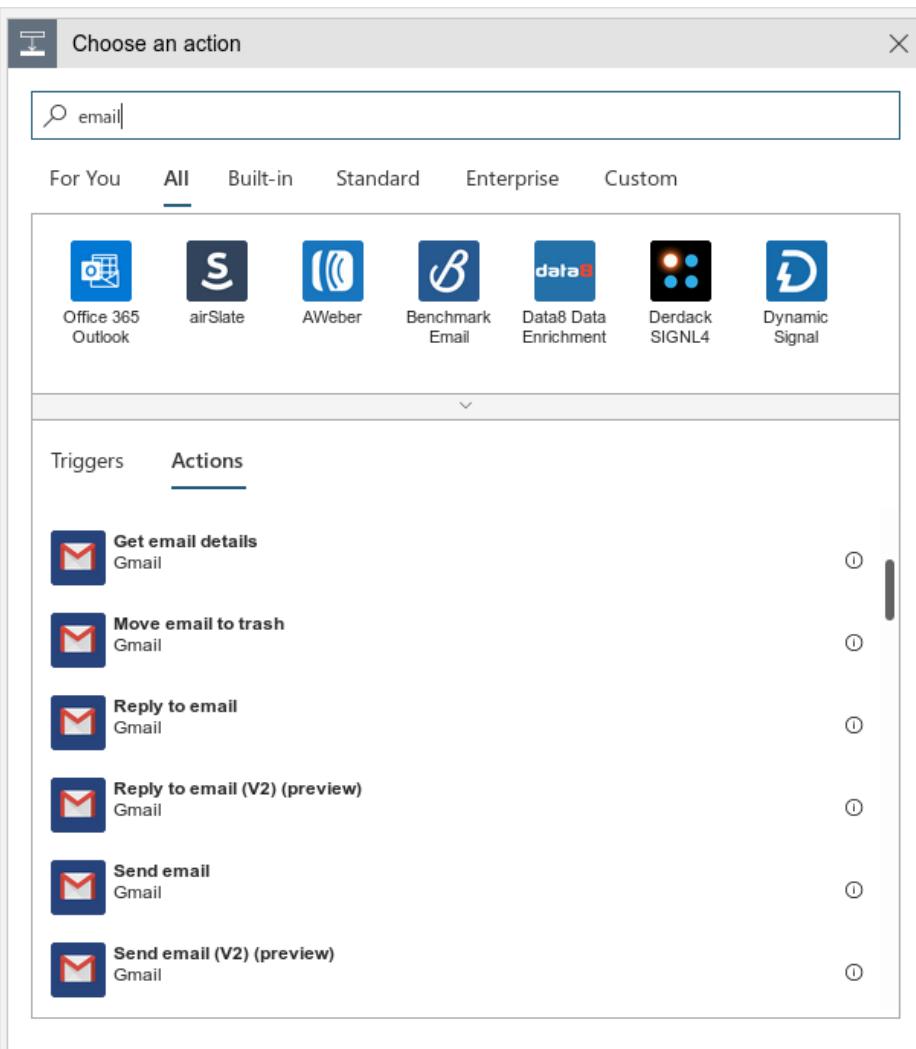
MicrosoftStorage.StorageAccounts

Enter custom value

3. Select which event(s) to be notified for. For example, the following screenshot RunCompleted.



4. Next, add a step to consume this event and search for email. There are several different mail accounts you can use to receive events. You can also configure conditions on when to send an email alert.



5. Select **Send an email** and fill in the parameters. In the subject, you can include the **Event Type** and **Topic** to help filter events. You can also include a link to the workspace page for runs in the message

body.

The screenshot shows the Logic Apps Designer interface. A workflow is being created with a trigger 'When a resource event occurs' followed by an action 'Send an email (V2)'. In the 'Send an email (V2)' step, the 'Subject' field contains 'Topic' and 'Event Type'. A red box highlights 'Event Type'. To the right, a 'Dynamic content' pane lists variables: 'Event Time' (Time of the event), 'Event Type' (Type of the event, also highlighted with a red box), 'ID' (ID for the event), 'Subject' (Subject of the event), and 'Topic' (Resource that fired the event).

6. To save this action, select **Save As** on the left corner of the page. From the right bar that appears, confirm creation of this action.

The screenshot shows the Logic Apps Designer with the 'Save As' dialog open. The 'Name' field is set to 'newLogicApp'. The 'Create' button is highlighted with a red box. To the right, a 'Logic App' configuration pane displays the following details:

- Name: newLogicApp
- Subscription: documentationteam
- Resource group: Use existing (larrygroup1029)
- Location: Central US
- Logic App Status: Enabled
- Log Analytics: Off

Example: Data drift triggers retraining

IMPORTANT

This example relies on a feature (data drift) that is only available when using Azure Machine Learning SDK v1 or Azure CLI extension v1 for Azure Machine Learning. For more information, see [What is Azure ML CLI & SDK v2](#).

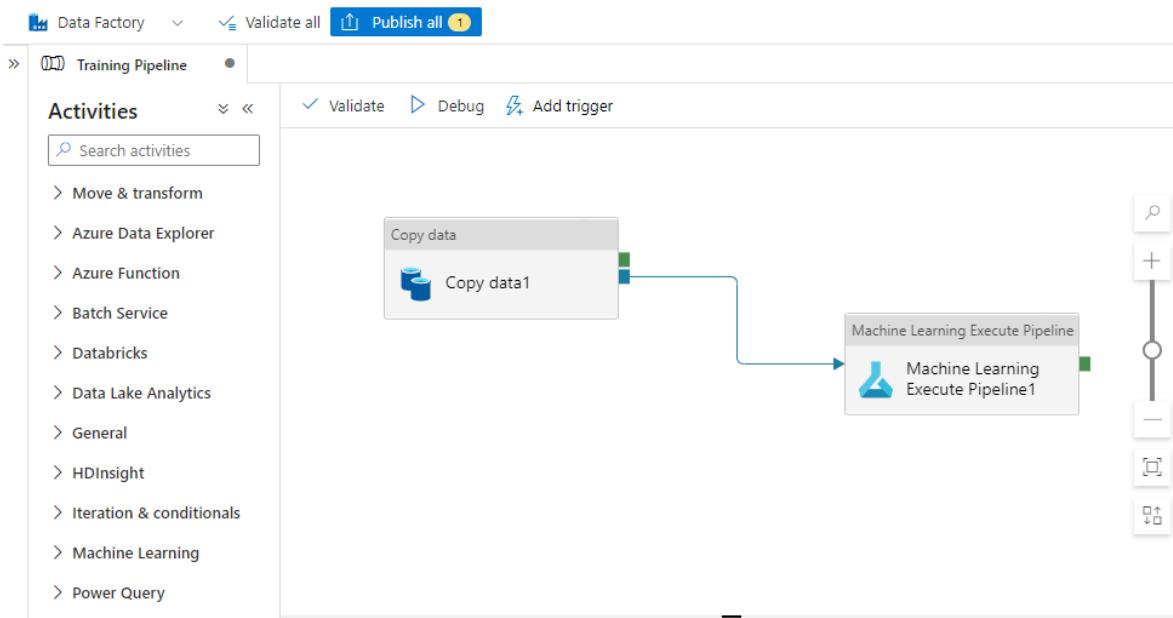
Models go stale over time, and not remain useful in the context it is running in. One way to tell if it's time to retrain the model is detecting data drift.

This example shows how to use event grid with an Azure Logic App to trigger retraining. The example triggers an Azure Data Factory pipeline when data drift occurs between a model's training and serving datasets.

Before you begin, perform the following actions:

- Set up a dataset monitor to [detect data drift](#) in a workspace
- Create a published [Azure Data Factory pipeline](#).

In this example, a simple Data Factory pipeline is used to copy files into a blob store and run a published Machine Learning pipeline. For more information on this scenario, see how to set up a [Machine Learning step in Azure Data Factory](#)



1. Start with creating the logic app. Go to the [Azure portal](#), search for Logic Apps, and select create.



2. Fill in the requested information. To simplify the experience, use the same subscription and resource group as your Azure Data Factory Pipeline and Azure Machine Learning workspace.

The screenshot shows the 'Logic App' creation form. It includes fields for 'Name' (set to 'triggerADF-demo'), 'Subscription' (set to 'documentationteam'), 'Resource group' (radio button selected for 'Create new'), 'Location' (set to 'Central US'), and 'Log Analytics' (set to 'Off'). A note at the bottom states: 'You can add triggers and actions to your Logic App after creation.' At the bottom of the form are 'Create' and 'Automation options' buttons.

- Once you have created the logic app, select **When an Event Grid resource event occurs**.

The screenshot shows the Logic Apps Designer interface with a header "Home > triggerADF-demo > Logic Apps Designer". Below the header, a section titled "Start with a common trigger" displays four options:

- "When a message is received in a Service Bus queue"
- "When a HTTP request is received"
- "When a new tweet is posted"
- "When an Event Grid resource event occurs" (this option is highlighted with a red border)

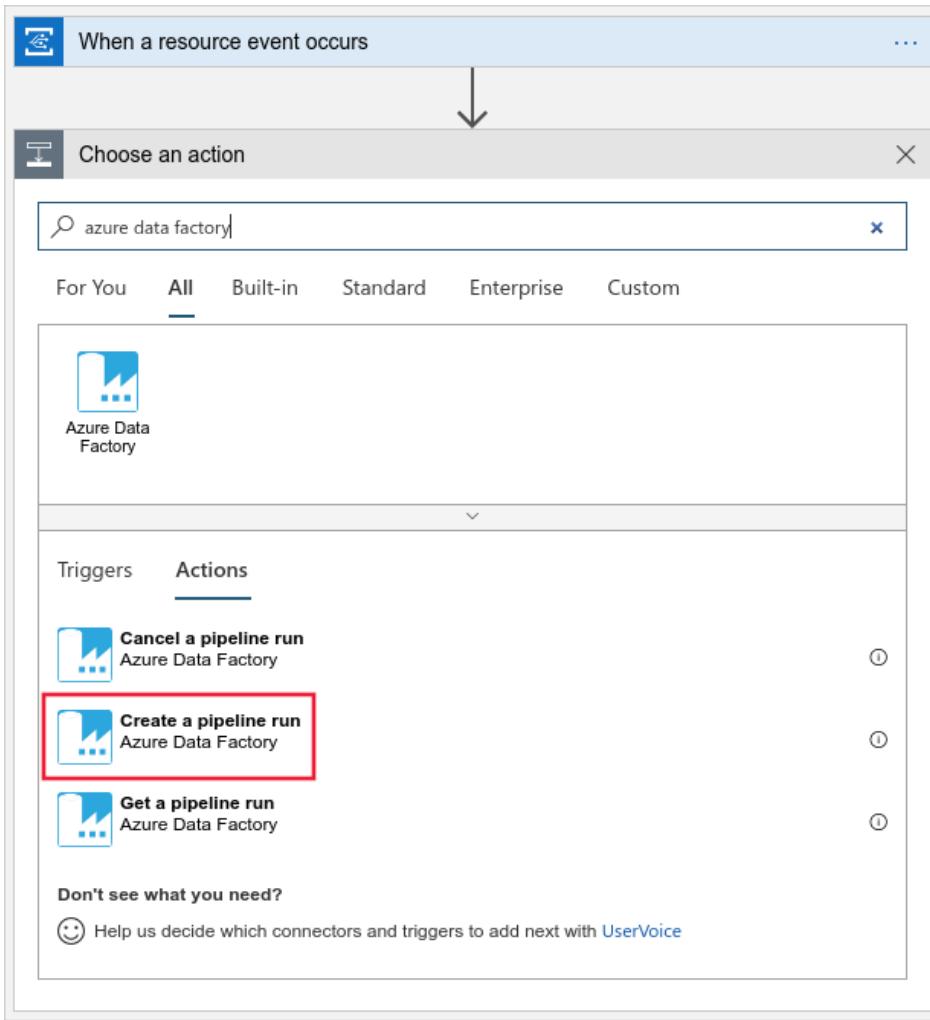
- Login and fill in the details for the event. Set the **Resource Name** to the workspace name. Set the **Event Type** to **DatasetDriftDetected**.

The screenshot shows the configuration of the "When a resource event occurs" trigger in the Logic Apps Designer. The configuration fields are:

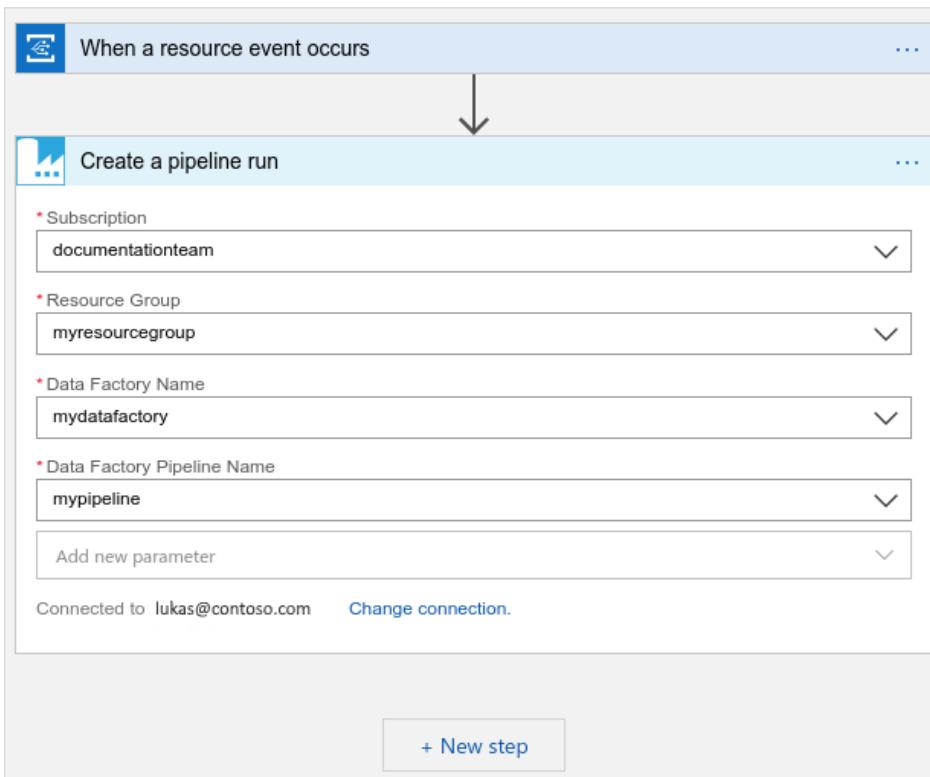
- * Subscription: {topicSubscriptionId}
- * Resource Type: Microsoft.MachineLearningServices.Workspaces
- * Resource Name: /subscriptions/mysubscription/resourceGroups/myresourcegroup/providers/Microsoft.MachineLearningServices/workspaces/myworkspace
- Event Type Item - 1: Microsoft.MachineLearningServices.DatasetDriftDetected (this field is highlighted with a red border)

At the bottom, it says "Connected to larryfr@microsoft.com. [Change connection.](#)". A "+ New step" button is visible at the bottom center.

- Add a new step, and search for **Azure Data Factory**. Select **Create a pipeline run**.



6. Login and specify the published Azure Data Factory pipeline to run.



7. Save and create the logic app using the **save** button on the top left of the page. To view your app, go to your workspace in the [Azure portal](#) and click on **Events**.

Search to filter items...

Name	Endpoint	Prefix Filter	Suffix Filter	Event Types
new-aml-events	EventHub			Microsoft.MachineLearningServices.ModelRe...
LogicApp56c7b13c-7c99-4...	WebHook			Microsoft.MachineLearningServices.Dataset...

Now the data factory pipeline is triggered when drift occurs. View details on your data drift run and machine learning pipeline in [Azure Machine Learning studio](#).

Pipelines

Pipeline jobs Pipeline endpoints Pipeline drafts

Refresh Disable Enable Edit columns Reset view Include disabled

Search Filter Clear all Page size: 25

Name	Description	Date updated
Datadrift pipeline	--	Jun 21, 2022 10:49 AM

Next steps

Learn more about Event Grid and give Azure Machine Learning events a try:

- [About Event Grid](#)
- [Event schema for Azure Machine Learning](#)

Schedule machine learning pipeline jobs (preview)

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

IMPORTANT

SDK v2 is currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In this article, you'll learn how to programmatically schedule a pipeline to run on Azure. You can create a schedule based on elapsed time. Time-based schedules can be used to take care of routine tasks, such as retrain models or do batch predictions regularly to keep them up-to-date. After learning how to create schedules, you'll learn how to retrieve, update and deactivate them via CLI and SDK.

Prerequisites

- You must have an Azure subscription to use Azure Machine Learning. If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.
- [Azure CLI](#)
- [Python SDK](#)
- Install the Azure CLI and the `ml` extension. Follow the installation steps in [Install, set up, and use the CLI \(v2\)](#).
- Create an Azure Machine Learning workspace if you don't have one. For workspace creation, see [Install, set up, and use the CLI \(v2\)](#).

Schedule a pipeline job

To run a pipeline job on a recurring basis, you'll need to create a schedule. A `Schedule` associates a job, and a trigger. The trigger can either be `cron` that use cron expression to describe the wait between runs or `recurrence` that specify using what frequency to trigger job. In each case, you need to define a pipeline job first, it can be existing pipeline jobs or a pipeline job define inline, refer to [Create a pipeline job in CLI](#) and [Create a pipeline job in SDK](#).

You can schedule a pipeline job yaml in local or an existing pipeline job in workspace.

Create a schedule

Create a time-based schedule with recurrence pattern

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO: Azure CLI ml extension v2 (current)

```

$schema: https://azurermschemas.azureedge.net/latest/schedule.schema.json
name: simple_recurrence_job_schedule
display_name: Simple recurrence job schedule
description: a simple hourly recurrence job schedule

trigger:
  type: recurrence
  frequency: day #can be minute, hour, day, week, month
  interval: 1 #every day
  schedule:
    hours: [4,5,10,11,12]
    minutes: [0,30]
  start_time: "2022-07-10T10:00:00" # optional - default will be schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

create_job: ./simple-pipeline-job.yml
# create_job: azureml:simple-pipeline-job

```

`trigger` contains the following properties:

- **(Required)** `type` specifies the schedule type is `recurrence`. It can also be `cron`, see details in the next section.

List continues below.

- **(Required)** `frequency` specifies the unit of time that describes how often the schedule fires. Can be `minute`, `hour`, `day`, `week`, `month`.
- **(Required)** `interval` specifies how often the schedule fires based on the frequency, which is the number of time units to wait until the schedule fires again.
- **(Optional)** `schedule` defines the recurrence pattern, containing `hours`, `minutes`, and `weekdays`.
 - When `frequency` is `day`, pattern can specify `hours` and `minutes`.
 - When `frequency` is `week` and `month`, pattern can specify `hours`, `minutes` and `weekdays`.
 - `hours` should be an integer or a list, from 0 to 23.
 - `minutes` should be an integer or a list, from 0 to 59.
 - `weekdays` can be a string or list from `monday` to `sunday`.
 - If `schedule` is omitted, the job(s) will be triggered according to the logic of `start_time`, `frequency` and `interval`.
- **(Optional)** `start_time` describes the start date and time with timezone. If `start_time` is omitted, `start_time` will be equal to the job created time. If the start time is in the past, the first job will run at the next calculated run time.
- **(Optional)** `end_time` describes the end date and time with timezone. If `end_time` is omitted, the schedule will continue trigger jobs until the schedule is manually disabled.
- **(Optional)** `time_zone` specifies the time zone of the recurrence. If omitted, by default is UTC. To learn more about timezone values, see [appendix for timezone values](#).

Create a time-based schedule with cron expression

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```

$schema: https://azurermschemas.azureedge.net/latest/schedule.schema.json
name: simple_cron_job_schedule
display_name: Simple cron job schedule
description: a simple hourly cron job schedule

trigger:
  type: cron
  expression: "0 * * * *"
  start_time: "2022-07-10T10:00:00" # optional - default will be schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

# create_job: azurerm:simple-pipeline-job
create_job: ./simple-pipeline-job.yml

```

The `trigger` section defines the schedule details and contains following properties:

- **(Required)** `type` specifies the schedule type is `cron`.

List continues below.

- **(Required)** `expression` uses standard crontab expression to express a recurring schedule. A single expression is composed of five space-delimited fields:

`MINUTES HOURS DAYS MONTHS DAYS-OF-WEEK`

- A single wildcard (`*`), which covers all values for the field. So a `*` in days means all days of a month (which varies with month and year).
- The `expression: "15 16 * * 1"` in the sample above means the 16:15PM on every Monday.
- The table below lists the valid values for each field:

FIELD	RANGE	COMMENT
<code>MINUTES</code>	0-59	-
<code>HOURS</code>	0-23	-
<code>DAYS</code>	-	Not supported. The value will be ignored and treat as <code>*</code> .
<code>MONTHS</code>	-	Not supported. The value will be ignored and treat as <code>*</code> .
<code>DAYS-OF-WEEK</code>	0-6	Zero (0) means Sunday. Names of days also accepted.

- To learn more about how to use crontab expression, see [Crontab Expression wiki on GitHub](#).

IMPORTANT

`DAYS` and `MONTH` are not supported. If you pass a value, it will be ignored and treat as `*`.

- **(Optional)** `start_time` specifies the start date and time with timezone of the schedule. `start_time: "2022-05-10T10:15:00-04:00"` means the schedule starts from 10:15:00AM on 2022-05-10 in UTC-4 timezone. If `start_time` is omitted, the `start_time` will be equal to schedule creation time. If the start time is in the past, the first job will run at the next calculated run time.

- (Optional) `end_time` describes the end date and time with timezone. If `end_time` is omitted, the schedule will continue trigger jobs until the schedule is manually disabled.
- (Optional) `time_zone` specifies the time zone of the expression. If omitted, by default is UTC. See [appendix for timezone values](#).

Change runtime settings when defining schedule

When defining a schedule using an existing job, you can change the runtime settings of the job. Using this approach, you can define multi-schedules using the same job with different inputs.

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
$schema: https://azuremlschemas.azureedge.net/latest/schedule.schema.json
name: cron_with_settings_job_schedule
display_name: Simple cron job schedule
description: a simple hourly cron job schedule

trigger:
  type: cron
  expression: "0 * * * *"
  start_time: "2022-07-10T10:00:00" # optional - default will be schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

create_job:
  type: pipeline
  job: ./simple-pipeline-job.yml
  # job: azureml:simple-pipeline-job
  # runtime settings
  settings:
    #default_compute: azureml:cpu-cluster
    continue_on_step_failure: true
  inputs:
    hello_string_top_level_input: ${{name}}
  tags:
    schedule: cron_with_settings_schedule
```

Following properties can be changed when defining schedule:

PROPERTY	DESCRIPTION
settings	A dictionary of settings to be used when running the pipeline job.
inputs	A dictionary of inputs to be used when running the pipeline job.
outputs	A dictionary of outputs to be used when running the pipeline job.
experiment_name	Experiment name of triggered job.

Expressions supported in schedule

When define schedule, we support following expression that will be resolved to real value during job runtime.

EXPRESSION	DESCRIPTION	SUPPORTED PROPERTIES
<code>\${{create_context.trigger_time}}</code>	The time when the schedule is triggered.	String type inputs of pipeline job
<code>\${{name}}</code>	The name of job.	outputs.path of pipeline job

Manage schedule

Create schedule

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

After you create the schedule yaml, you can use the following command to create a schedule via CLI.

```
# This action will create related resources for a schedule. It will take dozens of seconds to complete.
az ml schedule create --file cron-schedule.yml --no-wait
```

Check schedule detail

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml schedule show -n simple_cron_schedule
```

List schedules in a workspace

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml schedule list
```

Update a schedule

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml schedule update -n simple_cron_schedule --set description="new description" --no-wait
```

Disable a schedule

- [Azure CLI](#)
- [Python SDK](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

```
az ml schedule disable -n simple_cron_schedule --no-wait
```

Enable a schedule

- Azure CLI
- Python SDK

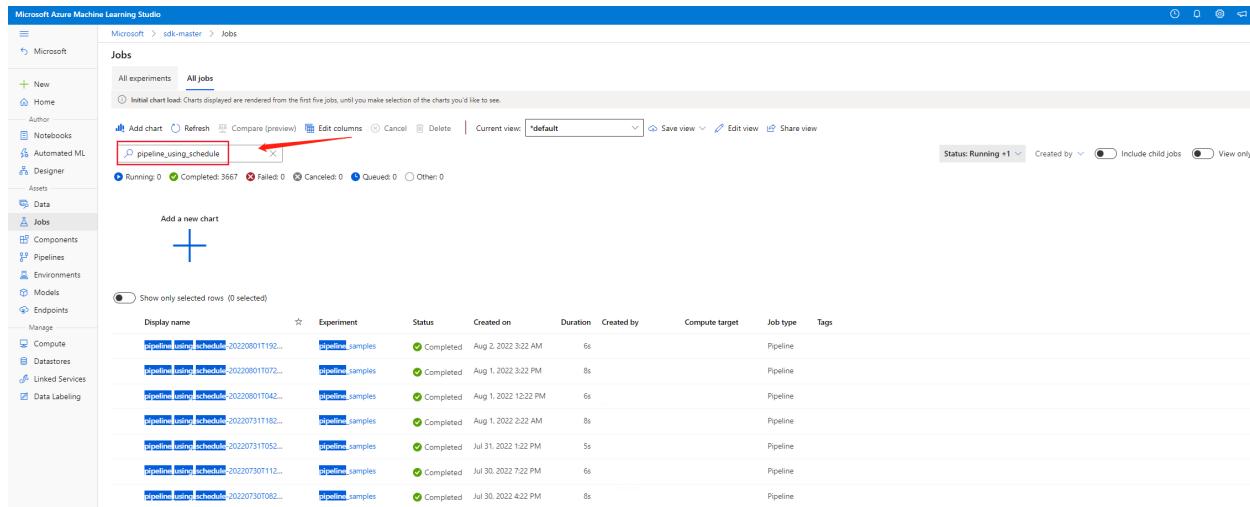
APPLIES TO:  Azure CLI ml extension v2 (current)

```
az ml schedule enable -n simple_cron_schedule --no-wait
```

Query triggered jobs from a schedule

All the display name of jobs triggered by schedule will have the display name as <schedule_name>-YYYYMMDDThhmmssZ. For example, if a schedule with a name of named-schedule is created with a scheduled run every 12 hours starting at 6 AM on Jan 1 2021, then the display names of the jobs created will be as follows:

- named-schedule-20210101T060000Z
- named-schedule-20210101T180000Z
- named-schedule-20210102T060000Z
- named-schedule-20210102T180000Z, and so on



The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a navigation sidebar with options like New, Home, Notebooks, Automated ML, Designer, Data, Jobs, Components, Pipelines, Environments, Models, Endpoints, Compute, Datasets, Linked Services, and Data Labeling. The 'Jobs' option is selected. The main area is titled 'Jobs' and shows a table of triggered jobs. The search bar at the top has 'pipeline_using_schedule' typed into it. The table columns include Display name, Experiment, Status, Created on, Duration, Created by, Compute target, Job type, and Tags. There are seven rows in the table, each corresponding to a job triggered by the specified schedule.

Display name	Experiment	Status	Created on	Duration	Created by	Compute target	Job type	Tags
simple_cron_schedule-20220801T190...	pipeline_samples	Completed	Aug 2, 2022 3:22 AM	6s			Pipeline	
simple_cron_schedule-20220801T072...	pipeline_samples	Completed	Aug 1, 2022 3:22 PM	8s			Pipeline	
simple_cron_schedule-20220801T042...	pipeline_samples	Completed	Aug 1, 2022 12:22 PM	6s			Pipeline	
simple_cron_schedule-20220731T112...	pipeline_samples	Completed	Aug 1, 2022 2:22 AM	8s			Pipeline	
simple_cron_schedule-20220731T052...	pipeline_samples	Completed	Jul 31, 2022 1:22 PM	5s			Pipeline	
simple_cron_schedule-20220730T112...	pipeline_samples	Completed	Jul 30, 2022 7:22 PM	6s			Pipeline	
simple_cron_schedule-20220730T082...	pipeline_samples	Completed	Jul 30, 2022 4:22 PM	8s			Pipeline	

You can also apply [Azure CLI JMESPath query](#) to query the jobs triggered by a schedule name.

```
# query triggered jobs from schedule, please replace the simple_cron_schedule to your schedule name
az ml job list --query "[?contains(display_name,'simple_cron_schedule')]"
```

Delete a schedule

IMPORTANT

A schedule must be disabled to be deleted.

- Azure CLI
- Python SDK

```
az ml schedule delete -n simple_cron_schedule
```

Next steps

- Learn more about the [CLI \(v2\) schedule YAML schema](#).
- Learn how to [create pipeline job](#) in CLI v2.
- Learn how to [create pipeline job](#) in SDK v2.
- Learn more about [CLI \(v2\) core YAML syntax](#).
- Learn more about [Pipelines](#).
- Learn more about [Component](#).

NOTE

Information the user should notice even if skimming

Create and run machine learning pipelines using components with the Azure Machine Learning CLI

9/22/2022 • 11 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

In this article, you learn how to create and run [machine learning pipelines](#) by using the Azure CLI and components (for more, see [What is an Azure Machine Learning component?](#)). You can [create pipelines without using components](#), but components offer the greatest amount of flexibility and reuse. AzureML Pipelines may be defined in YAML and run from the CLI, authored in Python, or composed in AzureML Studio Designer with a drag-and-drop UI. This document focuses on the CLI.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace. [Create workspace resources](#).
- [Install and set up the Azure CLI extension for Machine Learning](#).
- Clone the examples repository:

```
git clone https://github.com/Azure/azureml-examples --depth 1
cd azureml-examples/cli/jobs/pipelines-with-components/basics
```

Suggested pre-reading

- [What is Azure Machine Learning pipeline](#)
- [What is Azure Machine Learning component](#)

Create your first pipeline with component

Let's create your first pipeline with component using an example. This section aims to give you an initial impression of what pipeline and component look like in AzureML with a concrete example.

From the `cli/jobs/pipelines-with-components/basics` directory of the [azureml-examples repository](#), navigate to the `3b_pipeline_with_data` subdirectory. There are three types of files in this directory. Those are the files you'll need to create when building your own pipeline.

- **pipeline.yml**: This YAML file defines the machine learning pipeline. This YAML file describes how to break a full machine learning task into a multistep workflow. For example, considering a simple machine learning task of using historical data to train a sales forecasting model, you may want to build a sequential workflow with data processing, model training, and model evaluation steps. Each step is a component that has well defined interface and can be developed, tested, and optimized independently. The pipeline YAML also defines how the child steps connect to other steps in the pipeline, for example the model training step generate a model file and the model file will pass to a model evaluation step.
- **component.yml**: This YAML file defines the component. It packages following information:
 - Metadata: name, display name, version, description, type etc. The metadata helps to describe and manage the component.

- Interface: inputs and outputs. For example, a model training component will take training data and number of epochs as input, and generate a trained model file as output. Once the interface is defined, different teams can develop and test the component independently.
- Command, code & environment: the command, code and environment to run the component. Command is the shell command to execute the component. Code usually refers to a source code directory. Environment could be an AzureML environment(curated or customer created), docker image or conda environment.
- **component_src**: This is the source code directory for a specific component. It contains the source code that will be executed in the component. You can use your preferred language(Python, R...). The code must be executed by a shell command. The source code can take a few inputs from shell command line to control how this step is going to be executed. For example, a training step may take training data, learning rate, number of epochs to control the training process. The argument of a shell command is used to pass inputs and outputs to the code.

Now let's create a pipeline using the `3b_pipeline_with_data` example. We'll explain the detailed meaning of each file in following sections.

First list your available compute resources with the following command:

```
az ml compute list
```

If you don't have it, create a cluster called `cpu-cluster` by running:

```
az ml compute create -n cpu-cluster --type amlcompute --min-instances 0 --max-instances 10
```

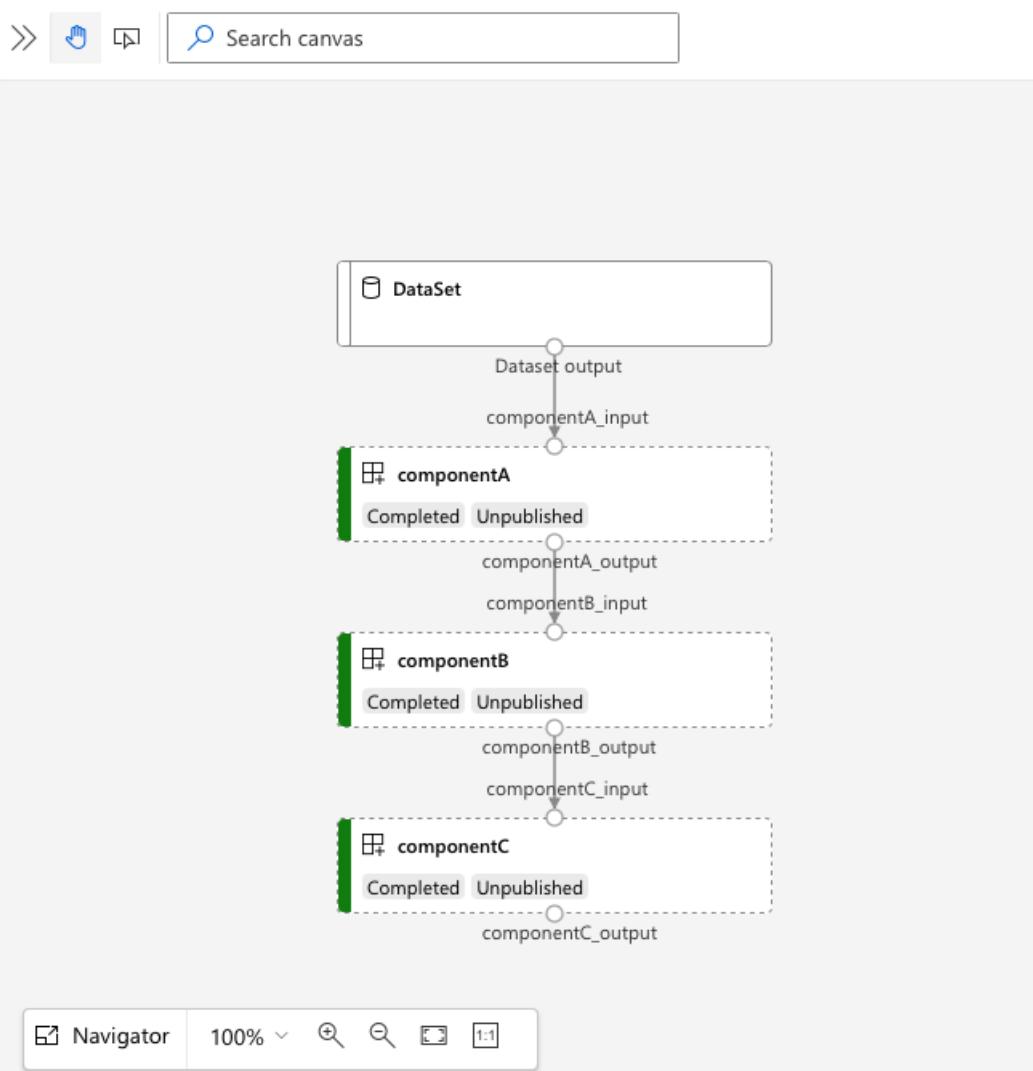
Now, create a pipeline job defined in the `pipeline.yml` file with the following command. The compute target will be referenced in the `pipeline.yml` file as `azureml:cpu-cluster`. If your compute target uses a different name, remember to update it in the `pipeline.yml` file.

```
az ml job create --file pipeline.yml
```

You should receive a JSON dictionary with information about the pipeline job, including:

KEY	DESCRIPTION
<code>name</code>	The GUID-based name of the job.
<code>experiment_name</code>	The name under which jobs will be organized in Studio.
<code>services.Studio.endpoint</code>	A URL for monitoring and reviewing the pipeline job.
<code>status</code>	The status of the job. This will likely be <code>Preparing</code> at this point.

Open the `services.Studio.endpoint` URL you'll see a graph visualization of the pipeline looks like below.



Understand the pipeline definition YAML

Let's take a look at the pipeline definition in the *3b_pipeline_with_data/pipeline.yml* file.

```

$schema: https://azurermschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline

display_name: 3b_pipeline_with_data
description: Pipeline with 3 component jobs with data dependencies

compute: azureml:cpu-cluster

outputs:
  final_pipeline_output:
    mode: rw_mount

jobs:
  component_a:
    type: command
    component: ./componentA.yml
    inputs:
      component_a_input:
        type: uri_folder
        path: ./data

    outputs:
      component_a_output:
        mode: rw_mount
  component_b:
    type: command
    component: ./componentB.yml
    inputs:
      component_b_input: ${{parent.jobs.component_a.outputs.component_a_output}}
    outputs:
      component_b_output:
        mode: rw_mount
  component_c:
    type: command
    component: ./componentC.yml
    inputs:
      component_c_input: ${{parent.jobs.component_b.outputs.component_b_output}}
    outputs:
      component_c_output: ${{parent.outputs.final_pipeline_output}}
      # mode: upload

```

Below table describes the most common used fields of pipeline YAML schema. See [full pipeline YAML schema here](#).

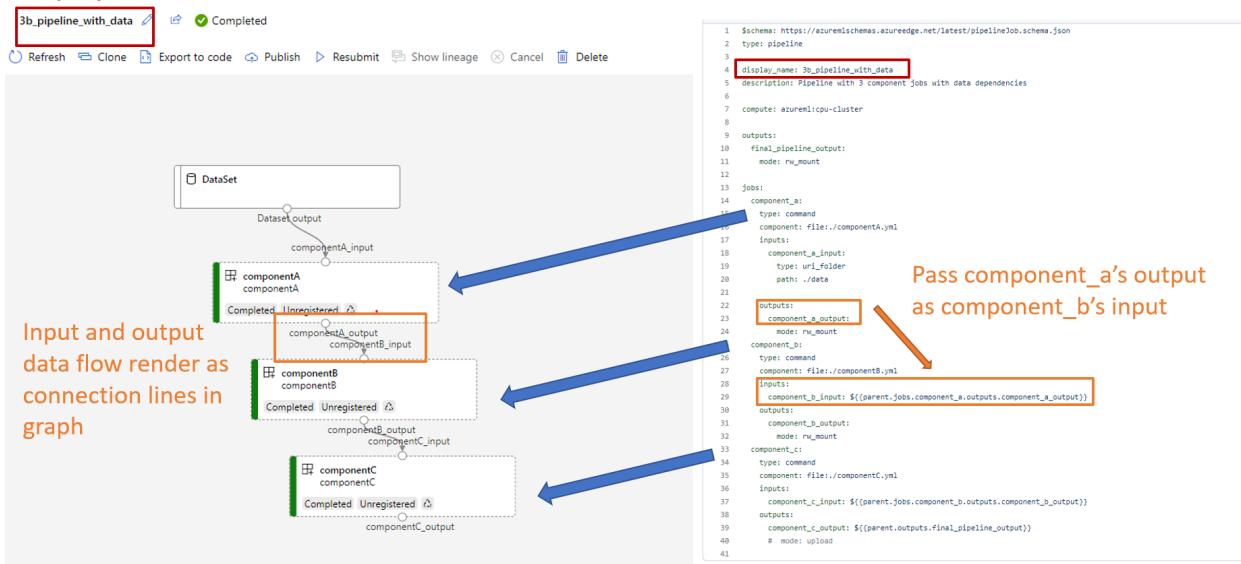
KEY	DESCRIPTION
type	Required. Job type, must be <code>pipeline</code> for pipeline jobs.
display_name	Display name of the pipeline job in Studio UI. Editable in Studio UI. Doesn't have to be unique across all jobs in the workspace.
jobs	Required. Dictionary of the set of individual jobs to run as steps within the pipeline. These jobs are considered child jobs of the parent pipeline job. In this release, supported job types in pipeline are <code>command</code> and <code>sweep</code>
inputs	Dictionary of inputs to the pipeline job. The key is a name for the input within the context of the job and the value is the input value. These pipeline inputs can be referenced by the inputs of an individual step job in the pipeline using the <code>\${{ parent.inputs.<input_name> }}</code> expression.

KEY	DESCRIPTION
outputs	Dictionary of output configurations of the pipeline job. The key is a name for the output within the context of the job and the value is the output configuration. These pipeline outputs can be referenced by the outputs of an individual step job in the pipeline using the \${{ parents.outputs. <output_name> }} expression.

In the `3b_pipeline_with_data` example, we've created a three steps pipeline.

- The three steps are defined under `jobs`. All three step type is command job. Each step's definition is in corresponding `component.yml` file. You can see the component YAML files under `3b_pipeline_with_data` directory. We'll explain the `componentA.yml` in next section.
- This pipeline has data dependency, which is common in most real world pipelines. Component_a takes data input from local folder under `./data` (line 17-20) and passes its output to componentB (line 29). Component_a's output can be referenced as `${{parent.jobs.component_a.outputs.component_a_output}}`.
- The `compute` defines the default compute for this pipeline. If a component under `jobs` defines a different compute for this component, the system will respect component specific setting.

Display name, editable in Studio UI



Read and write data in pipeline

One common scenario is to read and write data in your pipeline. In AzureML, we use the same schema to [read and write data](#) for all type of jobs (pipeline job, command job, and sweep job). Below are pipeline job examples of using data for common scenarios.

- [local data](#)
- [web file with public URL](#)
- [AzureML datastore and path](#)
- [AzureML data asset](#)

Understand the component definition YAML

Now let's look at the `componentA.yml` as an example to understand component definition YAML.

```

$schema: https://azurermschemas.azureedge.net/latest/commandComponent.schema.json
type: command

name: component_a
display_name: componentA
version: 1

inputs:
  component_a_input:
    type: uri_folder

outputs:
  component_a_output:
    type: uri_folder

code: ./componentA_src

environment:
  image: python

command: >-
  python hello.py --componentA_input ${{inputs.component_a_input}} --componentA_output
  ${{outputs.component_a_output}}

```

The most common used schema of the component YAML is described in below table. See [full component YAML schema here](#).

KEY	DESCRIPTION
name	Required. Name of the component. Must be unique across the AzureML workspace. Must start with lowercase letter. Allow lowercase letters, numbers and underscore(_). Maximum length is 255 characters.
display_name	Display name of the component in the studio UI. Can be non-unique within the workspace.
command	Required the command to execute
code	Local path to the source code directory to be uploaded and used for the component.
environment	Required. The environment that will be used to execute the component.
inputs	Dictionary of component inputs. The key is a name for the input within the context of the component and the value is the component input definition. Inputs can be referenced in the command using the \${{ inputs.<input_name> }} expression.
outputs	Dictionary of component outputs. The key is a name for the output within the context of the component and the value is the component output definition. Outputs can be referenced in the command using the \${{ outputs.<output_name> }} expression.

KEY	DESCRIPTION
is_deterministic	Whether to reuse the previous job's result if the component inputs did not change. Default value is <code>true</code> , also known as reuse by default. The common scenario when set as <code>false</code> is to force reload data from a cloud storage or URL.

For the example in `3b_pipeline_with_data/componentA.yml`, componentA has one data input and one data output, which can be connected to other steps in the parent pipeline. All the files under `code` section in component YAML will be uploaded to AzureML when submitting the pipeline job. In this example, files under `./componentA_src` will be uploaded (line 16 in `componentA.yml`). You can see the uploaded source code in Studio UI: double select the ComponentA step and navigate to Snapshot tab, as shown in below screenshot. We can see it's a hello-world script just doing some simple printing, and write current datetime to the `componentA_output` path. The component takes input and output through command line argument, and it's handled in the `hello.py` using `argparse`.

Double click ComponentA on the graph to open the right panel.
Files under `code` in component YAML will be uploaded to snapshot.

```

componentA
Details Parameters Outputs + logs Metrics Child runs Images Explanations (preview) Fairness (preview) Monitoring (preview)
PY hello.py
1 import argparse
2 import os
3 import datetime
4 parser = argparse.ArgumentParser()
5 parser.add_argument('--componentA_input', type=str)
6 parser.add_argument('--componentA_output', type=str)
7 args = parser.parse_args()
8 print("Hello Python World...\\nI'm componentA :-)")
9
10 args = parser.parse_args()
11
12 print("componentA_input path: %s" % args.componentA_input)
13 print("componentA_output path: %s" % args.componentA_output)
14
15 print("files in input path: ")
16 print(os.listdir(args.componentA_input))
17 print(args)
18
19 for filename in args:
20     print("reading file: %s" % filename)
21     with open(os.path.join(args.componentA_input, filename), "r") as handle:
22         print(handle.read())
23
24 cur_time_str = datetime.datetime.now().strftime("%b-%d-%Y-%H-%M-%S")
25
26 print(
27     "writing file: %s"
28     % os.path.join(args.componentA_output, "file-" + cur_time_str + ".txt")
29 )
30 user_file = os.path.join(args.componentA_output, "file-" + cur_time_str + ".txt")
31 with open(user_file, "wt") as file:
32     print(datetime.datetime.now(), file=file)
33
34 print("logging date time: (%s)" % cur_time_str)

```

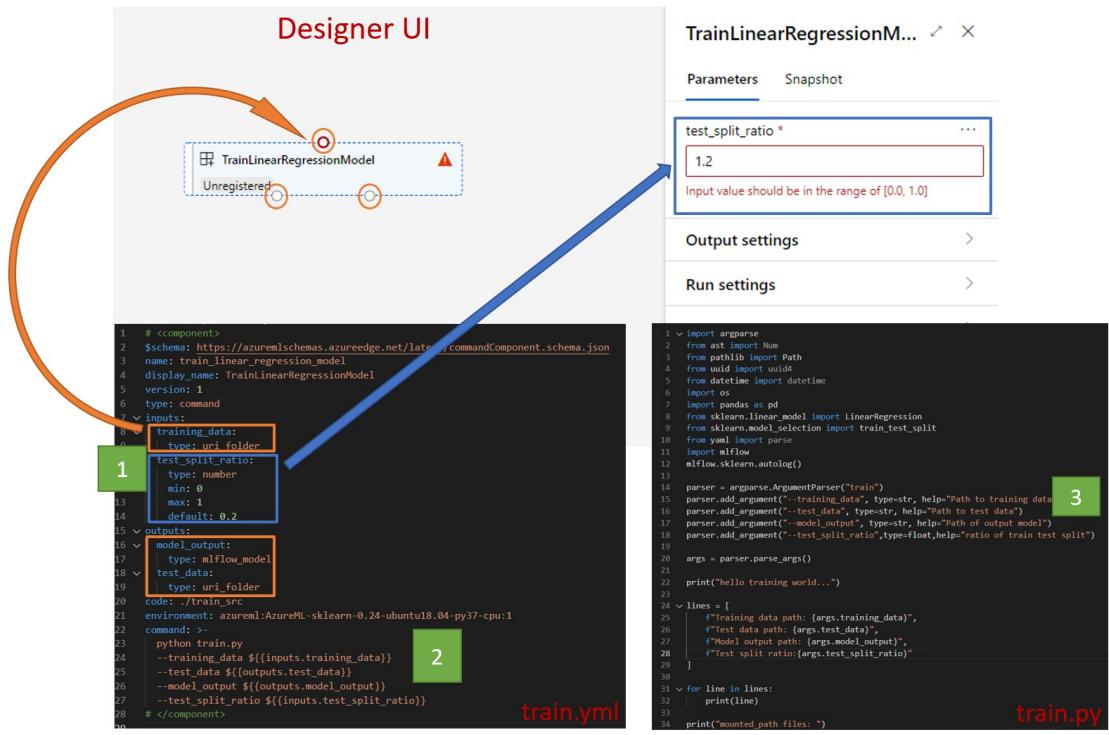
The left sidebar shows the pipeline structure with components A, B, and C. The componentA_src folder is highlighted with a red box. The componentA step in the graph is also highlighted with a red box. The componentA step in the graph has a blue arrow pointing to its 'Snapshot' tab in the details panel.

Input and output

Input and output define the interface of a component. Input and output could be either of a literal value(of type `string`, `number`, `integer`, or `boolean`) or an object containing input schema.

Object input (of type `uri_file`, `uri_folder`, `mltable`, `mlflow_model`, `custom_model`) can connect to other steps in the parent pipeline job and hence pass data/model to other steps. In pipeline graph, the object type input will render as a connection dot.

Literal value inputs (`string`, `number`, `integer`, `boolean`) are the parameters you can pass to the component at run time. You can add default value of literal inputs under `default` field. For `number` and `integer` type, you can also add minimum and maximum value of the accepted value using `min` and `max` fields. If the input value exceeds the min and max, pipeline will fail at validation. Validation happens before you submit a pipeline job to save your time. Validation works for CLI, Python SDK and designer UI. Below screenshot shows a validation example in designer UI. Similarly, you can define allowed values in `enum` field.



If you want to add an input to a component, remember to edit three places: 1) `inputs` field in component YAML 2) `command` field in component YAML. 3) component source code to handle the command line input. It's marked in green box in above screenshot.

Environment

Environment defines the environment to execute the component. It could be an AzureML environment(curated or custom registered), docker image or conda environment. See examples below.

- [AzureML registered environment asset](#). It's referenced in component following `azurerm:<environment-name>:<environment-version>` syntax.
- [public docker image](#)
- [conda file](#) Conda file needs to be used together with a base image.

Register component for reuse and sharing

While some components will be specific to a particular pipeline, the real benefit of components comes from reuse and sharing. Register a component in your Machine Learning workspace to make it available for reuse. Registered components support automatic versioning so you can update the component but assure that pipelines that require an older version will continue to work.

In the `azureml-examples` repository, navigate to the

`cli/jobs/pipelines-with-components/basics/1b_e2e_registered_components` directory.

To register a component, use the `az ml component create` command:

```

az ml component create --file train.yml
az ml component create --file score.yml
az ml component create --file eval.yml

```

After these commands run to completion, you can see the components in Studio, under Asset -> Components:

Microsoft Azure Machine Learning Studio

Display name	Name	Default version	Type	Description	Created on
Score Data	score_data	0.0.2	command	A dummy scoring component	Apr 22, 2022 1:24 PM
Train Model	train_model	0.0.2	command	A dummy training component	Apr 22, 2022 1:24 PM
Eval Model	eval_model	0.0.8	command	A dummy eval component defi...	Apr 22, 2022 1:19 PM

Select a component. You'll see detailed information for each version of the component.

Under **Details** tab, you'll see basic information of the component like name, created by, version etc. You'll see editable fields for Tags and Description. The tags can be used for adding rapidly searched keywords. The description field supports Markdown formatting and should be used to describe your component's functionality and basic use.

Under **Jobs** tab, you'll see the history of all jobs that use this component.

Microsoft Azure Machine Learning Studio

Display name	Name	Default version	Type	Description	Created on
Score Data	score_data	0.0.2	command	A dummy scoring component	Apr 22, 2022 1:24 PM
Train Model	train_model	0.0.2	command	A dummy training component	Apr 22, 2022 1:24 PM
Eval Model	eval_model	0.0.8	command	A dummy eval component defi...	Apr 22, 2022 1:19 PM

Use registered components in a pipeline job YAML file

Let's use `1b_e2e_registered_components` to demo how to use registered component in pipeline YAML. Navigate to `1b_e2e_registered_components` directory, open the `pipeline.yml` file. The keys and values in the `inputs` and `outputs` fields are similar to those already discussed. The only significant difference is the value of the `component` field in the `jobs.<JOB_NAME>.component` entries. The `component` value is of the form `azureml:<COMPONENT_NAME>:<COMPONENT_VERSION>`. The `train-job` definition, for instance, specifies the latest version of the registered component `my_train` should be used:

```

jobs:
  train_job:
    type: command
    component: azureml:my_train@latest
    inputs:
      training_data:
        type: uri_folder
        path: ./data
    max_epochs: ${{parent.inputs.pipeline_job_training_max_epochs}}
    learning_rate: ${{parent.inputs.pipeline_job_training_learning_rate}}
    learning_rate_schedule: ${{parent.inputs.pipeline_job_learning_rate_schedule}}
    outputs:
      model_output: ${{parent.outputs.pipeline_job_trained_model}}

```

Manage components

You can check component details and manage the component using CLI (v2). Use `az ml component -h` to get detailed instructions on component command. Below table lists all available commands. See more examples in [Azure CLI reference](#)

COMMANDS	DESCRIPTION
<code>az ml component create</code>	Create a component
<code>az ml component list</code>	List components in a workspace
<code>az ml component show</code>	Show details of a component
<code>az ml component update</code>	Update a component. Only a few fields(description, display_name) support update
<code>az ml component archive</code>	Archive a component container
<code>az ml component restore</code>	Restore an archived component

Next steps

- Try out [CLI v2 component example](#)

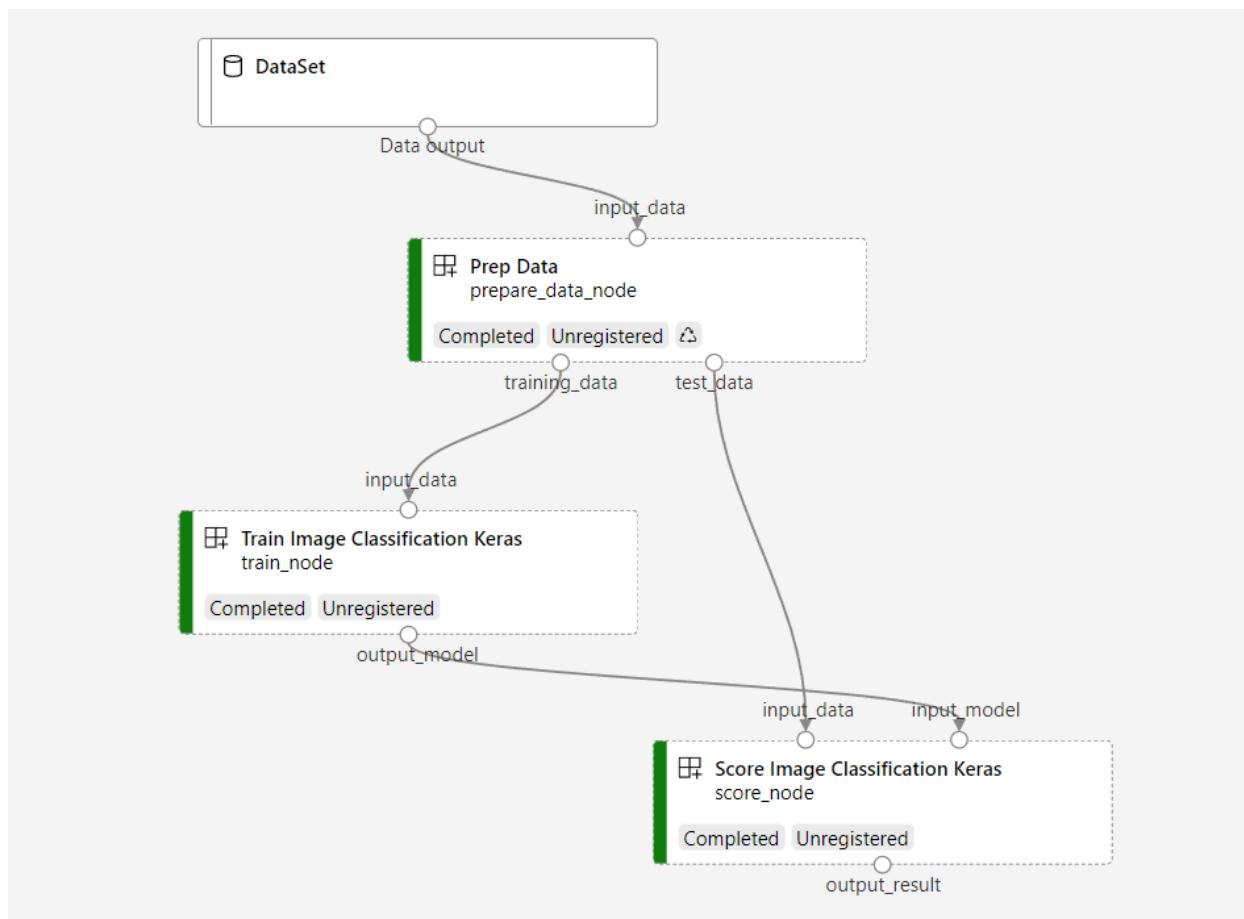
Create and run machine learning pipelines using components with the Azure Machine Learning SDK v2 (Preview)

9/22/2022 • 15 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

In this article, you learn how to build an [Azure Machine Learning pipeline](#) using Python SDK v2 to complete an image classification task containing three steps: prepare data, train an image classification model, and score the model. Machine learning pipelines optimize your workflow with speed, portability, and reuse, so you can focus on machine learning instead of infrastructure and automation.

The example trains a small [Keras](#) convolutional neural network to classify images in the [Fashion MNIST](#) dataset. The pipeline looks like following.



In this article, you complete the following tasks:

- Prepare input data for the pipeline job
- Create three components to prepare the data, train and score
- Compose a Pipeline from the components
- Get access to workspace with compute
- Submit the pipeline job
- Review the output of the components and the trained neural network
- (Optional) Register the component for further reuse and sharing within workspace

If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#) today.

Prerequisites

- Complete the [Quickstart: Get started with Azure Machine Learning](#) if you don't already have an Azure Machine Learning workspace.
- A Python environment in which you've installed Azure Machine Learning Python SDK v2 - [install instructions](#) - check the getting started section. This environment is for defining and controlling your Azure Machine Learning resources and is separate from the environment used at runtime for training.
- Clone examples repository

To run the training examples, first clone the examples repository and change into the `sdk` directory:

```
git clone --depth 1 https://github.com/Azure/azureml-examples --branch sdk-preview  
cd azureml-examples/sdk
```

Start an interactive Python session

This article uses the Python SDK for Azure ML to create and control an Azure Machine Learning pipeline. The article assumes that you'll be running the code snippets interactively in either a Python REPL environment or a Jupyter notebook.

This article is based on the [image_classification_keras_minist_convnet.ipynb](#) notebook found in the `sdk/jobs/pipelines/2e_image_classification_keras_minist_convnet` directory of the [AzureML Examples](#) repository.

Import required libraries

Import all the Azure Machine Learning required libraries that you'll need for this article:

```
# import required libraries  
from azure.identity import DefaultAzureCredential, InteractiveBrowserCredential  
  
from azure.ai.ml import MLClient  
from azure.ai.ml.dsl import pipeline  
from azure.ai.ml import load_component
```

Prepare input data for your pipeline job

You need to prepare the input data for this image classification pipeline.

Fashion-MNIST is a dataset of fashion images divided into 10 classes. Each image is a 28x28 grayscale image and there are 60,000 training and 10,000 test images. As an image classification problem, Fashion-MNIST is harder than the classic MNIST handwritten digit database. It's distributed in the same compressed binary form as the original [handwritten digit database](#).

To define the input data of a job that references the Web-based data, run:

```
from azure.ai.ml import Input

fashion_ds = Input(
    path="wasbs://demo@data4mldemo6150520719.blob.core.windows.net/mnist-fashion/"
)
```

By defining an `Input`, you create a reference to the data source location. The data remains in its existing location, so no extra storage cost is incurred.

Create components for building pipeline

The image classification task can be split into three steps: prepare data, train model and score model.

[Azure Machine Learning component](#) is a self-contained piece of code that does one step in a machine learning pipeline. In this article, you'll create three components for the image classification task:

- Prepare data for training and test
- Train a neural networking for image classification using training data
- Score the model using test data

For each component, you need to prepare the following staff:

1. Prepare the python script containing the execution logic
2. Define the interface of the component,
3. Add other metadata of the component, including run-time environment, command to run the component, and etc.

The next section will show create components in two different ways: the first two components using python function and the third component using yaml definition.

Create the data-preparation component

The first component in this pipeline will convert the compressed data files of `fashion_ds` into two csv files, one for training and the other for scoring. You'll use python function to define this component.

If you're following along with the example in the [AzureML Examples repo](#), the source files are already available in `prep/` folder. This folder contains two files to construct the component: `prep_component.py`, which defines the component and `conda.yaml`, which defines the run-time environment of the component.

Define component using python function

By using `command_component()` function as a decorator, you can easily define the component's interface, metadata and code to execute from a python function. Each decorated Python function will be transformed into a single static specification (YAML) that the pipeline service can process.

```

# Converts MNIST-formatted files at the passed-in input path to training data output path and test data
output path
import os
from pathlib import Path
from mlDesigner import command_component, Input, Output


@command_component(
    name="prep_data",
    version="1",
    display_name="Prep Data",
    description="Convert data to CSV file, and split to training and test data",
    environment=dict(
        conda_file=Path(__file__).parent / "conda.yaml",
        image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    ),
)
def prepare_data_component(
    input_data: Input(type="uri_folder"),
    training_data: Output(type="uri_folder"),
    test_data: Output(type="uri_folder"),
):
    convert(
        os.path.join(input_data, "train-images-idx3-ubyte"),
        os.path.join(input_data, "train-labels-idx1-ubyte"),
        os.path.join(training_data, "mnist_train.csv"),
        60000,
    )
    convert(
        os.path.join(input_data, "t10k-images-idx3-ubyte"),
        os.path.join(input_data, "t10k-labels-idx1-ubyte"),
        os.path.join(test_data, "mnist_test.csv"),
        10000,
    )

def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    l = open(labelf, "rb")
    o = open(outf, "w")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28 * 28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image) + "\n")

    f.close()
    o.close()
    l.close()

```

The code above define a component with display name `Prep Data` using `@command_component` decorator:

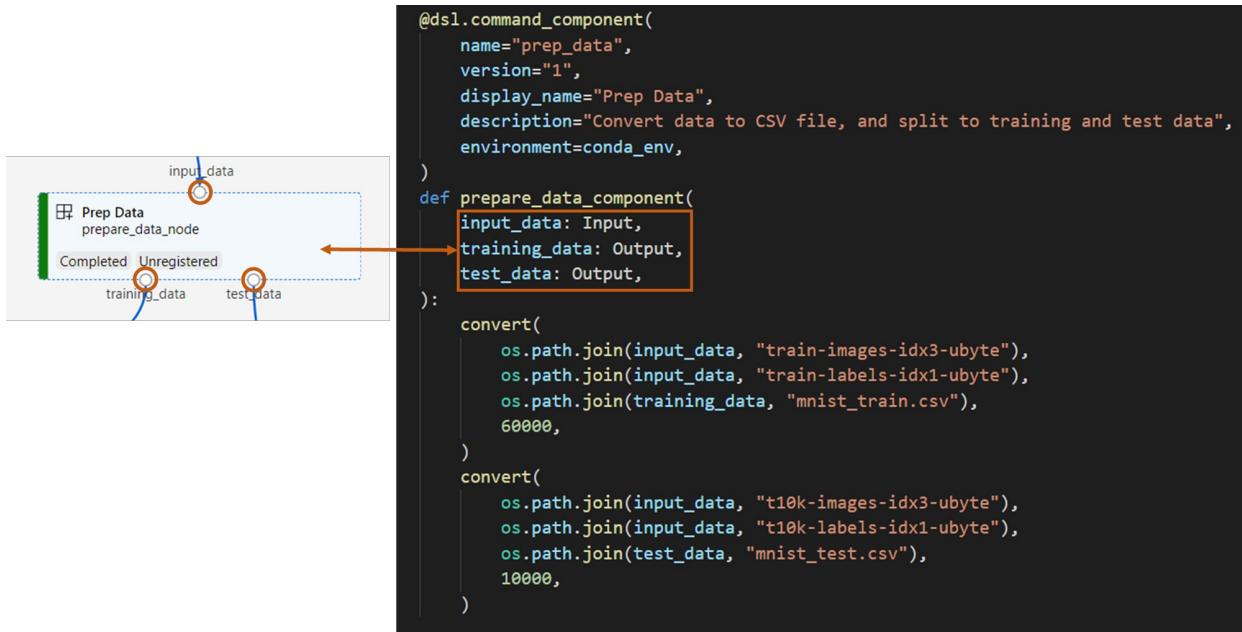
- `name` is the unique identifier of the component.
- `version` is the current version of the component. A component can have multiple versions.
- `display_name` is a friendly display name of the component in UI, which isn't unique.
- `description` usually describes what task this component can complete.
- `environment` specifies the run-time environment for this component. The environment of this component

specifies a docker image and refers to the `conda.yaml` file.

- The `prepare_data_component` function defines one input for `input_data` and two outputs for `training_data` and `test_data`. `input_data` is input data path. `training_data` and `test_data` are output data paths for training data and test data.
- This component converts the data from `input_data` into a training data csv to `training_data` and a test data csv to `test_data`.

Following is what a component looks like in the studio UI.

- A component is a block in a pipeline graph.
- The `input_data`, `training_data` and `test_data` are ports of the component, which connects to other components for data streaming.



Specify component run-time environment

You'll need to modify the runtime environment in which your component runs.

```
@command_component(
    name="prep_data",
    version="1",
    display_name="Prep Data",
```

The above code creates an object of `Environment` class, which represents the runtime environment in which the component runs.

The `conda.yaml` file contains all packages used for the component like following:

```
name: imagekeras_prep_conda_env
channels:
  - defaults
dependencies:
  - python=3.7.11
  - pip=20.0
  - pip:
    - mlDesigner==0.1.0b4
```

Now, you've prepared all source files for the `Prep Data` component.

Create the train-model component

In this section, you'll create a component for training the image classification model in the python function like the `Prep Data` component.

The difference is that since the training logic is more complicated, you can put the original training code in a separate Python file.

The source files of this component are under `train/` folder in the [AzureML Examples repo](#). This folder contains three files to construct the component:

- `train.py` : contains the actual logic to train model.
- `train_component.py` : defines the interface of the component and imports the function in `train.py`.
- `conda.yaml` : defines the run-time environment of the component.

Get a script containing execution logic

The `train.py` file contains a normal python function, which performs the training model logic to train a Keras neural network for image classification. You can find the code [here](#).

Define component using python function

After defining the training function successfully, you can use `@command_component` in Azure Machine Learning SDK v2 to wrap your function as a component, which can be used in AzureML pipelines.

```
import os
from pathlib import Path
from mlDesigner import command_component, Input, Output

@command_component(
    name="train_image_classification_keras",
    version="1",
    display_name="Train Image Classification Keras",
    description="train image classification with keras",
    environment=dict(
        conda_file=Path(__file__).parent / "conda.yaml",
        image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    ),
)
def keras_train_component(
    input_data: Input(type="uri_folder"),
    output_model: Output(type="uri_folder"),
    epochs=10,
):
    # avoid dependency issue, execution logic is in train() func in train.py file
    from train import train

    train(input_data, output_model, epochs)
```

The code above define a component with display name `Train Image Classification Keras` using `@command_component`:

- The `keras_train_component` function defines one input `input_data` where training data comes from, one input `epochs` specifying epochs during training, and one output `output_model` where outputs the model file. The default value of `epochs` is 10. The execution logic of this component is from `train()` function in `train.py` above.

Specify component run-time environment

The train-model component has a slightly more complex configuration than the prep-data component. The `conda.yaml` is like following:

```
name: imagekeras_train_conda_env
channels:
  - defaults
dependencies:
  - python=3.7.11
  - pip=20.0
  - pip:
    - mldesigner==0.1.0b4
    - azureml-mlflow
    - tensorflow==2.7.0
    - numpy==1.21.4
    - scikit-learn==1.0.1
    - pandas==1.3.4
    - matplotlib==3.2.2
    - protobuf==3.20.0
```

Now, you've prepared all source files for the `Train Image Classification Keras` component.

Create the score-model component

In this section, other than the previous components, you'll create a component to score the trained model via Yaml specification and script.

If you're following along with the example in the [AzureML Examples repo](#), the source files are already available in `score/` folder. This folder contains three files to construct the component:

- `score.py` : contains the source code of the component.
- `score.yaml` : defines the interface and other details of the component.
- `conda.yaml` : defines the run-time environment of the component.

Get a script containing execution logic

The `score.py` file contains a normal python function, which performs the training model logic.

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.utils import to_categorical
from keras.callbacks import Callback
from keras.models import load_model

import argparse
from pathlib import Path
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import mlflow

def get_file(f):

    f = Path(f)
    if f.is_file():
        return f
    else:
        files = list(f.iterdir())
        if len(files) == 1:
            return files[0]
        else:
            raise Exception("*****This path contains more than one file*****")
```

```

def parse_args():
    # setup argparse
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument(
        "--input_data", type=str, help="path containing data for scoring"
    )
    parser.add_argument(
        "--input_model", type=str, default="./", help="input path for model"
    )

    parser.add_argument(
        "--output_result", type=str, default="./", help="output path for model"
    )

# parse args
args = parser.parse_args()

# return args
return args

def score(input_data, input_model, output_result):

    test_file = get_file(input_data)
    data_test = pd.read_csv(test_file, header=None)

    img_rows, img_cols = 28, 28
    input_shape = (img_rows, img_cols, 1)

    # Read test data
    X_test = np.array(data_test.iloc[:, 1:])
    y_test = to_categorical(np.array(data_test.iloc[:, 0]))
    X_test = (
        X_test.reshape(X_test.shape[0], img_rows, img_cols, 1).astype("float32") / 255
    )

    # Load model
    files = [f for f in os.listdir(input_model) if f.endswith(".h5")]
    model = load_model(input_model + "/" + files[0])

    # Log metrics of the model
    eval = model.evaluate(X_test, y_test, verbose=0)

    mlflow.log_metric("Final test loss", eval[0])
    print("Test loss:", eval[0])

    mlflow.log_metric("Final test accuracy", eval[1])
    print("Test accuracy:", eval[1])

    # Score model using test data
    y_predict = model.predict(X_test)
    y_result = np.argmax(y_predict, axis=1)

    # Output result
    np.savetxt(output_result + "/predict_result.csv", y_result, delimiter=",")

def main(args):
    score(args.input_data, args.input_model, args.output_result)

# run script
if __name__ == "__main__":
    # parse args
    args = parse_args()

    # call main function

```

```
main(args)
```

The code in score.py takes three command-line arguments: `input_data`, `input_model` and `output_result`. The program score the input model using input data and then output the scoring result.

Define component via Yaml

In this section, you'll learn to create a component specification in the valid YAML component specification format. This file specifies the following information:

- Metadata: name, display_name, version, type, and so on.
- Interface: inputs and outputs
- Command, code, & environment: The command, code, and environment used to run the component

```
$schema: https://azurermschemas.azureedge.net/latest/commandComponent.schema.json
type: command

name: score_image_classification_keras
display_name: Score Image Classification Keras
inputs:
  input_data:
    type: uri_folder
  input_model:
    type: uri_folder
outputs:
  output_result:
    type: uri_folder
code: ../
command: python score.py --input_data ${{inputs.input_data}} --input_model ${{inputs.input_model}} --
output_result ${{outputs.output_result}}
environment:
  conda_file: ./conda.yaml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04
```

- `name` is the unique identifier of the component. Its display name is `Score Image Classification Keras`.
- This component has two inputs and one output.
- The source code path of it's defined in the `code` section and when the component is run in cloud, all files from that path will be uploaded as the snapshot of this component.
- The `command` section specifies the command to execute while running this component.
- The `environment` section contains a docker image and a conda yaml file.

Specify component run-time environment

The score component uses the same image and conda.yaml file as the train component. The source file is in the [sample repository](#).

Now, you've got all source files for score-model component.

Load components to build pipeline

For prep-data component and train-model component defined by python function, you can import the components just like normal python functions.

In the following code, you import `prepare_data_component()` and `keras_train_component()` function from the `prep_component.py` file under `prep` folder and `train_component` file under `train` folder respectively.

```
%load_ext autoreload
%autoreload 2

# load component function from component python file
from prep.prep_component import prepare_data_component
from train.train_component import keras_train_component

# print hint of components
help(prepare_data_component)
help(keras_train_component)
```

For score component defined by yaml, you can use `load_component()` function to load.

```
# load component function from yaml
keras_score_component = load_component(path="./score/score.yaml")
```

Build your pipeline

Now that you've created and loaded all components and input data to build the pipeline. You can compose them into a pipeline:

```
# define a pipeline containing 3 nodes: Prepare data node, train node, and score node
@pipeline(
    default_compute=cpu_compute_target,
)
def image_classification_keras_minist_convnet(pipeline_input_data):
    """E2E image classification pipeline with keras using python sdk."""
    prepare_data_node = prepare_data_component(input_data=pipeline_input_data)

    train_node = keras_train_component(
        input_data=prepare_data_node.outputs.training_data
    )
    train_node.compute = gpu_compute_target

    score_node = keras_score_component(
        input_data=prepare_data_node.outputs.test_data,
        input_model=train_node.outputs.output_model,
    )

# create a pipeline
pipeline_job = image_classification_keras_minist_convnet(pipeline_input_data=fashion_ds)
```

The pipeline has a default compute `cpu_compute_target`, which means if you don't specify compute for a specific node, that node will run on the default compute.

The pipeline has a pipeline level input `pipeline_input_data`. You can assign value to pipeline input when you submit a pipeline job.

The pipeline contains three nodes, `prepare_data_node`, `train_node` and `score_node`.

- The `input_data` of `prepare_data_node` uses the value of `pipeline_input_data`.
- The `input_data` of `train_node` is from the `training_data` output of the `prepare_data_node`.
- The `input_data` of `score_node` is from the `test_data` output of `prepare_data_node`, and the `input_model` is from the `output_model` of `train_node`.
- Since `train_node` will train a CNN model, you can specify its compute as the `gpu_compute_target`, which

can improve the training performance.

Submit your pipeline job

Now you've constructed the pipeline, you can submit to your workspace. To submit a job, you need to firstly connect to a workspace.

Get access to your workspace

Configure credential

We'll use `DefaultAzureCredential` to get access to workspace. `DefaultAzureCredential` to get access to workspace.

`DefaultAzureCredential` should be capable of handling most Azure SDK authentication scenarios.

Reference for more available credentials if it doesn't work for you: [configure credential example](#), [azure-identity reference doc](#).

```
try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()
```

Get a handle to a workspace with compute

Create a `MLClient` object to manage Azure Machine Learning services.

```
# Get a handle to workspace
ml_client = MLClient.from_config(credential=credential)

# Retrieve an already attached Azure Machine Learning Compute.
cpu_compute_target = "cpu-cluster"
print(ml_client.compute.get(cpu_compute_target))
gpu_compute_target = "gpu-cluster"
print(ml_client.compute.get(gpu_compute_target))
```

IMPORTANT

This code snippet expects the workspace configuration json file to be saved in the current directory or its parent. For more information on creating a workspace, see [Create workspace resources](#). For more information on saving the configuration to file, see [Create a workspace configuration file](#).

Submit pipeline job to workspace

Now you've get a handle to your workspace, you can submit your pipeline job.

```
pipeline_job = ml_client.jobs.create_or_update(
    pipeline_job, experiment_name="pipeline_samples"
)
pipeline_job
```

The code above submit this image classification pipeline job to experiment called `pipeline_samples`. It will auto create the experiment if not exists. The `pipeline_input_data` uses `fashion_ds`.

The call to `pipeline_job` produces output similar to:

The call to `submit` the `Experiment` completes quickly, and produces output similar to:

EXPERIMENT	NAME	TYPE	STATUS	DETAILS PAGE
pipeline_samples	sharp_pipe_4gvqx6h 1fb	pipeline	Preparing	Link to Azure Machine Learning studio.

You can monitor the pipeline run by opening the link or you can block until it completes by running:

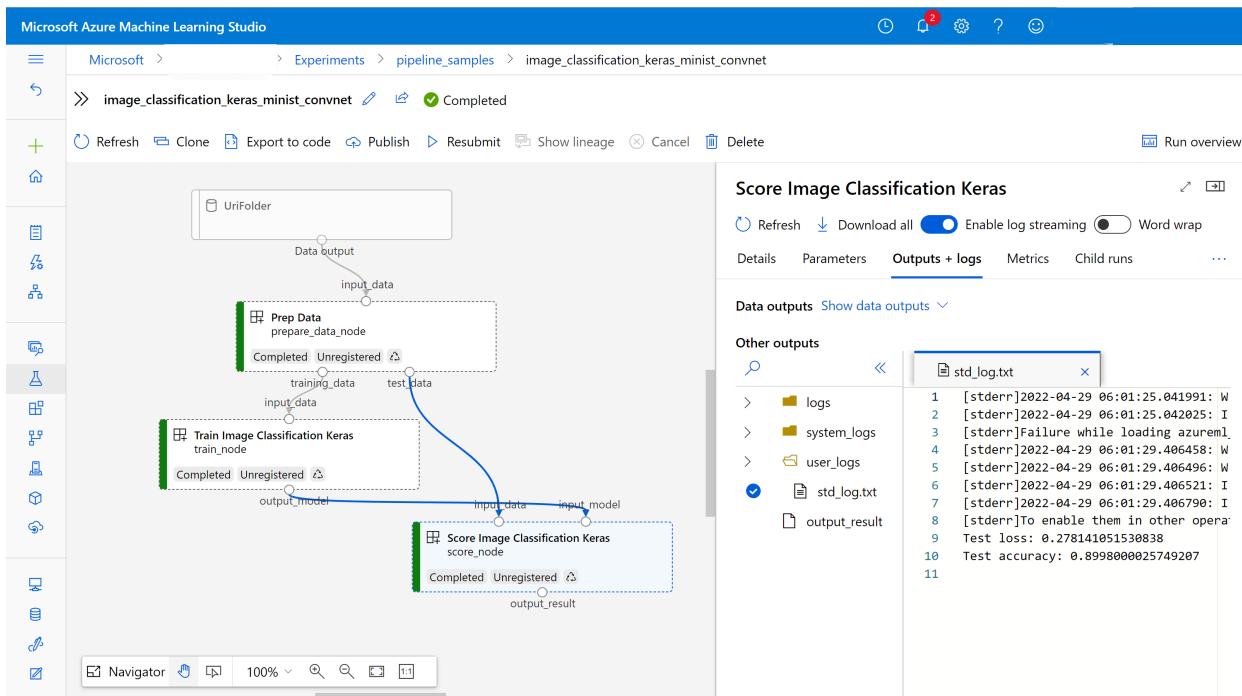
```
# wait until the job completes
ml_client.jobs.stream(pipeline_job.name)
```

IMPORTANT

The first pipeline run takes roughly *15 minutes*. All dependencies must be downloaded, a Docker image is created, and the Python environment is provisioned and created. Running the pipeline again takes significantly less time because those resources are reused instead of created. However, total run time for the pipeline depends on the workload of your scripts and the processes that are running in each pipeline step.

Checkout outputs and debug your pipeline in UI

You can open the [Link to Azure Machine Learning studio](#), which is the job detail page of your pipeline. You'll see the pipeline graph like following.



You can check the logs and outputs of each component by right clicking the component, or select the component to open its detail pane. To learn more about how to debug your pipeline in UI, see [How to use studio UI to build and debug Azure ML pipelines](#).

(Optional) Register components to workspace

In the previous section, you have built a pipeline using three components to E2E complete an image classification task. You can also register components to your workspace so that they can be shared and resued within the workspace. Following is an example to register prep-data component.

```
try:  
    # try get back the component  
    prep = ml_client.components.get(name="prep_data", version="1")  
except:  
    # if not exists, register component using following code  
    prep = ml_client.components.create_or_update(prepare_data_component)  
  
# list all components registered in workspace  
for c in ml_client.components.list():  
    print(c)
```

Using `ml_client.components.get()`, you can get a registered component by name and version. Using `ml_client.components.create_or_update()`, you can register a component previously loaded from python function or yaml.

Next steps

- For more examples of how to build pipelines by using the machine learning SDK, see the [example repository](#).
- For how to use studio UI to submit and debug your pipeline, refer to [how to create pipelines using component in the UI](#).
- For how to use Azure Machine Learning CLI to create components and pipelines, refer to [how to create pipelines using component with CLI](#).

Create and run machine learning pipelines using components with the Azure Machine Learning studio (Preview)

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

In this article, you'll learn how to create and run [machine learning pipelines](#) by using the Azure Machine Learning studio and [Components](#). You can [create pipelines without using components](#), but components offer better amount of flexibility and reuse. Azure ML Pipelines may be defined in YAML and [run from the CLI](#), [authored in Python](#), or composed in Azure ML Studio Designer with a drag-and-drop UI. This document focuses on the AzureML studio designer UI.

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace[Create workspace resources](#).
- [Install and set up the Azure CLI extension for Machine Learning](#).
- Clone the examples repository:

```
git clone https://github.com/Azure/azureml-examples --depth 1
cd azureml-examples/cli/jobs/pipelines-with-components/
```

Register component in your workspace

To build pipeline using components in UI, you need to register components to your workspace first. You can use CLI or SDK to register components to your workspace, so that you can share and reuse the component within the workspace. Registered components support automatic versioning so you can update the component but assure that pipelines that require an older version will continue to work.

In the example below take using CLI for example. If you want to learn more about how to build a component, see [Create and run pipelines using components with CLI](#).

1. From the `cli/jobs/pipelines-with-components/basics` directory of the [azureml-examples](#) repository, navigate to the `1b_e2e_registered_components` subdirectory.
2. Register the components to AzureML workspace using following commands. Learn more about [ML components](#).

```
az ml component create --file train.yml
az ml component create --file score.yml
az ml component create --file eval.yml
```

3. After register component successfully, you can see your component in the studio UI.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The left sidebar has a 'Components' section. The main area is titled 'Components' with a sub-section 'Components are basic building blocks to perform a specific task (e.g. data processing, model training, scoring, etc.) with predefined input/output ports, parameters and environment that can be reused in different pipelines.' Below this is a search bar and filter options ('Created by', 'Created on: Last 30 minutes', 'All filters', 'Clear all'). A table lists three components:

Display name	Name	Default version	Type	Description	Created on
Eval	my_eval	1	command		May 9, 2022 4:56 PM
Score	my_score	1	command		May 9, 2022 4:55 PM
Train_upper_case	my_train	1	command		May 9, 2022 4:55 PM

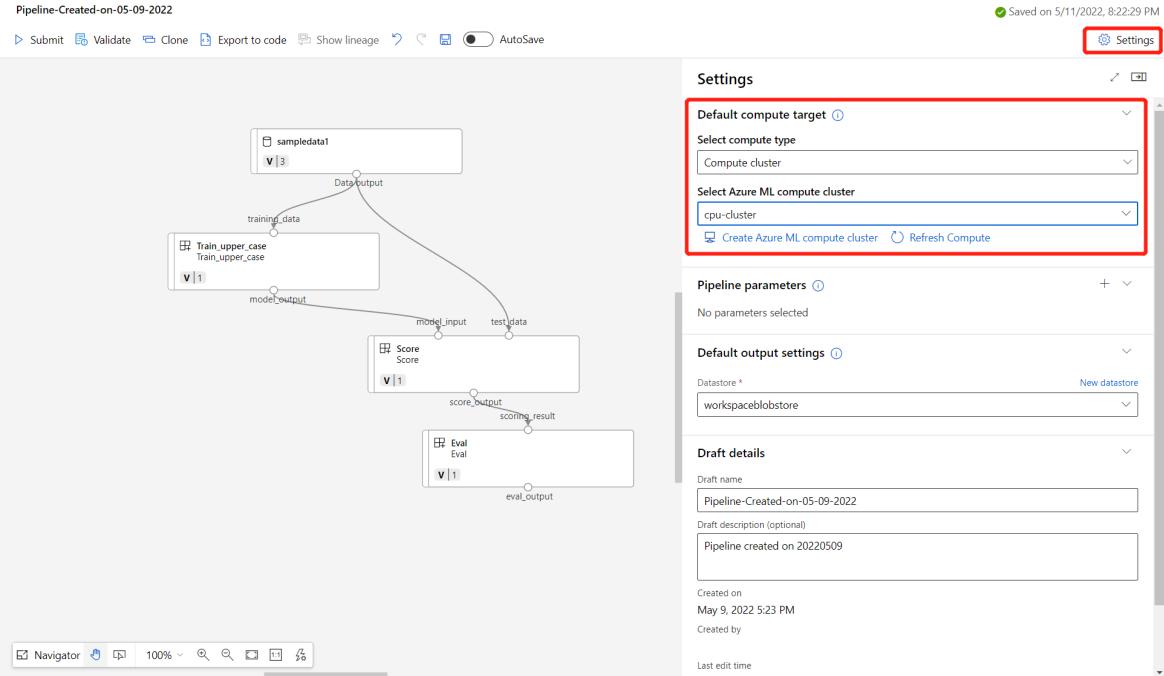
Create pipeline using registered component

1. Create a new pipeline in the designer.

The screenshot shows the Microsoft Azure Machine Learning Studio Designer interface. On the left, there's a sidebar with sections like 'New', 'Designer' (which is selected), 'Data', 'Jobs', 'Components', 'Pipelines', 'Environments', 'Models', 'Endpoints', and 'Manage'. The main area has a 'New pipeline' section with a large red box around a '+' button. Below it are several pre-built pipeline samples. The 'Pipelines' section shows a list of 'Pipeline drafts' with 315 items. The table headers are 'Name', 'Pipeline type', 'Updated on', and 'Created by'. One item in the list is 'Regression - Automobile Price Prediction (Basic)-real time inference'.

2. Set the default compute target of the pipeline.

Select the **Gear icon** at the top right of the canvas to open the **Settings** pane. Select the default compute target for your pipeline.



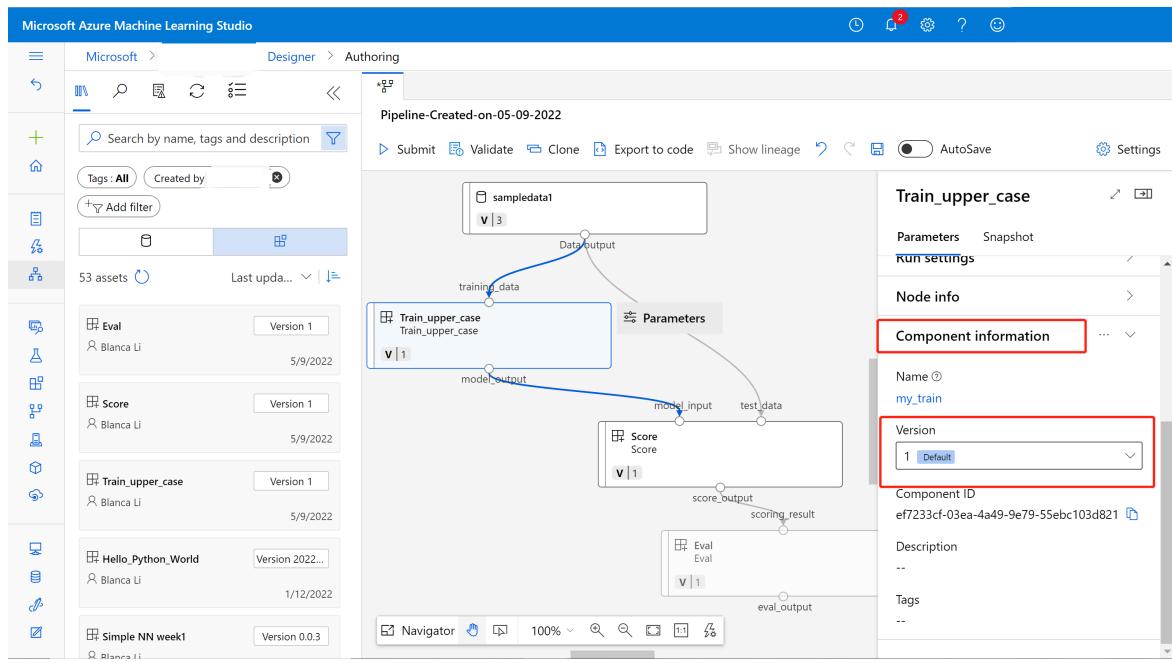
IMPORTANT

Attached compute is not supported, use [compute instances or clusters](#) instead.

3. In asset library, you can see **Data assets** and **Components** tabs. Switch to **Components** tab, you can see the components registered from previous section.

The screenshot shows the Microsoft Azure Machine Learning Studio Asset Library. The left sidebar has 'Automated ML' selected. The main area shows the 'Components' tab, which lists various registered components: 'Eval' (Blanca U, Version 1, 5/9/2022), 'Score' (Blanca U, Version 1, 5/9/2022), 'Train_upper_case' (Blanca U, Version 1, 5/9/2022), 'Filter Based Feature Selection' (Microsoft, Version 1, 5/9/2022), 'K-Means Clustering' (Microsoft, Version 1, 5/9/2022), and 'Linear Regression' (Microsoft, Version 1, 5/9/2022). The 'Components' tab is highlighted with a red box. The pipeline designer canvas is visible at the bottom.

Drag the components and drop on the canvas. By default it will use the default version of the component, and you can change to a specific version in the right pane of component if your component has multiple versions.

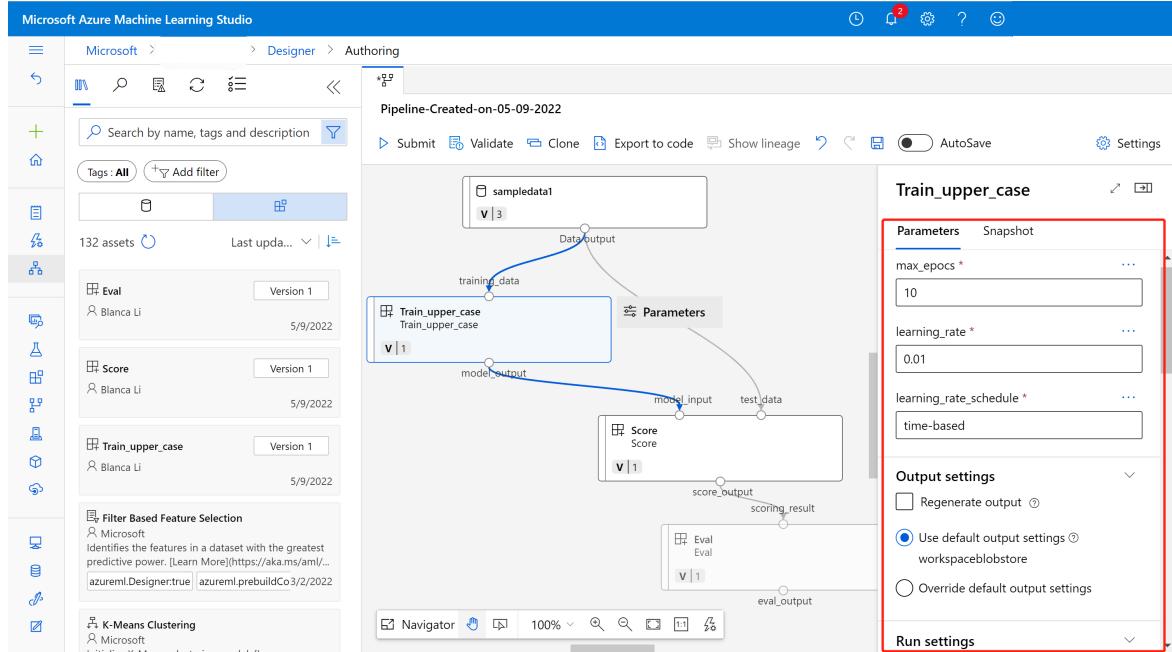


4. Connect the upstream component output ports to the downstream component input ports.

5. Select one component, you'll see a right pane where you can configure the component.

For components with primitive type inputs like number, integer, string and boolean, you can change values of such inputs in the component detailed pane.

You can also change the output settings and compute target where this component run in the right pane.

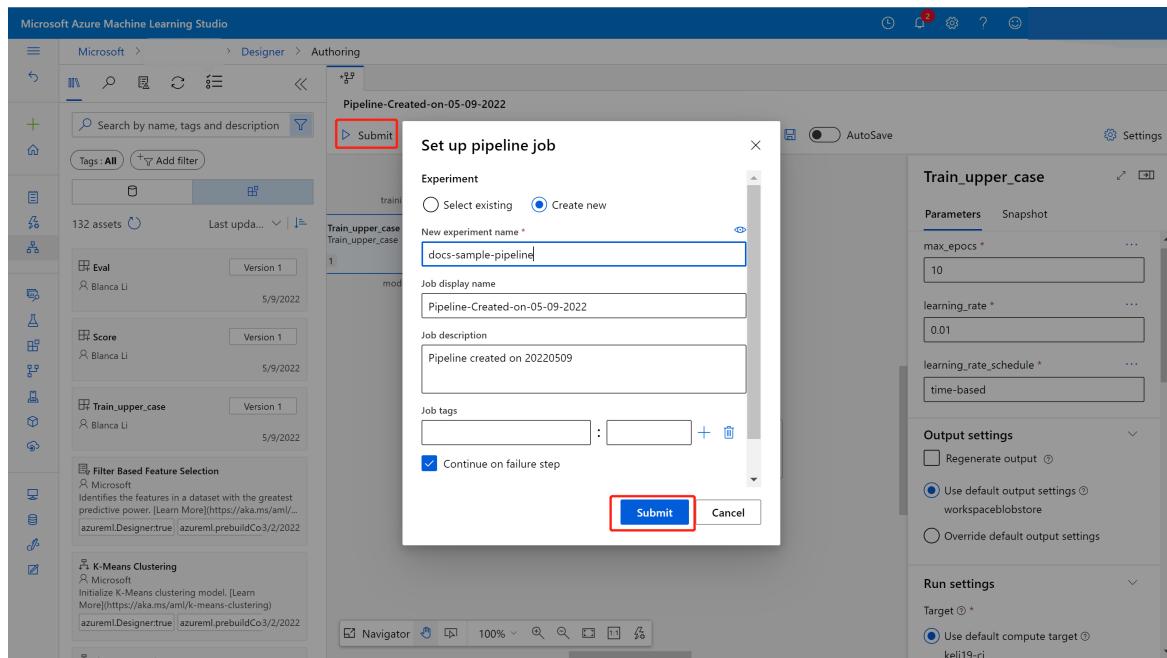


NOTE

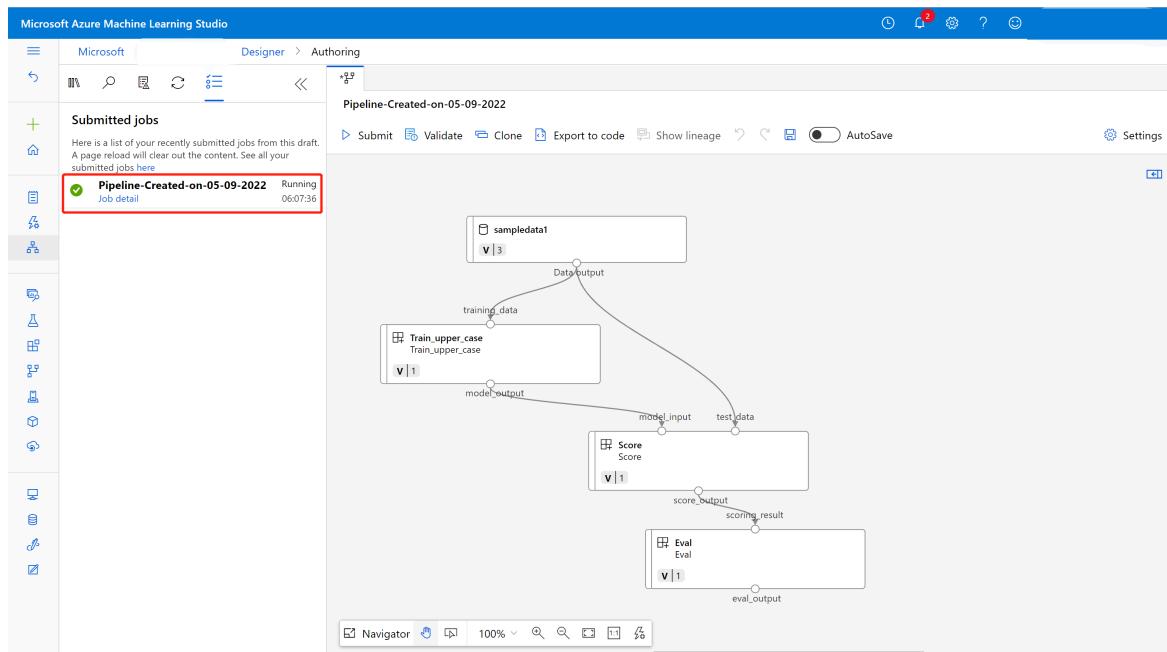
Currently registered components and the designer built-in components cannot be used together.

Submit pipeline

1. Select submit, and fill in the required information for your pipeline job.



- After submit successfully, you'll see a job detail page link in the left page. Select **Job detail** to go to pipeline job detail page for checking status and debugging.



NOTE

The **Submitted jobs** list only contains pipeline jobs submitted during an active session. A page reload will clear out the content.

Next steps

- Use [these Jupyter notebooks on GitHub](#) to explore machine learning pipelines further
- Learn [how to use CLI v2 to create pipeline using components](#).
- Learn [how to use SDK v2 \(preview\) to create pipeline using components](#)

How to do hyperparameter tuning in pipeline (V2) (preview)

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

In this article, you'll learn how to do hyperparameter tuning in Azure Machine Learning pipeline.

Prerequisite

1. Understand what is [hyperparameter tuning](#) and how to do hyperparameter tuning in Azure Machine Learning use SweepJob.
2. Understand what is a [Azure Machine Learning pipeline](#)
3. Build a command component that takes hyperparameter as input.

How to do hyperparameter tuning in Azure Machine Learning pipeline

This section explains how to do hyperparameter tuning in Azure Machine Learning pipeline using CLI v2 and Python SDK. Both approaches share the same prerequisite: you already have a command component created and the command component takes hyperparameters as inputs. If you don't have a command component yet. Follow below links to create a command component first.

- [AzureML CLI v2](#)
- [AzureML Python SDK v2](#)

CLI v2

The example used in this article can be found in [azureml-example repo](#). Navigate to `/azureml-examples/cli/jobs/pipelines-with-components/pipeline_with_hyperparameter_sweep` to check the example.

Assume you already have a command component defined in `train.yaml`. A two-step pipeline job (train and predict) YAML file looks like below.

```

$schema: https://azurermschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: pipeline_with_hyperparameter_sweep
description: Tune hyperparameters using TF component
settings:
  default_compute: azureml:cpu-cluster
jobs:
  sweep_step:
    type: sweep
    inputs:
      data:
        type: uri_file
        path: wasbs://datasets@azurermexamples.blob.core.windows.net/iris.csv
    degree: 3
    gamma: "scale"
    shrinking: False
    probability: False
    tol: 0.001
    cache_size: 1024
    verbose: False
    max_iter: -1
    decision_function_shape: "ovr"
    break_ties: False
    random_state: 42
    outputs:
      model_output:
        test_data:
          sampling_algorithm: random
        trial: ./train.yml
        search_space:
          c_value:
            type: uniform
            min_value: 0.5
            max_value: 0.9
          kernel:
            type: choice
            values: ["rbf", "linear", "poly"]
        coef0:
          type: uniform
          min_value: 0.1
          max_value: 1
    objective:
      goal: minimize
      primary_metric: training_f1_score
    limits:
      max_total_trials: 5
      max_concurrent_trials: 3
      timeout: 7200

  predict_step:
    type: command
    inputs:
      model: ${{parent.jobs.sweep_step.outputs.model_output}}
      test_data: ${{parent.jobs.sweep_step.outputs.test_data}}
    outputs:
      predict_result:
        component: ./predict.yml

```

The `sweep_step` is the step for hyperparameter tuning. Its type needs to be `sweep`. And `trial` refers to the command component defined in `train.yml`. From the `search_space` field we can see three hyperparameters (`c_value`, `kernel`, and `coef`) are added to the search space. After you submit this pipeline job, Azure Machine Learning will run the trial component multiple times to sweep over hyperparameters based on the search space and terminate policy you defined in `sweep_step`. Check [sweep job YAML schema](#) for full schema of sweep job.

Below is the trial component definition (train.yml file).

```
$schema: https://azurermschemas.azureedge.net/latest/commandComponent.schema.json
type: command

name: train_model
display_name: train_model
version: 1

inputs:
  data:
    type: uri_folder
  c_value:
    type: number
    default: 1.0
  kernel:
    type: string
    default: rbf
  degree:
    type: integer
    default: 3
  gamma:
    type: string
    default: scale
  coef0:
    type: number
    default: 0
  shrinking:
    type: boolean
    default: false
  probability:
    type: boolean
    default: false
  tol:
    type: number
    default: 1e-3
  cache_size:
    type: number
    default: 1024
  verbose:
    type: boolean
    default: false
  max_iter:
    type: integer
    default: -1
  decision_function_shape:
    type: string
    default: ovr
  break_ties:
    type: boolean
    default: false
  random_state:
    type: integer
    default: 42

outputs:
  model_output:
    type: mlflow_model
  test_data:
    type: uri_folder

code: ./train-src

environment: azurerm:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest

command: >-
  python train.py
  --data ${inputs.data}
  --C ${inputs.c_value}
  --kernel ${inputs.kernel}
```

```
--degree ${{inputs.degree}}
--gamma ${{inputs.gamma}}
--coef0 ${{inputs.coef0}}
--shrinking ${{inputs.shrinking}}
--probability ${{inputs.probability}}
--tol ${{inputs.tol}}
--cache_size ${{inputs.cache_size}}
--verbose ${{inputs.verbose}}
--max_iter ${{inputs.max_iter}}
--decision_function_shape ${{inputs.decision_function_shape}}
--break_ties ${{inputs.break_ties}}
--random_state ${{inputs.random_state}}
--model_output ${outputs.model_output}
--test_data ${outputs.test_data}
```

The hyperparameters added to search space in pipeline.yml need to be inputs for the trial component. The source code of the trial component is under `./train-src` folder. In this example, it's a single `train.py` file. This is the code that will be executed in every trial of the sweep job. Make sure you've logged the metrics in the trial component source code with exactly the same name as `primary_metric` value in pipeline.yml file. In this example, we use `mlflow.autolog()`, which is the recommended way to track your ML experiments. See more about mlflow [here](#)

Below code snippet is the source code of trial component.

```
# imports
import os
import mlflow
import argparse

import pandas as pd
from pathlib import Path

from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# define functions
def main(args):
    # enable auto logging
    mlflow.autolog()

    # setup parameters
    params = {
        "C": args.C,
        "kernel": args.kernel,
        "degree": args.degree,
        "gamma": args.gamma,
        "coef0": args.coef0,
        "shrinking": args.shrinking,
        "probability": args.probability,
        "tol": args.tol,
        "cache_size": args.cache_size,
        "class_weight": args.class_weight,
        "verbose": args.verbose,
        "max_iter": args.max_iter,
        "decision_function_shape": args.decision_function_shape,
        "break_ties": args.break_ties,
        "random_state": args.random_state,
    }

    # read in data
    df = pd.read_csv(args.data)

    # process data
    X_train, X_test, y_train, y_test = process_data(df, args.random_state)
```

```

# train model
model = train_model(params, X_train, X_test, y_train, y_test)
# Output the model and test data
# write to local folder first, then copy to output folder

mlflow.sklearn.save_model(model, "model")

from distutils.dir_util import copy_tree

# copy subdirectory example
from_directory = "model"
to_directory = args.model_output

copy_tree(from_directory, to_directory)

X_test.to_csv(Path(args.test_data) / "X_test.csv", index=False)
y_test.to_csv(Path(args.test_data) / "y_test.csv", index=False)

def process_data(df, random_state):
    # split dataframe into X and y
    X = df.drop(["species"], axis=1)
    y = df["species"]

    # train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=random_state
    )

    # return split data
    return X_train, X_test, y_train, y_test

def train_model(params, X_train, X_test, y_train, y_test):
    # train model
    model = SVC(**params)
    model = model.fit(X_train, y_train)

    # return model
    return model

def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # add arguments
    parser.add_argument("--data", type=str)
    parser.add_argument("--C", type=float, default=1.0)
    parser.add_argument("--kernel", type=str, default="rbf")
    parser.add_argument("--degree", type=int, default=3)
    parser.add_argument("--gamma", type=str, default="scale")
    parser.add_argument("--coef0", type=float, default=0)
    parser.add_argument("--shrinking", type=bool, default=False)
    parser.add_argument("--probability", type=bool, default=False)
    parser.add_argument("--tol", type=float, default=1e-3)
    parser.add_argument("--cache_size", type=float, default=1024)
    parser.add_argument("--class_weight", type=dict, default=None)
    parser.add_argument("--verbose", type=bool, default=False)
    parser.add_argument("--max_iter", type=int, default=-1)
    parser.add_argument("--decision_function_shape", type=str, default="ovr")
    parser.add_argument("--break_ties", type=bool, default=False)
    parser.add_argument("--random_state", type=int, default=42)
    parser.add_argument("--model_output", type=str, help="Path of output model")
    parser.add_argument("--test_data", type=str, help="Path of output model")

    # parse args
    args = parser.parse_args()

```

```
# return args
return args

# run script
if __name__ == "__main__":
    # parse args
    args = parse_args()

    # run main function
    main(args)
```

Python SDK

The python SDK example can be found in [azureml-example repo](#). Navigate to *azureml-examples/sdk/jobs/pipelines/1c_pipeline_with_hyperparameter_sweep* to check the example.

In Azure Machine Learning Python SDK v2, you can enable hyperparameter tuning for any command component by calling `.sweep()` method.

Below code snippet shows how to enable sweep for `train_model`.

```

train_component_func = load_component(path="./train.yml")
score_component_func = load_component(path="./predict.yml")

# define a pipeline
@pipeline()
def pipeline_with_hyperparameter_sweep():
    """Tune hyperparameters using sample components."""
    train_model = train_component_func(
        data=Input(
            type="uri_file",
            path="wasbs://datasets@azuremlexamples.blob.core.windows.net/iris.csv",
        ),
        c_value=Uniform(min_value=0.5, max_value=0.9),
        kernel=Choice(["rbf", "linear", "poly"]),
        coef0=Uniform(min_value=0.1, max_value=1),
        degree=3,
        gamma="scale",
        shrinking=False,
        probability=False,
        tol=0.001,
        cache_size=1024,
        verbose=False,
        max_iter=-1,
        decision_function_shape="ovr",
        break_ties=False,
        random_state=42,
    )
    sweep_step = train_model.sweep(
        primary_metric="training_f1_score",
        goal="minimize",
        sampling_algorithm="random",
        compute="cpu-cluster",
    )
    sweep_step.set_limits(max_total_trials=20, max_concurrent_trials=10, timeout=7200)

    score_data = score_component_func(
        model=sweep_step.outputs.model_output, test_data=sweep_step.outputs.test_data
    )

pipeline_job = pipeline_with_hyperparameter_sweep()

# set pipeline level compute
pipeline_job.settings.default_compute = "cpu-cluster"

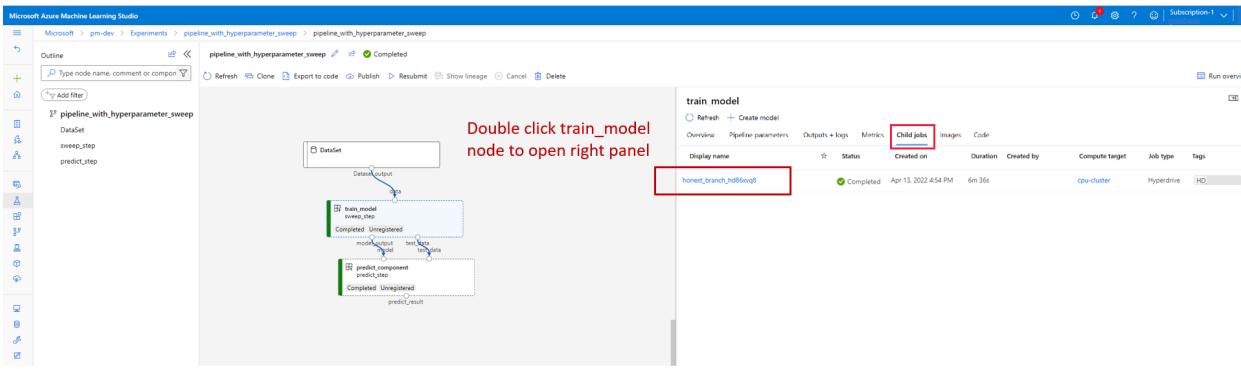
```

We first load `train_component_func` defined in `train.yml` file. When creating `train_model`, we add `c_value`, `kernel` and `coef0` into search space(line 15-17). Line 30-35 defines the primary metric, sampling algorithm etc.

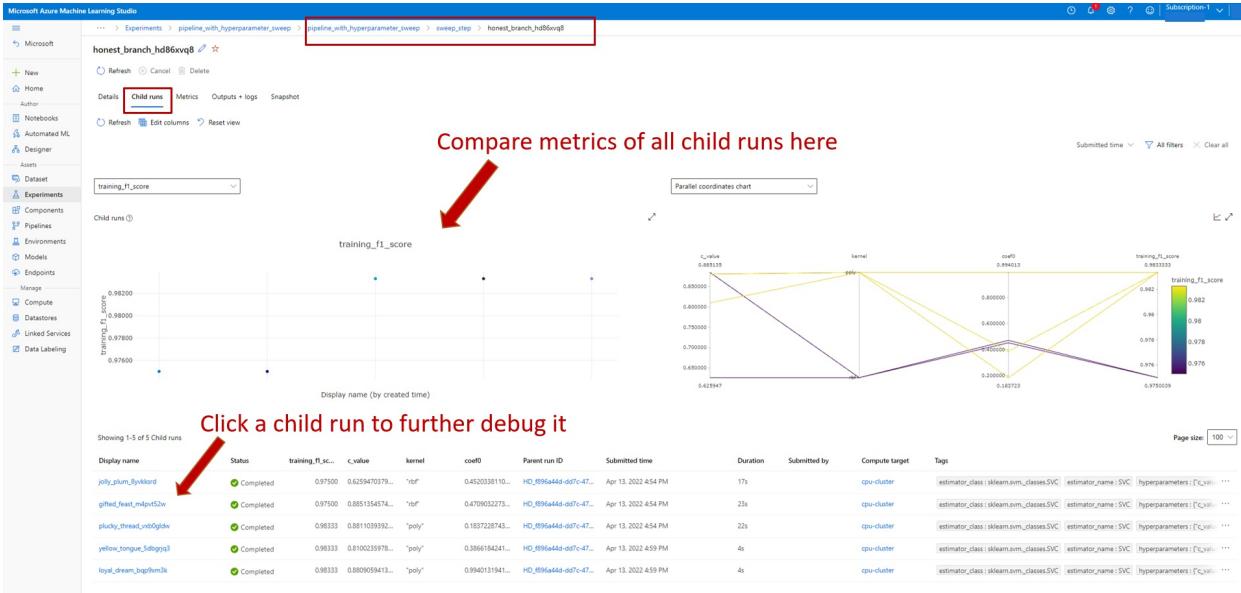
Check pipeline job with sweep step in Studio

After you submit a pipeline job, the SDK or CLI widget will give you a web URL link to Studio UI. The link will guide you to the pipeline graph view by default.

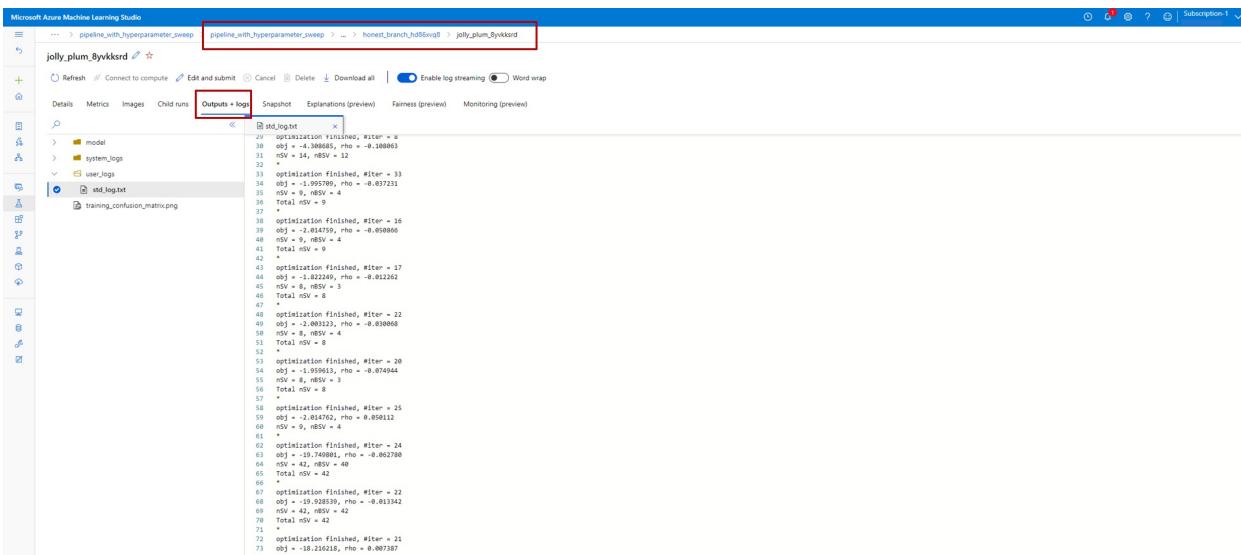
To check details of the sweep step, double click the sweep step and navigate to the **child job** tab in the panel on the right.



This will link you to the sweep job page as seen in the below screenshot. Navigate to **child job** tab, here you can see the metrics of all child jobs and list of all child jobs.



If a child jobs failed, select the name of that child job to enter detail page of that specific child job (see screenshot below). The useful debug information is under **Outputs + Logs**.



Sample notebooks

- Build pipeline with sweep node
- Run hyperparameter sweep on a command job

Next steps

- Track an experiment
- Deploy a trained model

How to use studio UI to build and debug Azure Machine Learning pipelines

9/22/2022 • 3 minutes to read • [Edit Online](#)

Azure Machine Learning studio provides UI to build and debug your pipeline. You can use components to author a pipeline in the designer, and you can debug your pipeline in the job detail page.

This article will introduce how to use the studio UI to build and debug machine learning pipelines.

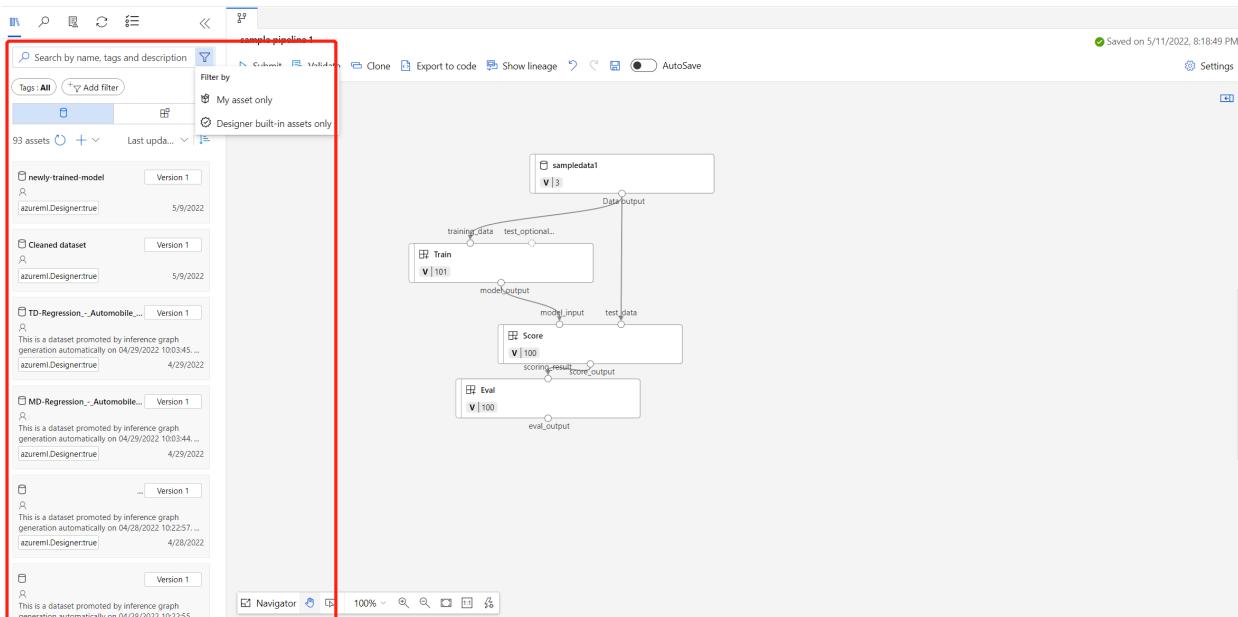
Build machine learning pipeline

Drag and drop components to build pipeline

In the designer homepage, you can select **New pipeline** to open a blank pipeline draft.

In the asset library left of the canvas, there are **Data assets** and **Components** tabs, which contain components and data registered to the workspace. For what is component and how to create custom component, you can refer to the [component concept article](#).

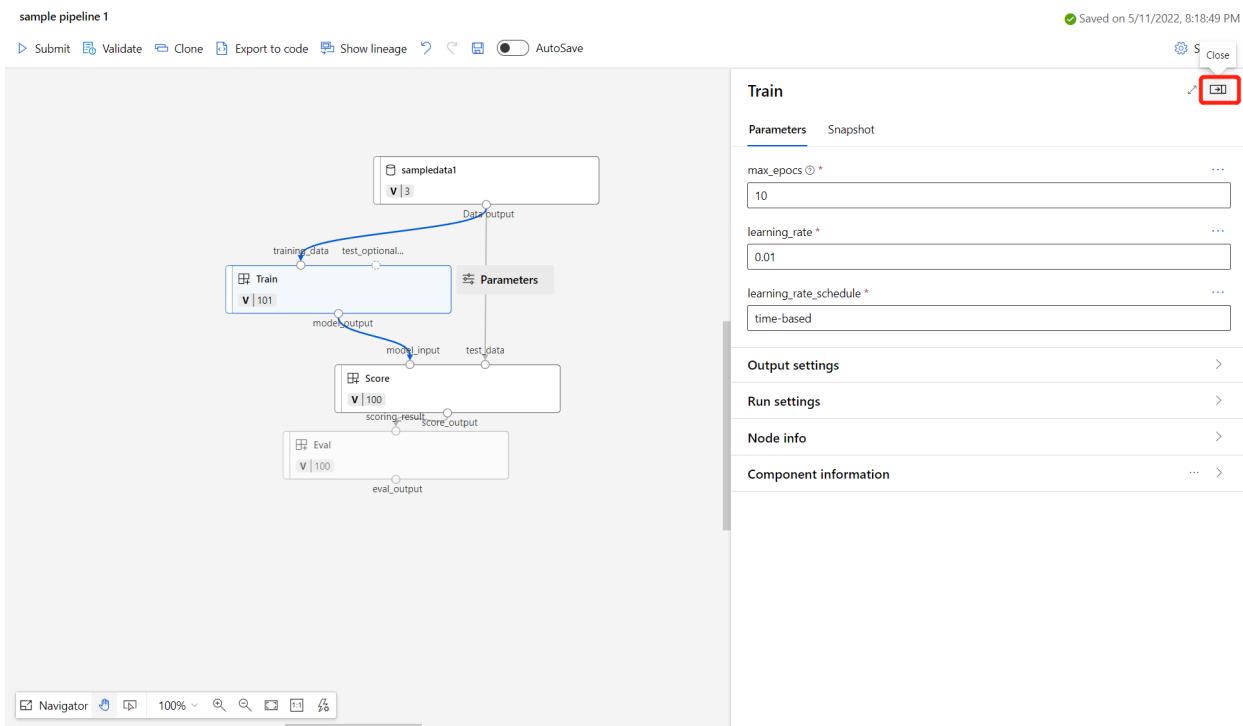
You can quickly filter **My assets** or **Designer built-in assets**.



Then you can drag and drop either built-in components or custom components to the canvas. You can construct your pipeline or configure your components in any order. Just hide the right pane to construct your pipeline first, and open the right pane to configure your component.

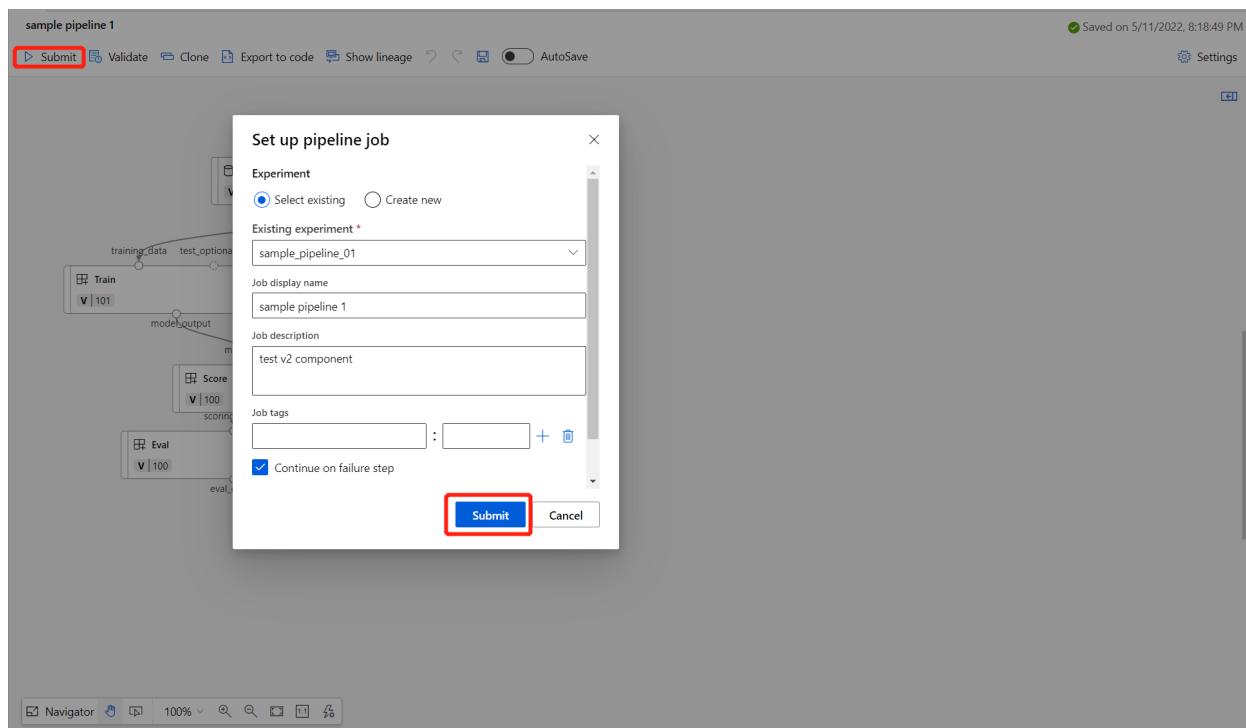
NOTE

Currently built-in components and custom components cannot be used together.



Submit pipeline

Now you've built your pipeline. Select **Submit** button above the canvas, and configure your pipeline job.

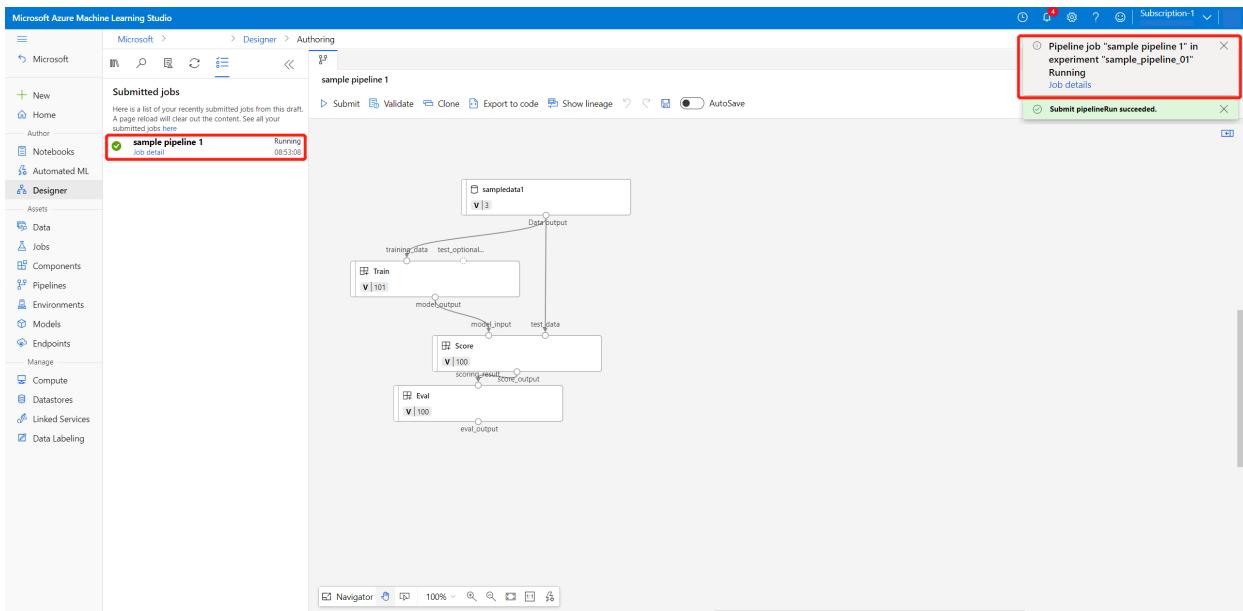


After you submit your pipeline job, you'll see a submitted job list in the left pane, which shows all the pipeline job you create from the current pipeline draft in the same session. There's also notification popping up from the notification center. You can select through the pipeline job link in the submission list or the notification to check pipeline job status or debugging.

NOTE

Pipeline job status and results will not be filled back to the authoring page.

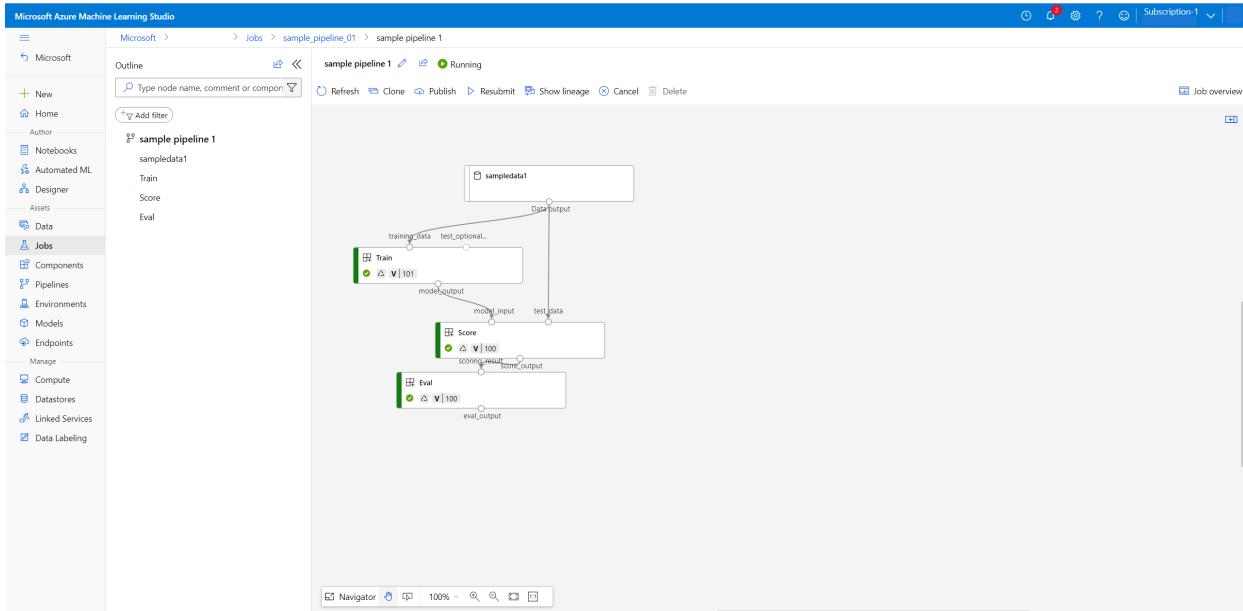
If you want to try a few different parameter values for the same pipeline, you can change values and submit for multiple times, without having to waiting for the running status.



NOTE

The submission list only contains jobs submitted in the same session. If you refresh current page, it will not preserve the previous submitted job list.

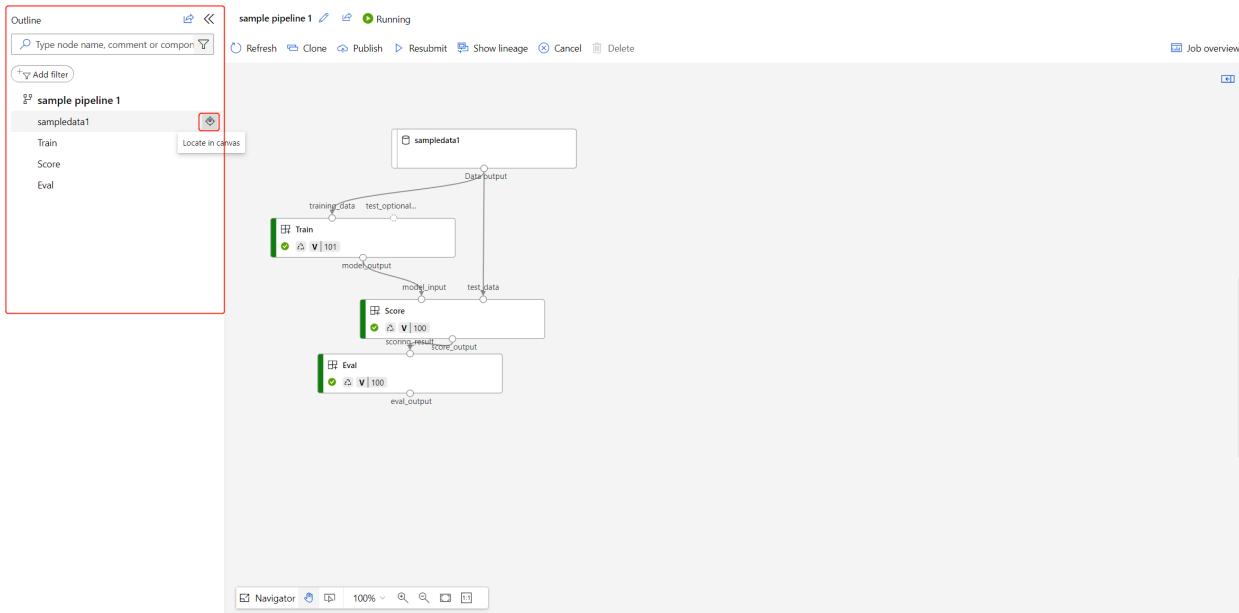
On the pipeline job detail page, you can check the status of the overall job and each node inside, and logs of each node.



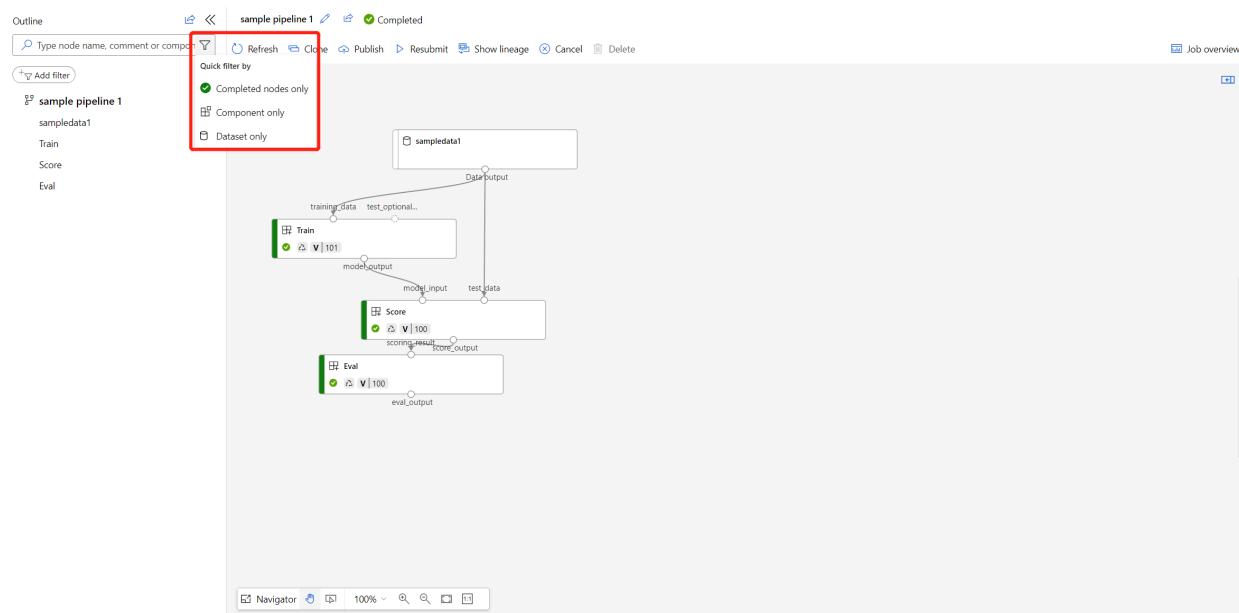
Debug your pipeline in job detail page

Using outline to quickly find node

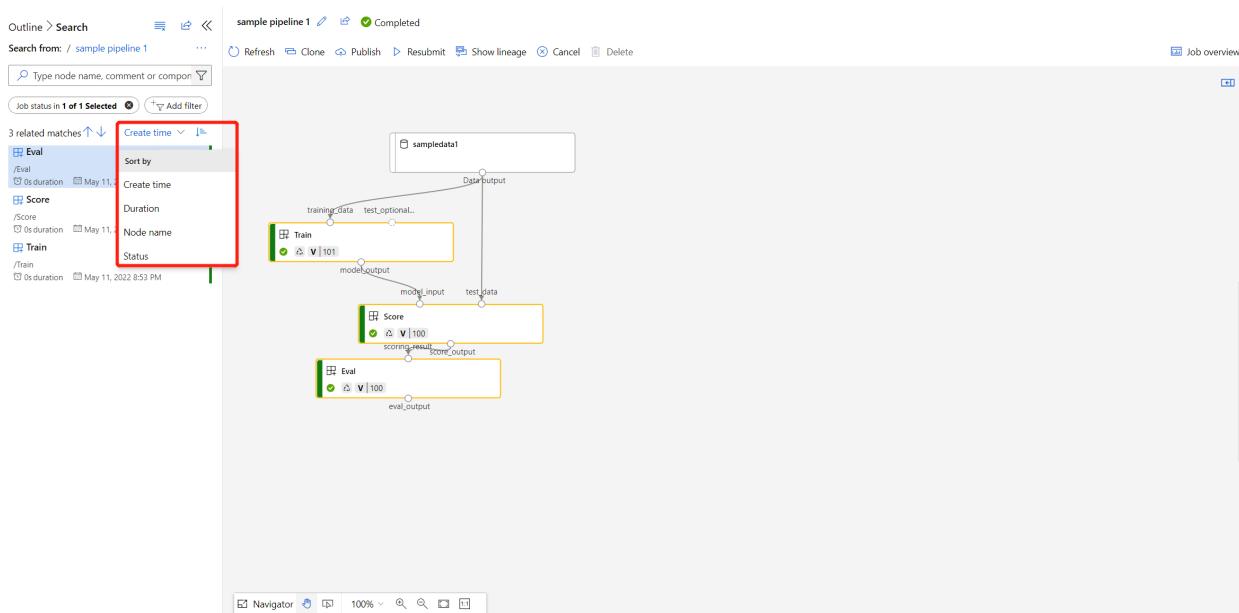
In pipeline job detail page, there's an outline left to the canvas, which shows the overall structure of your pipeline job. Hovering on any row, you can select the "Locate" button to locate that node in the canvas.



You can filter failed or completed nodes, and filter by only components or dataset for further search. The left pane will show the matched nodes with more information including status, duration, and created time.



You can also sort the filtered nodes.



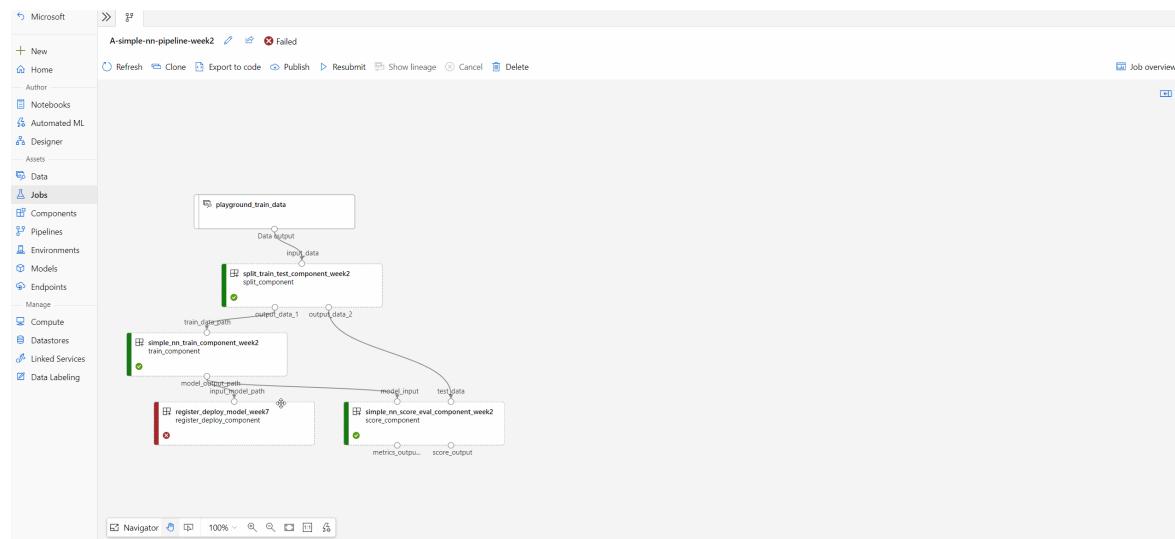
Check logs and outputs of component

If your pipeline fails or gets stuck on a node, first view the logs.

1. You can select the specific node and open the right pane.
2. Select **Outputs+logs** tab and you can explore all the outputs and logs of this node.

The **user_logs** folder contains information about user code generated logs. This folder is open by default, and the **std_log.txt** log is selected. The **std_log.txt** is where your code's logs (for example, print statements) show up.

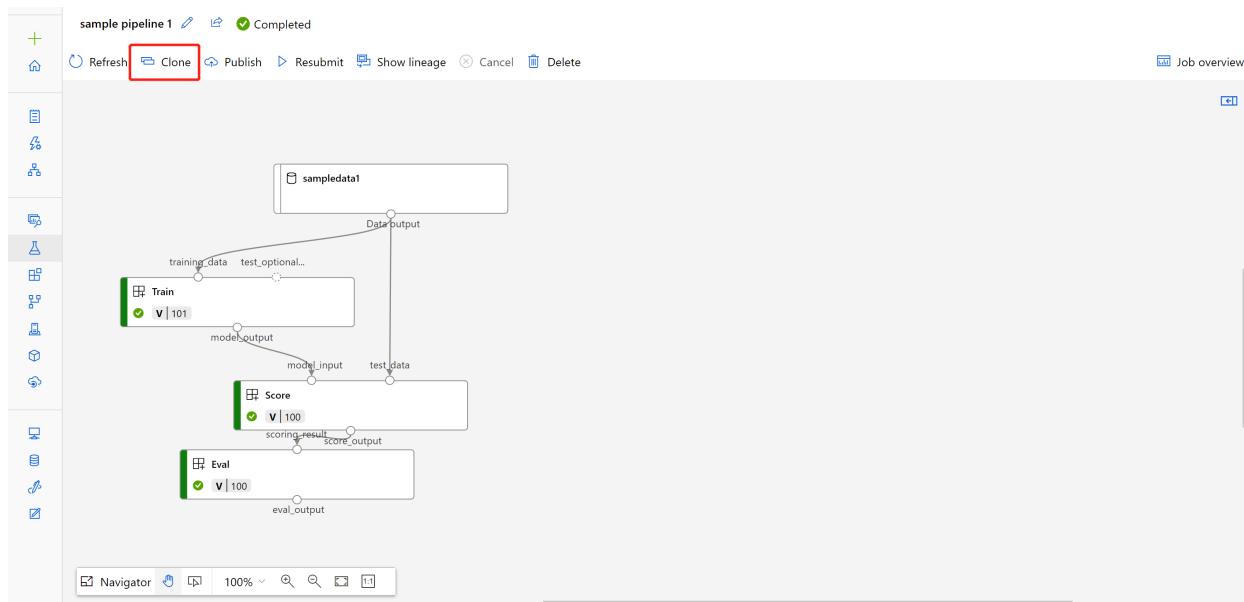
The **system_logs** folder contains logs generated by Azure Machine Learning. Learn more about [View and download diagnostic logs](#).



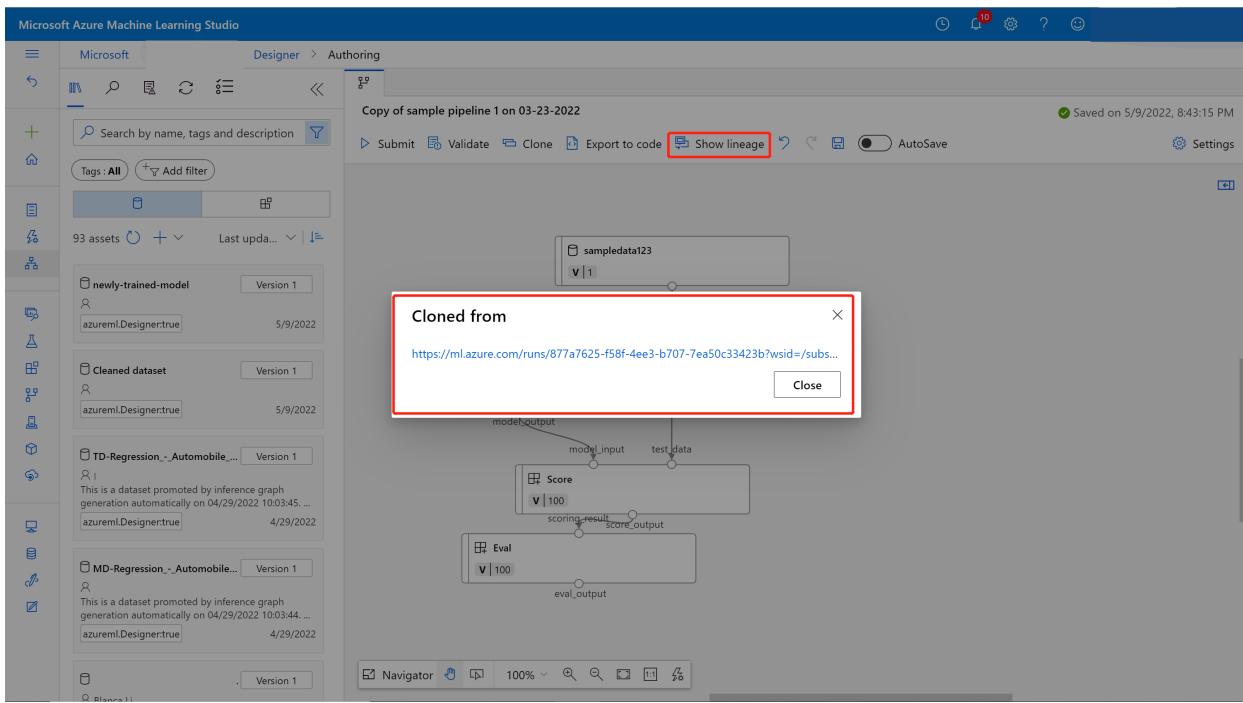
If you don't see those folders, this is due to the compute run time update isn't released to the compute cluster yet, and you can look at **70_driver_log.txt** under **azureml-logs** folder first.

Clone a pipeline job to continue editing

If you would like to work based on an existing pipeline job in the workspace, you can easily clone it into a new pipeline draft to continue editing.



After cloning, you can also know which pipeline job it's cloned from by selecting **Show lineage**.



You can edit your pipeline and then submit again. After submitting, you can see the lineage between the job you submit and the original job by selecting **Show lineage** in the job detail page.

Next steps

In this article, you learned the key features in how to create, explore, and debug a pipeline in UI. To learn more about how you can use the pipeline, see the following articles:

- [How to train a model in the designer](#)
- [How to deploy model to real-time endpoint in the designer](#)
- [What is machine learning component](#)

Enable logging in Azure Machine Learning designer pipelines

9/22/2022 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to add logging code to designer pipelines. You also learn how to view those logs using the Azure Machine Learning studio web portal.

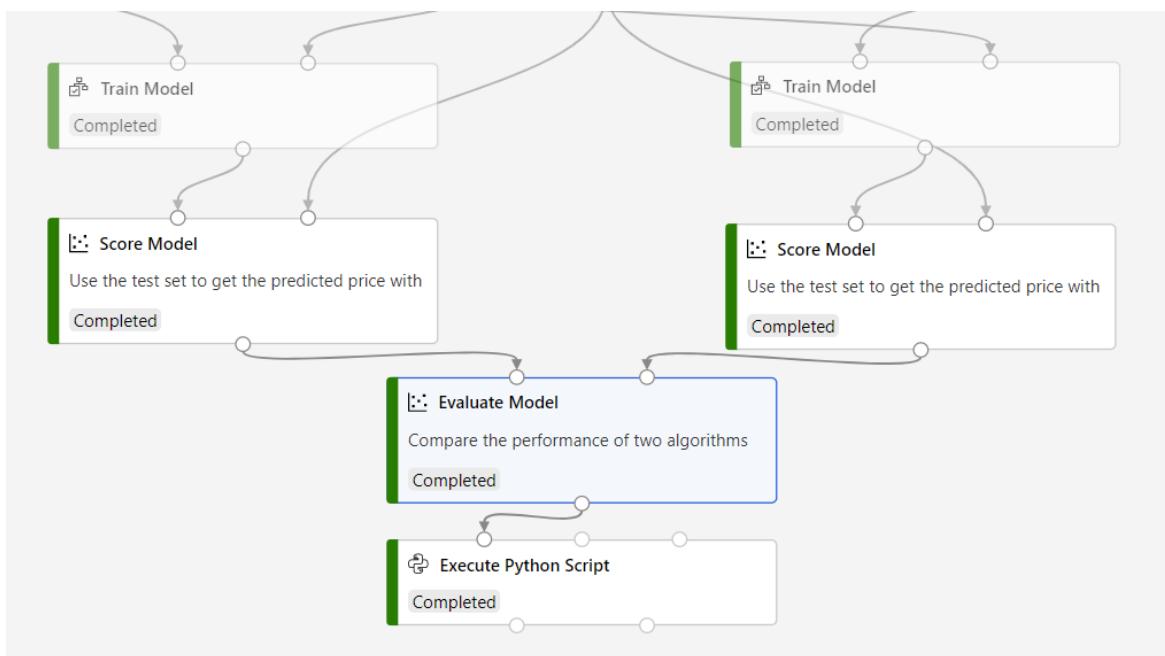
For more information on logging metrics using the SDK authoring experience, see [Monitor Azure ML experiment runs and metrics](#).

Enable logging with Execute Python Script

Use the [Execute Python Script](#) component to enable logging in designer pipelines. Although you can log any value with this workflow, it's especially useful to log metrics from the **Evaluate Model** component to track model performance across runs.

The following example shows you how to log the mean squared error of two trained models using the Evaluate Model and Execute Python Script components.

1. Connect an **Execute Python Script** component to the output of the **Evaluate Model** component.



2. Paste the following code into the **Execute Python Script** code editor to log the mean absolute error for your trained model. You can use a similar pattern to log any other value in the designer:

APPLIES TO: [Python SDK azureml v1](#)

```

# dataframe1 contains the values from Evaluate Model
def azureml_main(dataframe1=None, dataframe2=None):
    print(f'Input pandas.DataFrame #1: {dataframe1}')

    from azureml.core import Run

    run = Run.get_context()

    # Log the mean absolute error to the parent run to see the metric in the run details page.
    # Note: 'run.parent.log()' should not be called multiple times because of performance issues.
    # If repeated calls are necessary, cache 'run.parent' as a local variable and call 'log()' on
    # that variable.
    parent_run = Run.get_context().parent

    # Log left output port result of Evaluate Model. This also works when evaluate only 1 model.
    parent_run.log(name='Mean_Absolute_Error (left port)', value=dataframe1['Mean_Absolute_Error']
[0])

    # Log right output port result of Evaluate Model. The following line should be deleted if you
    # only connect one Score component to the` left port of Evaluate Model component.
    parent_run.log(name='Mean_Absolute_Error (right port)', value=dataframe1['Mean_Absolute_Error']
[1])

    return dataframe1,

```

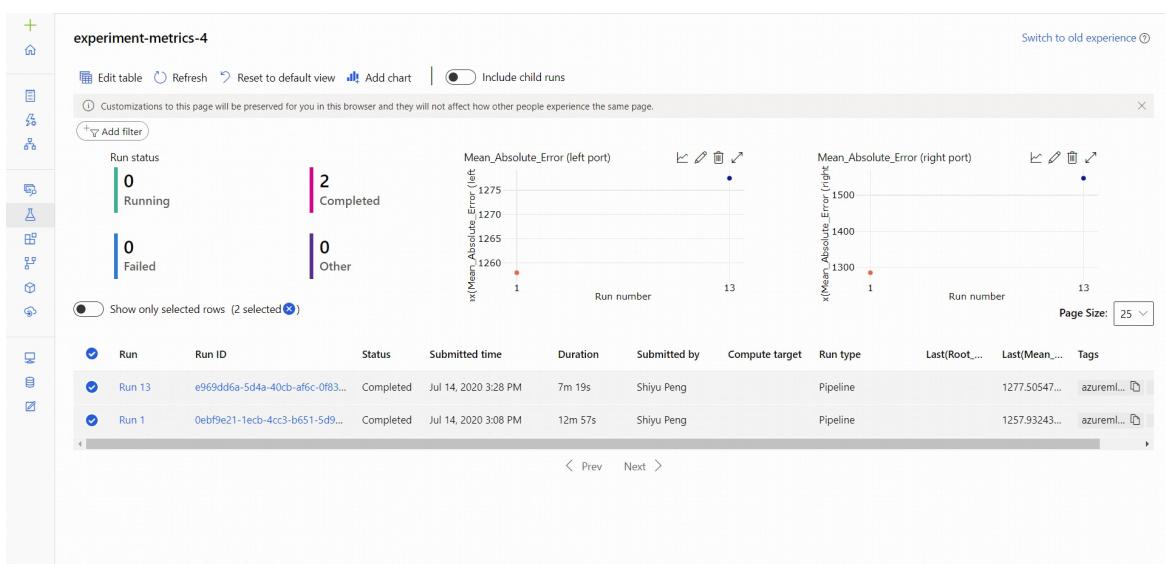
This code uses the Azure Machine Learning Python SDK to log values. It uses `Run.get_context()` to get the context of the current run. It then logs values to that context with the `run.parent.log()` method. It uses `parent` to log values to the parent pipeline run rather than the component run.

For more information on how to use the Python SDK to log values, see [Enable logging in Azure ML training runs](#).

View logs

After the pipeline run completes, you can see the *Mean_Absolute_Error* in the Experiments page.

1. Navigate to the **Jobs** section.
2. Select your experiment.
3. Select the job in your experiment you want to view.
4. Select **Metrics**.



Next steps

In this article, you learned how to use logs in the designer. For next steps, see these related articles:

- Learn how to troubleshoot designer pipelines, see [Debug & troubleshoot ML pipelines](#).
- Learn how to use the Python SDK to log metrics in the SDK authoring experience, see [Enable logging in Azure ML training runs](#).
- Learn how to use [Execute Python Script](#) in the designer.

Transform data in Azure Machine Learning designer

9/22/2022 • 6 minutes to read • [Edit Online](#)

In this article, you learn how to transform and save datasets in Azure Machine Learning designer so that you can prepare your own data for machine learning.

You will use the sample [Adult Census Income Binary Classification](#) dataset to prepare two datasets: one dataset that includes adult census information from only the United States and another dataset that includes census information from non-US adults.

In this article, you learn how to:

1. Transform a dataset to prepare it for training.
2. Export the resulting datasets to a datastore.
3. View results.

This how-to is a prerequisite for the [how to retrain designer models](#) article. In that article, you will learn how to use the transformed datasets to train multiple models with pipeline parameters.

IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Transform a dataset

In this section, you learn how to import the sample dataset and split the data into US and non-US datasets. For more information on how to import your own data into the designer, see [how to import data](#).

Import data

Use the following steps to import the sample dataset.

1. Sign in to [ml.azure.com](#), and select the workspace you want to work with.
2. Go to the designer. Select **Easy-to-use-prebuild components** to create a new pipeline.
3. Select a default compute target to run the pipeline.
4. To the left of the pipeline canvas is a palette of datasets and components. Select **Datasets**. Then view the **Samples** section.
5. Drag and drop the **Adult Census Income Binary classification** dataset onto the canvas.
6. Right-click the **Adult Census Income** dataset component, and select **Visualize > Dataset output**
7. Use the data preview window to explore the dataset. Take special note of the "native-country" column values.

Split the data

In this section, you use the **Split Data component** to identify and split rows that contain "United-States" in the "native-country" column.

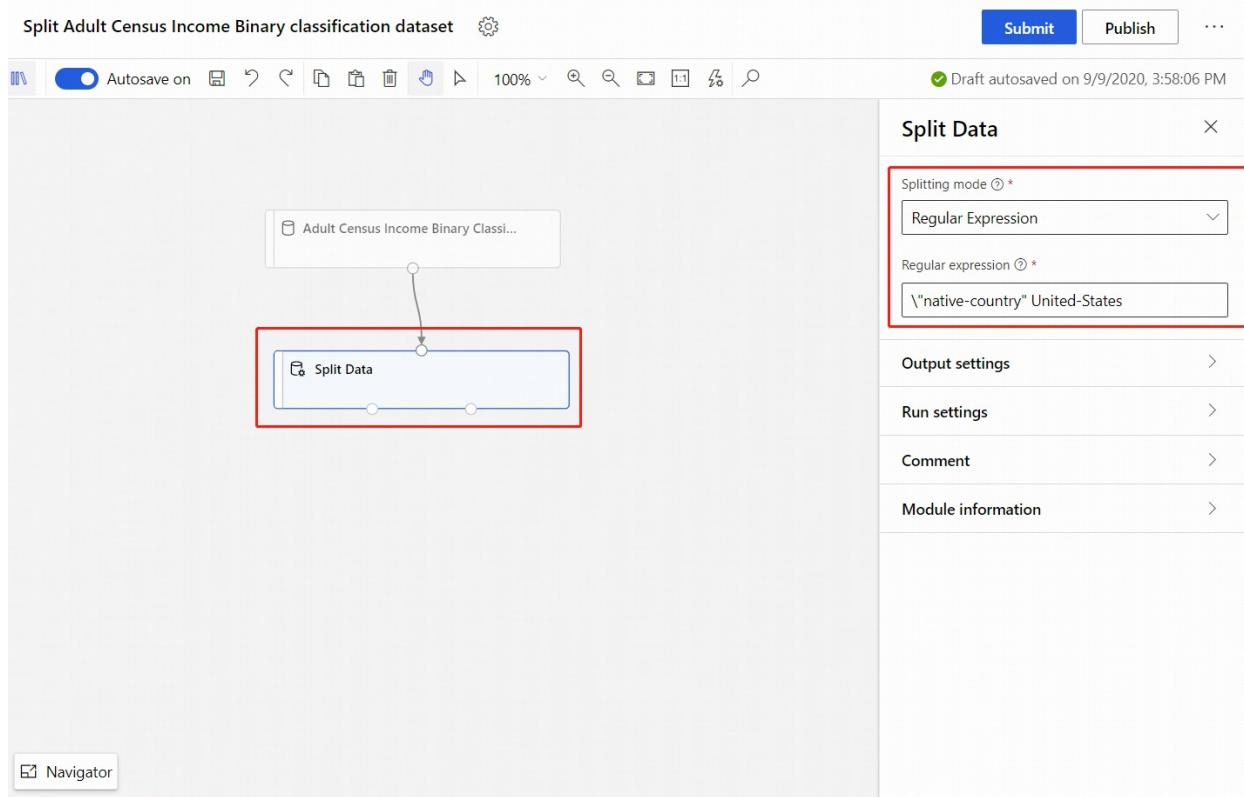
1. In the component palette to the left of the canvas, expand the **Data Transformation** section and find the

Split Data component.

2. Drag the **Split Data** component onto the canvas, and drop the component below the dataset component.
3. Connect the dataset component to the **Split Data** component.
4. Select the **Split Data** component.
5. In the component details pane to the right of the canvas, set **Splitting mode** to **Regular Expression**.
6. Enter the **Regular Expression**: `\"native-country" United-States`.

The **Regular expression** mode tests a single column for a value. For more information on the Split Data component, see the related [algorithm component reference page](#).

Your pipeline should look like this:

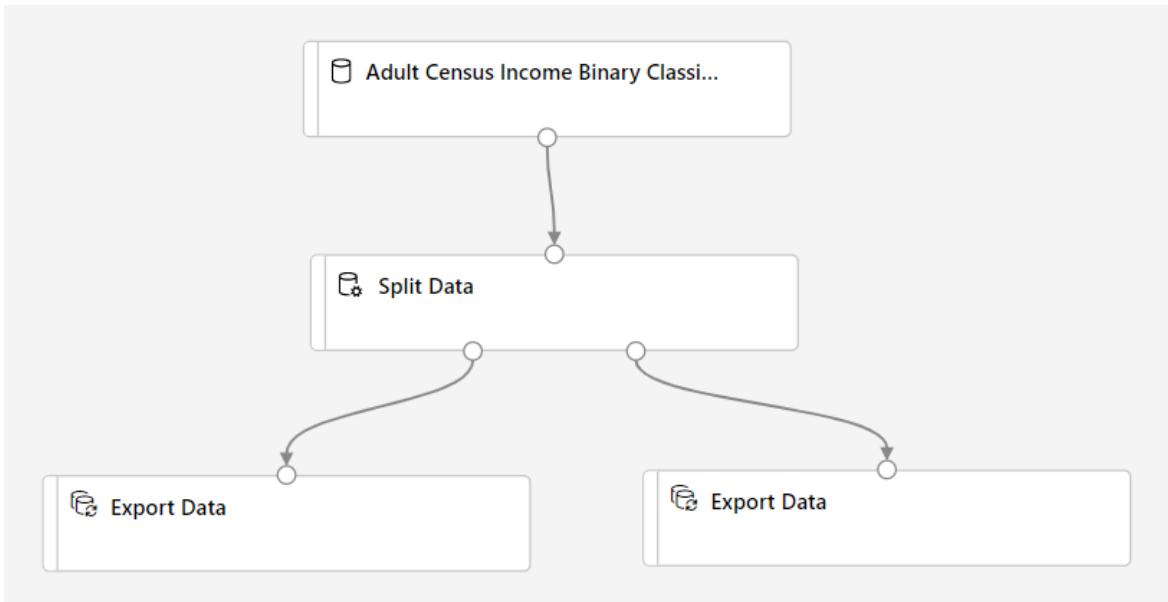


Save the datasets

Now that your pipeline is set up to split the data, you need to specify where to persist the datasets. For this example, use the **Export Data** component to save your dataset to a datastore. For more information on datastores, see [Connect to Azure storage services](#)

1. In the component palette to the left of the canvas, expand the **Data Input and Output** section and find the **Export Data** component.
2. Drag and drop two **Export Data** components below the **Split Data** component.
3. Connect each output port of the **Split Data** component to a different **Export Data** component.

Your pipeline should look something like this:



4. Select the **Export Data** component that is connected to the *left*-most port of the **Split Data** component.

The order of the output ports matter for the **Split Data** component. The first output port contains the rows where the regular expression is true. In this case, the first port contains rows for US-based income, and the second port contains rows for non-US based income.

5. In the component details pane to the right of the canvas, set the following options:

Datastore type: Azure Blob Storage

Datastore: Select an existing datastore or select "New datastore" to create one now.

Path: `/data/us-income`

File format: csv

NOTE

This article assumes that you have access to a datastore registered to the current Azure Machine Learning workspace. For instructions on how to setup a datastore, see [Connect to Azure storage services](#).

If you don't have a datastore, you can create one now. For example purposes, this article will save the datasets to the default blob storage account associated with the workspace. It will save the datasets into the `azureml` container in a new folder called `data`.

6. Select the **Export Data** component connected to the *right*-most port of the **Split Data** component.

7. In the component details pane to the right of the canvas, set the following options:

Datastore type: Azure Blob Storage

Datastore: Select the same datastore as above

Path: `/data/non-us-income`

File format: csv

8. Confirm the **Export Data** component connected to the left port of the **Split Data** has the **Path** `/data/us-income`.

9. Confirm the **Export Data** component connected to the right port has the **Path** `/data/non-us-income`.

Your pipeline and settings should look like this:



Submit the job

Now that your pipeline is setup to split and export the data, submit a pipeline job.

1. At the top of the canvas, select **Submit**.
2. In the **Set up pipeline job** dialog, select **Create new** to create an experiment.

Experiments logically group together related pipeline jobs. If you run this pipeline in the future, you should use the same experiment for logging and tracking purposes.
3. Provide a descriptive experiment name like "split-census-data".
4. Select **Submit**.

View results

After the pipeline finishes running, you can view your results by navigating to your blob storage in the Azure portal. You can also view the intermediary results of the **Split Data** component to confirm that your data has been split correctly.

1. Select the **Split Data** component.
2. In the component details pane to the right of the canvas, select **Outputs + logs**.
3. Select the visualize icon next to **Results dataset1**.
4. Verify that the "native-country" column only contains the value "United-States".
5. Select the visualize icon next to **Results dataset2**.
6. Verify that the "native-country" column does not contain the value "United-States".

Clean up resources

Skip this section if you want to continue on with part 2 of this how to, [Retrain models with Azure Machine Learning designer](#).

IMPORTANT

You can use the resources that you created as prerequisites for other Azure Machine Learning tutorials and how-to articles.

Delete everything

If you don't plan to use anything that you created, delete the entire resource group so you don't incur any charges.

1. In the Azure portal, select **Resource groups** on the left side of the window.

The screenshot shows the Azure portal's Resource groups blade. On the left, there's a list of resource groups with one named 'my-rg' selected and highlighted with a red box. On the right, there's a detailed view of 'my-rg' including sections for Overview, Activity log, Access control (IAM), Tags, Events, and Settings. At the top right of this view, there are several buttons: '+ Add', 'Edit columns', 'Delete resource group' (which is highlighted with a red box), and 'Refresh'. Below the main view, there's a list of other resource groups: 'my-ws', 'myws22', 'myws70', and 'myws74'. There are also filters at the bottom: 'Filter by name...', 'Type == all', and 'Location == all'.

2. In the list, select the resource group that you created.

3. Select **Delete resource group**.

Deleting the resource group also deletes all resources that you created in the designer.

Delete individual assets

In the designer where you created your experiment, delete individual assets by selecting them and then selecting the **Delete** button.

The compute target that you created here *automatically autoscales* to zero nodes when it's not being used. This action is taken to minimize charges. If you want to delete the compute target, take these steps:

The screenshot shows the Compute blade in the Azure Machine Learning studio. On the left, there's a sidebar with links like Home, Author, Automated ML, Designer, Notebooks, Datasets, Experiments, Pipelines, Models, Endpoints, Manage, and Compute (which is highlighted with a red box). The main area is titled 'Compute' and shows tabs for Compute Instances, Training Clusters, **Inference Clusters** (which is underlined), and Attached Compute. Below the tabs, there's a toolbar with 'New', 'Refresh', 'Delete' (highlighted with a red box), and a search bar. The main table lists inference clusters with columns: Name, Type, Provisioning Status, Creation Date, and Virtual Machines. One entry, 'aks-compute', is selected and has a checkmark icon next to its name. Navigation arrows for 'Prev' and 'Next' are at the bottom of the table.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a sidebar with various navigation options like New, Home, Author, Notebooks, Automated ML, Designer, Assets, and Datasets (which is highlighted with a red box). The main content area displays a dataset named "TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1". The top right shows "Version 1 (latest) <". Below the title, there are tabs for Details, Explore, Models, and Datasheet. A toolbar includes Refresh, Generate profile, Unregister (which has a red box around it), and New version. The "Attributes" section contains fields for Properties (File), Description (a note about inference graph generation), Datastore (workspaceblobstore), Relative path (azureml://4393076b-19ff-4e41-81d9-a1146d905696/Cleaning_transformation), Profile (No profile generated), Files in dataset (4), Current version (1), and Latest version (1). The "Tags" section lists "CreatedByAMLStudio" and "true". The "Sample usage" section contains a code snippet:

```
# azureml-core of version 1.0.72 or higher is required
from azureml.core import Workspace, Dataset

subscription_id = 'ee85ed72-2b26-48f6-a0e8-cb5bcf98fb9'
resource_group = 'test-like'
workspace_name = 'like_test'

workspace = Workspace(subscription_id, resource_group, workspace_name)

dataset = Dataset.get_by_name(workspace, name='TD-Sample_1:_Regression_-_Aut
dataset.download(target_path='.', overwrite=False)
```

To delete a dataset, go to the storage account by using the Azure portal or Azure Storage Explorer and manually delete those assets.

Next steps

In this article, you learned how to transform a dataset and save it to a registered datastore.

Continue to the next part of this how-to series with [Retrain models with Azure Machine Learning designer](#) to use your transformed datasets and pipeline parameters to train machine learning models.

Use pipeline parameters in the designer to build versatile pipelines

9/22/2022 • 4 minutes to read • [Edit Online](#)

Use pipeline parameters to build flexible pipelines in the designer. Pipeline parameters let you dynamically set values at runtime to encapsulate pipeline logic and reuse assets.

Pipeline parameters are especially useful when resubmitting a pipeline job, [retraining models](#), or performing [batch predictions](#).

In this article, you learn how to do the following:

- Create pipeline parameters
- Delete and manage pipeline parameters
- Trigger pipeline jobs while adjusting pipeline parameters

Prerequisites

- An Azure Machine Learning workspace. See [Create workspace resources](#).
- For a guided introduction to the designer, complete the [designer tutorial](#).

IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Create pipeline parameter

There are three ways to create a pipeline parameter in the designer:

- Create a pipeline parameter in the settings panel, and bind it to a component.
- Promote a component parameter to a pipeline parameter.
- Promote a dataset to a pipeline parameter

NOTE

Pipeline parameters only support basic data types like `int`, `float`, and `string`.

Option 1: Create a pipeline parameter in the settings panel

In this section, you create a pipeline parameter in the settings panel.

In this example, you create a pipeline parameter that defines how a pipeline fills in missing data using the [Clean missing data](#) component.

1. Next to the name of your pipeline draft, select the **gear icon** to open the **Settings** panel.
2. In the **Pipeline parameters** section, select the **+** icon.
3. Enter a name for the parameter and a default value.

For example, enter `replace-missing-value` as parameter name and `0` as default value.

The screenshot shows the Azure Machine Learning studio interface. On the left, a pipeline diagram is displayed with various components: Dataset1, Import Data, Clean Missing Data, Two-Class Boosted Decision Tree, Train Model, Score Model, and Evaluate Model. Arrows indicate the flow of data between these components. On the right, the 'Settings' pane is open. Under 'Pipeline parameters', there is a parameter named 'replace-miss...' with a value of '0'. A red box highlights the ellipsis (...) button next to the value field. Below this, a tooltip says 'Delete parameter'. Other settings in the pane include 'Default compute target' set to 'cpucluster' and 'Draft details' with a draft name of 'pipeline parameter sample'.

After you create a pipeline parameter, you must [attach it to the component parameter](#) that you want to dynamically set.

Option 2: Promote a component parameter

The simplest way to create a pipeline parameter for a component value is to promote a component parameter. Use the following steps to promote a component parameter to a pipeline parameter:

1. Select the component you want to attach a pipeline parameter to.
2. In the component detail pane, mouseover the parameter you want to specify.
3. Select the ellipses (...) that appear.
4. Select **Add to pipeline parameter**.

The screenshot shows the Azure Machine Learning studio interface. The pipeline diagram is the same as the previous screenshot. On the right, the 'Import Data' component detail pane is open. In the 'Path' section, there is a parameter named 'dat' with a value of '0'. A red box highlights the ellipsis (...) button next to the value field. A tooltip for this button says 'Add to pipeline parameter'. Other sections in the pane include 'Data source', 'Output settings', 'Run settings', 'Comment', and 'Module information'.

5. Enter a parameter name and default value.

6. Select **Save**

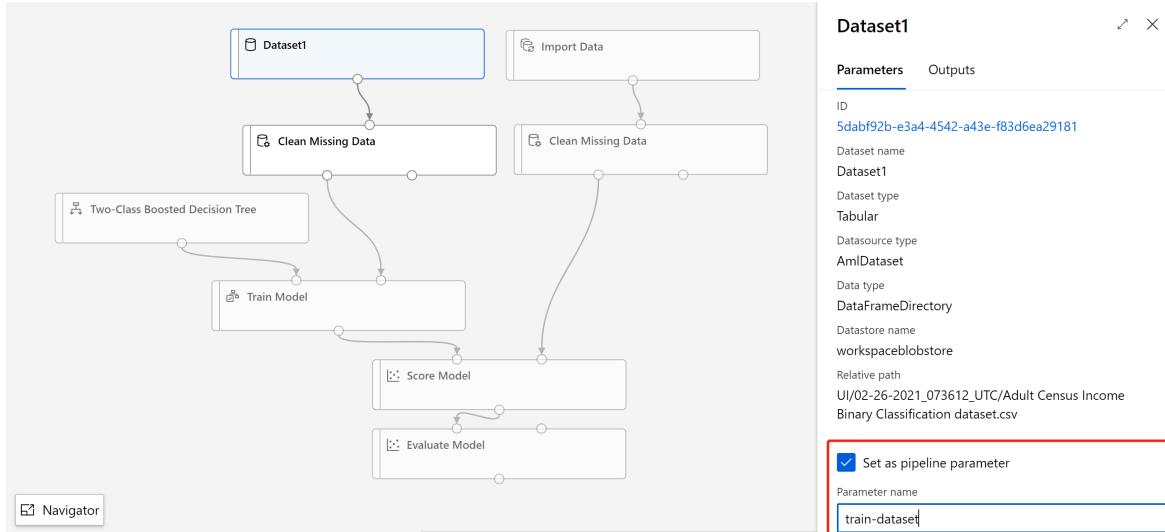
You can now specify new values for this parameter anytime you submit this pipeline.

Option 3: Promote a dataset to a pipeline parameter

If you want to submit your pipeline with variable datasets, you must promote your dataset to a pipeline parameter:

1. Select the dataset you want to turn into a pipeline parameter.

2. In the detail panel of dataset, check **Set as pipeline parameter**.



You can now specify a different dataset by using the pipeline parameter the next time you run the pipeline.

Attach and detach component parameter to pipeline parameter

In this section, you will learn how to attach and detach component parameter to pipeline parameter.

Attach component parameter to pipeline parameter

You can attach the same component parameters of duplicated components to the same pipeline parameter if you want to alter the value at one time when triggering the pipeline job.

The following example has duplicated **Clean Missing Data** component. For each **Clean Missing Data** component, attach **Replacement value** to pipeline parameter **replace-missing-value**:

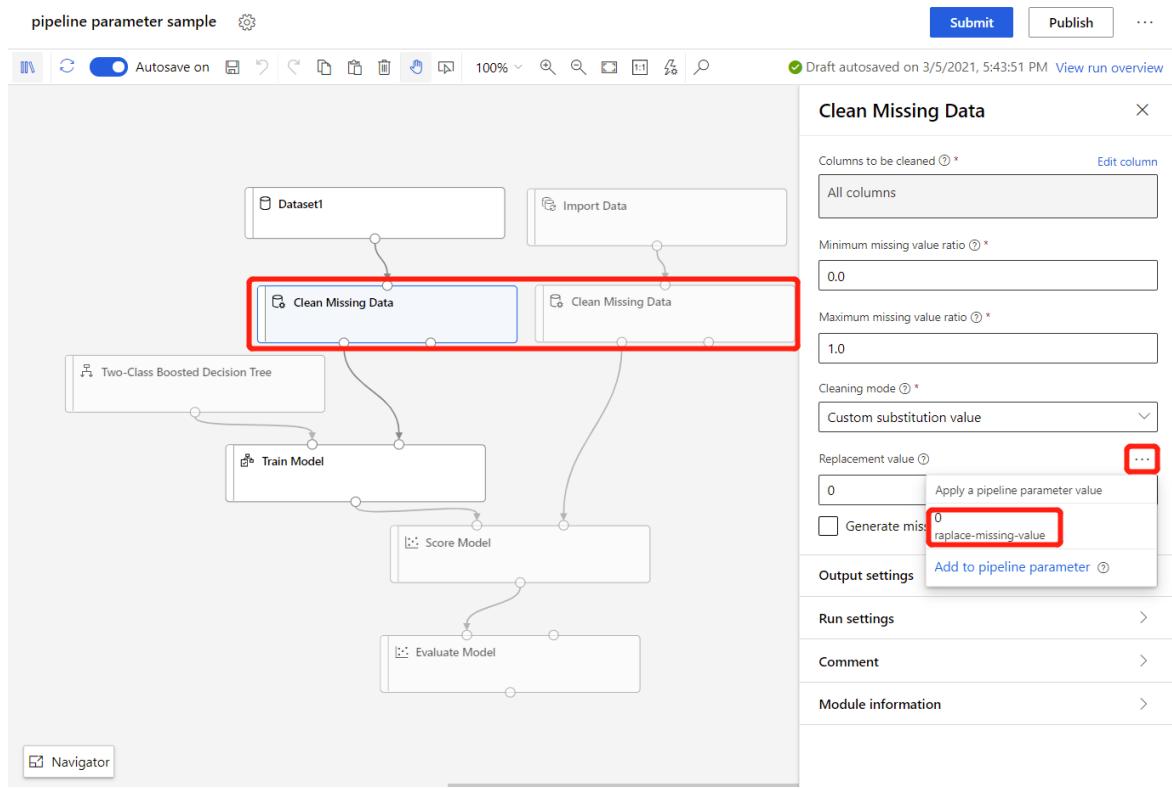
1. Select the **Clean Missing Data** component.

2. In the component detail pane, to the right of the canvas, set the **Cleaning mode** to "Custom substitution value".

3. Mouseover the **Replacement value** field.

4. Select the ellipses (...) that appear.

5. Select the pipeline parameter `replace-missing-value`.

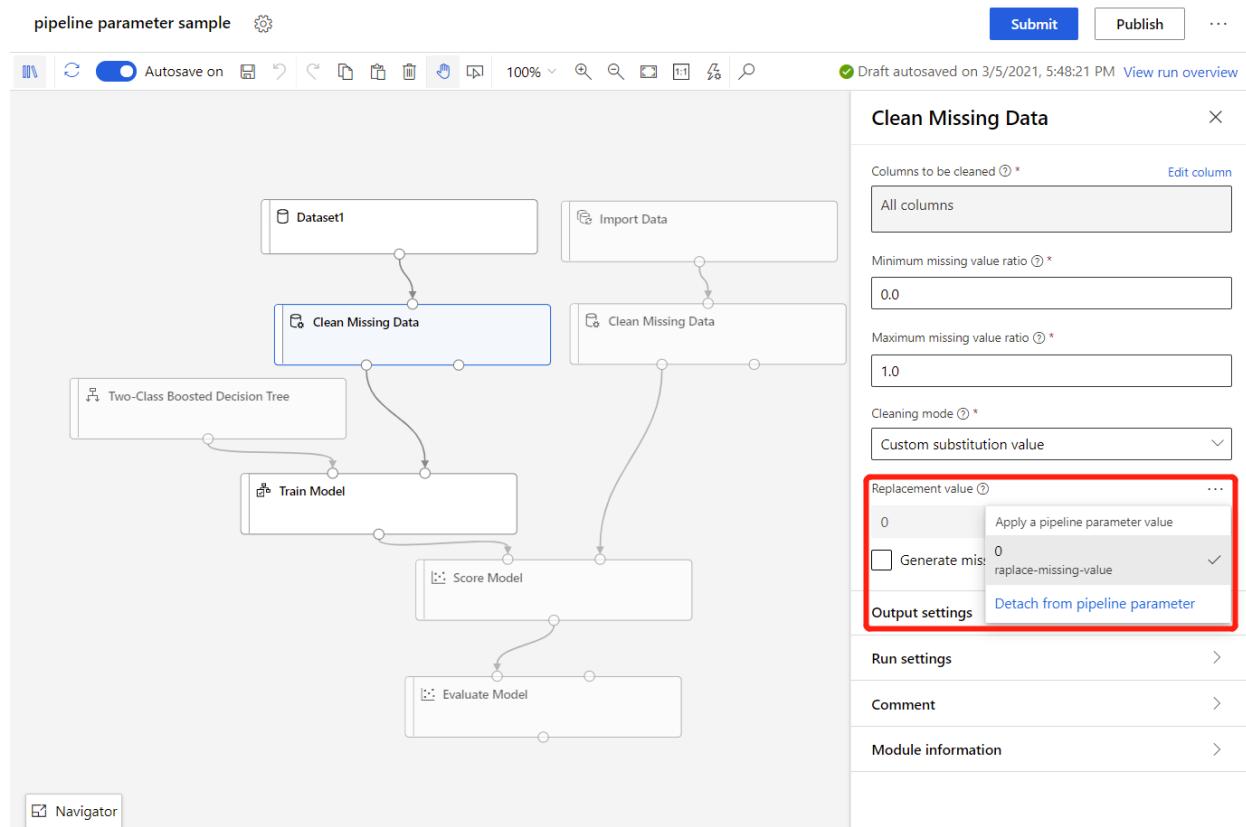


You have successfully attached the **Replacement value** field to your pipeline parameter.

Detach component parameter to pipeline parameter

After you attach **Replacement value** to pipeline parameter, it is non-actionable.

You can detach component parameter to pipeline parameter by clicking the ellipses (...) next to the component parameter, and select **Detach from pipeline parameter**.



Update and delete pipeline parameters

In this section, you learn how to update and delete pipeline parameters.

Update pipeline parameters

Use the following steps to update a component pipeline parameter:

1. At the top of the canvas, select the gear icon.
2. In the **Pipeline parameters** section, you can view and update the name and default value for all of your pipeline parameter.

Delete a dataset pipeline parameter

Use the following steps to delete a dataset pipeline parameter:

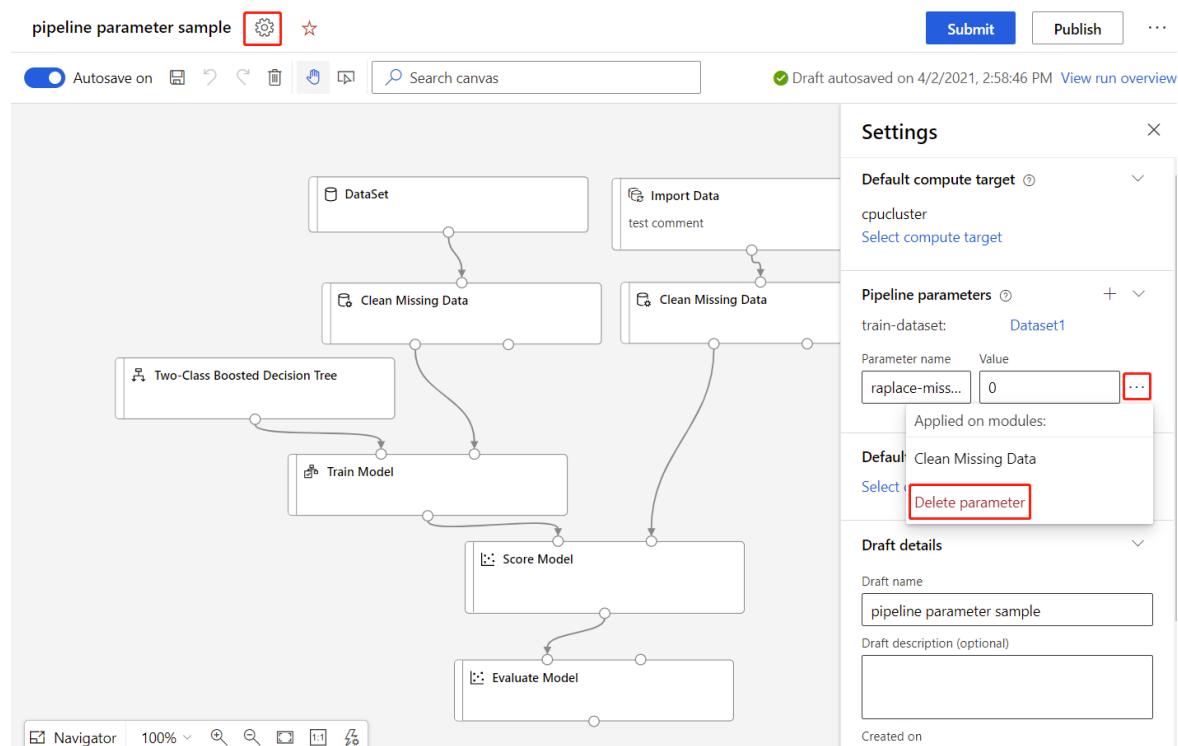
1. Select the dataset component.
2. Uncheck the option **Set as pipeline parameter**.

Delete component pipeline parameters

Use the following steps to delete a component pipeline parameter:

1. At the top of the canvas, select the gear icon.
2. Select the ellipses (...) next to the pipeline parameter.

This view shows you which components the pipeline parameter is attached to.



3. Select **Delete parameter** to delete the pipeline parameter.

NOTE

Deleting a pipeline parameter will cause all attached component parameters to be detached and the value of detached component parameters will keep current pipeline parameter value.

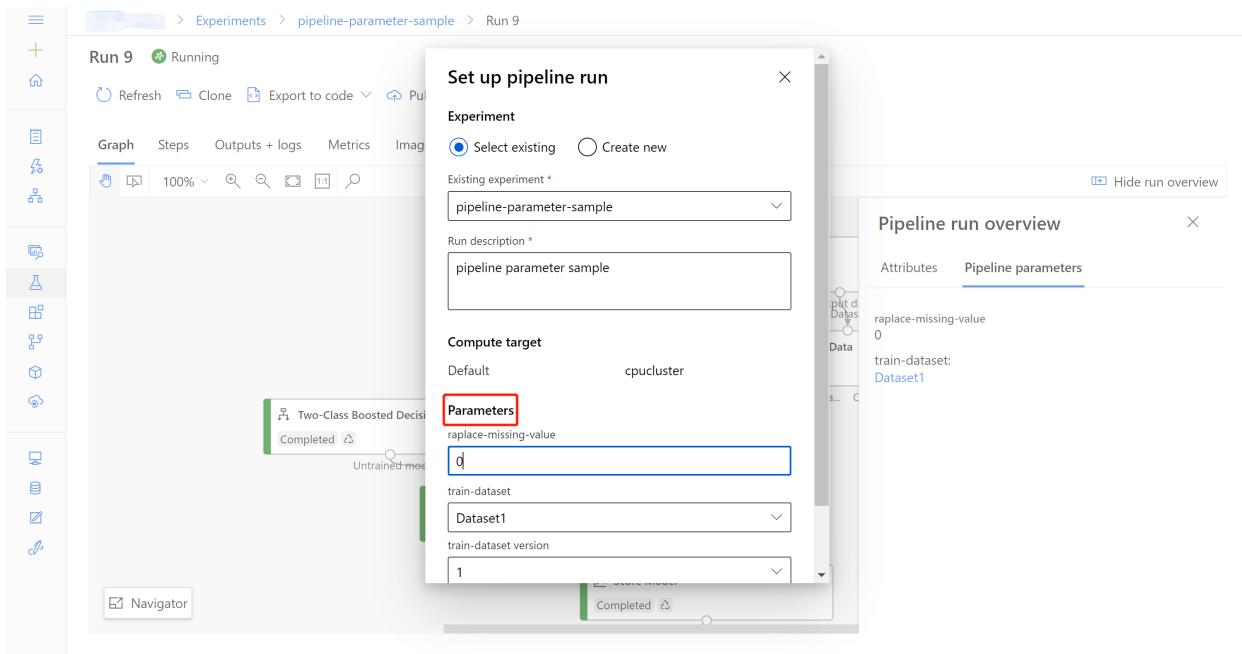
Trigger a pipeline job with pipeline parameters

In this section, you learn how to submit a pipeline job while setting pipeline parameters.

Resubmit a pipeline job

After submitting a pipeline with pipeline parameters, you can resubmit a pipeline job with different parameters:

1. Go to pipeline detail page. In the **Pipeline job overview** window, you can check current pipeline parameters and values.
2. Select **Resubmit**.
3. In the **Setup pipeline job**, specify your new pipeline parameters.



Use published pipelines

You can also publish a pipeline to use its pipeline parameters. A **published pipeline** is a pipeline that has been deployed to a compute resource, which client applications can invoke via a REST endpoint.

Published endpoints are especially useful for retraining and batch prediction scenarios. For more information, see [How to retrain models in the designer](#) or [Run batch predictions in the designer](#).

Next steps

In this article, you learned how to create pipeline parameters in the designer. Next, see how you can use pipeline parameters to [retrain models](#) or perform [batch predictions](#).

You can also learn how to [use pipelines programmatically with the SDK v1](#).

Use pipeline parameters to retrain models in the designer

9/22/2022 • 4 minutes to read • [Edit Online](#)

In this how-to article, you learn how to use Azure Machine Learning designer to retrain a machine learning model using pipeline parameters. You will use published pipelines to automate your workflow and set parameters to train your model on new data. Pipeline parameters let you re-use existing pipelines for different jobs.

In this article, you learn how to:

- Train a machine learning model.
- Create a pipeline parameter.
- Publish your training pipeline.
- Retrain your model with new parameters.

Prerequisites

- An Azure Machine Learning workspace
- Complete part 1 of this how-to series, [Transform data in the designer](#)

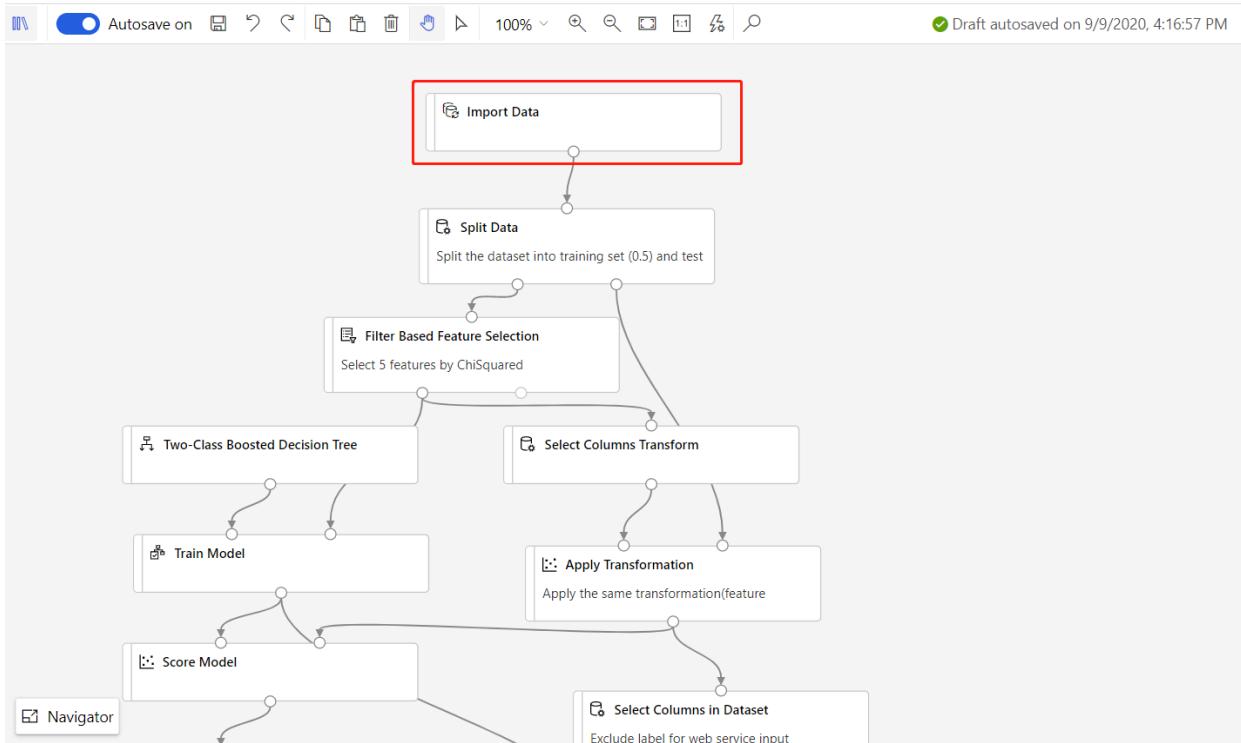
IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

This article also assumes that you have some knowledge of building pipelines in the designer. For a guided introduction, complete the [tutorial](#).

Sample pipeline

The pipeline used in this article is an altered version of a sample pipeline [Income prediction](#) in the designer homepage. The pipeline uses the [Import Data](#) component instead of the sample dataset to show you how to train models using your own data.



Create a pipeline parameter

Pipeline parameters are used to build versatile pipelines which can be resubmitted later with varying parameter values. Some common scenarios are updating datasets or some hyper-parameters for retraining. Create pipeline parameters to dynamically set variables at runtime.

Pipeline parameters can be added to data source or component parameters in a pipeline. When the pipeline is resubmitted, the values of these parameters can be specified.

For this example, you will change the training data path from a fixed value to a parameter, so that you can retrain your model on different data. You can also add other component parameters as pipeline parameters according to your use case.

1. Select the **Import Data** component.

NOTE

This example uses the Import Data component to access data in a registered datastore. However, you can follow similar steps if you use alternative data access patterns.

2. In the component detail pane, to the right of the canvas, select your data source.
3. Enter the path to your data. You can also select **Browse path** to browse your file tree.
4. Mouseover the **Path** field, and select the ellipses above the **Path** field that appear.
5. Select **Add to pipeline parameter**.
6. Provide a parameter name and a default value.

Import Data

Data source *

Datastore

Datastore *

New datastore

mydatastore

Path *

Path

To include files in subfolders, enter a path relative to the root folder. Please input a value.

Parameter name

input-data-path

Value

data/us-income

Preview schema

Save Cancel

Regenerate

The screenshot shows the 'Import Data' dialog box. It has sections for 'Data source *' (set to 'Datastore'), 'Datastore *' (set to 'mydatastore'), and a 'Path' parameter named 'input-data-path' with the value 'data/us-income'. There is a 'Preview schema' section and a 'Save' button at the bottom.

7. Select **Save**.

NOTE

You can also detach a component parameter from pipeline parameter in the component detail pane, similar to adding pipeline parameters.

You can inspect and edit your pipeline parameters by selecting the **Settings** gear icon next to the title of your pipeline draft.

- After detaching, you can delete the pipeline parameter in the **Settings** pane.
- You can also add a pipeline parameter in the **Settings** pane, and then apply it on some component parameter.

8. Submit the pipeline job.

Publish a training pipeline

Publish a pipeline to a pipeline endpoint to easily reuse your pipelines in the future. A pipeline endpoint creates a REST endpoint to invoke pipeline in the future. In this example, your pipeline endpoint lets you reuse your pipeline to retrain a model on different data.

1. Select **Publish** above the designer canvas.

2. Select or create a pipeline endpoint.

NOTE

You can publish multiple pipelines to a single endpoint. Each pipeline in a given endpoint is given a version number, which you can specify when you call the pipeline endpoint.

3. Select **Publish**.

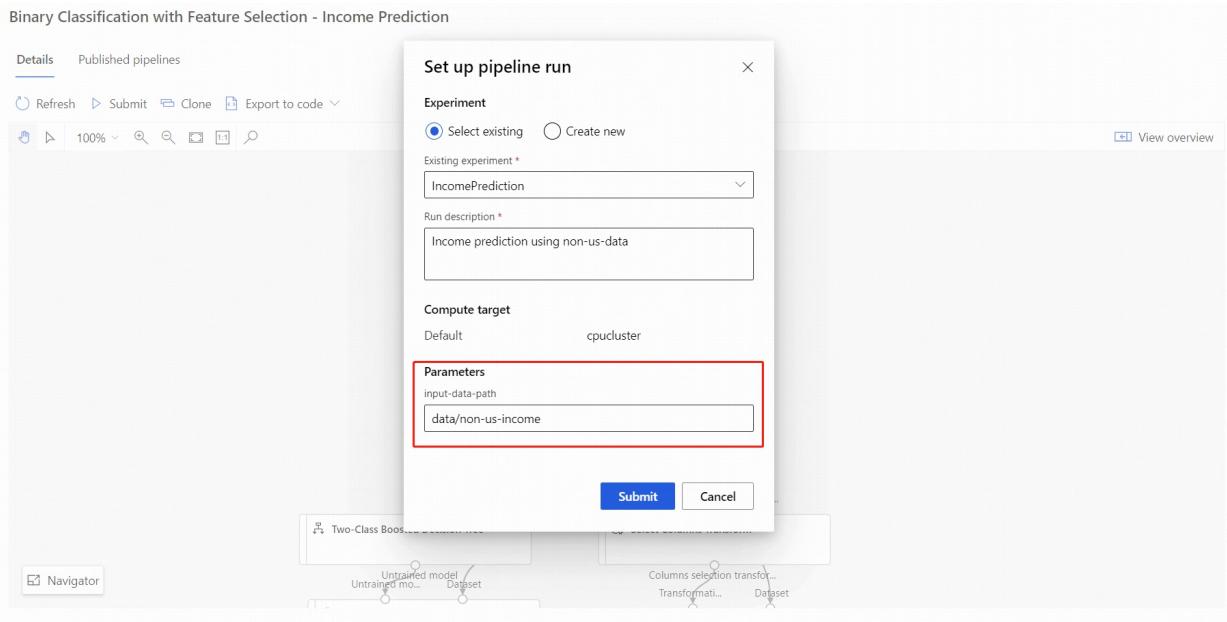
Retrain your model

Now that you have a published training pipeline, you can use it to retrain your model on new data. You can submit jobs from a pipeline endpoint from the studio workspace or programmatically.

Submit jobs by using the studio portal

Use the following steps to submit a parameterized pipeline endpoint job from the studio portal:

1. Go to the **Endpoints** page in your studio workspace.
2. Select the **Pipeline endpoints** tab. Then, select your pipeline endpoint.
3. Select the **Published pipelines** tab. Then, select the pipeline version that you want to run.
4. Select **Submit**.
5. In the setup dialog box, you can specify the parameters values for the job. For this example, update the data path to train your model using a non-US dataset.



Submit jobs by using code

You can find the REST endpoint of a published pipeline in the overview panel. By calling the endpoint, you can retrain the published pipeline.

To make a REST call, you need an OAuth 2.0 bearer-type authentication header. For information about setting up authentication to your workspace and making a parameterized REST call, see [Build an Azure Machine Learning pipeline for batch scoring](#).

Next steps

In this article, you learned how to create a parameterized training pipeline endpoint using the designer.

For a complete walkthrough of how you can deploy a model to make predictions, see the [designer tutorial](#) to train and deploy a regression model.

For how to publish and submit a job to pipeline endpoint using the SDK v1, see [this article](#).

Run batch predictions using Azure Machine Learning designer

9/22/2022 • 5 minutes to read • [Edit Online](#)

In this article, you learn how to use the designer to create a batch prediction pipeline. Batch prediction lets you continuously score large datasets on-demand using a web service that can be triggered from any HTTP library.

In this how-to, you learn to do the following tasks:

- Create and publish a batch inference pipeline
- Consume a pipeline endpoint
- Manage endpoint versions

To learn how to set up batch scoring services using the SDK, see the accompanying [tutorial on pipeline batch scoring](#).

NOTE

Azure Machine Learning Endpoints (preview) provide an improved, simpler deployment experience. Endpoints support both real-time and batch inference scenarios. Endpoints provide a unified interface to invoke and manage model deployments across compute types. See [What are Azure Machine Learning endpoints \(preview\)?](#).

Prerequisites

This how-to assumes you already have a training pipeline. For a guided introduction to the designer, complete [part one of the designer tutorial](#).

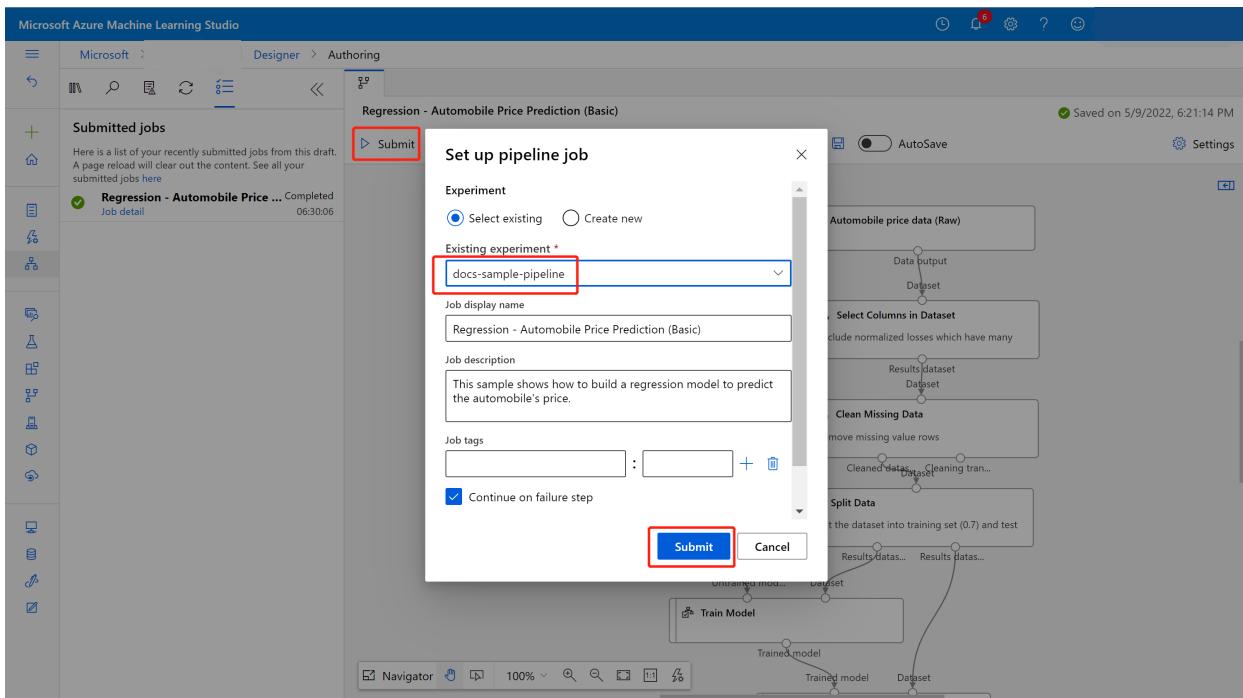
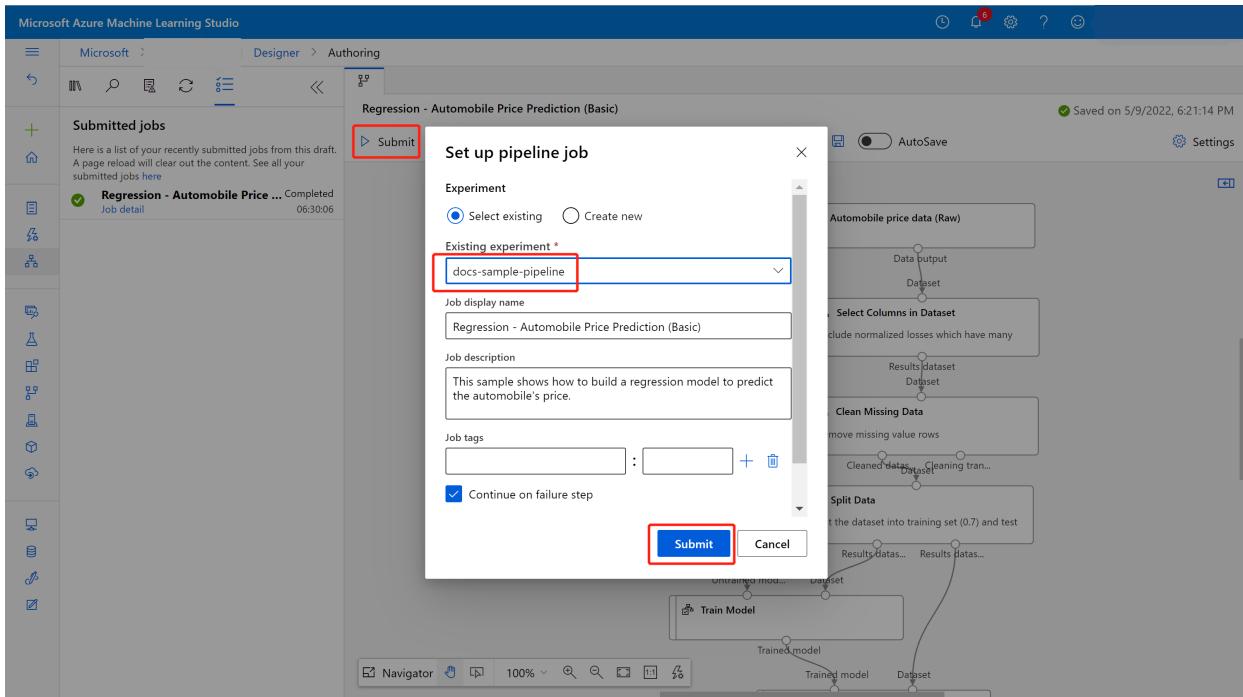
IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

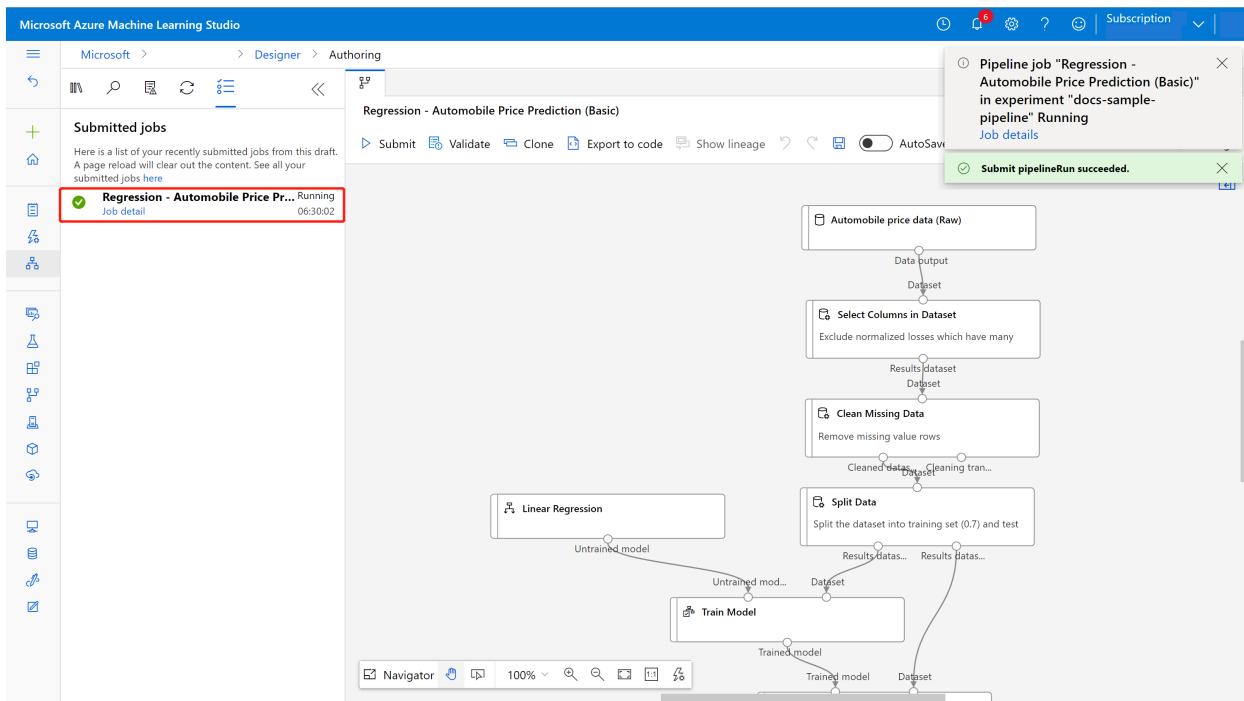
Create a batch inference pipeline

Your training pipeline must be run at least once to be able to create an inferencing pipeline.

1. Go to the **Designer** tab in your workspace.
2. Select the training pipeline that trains the model you want to use to make prediction.
3. **Submit** the pipeline.



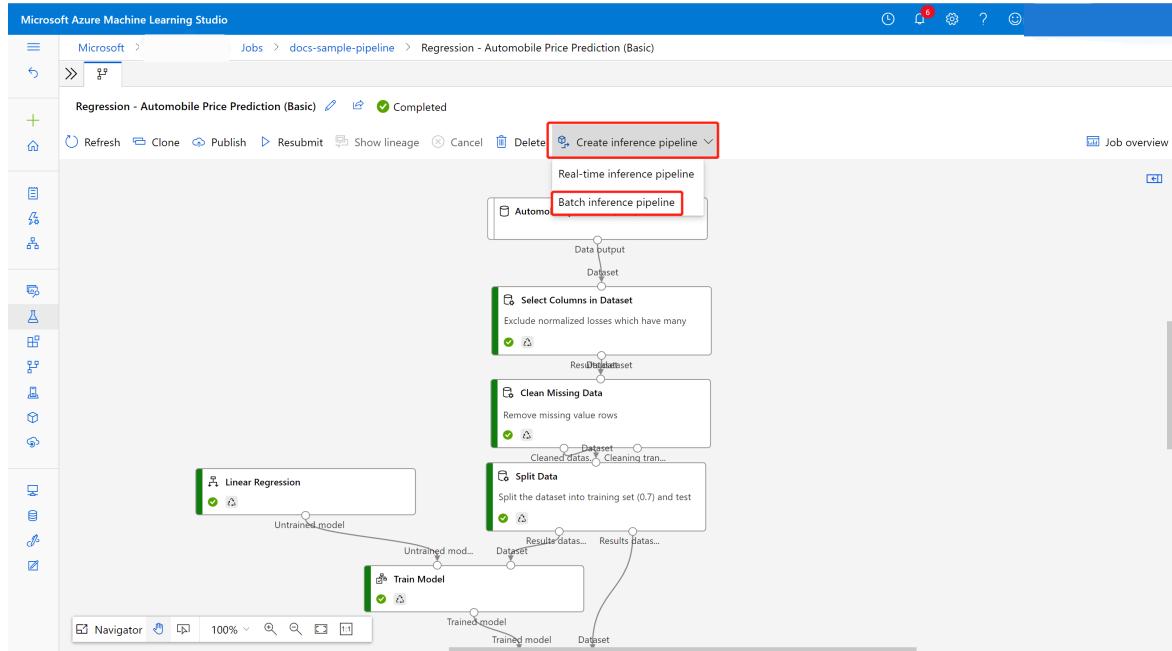
You'll see a submission list on the left of canvas. You can select the job detail link to go to the job detail page, and after the training pipeline job completes, you can create a batch inference pipeline.



1. In job detail page, above the canvas, select the dropdown **Create inference pipeline**. Select **Batch inference pipeline**.

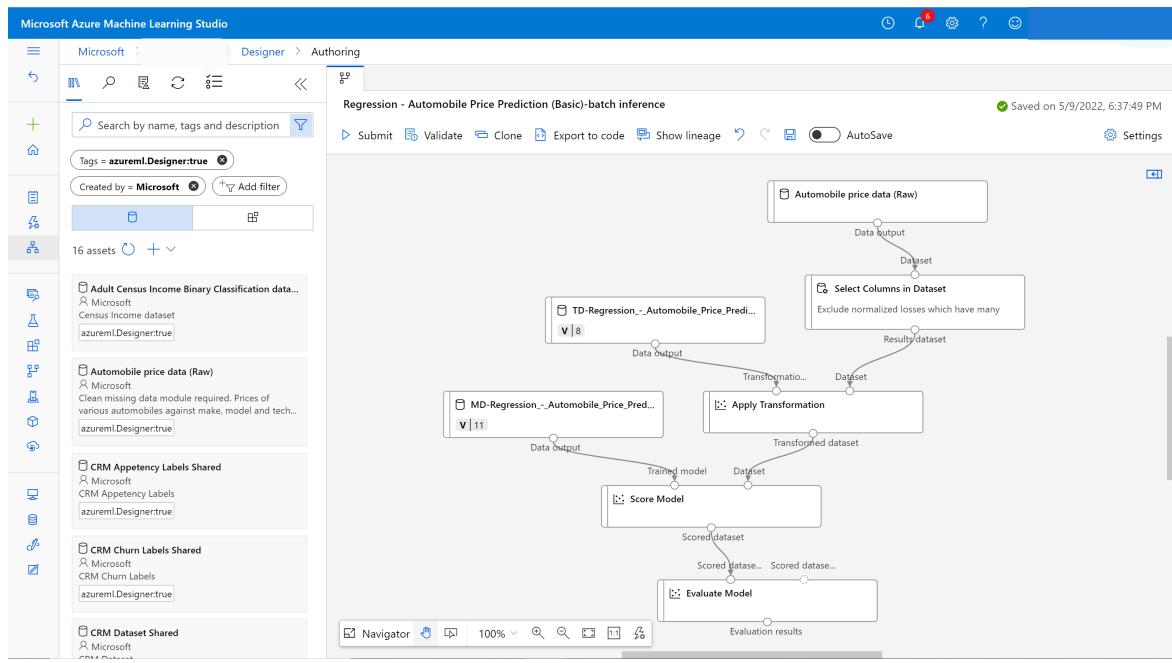
NOTE

Currently auto-generating inference pipeline only works for training pipeline built purely by the designer built-in components.



It will create a batch inference pipeline draft for you. The batch inference pipeline draft uses the trained model as MD- node and transformation as TD- node from the training pipeline job.

You can also modify this inference pipeline draft to better handle your input data for batch inference.



Add a pipeline parameter

To create predictions on new data, you can either manually connect a different dataset in this pipeline draft view or create a parameter for your dataset. Parameters let you change the behavior of the batch inferencing process at runtime.

In this section, you create a dataset parameter to specify a different dataset to make predictions on.

1. Select the dataset component.
2. A pane will appear to the right of the canvas. At the bottom of the pane, select **Set as pipeline parameter**.

Enter a name for the parameter, or accept the default value.

Parameters	Outputs
Data name Cleaned dataset	ID
Data type File	Datasource type
Description	AmIDataset
Data type DataFrameDirectory	Description
Datastore name workspaceblobstore	Data type
Created time May 9, 2022 6:44 PM	Datastore name
Modified time May 9, 2022 6:44 PM	Created time

Set as pipeline parameter
Parameter name: **DataSet1**

3. Submit the batch inference pipeline and go to job detail page by selecting the job link in the left pane.

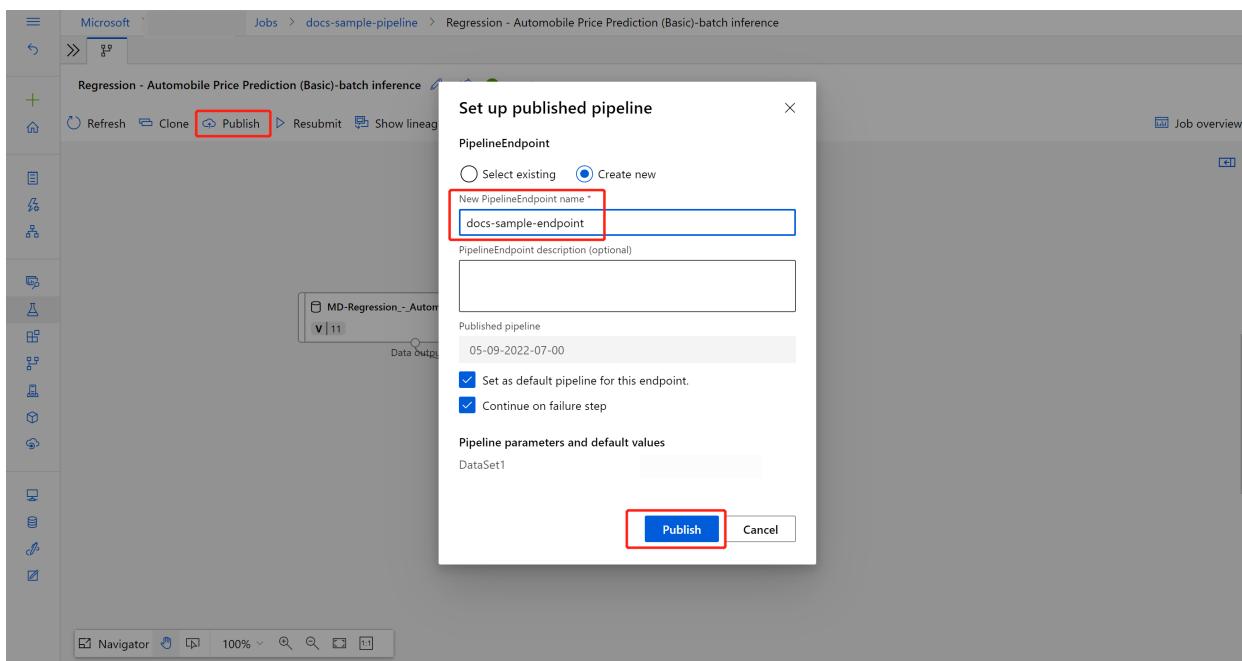
Publish your batch inference pipeline

Now you're ready to deploy the inference pipeline. This will deploy the pipeline and make it available for others to use.

1. Select the **Publish** button.
2. In the dialog that appears, expand the drop-down for **PipelineEndpoint**, and select **New PipelineEndpoint**.
3. Provide an endpoint name and optional description.

Near the bottom of the dialog, you can see the parameter you configured with a default value of the dataset ID used during training.

4. Select **Publish**.



Consume an endpoint

Now, you have a published pipeline with a dataset parameter. The pipeline will use the trained model created in the training pipeline to score the dataset you provide as a parameter.

Submit a pipeline job

In this section, you'll set up a manual pipeline job and alter the pipeline parameter to score new data.

1. After the deployment is complete, go to the **Endpoints** section.
2. Select **Pipeline endpoints**.
3. Select the name of the endpoint you created.

Pipelines

Pipeline jobs **Pipeline endpoints** Pipeline drafts

Refresh Disable Enable Edit columns

Search

Showing 1-25 of 27 pipeline endpoints

Name	Star
docs-sample-endpoint	
showcasing-datapath-test	
test regression 20220224-batch infer...	

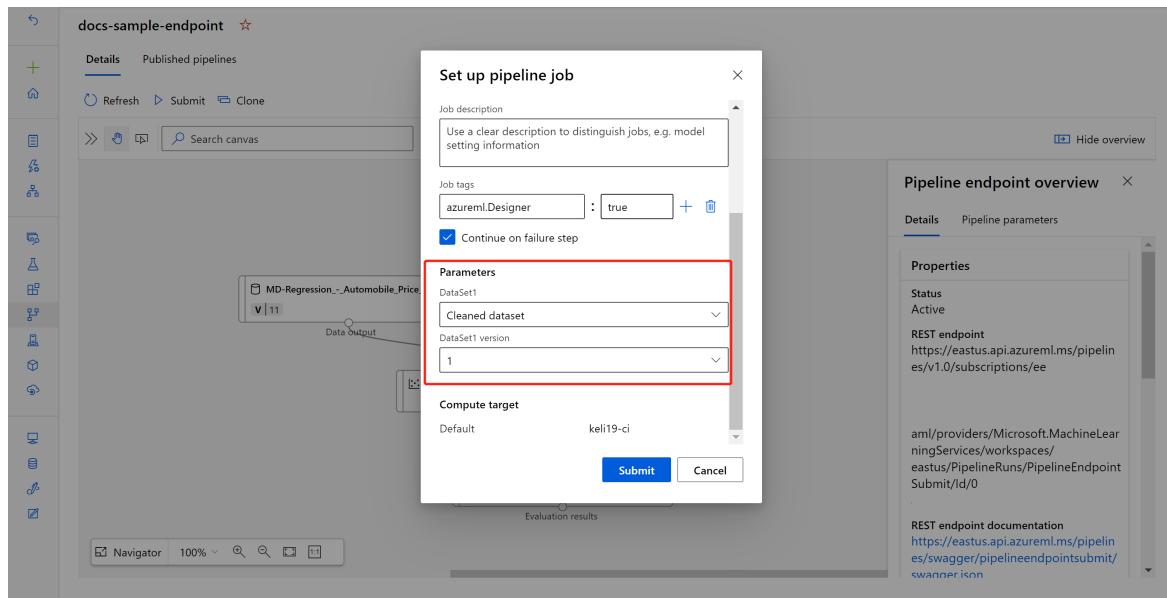
1. Select **Published pipelines**.

This screen shows all published pipelines published under this endpoint.

2. Select the pipeline you published.

The pipeline details page shows you a detailed job history and connection string information for your pipeline.

3. Select **Submit** to create a manual run of the pipeline.



4. Change the parameter to use a different dataset.

5. Select **Submit** to run the pipeline.

Use the REST endpoint

You can find information on how to consume pipeline endpoints and published pipeline in the **Endpoints** section.

You can find the REST endpoint of a pipeline endpoint in the job overview panel. By calling the endpoint, you're consuming its default published pipeline.

You can also consume a published pipeline in the **Published pipelines** page. Select a published pipeline and you can find the REST endpoint of it in the **Published pipeline overview** panel to the right of the graph.

To make a REST call, you'll need an OAuth 2.0 bearer-type authentication header. See the following [tutorial section](#) for more detail on setting up authentication to your workspace and making a parameterized REST call.

Versioning endpoints

The designer assigns a version to each subsequent pipeline that you publish to an endpoint. You can specify the pipeline version that you want to execute as a parameter in your REST call. If you don't specify a version number, the designer will use the default pipeline.

When you publish a pipeline, you can choose to make it the new default pipeline for that endpoint.

Set up published pipeline

PipelineEndpoint *

+ New PipelineEndpoint

New PipelineEndpoint name *

Sample pipeline

PipelineEndpoint description (optional)

Published pipeline

Sample pipeline 01-16-2020-02-34

- Set as default pipeline for this endpoint.
- Continue on failure step

Pipeline parameters and default values

DataSet1

Publish

Cancel

You can also set a new default pipeline in the **Published pipelines** tab of your endpoint.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a sidebar with icons for preview, workspace, endpoints, and datasets. The main area shows a navigation path: docs-ws > Endpoints > Sample pipeline. Below this, there are two tabs: 'Details' and 'Published pipelines'. The 'Published pipelines' tab is selected and has a red box around its header. It contains several buttons: Refresh, Disable, Enable, Set as Default (which is highlighted with a red box), and View disabled. Below these are three rows of pipeline details:

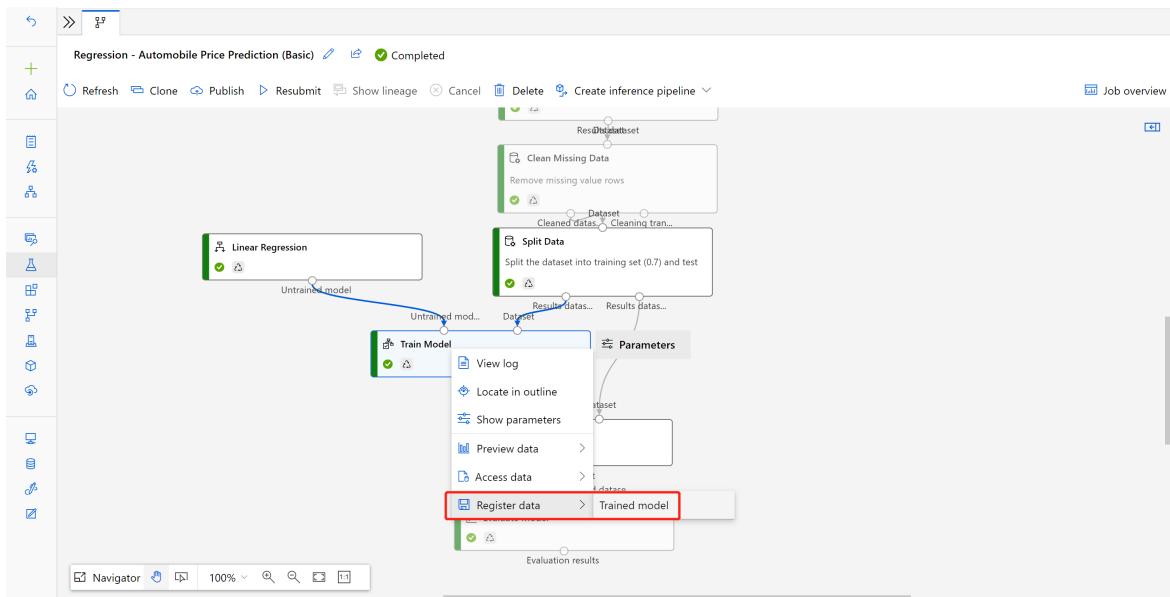
Name	Description	Data updated	Updated by	Last run submit time	Last run status	Version	
default	Pipeline-Create...	--	January 13, 2020 10:38 PM	John Doe	Not started	2	
<input checked="" type="checkbox"/>	Sample pipeline-batch...	--	January 13, 2020 10:31 PM	Jane Hamilton	Not started	1	
	Pipeline-Created-on-01...	--	January 13, 2020 9:57 PM	John Doe	January 13, 2020 10:03 PM	Failed	0

At the bottom, there are navigation arrows for 'Prev' and 'Next'.

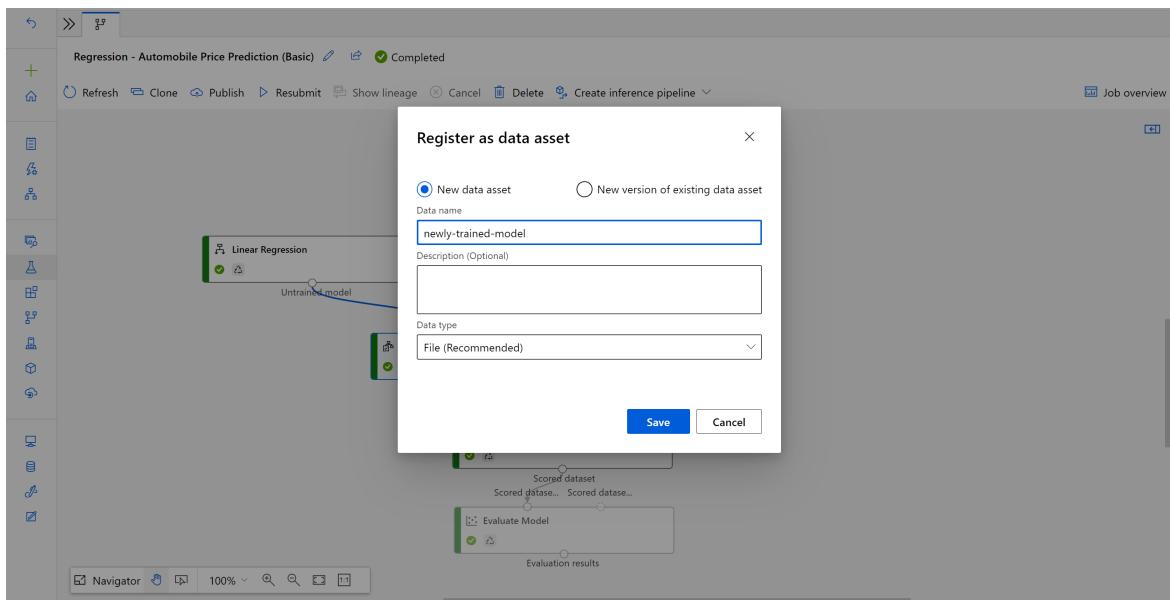
Update pipeline endpoint

If you make some modifications in your training pipeline, you may want to update the newly trained model to the pipeline endpoint.

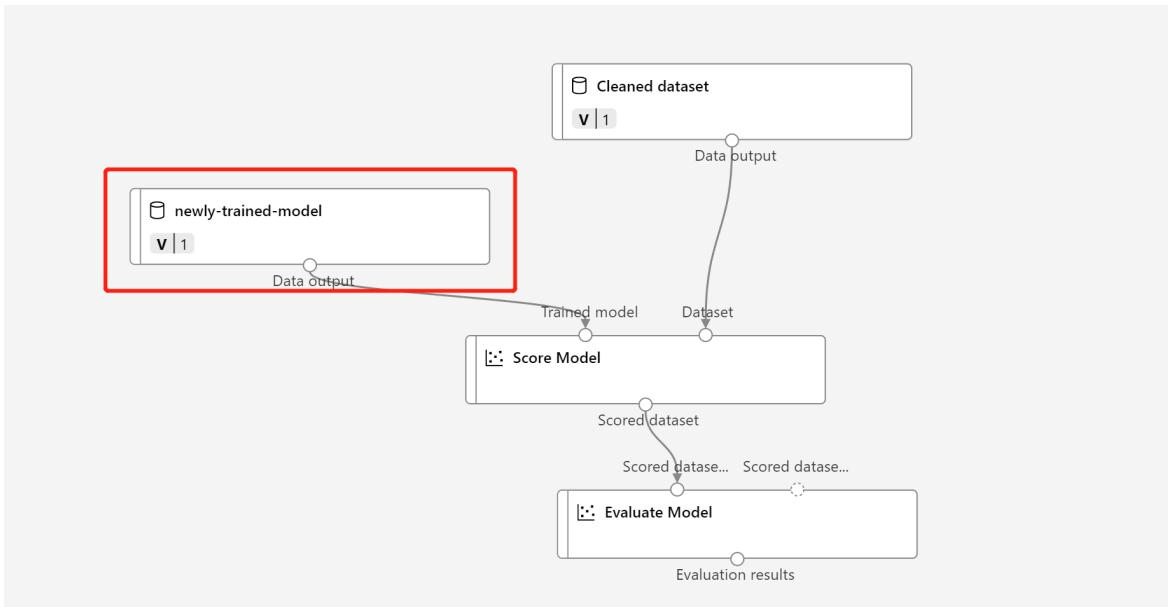
1. After your modified training pipeline completes successfully, go to the job detail page.
2. Right click **Train Model** component and select **Register data**



Input name and select **File** type.



3. Find the previous batch inference pipeline draft, or you can just **Clone** the published pipeline into a new draft.
4. Replace the **MD-** node in the inference pipeline draft with the registered data in the step above.



5. Updating data transformation node TD- is the same as the trained model.
6. Then you can submit the inference pipeline with the updated model and transformation, and publish again.

Next steps

- Follow the [designer tutorial to train and deploy a regression model](#).
- For how to publish and run a published pipeline using the SDK v1, see the [How to deploy pipelines](#) article.

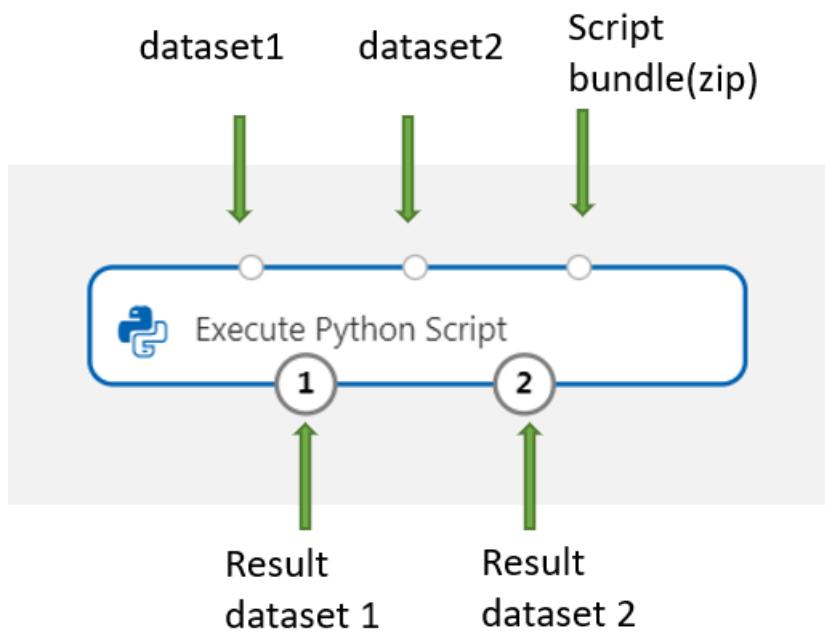
Run Python code in Azure Machine Learning designer

9/22/2022 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to use the [Execute Python Script](#) component to add custom logic to Azure Machine Learning designer. In the following how-to, you use the Pandas library to do simple feature engineering.

You can use the in-built code editor to quickly add simple Python logic. If you want to add more complex code or upload additional Python libraries, you should use the zip file method.

The default execution environment uses the Anacondas distribution of Python. For a complete list of pre-installed packages, see the [Execute Python Script component reference](#) page.



IMPORTANT

If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Execute Python written in the designer

Add the Execute Python Script component

1. Find the **Execute Python Script** component in the designer palette. It can be found in the **Python Language** section.
2. Drag and drop the component onto the pipeline canvas.

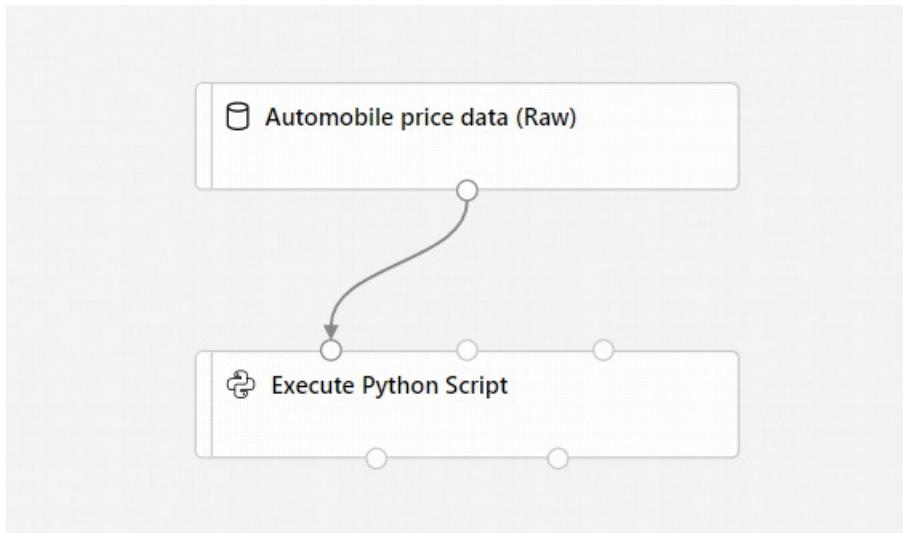
Connect input datasets

This article uses the sample dataset, **Automobile price data (Raw)**.

1. Drag and drop your dataset to the pipeline canvas.

2. Connect the output port of the dataset to the top-left input port of the **Execute Python Script** component. The designer exposes the input as a parameter to the entry point script.

The right input port is reserved for zipped Python libraries.



3. Take note of which input port you use. The designer assigns the left input port to the variable `dataset1` and the middle input port to `dataset2`.

Input components are optional since you can generate or import data directly in the **Execute Python Script** component.

Write your Python code

The designer provides an initial entry point script for you to edit and enter your own Python code.

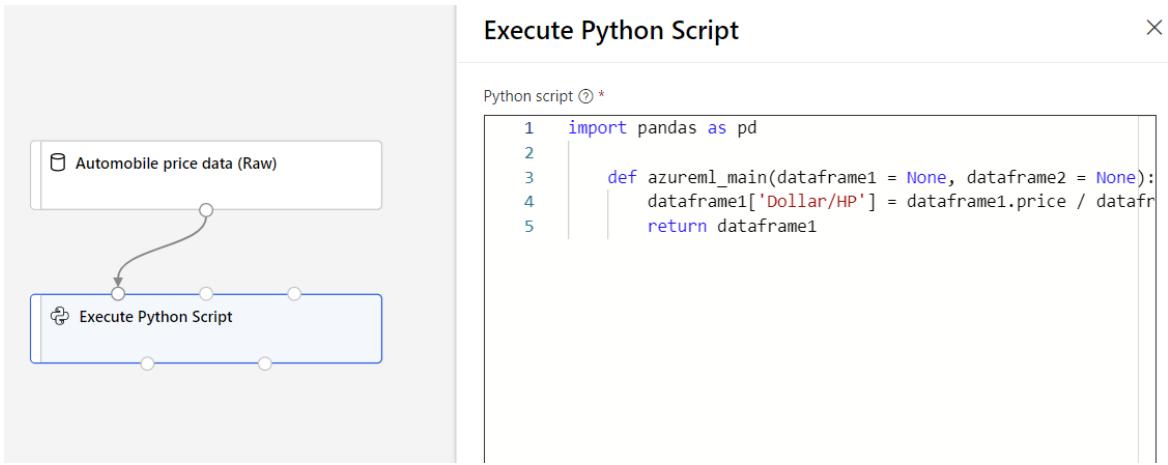
In this example, you use Pandas to combine two columns found in the automobile dataset, **Price** and **Horsepower**, to create a new column, **Dollars per horsepower**. This column represents how much you pay for each horsepower, which could be a useful feature to decide if a car is a good deal for the money.

1. Select the **Execute Python Script** component.
2. In the pane that appears to the right of the canvas, select the **Python script** text box.
3. Copy and paste the following code into the text box.

```
import pandas as pd

def azureml_main(dataframe1 = None, dataframe2 = None):
    dataframe1['Dollar/HP'] = dataframe1.price / dataframe1.horsepower
    return dataframe1
```

Your pipeline should look the following image:



The entry point script must contain the function `azureml_main`. There are two function parameters that map to the two input ports for the **Execute Python Script** component.

The return value must be a Pandas Dataframe. You can return up to two dataframes as component outputs.

4. Submit the pipeline.

Now, you have a dataset with the new feature **Dollars/HP**, which could be useful in training a car recommender. This is an example of feature extraction and dimensionality reduction.

Next steps

Learn how to [import your own data](#) in Azure Machine Learning designer.

Manage and increase quotas for resources with Azure Machine Learning

9/22/2022 • 10 minutes to read • [Edit Online](#)

Azure uses limits and quotas to prevent budget overruns due to fraud, and to honor Azure capacity constraints. Consider these limits as you scale for production workloads. In this article, you learn about:

- Default limits on Azure resources related to [Azure Machine Learning](#).
- Creating workspace-level quotas.
- Viewing your quotas and limits.
- Requesting quota increases.

Along with managing quotas, you can learn how to [plan and manage costs for Azure Machine Learning](#) or learn about the [service limits in Azure Machine Learning](#).

Special considerations

- A quota is a credit limit, not a capacity guarantee. If you have large-scale capacity needs, [contact Azure support to increase your quota](#).
- A quota is shared across all the services in your subscriptions, including Azure Machine Learning. Calculate usage across all services when you're evaluating capacity.

Azure Machine Learning compute is an exception. It has a separate quota from the core compute quota.

- Default limits vary by offer category type, such as free trial, pay-as-you-go, and virtual machine (VM) series (such as Dv2, F, and G).

Default resource quotas

In this section, you learn about the default and maximum quota limits for the following resources:

- Azure Machine Learning assets
 - Azure Machine Learning compute
 - Azure Machine Learning managed online endpoints
 - Azure Machine Learning pipelines
- Virtual machines
- Azure Container Instances
- Azure Storage

IMPORTANT

Limits are subject to change. For the latest information, see [Service limits in Azure Machine Learning](#).

Azure Machine Learning assets

The following limits on assets apply on a per-workspace basis.

RESOURCE	MAXIMUM LIMIT
Datasets	10 million
Runs	10 million
Models	10 million
Artifacts	10 million

In addition, the maximum **run time** is 30 days and the maximum number of **metrics logged per run** is 1 million.

Azure Machine Learning Compute

[Azure Machine Learning Compute](#) has a default quota limit on both the number of cores (split by each VM Family and cumulative total cores) and the number of unique compute resources allowed per region in a subscription. This quota is separate from the VM core quota listed in the previous section as it applies only to the managed compute resources of Azure Machine Learning.

[Request a quota increase](#) to raise the limits for various VM family core quotas, total subscription core quotas, cluster quota and resources in this section.

Available resources:

- **Dedicated cores per region** have a default limit of 24 to 300, depending on your subscription offer type. You can increase the number of dedicated cores per subscription for each VM family. Specialized VM families like NCv2, NCv3, or ND series start with a default of zero cores. GPUs also default to zero cores.
- **Low-priority cores per region** have a default limit of 100 to 3,000, depending on your subscription offer type. The number of low-priority cores per subscription can be increased and is a single value across VM families.
- **Clusters per region** have a default limit of 200. This limit is shared between training clusters, compute instances and MIR endpoint deployments. (A compute instance is considered a single-node cluster for quota purposes.) Cluster quota can be increased up to a value of 500 per region within a given subscription.

TIP

To learn more about which VM family to request a quota increase for, check out [virtual machine sizes in Azure](#). For instance GPU VM families start with an "N" in their family name (eg. NCv3 series)

The following table shows more limits in the platform. Reach out to the AzureML product team through a [technical support ticket](#) to request an exception.

RESOURCE OR ACTION	MAXIMUM LIMIT
Workspaces per resource group	800
Nodes in a single Azure Machine Learning Compute (AmlCompute) cluster set up as a non communication-enabled pool (that is, can't run MPI jobs)	100 nodes but configurable up to 65,000 nodes

RESOURCE OR ACTION	MAXIMUM LIMIT
Nodes in a single Parallel Run Step run on an Azure Machine Learning Compute (AmlCompute) cluster	100 nodes but configurable up to 65,000 nodes if your cluster is set up to scale per above
Nodes in a single Azure Machine Learning Compute (AmlCompute) cluster set up as a communication-enabled pool	300 nodes but configurable up to 4000 nodes
Nodes in a single Azure Machine Learning Compute (AmlCompute) cluster set up as a communication-enabled pool on an RDMA enabled VM Family	100 nodes
Nodes in a single MPI run on an Azure Machine Learning Compute (AmlCompute) cluster	100 nodes but can be increased to 300 nodes
Job lifetime	21 days ¹
Job lifetime on a low-priority node	7 days ²
Parameter servers per node	1

¹ Maximum lifetime is the duration between when a job starts and when it finishes. Completed jobs persist indefinitely. Data for jobs not completed within the maximum lifetime isn't accessible.

² Jobs on a low-priority node can be preempted whenever there's a capacity constraint. We recommend that you implement checkpoints in your job.

Azure Machine Learning managed online endpoints

Azure Machine Learning managed online endpoints have limits described in the following table.

RESOURCE	LIMIT
Endpoint name	Endpoint names must <ul style="list-style-type: none">• Begin with a letter• Be 3-32 characters in length• Only consist of letters and numbers ¹
Deployment name	Deployment names must <ul style="list-style-type: none">• Begin with a letter• Be 3-32 characters in length• Only consist of letters and numbers ¹
Number of endpoints per subscription	50
Number of deployments per subscription	200
Number of deployments per endpoint	20
Number of instances per deployment	20 ²
Max request time-out at endpoint level	90 seconds
Total requests per second at endpoint level for all deployments	500 ³

RESOURCE	LIMIT
Total connections per second at endpoint level for all deployments	500 ³
Total connections active at endpoint level for all deployments	500 ³
Total bandwidth at endpoint level for all deployments	5 MBPS ³

¹ Single dashes like, `my-endpoint-name`, are accepted in endpoint and deployment names.

² We reserve 20% extra compute resources for performing upgrades. For example, if you request 10 instances in a deployment, you must have a quota for 12. Otherwise, you'll receive an error.

³ If you request a limit increase, be sure to calculate related limit increases you might need. For example, if you request a limit increase for requests per second, you might also want to compute the required connections and bandwidth limits and include these limit increases in the same request.

To determine the current usage for an endpoint, [view the metrics](#).

To request an exception from the Azure Machine Learning product team, use the steps in the [Request quota increases](#).

Azure Machine Learning pipelines

Azure Machine Learning pipelines have the following limits.

RESOURCE	LIMIT
Steps in a pipeline	30,000
Workspaces per resource group	800

Virtual machines

Each Azure subscription has a limit on the number of virtual machines across all services. Virtual machine cores have a regional total limit and a regional limit per size series. Both limits are separately enforced.

For example, consider a subscription with a US East total VM core limit of 30, an A series core limit of 30, and a D series core limit of 30. This subscription would be allowed to deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two that doesn't exceed a total of 30 cores.

You can't raise limits for virtual machines above the values shown in the following table.

RESOURCE	LIMIT
Subscriptions associated with an Azure Active Directory tenant	Unlimited
Coadministrators per subscription	Unlimited
Resource groups per subscription	980
Azure Resource Manager API request size	4,194,304 bytes
Tags per subscription ¹	50

RESOURCE	LIMIT
Unique tag calculations per subscription ²	80,000
Subscription-level deployments per location	800 ³
Locations of Subscription-level deployments	10

¹You can apply up to 50 tags directly to a subscription. However, the subscription can contain an unlimited number of tags that are applied to resource groups and resources within the subscription. The number of tags per resource or resource group is limited to 50.

²Resource Manager returns a [list of tag name and values](#) in the subscription only when the number of unique tags is 80,000 or less. A unique tag is defined by the combination of resource ID, tag name, and tag value. For example, two resources with the same tag name and value would be calculated as two unique tags. You still can find a resource by tag when the number exceeds 80,000.

³Deployments are automatically deleted from the history as you near the limit. For more information, see [Automatic deletions from deployment history](#).

Container Instances

For more information, see [Container Instances limits](#).

Storage

Azure Storage has a limit of 250 storage accounts per region, per subscription. This limit includes both Standard and Premium storage accounts.

To increase the limit, make a request through [Azure Support](#). The Azure Storage team will review your case and can approve up to 250 storage accounts for a region.

Workspace-level quotas

Use workspace-level quotas to manage Azure Machine Learning compute target allocation between multiple [workspaces](#) in the same subscription.

By default, all workspaces share the same quota as the subscription-level quota for VM families. However, you can set a maximum quota for individual VM families on workspaces in a subscription. This lets you share capacity and avoid resource contention issues.

1. Go to any workspace in your subscription.
2. In the left pane, select **Usages + quotas**.
3. Select the **Configure quotas** tab to view the quotas.
4. Expand a VM family.
5. Set a quota limit on any workspace listed under that VM family.

You can't set a negative value or a value higher than the subscription-level quota.

Subscription *	Resource	Location *
Azure ML Team Testing	Machine Learning Compute	West Europe
Subscription View		Configure quotas
Resource name	Current limit	New limit
> Standard D Family vCPUs	100	
> Standard DSV2 Family vCPUs	100	
Standard Dv2 Family vCPUs	100	
azureml-dogbreeds (azureml-webinar)	10	<input type="text" value="10"/>
azureml-webinar (azureml)	20	<input type="text" value="20"/>
azuremlwbatchai_yopzkjia (azureml)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
build19-weurope (build19)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
danielsc (aml-workshop)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
ignite2019 (ignite2019)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
keras (azureml-webinar)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
maxluk-azml (azureml-webinar)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
pytorch-bert-squad (azureml-webinar)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
test (amlworkspace)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
test-sku-ws (azureml)	50	<input type="text" value="50"/>
test2 (aml-workshop)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
testws1 (azureml)	100	<input type="text" value="Unallocated cores: 0, Maximum: 100"/>
> Standard FSv2 Family vCPUs	100	

NOTE

You need subscription-level permissions to set a quota at the workspace level.

View quotas in the studio

- When you create a new compute resource, by default you'll see only VM sizes that you already have quota to use. Switch the view to **Select from all options**.

Create compute cluster

Virtual Machine

Advanced Settings

Dedicated Low priority

Virtual machine type

CPU GPU

Virtual machine size

Select from recommended options **Select from all options**

- Scroll down until you see the list of VM sizes you don't have quota for.

Create compute cluster

Virtual Machine

Advanced Settings

Standard_F1s_v2 72 cores, 144GB RAM, 576GB storage	Compute optimized	96 cores
Standard_F8s_v2 8 cores, 16GB RAM, 64GB storage	Compute optimized	96 cores
You do not have enough quota for the following VM sizes. It is possible that you have quota in a different location. Click here to view and request quota.		
Standard_D16_v3 16 cores, 64GB RAM, 400GB storage	General purpose	0 cores
Standard_D16as_v4 16 cores, 64GB RAM, 128GB storage	General purpose	0 cores
Standard_D16ds_v4 16 cores, 64GB RAM, 600GB storage	General purpose	0 cores

- Use the link to go directly to the online customer support request for more quota.

View your usage and quotas in the Azure portal

To view your quota for various Azure resources like virtual machines, storage, or network, use the [Azure portal](#):

1. On the left pane, select **All services** and then select **Subscriptions** under the **General** category.
2. From the list of subscriptions, select the subscription whose quota you're looking for.
3. Select **Usage + quotas** to view your current quota limits and usage. Use the filters to select the provider and locations.

You manage the Azure Machine Learning compute quota on your subscription separately from other Azure quotas:

1. Go to your **Azure Machine Learning** workspace in the Azure portal.
2. On the left pane, in the **Support + troubleshooting** section, select **Usage + quotas** to view your current quota limits and usage.
3. Select a subscription to view the quota limits. Filter to the region you're interested in.
4. You can switch between a subscription-level view and a workspace-level view.

Request quota increases

To raise the limit or quota above the default limit, [open an online customer support request](#) at no charge.

You can't raise limits above the maximum values shown in the preceding tables. If there's no maximum limit, you can't adjust the limit for the resource.

When you're requesting a quota increase, select the service that you have in mind. For example, select Azure Machine Learning, Container Instances, or Storage. For Azure Machine Learning compute, you can select the **Request Quota** button while viewing the quota in the preceding steps.

NOTE

Free trial subscriptions are not eligible for limit or quota increases. If you have a free trial subscription, you can upgrade to a pay-as-you-go subscription. For more information, see [Upgrade Azure free trial to pay-as-you-go](#) and [Azure free account FAQ](#).

Endpoint quota increases

When requesting the quota increase, provide the following information:

1. When opening the support request, select **Machine Learning Service: Endpoint Limits** as the **Quota type**.
2. On the **Additional details** tab, select **Enter details** and then provide the quota you'd like to increase and the new value, the reason for the quota increase request, and **location(s)** where you need the quota increase. Finally, select **Save and continue** to continue.

1. Problem description 2. Recommended solution 3. Additional details 4. Review + create

Tell us a little more information.

Providing detailed, accurate information helps us resolve your issue faster.

Problem details

Additional information is required to promptly process your request for a quota increase.

Request details *

Provide details for the request

[Enter details](#)

File upload

Select a file

Quota details

To help us better understand your usage scenario, what is your business reason for requesting a quota increase? If possible, provide the peak QPS (queries per second) for scoring your endpoints, any schedule expected for QPS, any expectation of performance, plan for usage of endpoints and deployments.

For example: our team expects to utilize 1k QPS (queries per second) most of the day with spikes of 30k QPS in the evenings. We plan to have about 100 endpoints each with 2 or so deployments.

Be sure to calculate all related limit changes. Fill in relevant values below with the new desired limits:

Number of endpoints per subscription (Default: 50)

N/A

Number of deployments per subscription (Default: 200)

N/A

Number of deployments per endpoint (Default: 20)

N/A

Number of instances per deployment (Default: 20)

N/A

Total requests per second at endpoint level for all deployments (RPS) (Default: 500)

N/A

Total connections per second at endpoint level for all deployments (Default: 100)

N/A

Total connections active at endpoint level for all deployments (Default: 100)

N/A

[Save and continue](#)

Support method

Support plan

Azure Support Plan - Internal

[Previous](#) [Next](#)

Next steps

- [Plan and manage costs for Azure Machine Learning](#)
- [Service limits in Azure Machine Learning](#)
- [Troubleshooting managed online endpoints deployment and scoring](#)

Manage and optimize Azure Machine Learning costs

9/22/2022 • 6 minutes to read • [Edit Online](#)

Learn how to manage and optimize costs when training and deploying machine learning models to Azure Machine Learning.

Use the following tips to help you manage and optimize your compute resource costs.

- Configure your training clusters for autoscaling
- Set quotas on your subscription and workspaces
- Set termination policies on your training job
- Use low-priority virtual machines (VM)
- Schedule compute instances to shut down and start up automatically
- Use an Azure Reserved VM Instance
- Train locally
- Parallelize training
- Set data retention and deletion policies
- Deploy resources to the same region

For information on planning and monitoring costs, see the [plan to manage costs for Azure Machine Learning](#) guide.

Use Azure Machine Learning compute cluster (AmlCompute)

With constantly changing data, you need fast and streamlined model training and retraining to maintain accurate models. However, continuous training comes at a cost, especially for deep learning models on GPUs.

Azure Machine Learning users can use the managed Azure Machine Learning compute cluster, also called AmlCompute. AmlCompute supports a variety of GPU and CPU options. The AmlCompute is internally hosted on behalf of your subscription by Azure Machine Learning. It provides the same enterprise grade security, compliance and governance at Azure IaaS cloud scale.

Because these compute pools are inside of Azure's IaaS infrastructure, you can deploy, scale, and manage your training with the same security and compliance requirements as the rest of your infrastructure. These deployments occur in your subscription and obey your governance rules. Learn more about [Azure Machine Learning compute](#).

Configure training clusters for autoscaling

Autoscaling clusters based on the requirements of your workload helps reduce your costs so you only use what you need.

AmlCompute clusters are designed to scale dynamically based on your workload. The cluster can be scaled up to the maximum number of nodes you configure. As each job completes, the cluster will release nodes and scale to your configured minimum node count.

IMPORTANT

To avoid charges when no jobs are running, **set the minimum nodes to 0**. This setting allows Azure Machine Learning to de-allocate the nodes when they aren't in use. Any value larger than 0 will keep that number of nodes running, even if they are not in use.

You can also configure the amount of time the node is idle before scale down. By default, idle time before scale down is set to 120 seconds.

- If you perform less iterative experimentation, reduce this time to save costs.
- If you perform highly iterative dev/test experimentation, you might need to increase the time so you aren't paying for constant scaling up and down after each change to your training script or environment.

AmlCompute clusters can be configured for your changing workload requirements in Azure portal, using the [AmlCompute SDK class](#), [AmlCompute CLI](#), with the [REST APIs](#).

Set quotas on resources

AmlCompute comes with a [quota \(or limit\) configuration](#). This quota is by VM family (for example, Dv2 series, NCv3 series) and varies by region for each subscription. Subscriptions start with small defaults to get you going, but use this setting to control the amount of Amlcompute resources available to be spun up in your subscription.

Also configure [workspace level quota by VM family](#), for each workspace within a subscription. Doing so allows you to have more granular control on the costs that each workspace might potentially incur and restrict certain VM families.

To set quotas at the workspace level, start in the [Azure portal](#). Select any workspace in your subscription, and select **Usages + quotas** in the left pane. Then select the **Configure quotas** tab to view the quotas. You need privileges at the subscription scope to set the quota, since it's a setting that affects multiple workspaces.

Set job autotermination policies

In some cases, you should configure your training runs to limit their duration or terminate them early. For example, when you are using Azure Machine Learning's built-in hyperparameter tuning or automated machine learning.

Here are a few options that you have:

- Define a parameter called `max_run_duration_seconds` in your RunConfiguration to control the maximum duration a run can extend to on the compute you choose (either local or remote cloud compute).
- For [hyperparameter tuning](#), define an early termination policy from a Bandit policy, a Median stopping policy, or a Truncation selection policy. To further control hyperparameter sweeps, use parameters such as `max_total_runs` or `max_duration_minutes`.
- For [automated machine learning](#), set similar termination policies using the `enable_early_stopping` flag. Also use properties such as `iteration_timeout_minutes` and `experiment_timeout_minutes` to control the maximum duration of a job or for the entire experiment.

Use low-priority VMs

Azure allows you to use excess unutilized capacity as Low-Priority VMs across virtual machine scale sets, Batch, and the Machine Learning service. These allocations are pre-emptible but come at a reduced price compared to dedicated VMs. In general, we recommend using Low-Priority VMs for Batch workloads. You should also use them where interruptions are recoverable either through resubmits (for Batch Inferencing) or through restarts.

(for deep learning training with checkpointing).

Low-Priority VMs have a single quota separate from the dedicated quota value, which is by VM family. Learn [more about AmlCompute quotas](#).

Low-Priority VMs don't work for compute instances, since they need to support interactive notebook experiences.

Schedule compute instances

When you create a [compute instance](#), the VM stays on so it is available for your work. [Set up a schedule](#) to automatically start and stop the compute instance (preview) to save cost when you aren't planning to use it.

Use reserved instances

Another way to save money on compute resources is Azure Reserved VM Instance. With this offering, you commit to one-year or three-year terms. These discounts range up to 72% of the pay-as-you-go prices and are applied directly to your monthly Azure bill.

Azure Machine Learning Compute supports reserved instances inherently. If you purchase a one-year or three-year reserved instance, we will automatically apply discount against your Azure Machine Learning managed compute.

Train locally

When prototyping and running training jobs that are small enough to run on your local computer, consider training locally. Using the Python SDK, setting your compute target to `local` executes your script locally.

Visual Studio Code provides a full-featured environment for developing your machine learning applications. Using the Azure Machine Learning visual Visual Studio Code extension and Docker, you can run and debug locally. For more information, see [interactive debugging with Visual Studio Code](#).

Parallelize training

One of the key methods of optimizing cost and performance is by parallelizing the workload with the help of a parallel run step in Azure Machine Learning. This step allows you to use many smaller nodes to execute the task in parallel, hence allowing you to scale horizontally. There is an overhead for parallelization. Depending on the workload and the degree of parallelism that can be achieved, this may or may not be an option. For further information, see the [ParallelRunStep](#) documentation.

Set data retention & deletion policies

Every time a pipeline is executed, intermediate datasets are generated at each step. Over time, these intermediate datasets take up space in your storage account. Consider setting up policies to manage your data throughout its lifecycle to archive and delete your datasets. For more information, see [optimize costs by automating Azure Blob Storage access tiers](#).

Deploy resources to the same region

Computes located in different regions may experience network latency and increased data transfer costs. Azure network costs are incurred from outbound bandwidth from Azure data centers. To help reduce network costs, deploy all your resources in the region. Provisioning your Azure Machine Learning workspace and dependent resources in the same region as your data can help lower cost and improve performance.

For hybrid cloud scenarios like those using ExpressRoute, it can sometimes be more cost effective to move all

resources to Azure to optimize network costs and latency.

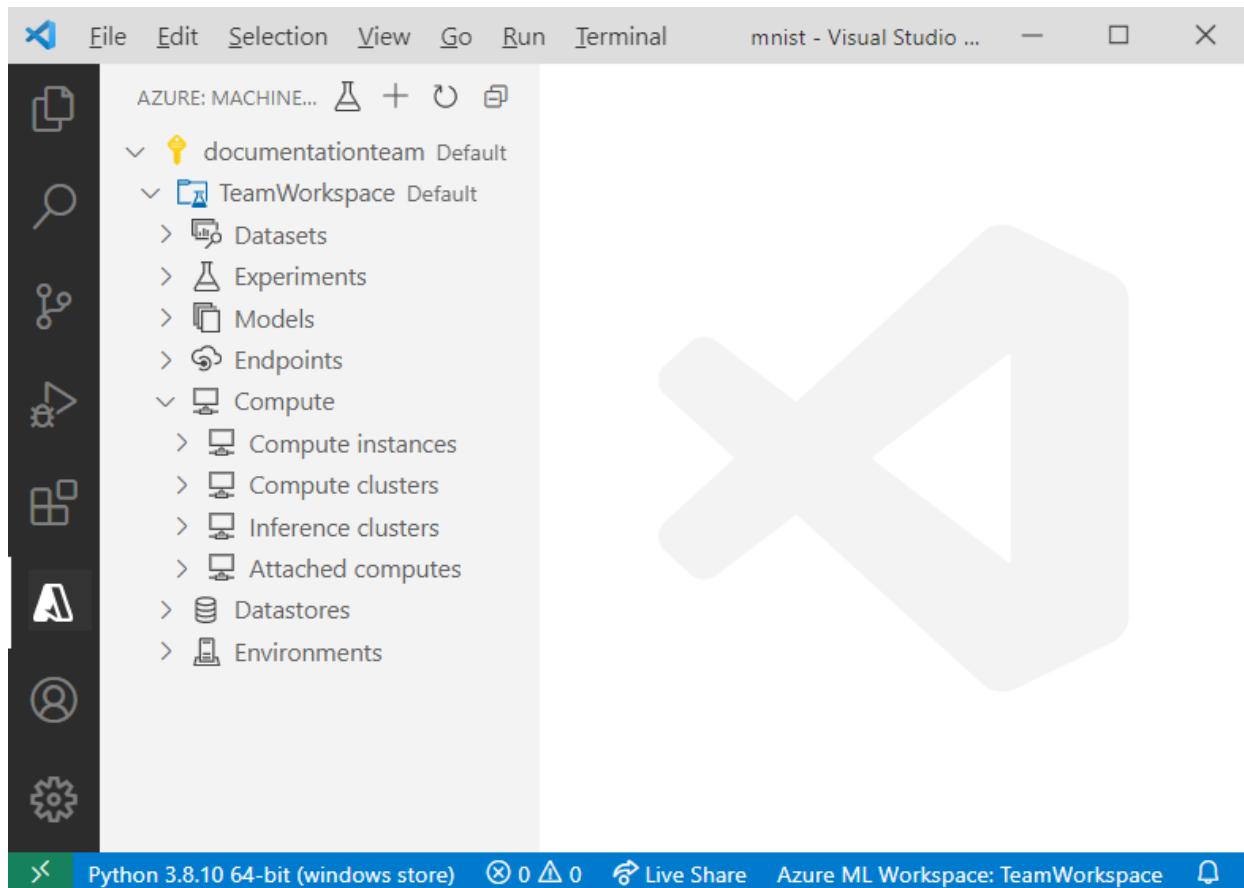
Next steps

- [Plan to manage costs for Azure Machine Learning](#)
- [Manage budgets, costs, and quota for Azure Machine Learning at organizational scale](#)

Manage Azure Machine Learning resources with the VS Code Extension (preview)

9/22/2022 • 11 minutes to read • [Edit Online](#)

Learn how to manage Azure Machine Learning resources with the VS Code extension.



Prerequisites

- Azure subscription. If you don't have one, sign up to try the [free or paid version of Azure Machine Learning](#).
- Visual Studio Code. If you don't have it, [install it](#).
- Azure Machine Learning extension. Follow the [Azure Machine Learning VS Code extension installation guide](#) to set up the extension.

Create resources

The quickest way to create resources is using the extension's toolbar.

1. Open the Azure Machine Learning view.
2. Select + in the activity bar.
3. Choose your resource from the dropdown list.
4. Configure the specification file. The information required depends on the type of resource you want to create.
5. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, you can create a resource by using the command palette:

1. Open the command palette **View > Command Palette**

2. Enter `> Azure ML: Create <RESOURCE-TYPE>` into the text box. Replace `<RESOURCE-TYPE>` with the type of resource you want to create.
3. Configure the specification file.
4. Open the command palette **View > Command Palette**
5. Enter `> Azure ML: Create Resource` into the text box.

Version resources

Some resources like environments, datasets, and models allow you to make changes to a resource and store the different versions.

To version a resource:

1. Use the existing specification file that created the resource or follow the create resources process to create a new specification file.
2. Increment the version number in the template.
3. Right-click the specification file and select **Azure ML: Execute YAML**.

As long as the name of the updated resource is the same as the previous version, Azure Machine Learning picks up the changes and creates a new version.

Workspaces

For more information, see [workspaces](#).

Create a workspace

1. In the Azure Machine Learning view, right-click your subscription node and select **Create Workspace**.
2. A specification file appears. Configure the specification file.
3. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Workspace` command in the command palette.

Remove workspace

1. Expand the subscription node that contains your workspace.
2. Right-click the workspace you want to remove.
3. Select whether you want to remove:
 - *Only the workspace*: This option deletes **only** the workspace Azure resource. The resource group, storage accounts, and any other resources the workspace was attached to are still in Azure.
 - *With associated resources*: This option deletes the workspace **and** all resources associated with it.

Alternatively, use the `> Azure ML: Remove Workspace` command in the command palette.

Datastores

The extension currently supports datastores of the following types:

- Azure Blob
- Azure Data Lake Gen 1
- Azure Data Lake Gen 2
- Azure File

For more information, see [datastore](#).

Create a datastore

1. Expand the subscription node that contains your workspace.
2. Expand the workspace node you want to create the datastore under.
3. Right-click the **Datastores** node and select **Create Datastore**.
4. Choose the datastore type.
5. A specification file appears. Configure the specification file.
6. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Datastore` command in the command palette.

Manage a datastore

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Datastores** node inside your workspace.
4. Right-click the datastore you want to:
 - *Unregister Datastore*. Removes datastore from your workspace.
 - *View Datastore*. Display read-only datastore settings

Alternatively, use the `> Azure ML: Unregister Datastore` and `> Azure ML: View Datastore` commands respectively in the command palette.

Datasets

The extension currently supports the following dataset types:

- *Tabular*: Allows you to materialize data into a DataFrame.
- *File*: A file or collection of files. Allows you to download or mount files to your compute.

For more information, see [datasets](#)

Create dataset

1. Expand the subscription node that contains your workspace.
2. Expand the workspace node you want to create the dataset under.
3. Right-click the **Datasets** node and select **Create Dataset**.
4. A specification file appears. Configure the specification file.
5. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Dataset` command in the command palette.

Manage a dataset

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Datasets** node.
4. Right-click the dataset you want to:
 - **View Dataset Properties**. Lets you view metadata associated with a specific dataset. If you have multiple version of a dataset, you can choose to only view the dataset properties of a specific version by expanding the dataset node and performing the same steps described in this section on the version of interest.
 - **Preview dataset**. View your dataset directly in the VS Code Data Viewer. Note that this option is only available for tabular datasets.
 - **Unregister dataset**. Removes a dataset and all versions of it from your workspace.

Alternatively, use the `> Azure ML: View Dataset Properties` and `> Azure ML: Unregister Dataset` commands

respectively in the command palette.

Environments

For more information, see [environments](#).

Create environment

1. Expand the subscription node that contains your workspace.
2. Expand the workspace node you want to create the datastore under.
3. Right-click the **Environments** node and select **Create Environment**.
4. A specification file appears. Configure the specification file.
5. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Environment` command in the command palette.

View environment configurations

To view the dependencies and configurations for a specific environment in the extension:

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Environments** node.
4. Right-click the environment you want to view and select **View Environment**.

Alternatively, use the `> Azure ML: View Environment` command in the command palette.

Experiments

For more information, see [experiments](#).

Create job

The quickest way to create a job is by clicking the **Create Job** icon in the extension's activity bar.

Using the resource nodes in the Azure Machine Learning view:

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Right-click the **Experiments** node in your workspace and select **Create Job**.
4. Choose your job type.
5. A specification file appears. Configure the specification file.
6. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Job` command in the command palette.

View job

To view your job in Azure Machine Learning studio:

1. Expand the subscription node that contains your workspace.
2. Expand the **Experiments** node inside your workspace.
3. Right-click the experiment you want to view and select **View Experiment in Studio**.
4. A prompt appears asking you to open the experiment URL in Azure Machine Learning studio. Select **Open**.

Alternatively, use the `> Azure ML: View Experiment in Studio` command respectively in the command palette.

Track job progress

As you're running your job, you may want to see its progress. To track the progress of a job in Azure Machine

Learning studio from the extension:

1. Expand the subscription node that contains your workspace.
2. Expand the **Experiments** node inside your workspace.
3. Expand the job node you want to track progress for.
4. Right-click the job and select **View Job in Studio**.
5. A prompt appears asking you to open the job URL in Azure Machine Learning studio. Select **Open**.

Download job logs & outputs

Once a job is complete, you may want to download the logs and assets such as the model generated as part of a job.

1. Expand the subscription node that contains your workspace.
2. Expand the **Experiments** node inside your workspace.
3. Expand the job node you want to download logs and outputs for.
4. Right-click the job:
 - To download the outputs, select **Download outputs**.
 - To download the logs, select **Download logs**.

Alternatively, use the `> Azure ML: Download Outputs` and `> Azure ML: Download Logs` commands respectively in the command palette.

Compute instances

For more information, see [compute instances](#).

Create compute instance

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Compute** node.
4. Right-click the **Compute instances** node in your workspace and select **Create Compute**.
5. A specification file appears. Configure the specification file.
6. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Compute` command in the command palette.

Connect to compute instance

To use a compute instance as a development environment or remote Jupyter server, see [Connect to a compute instance](#).

Stop or restart compute instance

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Compute instances** node inside your **Compute** node.
4. Right-click the compute instance you want to stop or restart and select **Stop Compute instance** or **Restart Compute instance** respectively.

Alternatively, use the `> Azure ML: Stop Compute instance` and `> Azure ML: Restart Compute instance` commands respectively in the command palette.

View compute instance configuration

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.

3. Expand the **Compute instances** node inside your **Compute** node.
4. Right-click the compute instance you want to inspect and select **View Compute instance Properties**.

Alternatively, use the `Azure ML: View Compute instance Properties` command in the command palette.

Delete compute instance

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Compute instances** node inside your **Compute** node.
4. Right-click the compute instance you want to delete and select **Delete Compute instance**.

Alternatively, use the `Azure ML: Delete Compute instance` command in the command palette.

Compute clusters

For more information, see [training compute targets](#).

Create compute cluster

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Compute** node.
4. Right-click the **Compute clusters** node in your workspace and select **Create Compute**.
5. A specification file appears. Configure the specification file.
6. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Compute` command in the command palette.

View compute configuration

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Compute clusters** node inside your **Compute** node.
4. Right-click the compute you want to view and select **View Compute Properties**.

Alternatively, use the `> Azure ML: View Compute Properties` command in the command palette.

Delete compute cluster

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Compute clusters** node inside your **Compute** node.
4. Right-click the compute you want to delete and select **Remove Compute**.

Alternatively, use the `> Azure ML: Remove Compute` command in the command palette.

Inference Clusters

For more information, see [compute targets for inference](#).

Manage inference clusters

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Inference clusters** node inside your **Compute** node.
4. Right-click the compute you want to:
 - **View Compute Properties**. Displays read-only configuration data about your attached compute.

- **Detach compute.** Detaches the compute from your workspace.

Alternatively, use the `> Azure ML: View Compute Properties` and `> Azure ML: Detach Compute` commands respectively in the command palette.

Delete inference clusters

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Attached computes** node inside your **Compute** node.
4. Right-click the compute you want to delete and select **Remove Compute**.

Alternatively, use the `> Azure ML: Remove Compute` command in the command palette.

Attached Compute

For more information, see [unmanaged compute](#).

Manage attached compute

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the **Attached computes** node inside your **Compute** node.
4. Right-click the compute you want to:
 - **View Compute Properties.** Displays read-only configuration data about your attached compute.
 - **Detach compute.** Detaches the compute from your workspace.

Alternatively, use the `> Azure ML: View Compute Properties` and `> Azure ML: Detach Compute` commands respectively in the command palette.

Models

For more information, see [models](#)

Create model

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Right-click the **Models** node in your workspace and select **Create Model**.
4. A specification file appears. Configure the specification file.
5. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Model` command in the command palette.

View model properties

1. Expand the subscription node that contains your workspace.
2. Expand the **Models** node inside your workspace.
3. Right-click the model whose properties you want to see and select **View Model Properties**. A file opens in the editor containing your model properties.

Alternatively, use the `> Azure ML: View Model Properties` command in the command palette.

Download model

1. Expand the subscription node that contains your workspace.
2. Expand the **Models** node inside your workspace.
3. Right-click the model you want to download and select **Download Model File**.

Alternatively, use the `> Azure ML: Download Model File` command in the command palette.

Delete a model

1. Expand the subscription node that contains your workspace.
2. Expand the **Models** node inside your workspace.
3. Right-click the model you want to delete and select **Remove Model**.
4. A prompt appears confirming you want to remove the model. Select **Ok**.

Alternatively, use the `> Azure ML: Remove Model` command in the command palette.

Endpoints

For more information, see [endpoints](#).

Create endpoint

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Right-click the **Models** node in your workspace and select **Create Endpoint**.
4. Choose your endpoint type.
5. A specification file appears. Configure the specification file.
6. Right-click the specification file and select **Azure ML: Execute YAML**.

Alternatively, use the `> Azure ML: Create Endpoint` command in the command palette.

Delete endpoint

1. Expand the subscription node that contains your workspace.
2. Expand the **Endpoints** node inside your workspace.
3. Right-click the deployment you want to remove and select **Remove Service**.
4. A prompt appears confirming you want to remove the service. Select **Ok**.

Alternatively, use the `> Azure ML: Remove Service` command in the command palette.

View service properties

In addition to creating and deleting deployments, you can view and edit settings associated with the deployment.

1. Expand the subscription node that contains your workspace.
2. Expand the **Endpoints** node inside your workspace.
3. Right-click the deployment you want to manage:
 - To view deployment configuration settings, select **View Service Properties**.

Alternatively, use the `> Azure ML: View Service Properties` command in the command palette.

Next steps

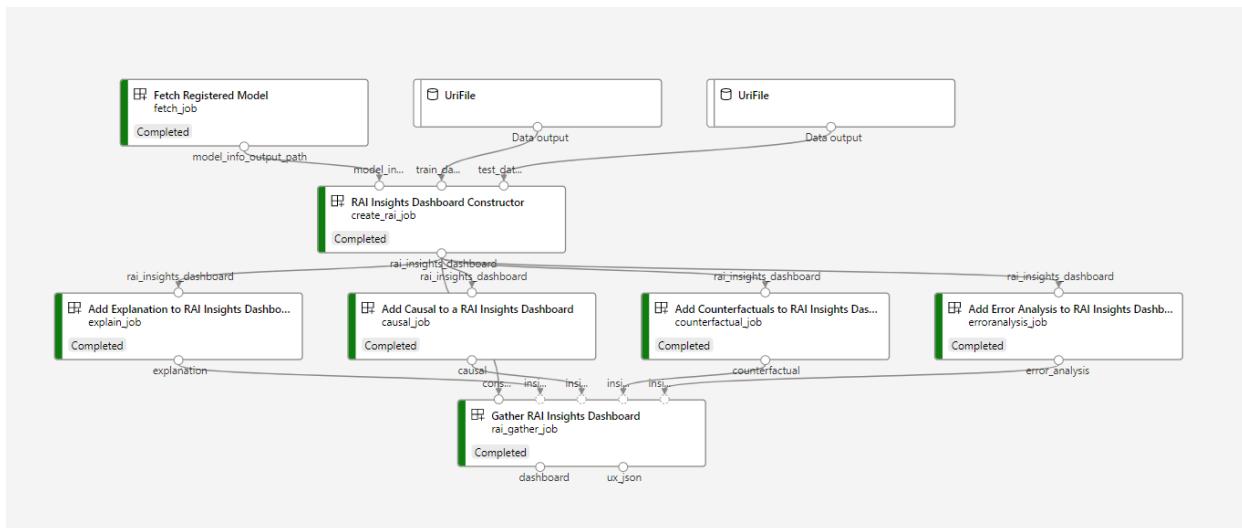
[Train an image classification model](#) with the VS Code extension.

Generate a Responsible AI dashboard with YAML and Python (preview)

9/22/2022 • 15 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

You can generate a Responsible AI dashboard via a pipeline job by using Responsible AI components. There are six core components for creating Responsible AI dashboards, along with a couple of helper components. Here's a sample experiment graph:



Get started

To use the Responsible AI components, you must first register them in your Azure Machine Learning workspace. This section documents the required steps.

Prerequisites

You'll need:

- An Azure Machine Learning workspace
- A Git installation
- A MiniConda installation
- An Azure CLI installation

Installation steps

1. Clone the repository:

```
git clone https://github.com/Azure/RAI-vNext-Preview.git  
cd RAI-vNext-Preview
```

2. Sign in to Azure:

```
Az login
```

3. From the repository root, run the following PowerShell setup script:

```
Quick-Setup.ps1
```

Running the script:

- Creates a new conda environment with a name you specify.
- Installs all the required Python packages.
- Registers all the Responsible AI components in your Azure Machine Learning workspace.
- Registers some sample datasets in your workspace.
- Sets the defaults for the Azure CLI to point to your workspace.

Running the script prompts for the desired conda environment name and Azure Machine Learning workspace details.

Alternatively, you can use the following Bash script:

```
./quick-setup.bash <CONDA-ENV-NAME> <SUBSCRIPTION-ID> <RESOURCE-GROUP-NAME> <WORKSPACE-NAME>
```

This script echoes the supplied parameters, and it pauses briefly before continuing.

Responsible AI components

The core components for constructing the Responsible AI dashboard in Azure Machine Learning are:

- RAI Insights dashboard constructor
- The tool components:
 - Add Explanation to RAI Insights dashboard
 - Add Causal to RAI Insights dashboard
 - Add Counterfactuals to RAI Insights dashboard
 - Add Error Analysis to RAI Insights dashboard
 - Gather RAI Insights dashboard

The RAI Insights dashboard constructor and Gather RAI Insights dashboard components are always required, plus at least one of the tool components. However, it isn't necessary to use all the tools in every Responsible AI dashboard.

In the following sections are specifications of the Responsible AI components and examples of code snippets in YAML and Python. To view the full code, see [sample YAML and Python notebook](#).

Limitations

The current set of components have a number of limitations on their use:

- All models must be registered in Azure Machine Learning in MLflow format with a sklearn (scikit-learn) flavor.
- The models must be loadable in the component environment.
- The models must be pickleable.
- The models must be supplied to the Responsible AI components by using the *fetch registered model* component, which we provide.
- The dataset inputs must be in pandas.DataFrame.to_parquet format.
- A model must be supplied even if only a causal analysis of the data is performed. You can use the DummyClassifier and DummyRegressor estimators from scikit-learn for this purpose.

RAI Insights dashboard constructor

This component has three input ports:

- The machine learning model
- The training dataset
- The test dataset

To generate model-debugging insights with components such as error analysis and Model explanations, use the training and test dataset that you used when you trained your model. For components like causal analysis, which doesn't require a model, you use the training dataset to train the causal model to generate the causal insights. You use the test dataset to populate your Responsible AI dashboard visualizations.

The easiest way to supply the model is to use the fetch registered model component, which we discuss later in this article.

NOTE

Currently, only models in MLflow format and with a sklearn flavor are supported.

The two datasets should be file datasets (of type `uri_file`) in Parquet format. Tabular datasets aren't supported, but we provide a `TabularDataset` to Parquet file component to help with conversions. The training and test datasets provided don't have to be the same datasets that are used in training the model, but they can be the same. By default, for performance reasons, the test dataset is restricted to 5,000 rows of the visualization UI.

The constructor component also accepts the following parameters:

PARAMETER NAME	DESCRIPTION	TYPE
<code>title</code>	Brief description of the dashboard.	String
<code>task_type</code>	Specifies whether the model is for classification or regression.	String, <code>classification</code> or <code>regression</code>
<code>target_column_name</code>	The name of the column in the input datasets, which the model is trying to predict.	String
<code>maximum_rows_for_test_dataset</code>	The maximum number of rows allowed in the test dataset, for performance reasons.	Integer, defaults to 5,000
<code>categorical_column_names</code>	The columns in the datasets, which represent categorical data.	Optional list of strings ¹
<code>classes</code>	The full list of class labels in the training dataset.	Optional list of strings ¹

¹ The lists should be supplied as a single JSON-encoded string for `categorical_column_names` and `classes` inputs.

The constructor component has a single output named `rai_insights_dashboard`. This is an empty dashboard, which the individual tool components operate on. All the results are assembled by the Gather RAI Insights dashboard component at the end.

- [YAML](#)
- [Python SDK](#)

```

create_rai_job:

type: command
component: azureml:rai_insights_constructor:1
inputs:
    title: From YAML snippet
    task_type: regression
    model_info_path: ${{parent.jobs.fetch_model_job.outputs.model_info_output_path}}
    train_dataset: ${{parent.inputs.my_training_data}}
    test_dataset: ${{parent.inputs.my_test_data}}
    target_column_name: ${{parent.inputs.target_column_name}}
    categorical_column_names: '["location", "style", "job title", "OS", "Employer", "IDE", "Programming language"]'

```

Export pre-built cohorts for scorecard generation

You can export pre-built cohorts for use in scorecard generation. For example, here are building cohorts in a Jupyter Notebook: [responsibleaidashboard-diabetes-decision-making.ipynb](#). After you've defined a cohort, you can export it to a JSON file, as shown here:

```

# cohort1, cohort2 are cohorts defined in sample notebook of type raiwidgets.cohort.Cohort
import json
json.dumps([cohort1.to_json(), cohort2.to_json])

```

A sample generated JSON string is shown here:

```

[
  {
    "name": "High Yoe",
    "cohort_filter_list": [
      {
        "method": "greater",
        "arg": [
          5
        ],
        "column": "YOE"
      }
    ]
  },
  {
    "name": "Low Yoe",
    "cohort_filter_list": [
      {
        "method": "less",
        "arg": [
          6.5
        ],
        "column": "YOE"
      }
    ]
  }
]

```

Add Causal to RAI Insights dashboard

This component performs a causal analysis on the supplied datasets. It has a single input port, which accepts the output of the RAI Insights dashboard constructor. It also accepts the following parameters:

PARAMETER NAME	DESCRIPTION	TYPE
<code>treatment_features</code>	A list of feature names in the datasets, which are potentially "treatable" to obtain different outcomes.	List of strings ² .
<code>heterogeneity_features</code>	A list of feature names in the datasets, which might affect how the "treatable" features behave. By default, all features will be considered.	Optional list of strings ² .
<code>nuisance_model</code>	The model used to estimate the outcome of changing the treatment features.	Optional string. Must be <code>linear</code> or <code>AutoML</code> , defaulting to <code>linear</code> .
<code>heterogeneity_model</code>	The model used to estimate the effect of the heterogeneity features on the outcome.	Optional string. Must be <code>linear</code> or <code>forest</code> , defaulting to <code>linear</code> .
<code>alpha</code>	Confidence level of confidence intervals.	Optional floating point number, defaults to 0.05.
<code>upper_bound_on_cat_expansion</code>	The maximum expansion of categorical features.	Optional integer, defaults to 50.
<code>treatment_cost</code>	<p>The cost of the treatments. If 0, all treatments will have zero cost. If a list is passed, each element is applied to one of the <code>treatment_features</code>.</p> <p>Each element can be a scalar value to indicate a constant cost of applying that treatment or an array indicating the cost for each sample. If the treatment is a discrete treatment, the array for that feature should be two dimensional, with the first dimension representing samples and the second representing the difference in cost between the non-default values and the default value.</p>	Optional integer or list ² .
<code>min_tree_leaf_samples</code>	The minimum number of samples per leaf in the policy tree.	Optional integer, defaults to 2.
<code>max_tree_depth</code>	The maximum depth of the policy tree.	Optional integer, defaults to 2.
<code>skip_cat_limit_checks</code>	By default, categorical features need to have several instances of each category in order for a model to be fit robustly. Setting this to <code>True</code> will skip these checks.	Optional Boolean, defaults to <code>False</code> .

PARAMETER NAME	DESCRIPTION	TYPE
<code>categories</code>	<p>The categories to use for the categorical columns. If <code>auto</code>, the categories will be inferred for all categorical columns. Otherwise, this argument should have as many entries as there are categorical columns.</p> <p>Each entry should be either <code>auto</code> to infer the values for that column or the list of values for the column. If explicit values are provided, the first value is treated as the "control" value for that column against which other values are compared.</p>	Optional, <code>auto</code> or list ² .
<code>n_jobs</code>	The degree of parallelism to use.	Optional integer, defaults to 1.
<code>verbose</code>	Expresses whether to provide detailed output during the computation.	Optional integer, defaults to 1.
<code>random_state</code>	Seed for the pseudorandom number generator (PRNG).	Optional integer.

² For the `list` parameters: Several of the parameters accept lists of other types (strings, numbers, even other lists). To pass these into the component, they must first be JSON-encoded into a single string.

This component has a single output port, which can be connected to one of the `insight_[n]` input ports of the Gather RAI Insights dashboard component.

- [YAML](#)
- [Python SDK](#)

```
causal_01:
  type: command
  component: azureml:rai_insights_causal:1
  inputs:
    rai_insights_dashboard: ${{parent.jobs.create_rai_job.outputs.rai_insights_dashboard}}
    treatment_features: `["Number of GitHub repos contributed to", "YOE"]'
```

Add Counterfactuals to RAI Insights dashboard

This component generates counterfactual points for the supplied test dataset. It has a single input port, which accepts the output of the RAI Insights dashboard constructor. It also accepts the following parameters:

PARAMETER NAME	DESCRIPTION	TYPE
<code>total_CFs</code>	The number of counterfactual points to generate for each row in the test dataset.	Optional integer, defaults to 10.
<code>method</code>	The <code>dice-ml</code> explainer to use.	Optional string. Either <code>random</code> , <code>genetic</code> , or <code>kdtree</code> . Defaults to <code>random</code> .

PARAMETER NAME	DESCRIPTION	TYPE
<code>desired_class</code>	Index identifying the desired counterfactual class. For binary classification, this should be set to <code>opposite</code> .	Optional string or integer. Defaults to 0.
<code>desired_range</code>	For regression problems, identify the desired range of outcomes.	Optional list of two numbers ³ .
<code>permitted_range</code>	Dictionary with feature names as keys and the permitted range in a list as values. Defaults to the range inferred from training data.	Optional string or list ³ .
<code>features_to_vary</code>	Either a string <code>all</code> or a list of feature names to vary.	Optional string or list ³ .
<code>feature_importance</code>	Flag to enable computation of feature importances by using <code>dice-m1</code> .	Optional Boolean. Defaults to <code>True</code> .

³ For the non-scalar parameters: Parameters that are lists or dictionaries should be passed as single JSON-encoded strings.

This component has a single output port, which can be connected to one of the `insight_[n]` input ports of the Gather RAI Insights dashboard component.

- [YAML](#)
- [Python SDK](#)

```
counterfactual_01:
  type: command
  component: azureml:rai_insights_counterfactual:1
  inputs:
    rai_insights_dashboard: ${{parent.jobs.create_rai_job.outputs.rai_insights_dashboard}}
    total_CFs: 10
    desired_range: "[5, 10]"
```

Add Error Analysis to RAI Insights dashboard

This component generates an error analysis for the model. It has a single input port, which accepts the output of the RAI Insights dashboard constructor. It also accepts the following parameters:

PARAMETER NAME	DESCRIPTION	TYPE
<code>max_depth</code>	The maximum depth of the error analysis tree.	Optional integer. Defaults to 3.
<code>num_leaves</code>	The maximum number of leaves in the error tree.	Optional integer. Defaults to 31.
<code>min_child_samples</code>	The minimum number of datapoints required to produce a leaf.	Optional integer. Defaults to 20.
<code>filter_features</code>	A list of one or two features to use for the matrix filter.	Optional list, to be passed as a single JSON-encoded string.

This component has a single output port, which can be connected to one of the `insight_[n]` input ports of the Gather RAI Insights dashboard component.

- [YAML](#)
- [Python SDK](#)

```
error_analysis_01:  
  type: command  
  component: azureml:rai_insights_erroranalysis:1  
  inputs:  
    rai_insights_dashboard: ${{parent.jobs.create_rai_job.outputs.rai_insights_dashboard}}  
    filter_features: `["style", "Employer"]'
```

Add Explanation to RAI Insights dashboard

This component generates an explanation for the model. It has a single input port, which accepts the output of the RAI Insights dashboard constructor. It accepts a single, optional comment string as a parameter.

This component has a single output port, which can be connected to one of the `insight_[n]` input ports of the Gather RAI Insights dashboard component.

- [YAML](#)
- [Python SDK](#)

```
explain_01:  
  type: command  
  component: azureml:rai_insights_explanation:VERSION_REPLACEMENT_STRING  
  inputs:  
    comment: My comment  
    rai_insights_dashboard: ${{parent.jobs.create_rai_job.outputs.rai_insights_dashboard}}
```

Gather RAI Insights dashboard

This component assembles the generated insights into a single Responsible AI dashboard. It has five input ports:

- The `constructor` port that must be connected to the RAI Insights dashboard constructor component.
- Four `insight_[n]` ports that can be connected to the output of the tool components. At least one of these ports must be connected.

There are two output ports:

- The `dashboard` port contains the completed `RAIInsights` object.
 - The `ux_json` port contains the data required to display a minimal dashboard.
- [YAML](#)
 - [Python SDK](#)

```
gather_01:  
  type: command  
  component: azureml:rai_insights_gather:1  
  inputs:  
    constructor: ${{parent.jobs.create_rai_job.outputs.rai_insights_dashboard}}  
    insight_1: ${{parent.jobs.causal_01.outputs.causal}}  
    insight_2: ${{parent.jobs.counterfactual_01.outputs.counterfactual}}  
    insight_3: ${{parent.jobs.error_analysis_01.outputs.error_analysis}}  
    insight_4: ${{parent.jobs.explain_01.outputs.explanation}}
```

Helper components

We provide two helper components to aid in connecting the Responsible AI components to your existing assets.

Fetch registered model

This component produces information about a registered model, which can be consumed by the `model_info_path` input port of the RAI Insights dashboard constructor component. It has a single input parameter: the Azure Machine Learning ID (`<NAME>:<VERSION>`) of the desired model.

- [YAML](#)
- [Python SDK](#)

```
fetch_model_job:  
  type: command  
  component: azureml:fetch_registered_model:1  
  inputs:  
    model_id: my_model_name:12
```

Tabular dataset to parquet file

This component converts the tabular dataset named in its sole input parameter into a Parquet file, which can be consumed by the `train_dataset` and `test_dataset` input ports of the RAI Insights dashboard constructor component. Its single input parameter is the name of the desired dataset.

- [YAML](#)
- [Python SDK](#)

```
convert_train_job:  
  type: command  
  component: azureml:convert_tabular_to_parquet:1  
  inputs:  
    tabular_dataset_name: tabular_dataset_name
```

Input constraints

What model formats and flavors are supported?

The model must be in the MLflow directory with a sklearn flavor available. Additionally, the model needs to be loadable in the environment that's used by the Responsible AI components.

What data formats are supported?

The supplied datasets should be file datasets (of type `uri_file`) in Parquet format. We provide the `TabularDataset to Parquet File` component to help convert the data into the required format.

Next steps

- After you've generated your Responsible AI dashboard, [view how to access and use it in Azure Machine Learning studio](#).
- Summarize and share your Responsible AI insights with the [Responsible AI scorecard as a PDF export](#).
- Learn more about the [concepts and techniques behind the Responsible AI dashboard](#).
- Learn more about how to [collect data responsibly](#).
- View [sample YAML and Python notebooks](#) to generate the Responsible AI dashboard with YAML or Python.
- Learn more about how to use the Responsible AI dashboard and scorecard to debug data and models and

inform better decision-making in this [tech community blog post](#).

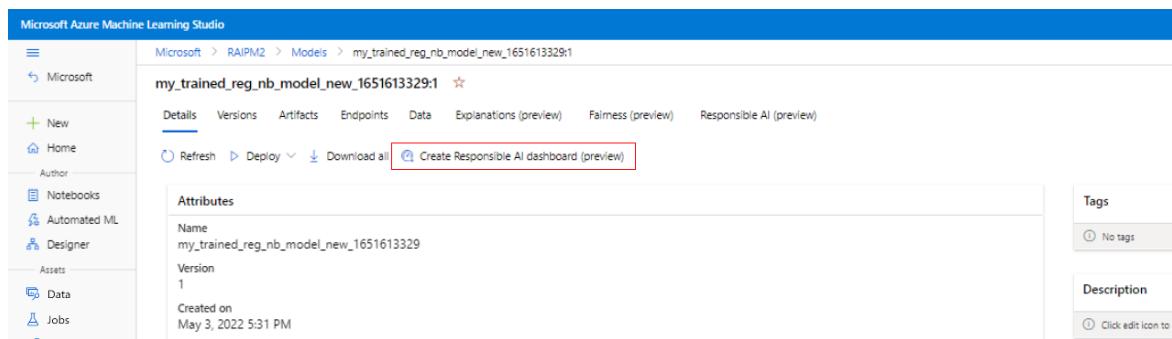
- Learn about how the Responsible AI dashboard and scorecard were used by the UK National Health Service (NHS) in a [real life customer story](#).
- Explore the features of the Responsible AI dashboard through this [interactive AI lab web demo](#).

Generate a Responsible AI dashboard (preview) in the studio UI

9/22/2022 • 6 minutes to read • [Edit Online](#)

In this article, you create a Responsible AI dashboard with a no-code experience in the [Azure Machine Learning studio UI](#). To access the dashboard generation wizard, do the following:

1. [Register your model](#) in Azure Machine Learning so that you can access the no-code experience.
2. On the left pane of Azure Machine Learning studio, select the **Models** tab.
3. Select the registered model that you want to create Responsible AI insights for, and then select the **Details** tab.
4. Select **Create Responsible AI dashboard (preview)**.



The screenshot shows the 'Details' tab of a registered model named 'my_trained_reg_nb_model_new_1651613329:1'. The 'Create Responsible AI dashboard (preview)' button is highlighted with a red box. The 'Attributes' section displays the model's name, version, and creation date. The 'Tags' and 'Description' sections are also visible.

To learn more, see the Responsible AI dashboard [supported model types and limitations](#).

The wizard provides an interface for entering all the necessary parameters to create your Responsible AI dashboard without having to touch code. The experience takes place entirely in the Azure Machine Learning studio UI. The studio presents a guided flow and instructional text to help contextualize the variety of choices about which Responsible AI components you'd like to populate your dashboard with.

The wizard is divided into five sections:

1. Datasets
2. Modeling task
3. Dashboard components
4. Component parameters
5. Experiment configuration

Select your datasets

In the first section, you select the train and test datasets that you used when you trained your model to generate model-debugging insights. For components like causal analysis, which doesn't require a model, you use the train dataset to train the causal model to generate the causal insights.

NOTE

Only tabular dataset formats are supported.

Datasets for training and testing
Select a training dataset to generate your Responsible AI dashboard with components specific to this model (like explanations), then choose a test dataset to populate the dashboard.

Currently only tabular dataset formats are supported.

Select a dataset for training * 1
ProgrammersTrain

Don't see a dataset you want? + New dataset 3

Select a dataset for testing * 2
Currently only up to 5,000 rows are allowed.
ProgrammersTest

Don't see a dataset you want? + New dataset

Back Next Cancel

- Select a dataset for training:** In the dropdown list of registered datasets in the Azure Machine Learning workspace, select the dataset you want to use to generate Responsible AI insights for components, such as model explanations and error analysis.
- Select a dataset for testing:** In the dropdown list, select the dataset you want to use to populate your Responsible AI dashboard visualizations.
- If the train or test dataset you want to use isn't listed, select **New dataset** to upload it.

Select your modeling task

After you've picked your datasets, select your modeling task type, as shown in the following image:

Modeling task type
Choose the type of modeling task (algorithm) you want to use for generating Responsible AI analytics. The task type you choose here will determine which components are available in the next step.

To generate dashboard components successfully, make sure your model is a .pkl file with sklearn format.

Regression
Predict continuous values.

Classification: binary labels
Predict values that belong to one of two discrete groups.

Classification: multi labels
Predict values that belong to one of multiple discrete groups.

Back Next Cancel

NOTE

The wizard supports only models in MLflow format and with a sklearn (scikit-learn) flavor.

Select your dashboard components

The Responsible AI dashboard offers two profiles for recommended sets of tools that you can generate:

- Model debugging:** Understand and debug erroneous data cohorts in your machine learning model by using error analysis, counterfactual what-if examples, and model explainability.
- Real-life interventions:** Understand and debug erroneous data cohorts in your machine learning

model by using causal analysis.

NOTE

Multi-class classification doesn't support the real-life interventions analysis profile.

The screenshot shows the 'Create Responsible AI dashboard' interface. On the left, a sidebar lists navigation options: Datasets, Modeling task (selected), Dashboard components (selected), Component parameters, and Experiment configuration. The main area is titled 'Dashboard components' with the sub-instruction: 'Choose which components to include in your dashboard analytics. Available components are dependent on which modeling task type was selected in the previous step.' A yellow note bar at the top states: 'Note: Multi-class classification currently does not support Real-life intervention analysis profile.' Below this, two components are listed: 'Model debugging' (selected) and 'Real-life interventions'. The 'Model debugging' card includes the sub-instruction: 'Understand and debug erroneous data cohorts in your ML model.' The 'Real-life interventions' card includes the sub-instruction: 'Understand causal relationships between your features and real-world outcomes.' At the bottom of the screen are 'Back', 'Next', and 'Cancel' buttons.

1. Select the profile you want to use.
2. Select **Next**.

Configure parameters for dashboard components

After you've selected a profile, the **Component parameters for model debugging** configuration pane for the corresponding components appears.

The screenshot shows the 'Component parameters for model debugging' configuration pane. The sidebar on the left shows the selected path: Datasets, Modeling task, Dashboard components, Component parameters (selected), and Experiment configuration. The main pane is titled 'Component parameters for model debugging' with the sub-instruction: 'Customize your dashboard analytics by choosing which parameters to generate and configuring their settings.' It contains several sections:

- Target feature:** A dropdown menu set to 'score' with a note: 'Choose the feature in your dataset that your ML model was trained to predict.'
- Categorical features:** A dropdown menu set to 'first_name, last_name, job title, style, IDE, P...' with a note: 'Choose one or more features you want to have categorical values.'
- Error tree and heat map:** Includes a toggle switch for generating visualizations of error distribution (set to 'On') and a 'Features' dropdown set to 'style, location'. To the right is a 'Heat map' visualization showing error rates across two feature types (labeled Feature 1 and Feature 2).
- Counterfactual what-if examples:** Includes a toggle switch for generating an example set of diverse 'what-if' datapoints (set to 'On'), a 'Number of counterfactuals (per datapoint)' input field set to '10', and a 'Range of value predictions' input field with minimum and maximum values set to '7' and '10' respectively. A note specifies: 'Specify a range of value predictions you want to generate counterfactual examples for.'
- Advanced configuration:** Includes fields for 'Maximum depth of error tree' (set to '3'), 'Number of leaves in error tree' (set to '31'), and 'Minimum number of samples in each leaf node' (set to '20'). Notes explain: 'The maximum depth of the surrogate tree visualization trained on errors.', 'The number of leaves on the surrogate tree visualization trained on errors.', and 'The minimum number of data required to create one leaf in the error tree.'
- Explanation:** Includes a toggle switch for generating individual and aggregate feature importances (set to 'On'). A note states: 'Parameters are automatically configured; a default opaque box mimic explainer will be used to generate aggregate and individual feature importances. Learn more about opaque box mimic explainers.'

At the bottom are 'Back', 'Next', and 'Cancel' buttons.

Component parameters for model debugging:

1. **Target feature (required):** Specify the feature that your model was trained to predict.
2. **Categorical features:** Indicate which features are categorical to properly render them as categorical

values in the dashboard UI. This field is pre-loaded for you based on your dataset metadata.

3. **Generate error tree and heat map:** Toggle on and off to generate an error analysis component for your Responsible AI dashboard.
4. **Features for error heat map:** Select up to two features that you want to pre-generate an error heatmap for.
5. **Advanced configuration:** Specify additional parameters, such as **Maximum depth of error tree**, **Number of leaves in error tree**, and **Minimum number of samples in each leaf node**.
6. **Generate counterfactual what-if examples:** Toggle on and off to generate a counterfactual what-if component for your Responsible AI dashboard.
7. **Number of counterfactuals (required):** Specify the number of counterfactual examples that you want generated per data point. A minimum of 10 should be generated to enable a bar chart view of the features that were most perturbed, on average, to achieve the desired prediction.
8. **Range of value predictions (required):** Specify for regression scenarios the range that you want counterfactual examples to have prediction values in. For binary classification scenarios, the range will automatically be set to generate counterfactuals for the opposite class of each data point. For multi-classification scenarios, use the dropdown list to specify which class you want each data point to be predicted as.
9. **Specify which features to perturb:** By default, all features will be perturbed. However, if you want only specific features to be perturbed, select **Specify which features to perturb for generating counterfactual explanations** to display a pane with a list of features to select.

When you select **Specify which features to perturb**, you can specify the range you want to allow perturbations in. For example: for the feature YOE (Years of experience), specify that counterfactuals should have feature values ranging from only 10 to 21 instead of the default values of 5 to 21.

Specify which features to perturb for generating counterfactual explanations
Counterfactual examples will be generated with only specified features and range to perturb.

Search

Features to perturb	Range to perturb
<input checked="" type="checkbox"/> first_name num of categories: 843	<input type="text"/>
<input checked="" type="checkbox"/> last_name num of categories: 868	<input type="text"/>
score (Target feature) min: 0, max: 10	
<input checked="" type="checkbox"/> style num of categories: 2	<input type="text"/>
<input checked="" type="checkbox"/> YOE min: 5, max: 21	5 <input type="text"/> 21
<input checked="" type="checkbox"/> IDE num of categories: 8	<input type="text"/>
<input checked="" type="checkbox"/> Programming language num of categories: 10	<input type="text"/>
<input checked="" type="checkbox"/> location num of categories: 5	<input type="text"/>
<input checked="" type="checkbox"/> Number of github repos contributed <input type="text"/> 11	

Save

10. **Generate explanations:** Toggle on and off to generate a model explanation component for your Responsible AI dashboard. No configuration is necessary, because a default opaque box mimic explainer will be used to generate feature importances.

Alternatively, if you select the **Real-life interventions** profile, you'll see the following screen generate a causal analysis. This will help you understand the causal effects of features you want to "treat" on a certain outcome you want to optimize.

Component parameters for real-life interventions

Customize your Responsible AI dashboard by filling in the below required parameters.

Target feature * score 1 Choose the outcome you want the causal effects to be calculated for.

Treatment features * YOE, Number of github repos contributed ... 2 Choose one or more features you're interested in changing ("treating") to optimize the target outcome.

Categorical features first_name, last_name, job title, style, IDE, P... 3 Choose one or more features you want to have categorical values.

Advanced settings 4

Heterogeneous features IDE, Programming language Additional features to understand causal segmentation in your analysis in addition to your treatment features.

Causal model used Linear Specify what model to use for estimating propensity models or models of how controls predict the outcome). If 'Linear', then LassoCV (for regression) and LogisticRegressionCV (for classification) are used. If 'AutoML', then a k-fold cross-validation and model selection is performed among several modes and the best is chosen. Linear will be faster but less accurate. AutoML may take longer to compute but may yield more accurate results.

Back **Next** **Cancel**

Component parameters for real-life interventions use causal analysis. Do the following:

- Target feature (required):** Choose the outcome you want the causal effects to be calculated for.
- Treatment features (required):** Choose one or more features that you're interested in changing ("treating") to optimize the target outcome.
- Categorical features:** Indicate which features are categorical to properly render them as categorical values in the dashboard UI. This field is pre-loaded for you based on your dataset metadata.
- Advanced settings:** Specify additional parameters for your causal analysis, such as heterogeneous features (that is, additional features to understand causal segmentation in your analysis, in addition to your treatment features) and which causal model you want to be used.

Configure your experiment

Finally, configure your experiment to kick off a job to generate your Responsible AI dashboard.

Training job or experiment configuration

Choose an existing job or experiment, or create a new one to start populating your Responsible AI dashboard with relevant analytics.

Name * Scoring Programmers RAI dashboard 1

Experiment name * 2 Select existing Create new

Existing experiment * RAI_Programmers_Example_Model_Training_1651182949 3

Select compute type Compute cluster 4

Select Azure ML compute cluster * rai-cluster 5

+ New Refresh computes

Description Add a description to your Responsible AI dashboard job 6

Tags 7

Back **Create** **Cancel**

On the Training job or experiment configuration pane, do the following:

- Name:** Give your dashboard a unique name so that you can differentiate it when you're viewing the list of dashboards for a given model.
- Experiment name:** Select an existing experiment to run the job in, or create a new experiment.
- Existing experiment:** In the dropdown list, select an existing experiment.
- Select compute type:** Specify which compute type you want to use to execute your job.
- Select compute:** In the dropdown list, select the compute you want to use. If there are no existing compute resources, select the plus sign (+), create a new compute resource, and then refresh the list.

6. **Description:** Add a longer description of your Responsible AI dashboard.

7. **Tags:** Add any tags to this Responsible AI dashboard.

After you've finished configuring your experiment, select **Create** to start generating your Responsible AI dashboard. You'll be redirected to the experiment page to track the progress of your job.

In the "Next steps" section, you can learn how to view and use your Responsible AI dashboard.

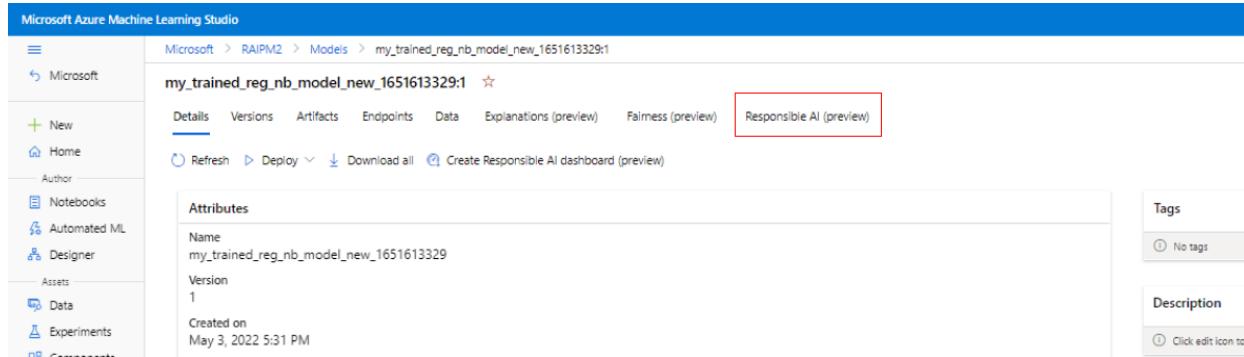
Next steps

- After you've generated your Responsible AI dashboard, [view how to access and use it in Azure Machine Learning studio](#).
- Summarize and share your Responsible AI insights with the [Responsible AI scorecard as a PDF export](#).
- Learn more about the [concepts and techniques behind the Responsible AI dashboard](#).
- Learn more about how to [collect data responsibly](#).
- Learn more about how to use the Responsible AI dashboard and scorecard to debug data and models and inform better decision-making in this [tech community blog post](#).
- Learn about how the Responsible AI dashboard and scorecard were used by the UK National Health Service (NHS) in a [real life customer story](#).
- Explore the features of the Responsible AI dashboard through this [interactive AI Lab web demo](#).

Use the Responsible AI dashboard (preview) in Azure Machine Learning studio

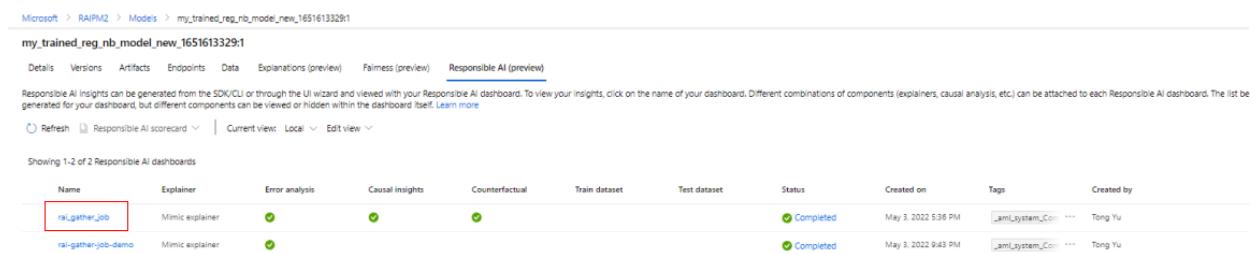
9/22/2022 • 19 minutes to read • [Edit Online](#)

Responsible AI dashboards are linked to your registered models. To view your Responsible AI dashboard, go into your model registry and select the registered model you've generated a Responsible AI dashboard for. Then, select the **Responsible AI (preview)** tab to view a list of generated dashboards.



The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a sidebar with options like New, Home, Author, Notebooks, Automated ML, Designer, Assets, Data, Experiments, and Commandline. The main area shows a breadcrumb path: Microsoft > RAIPM2 > Models > my_trained_reg_nb_model_new_1651613329:1. Below the path is the model name "my_trained_reg_nb_model_new_1651613329:1" with a star icon. A horizontal navigation bar below the path includes Details, Versions, Artifacts, Endpoints, Data, Explanations (preview), Fairness (preview), and Responsible AI (preview). The "Responsible AI (preview)" tab is highlighted with a red box. Below the navigation bar are buttons for Refresh, Deploy, Download all, and Create Responsible AI dashboard (preview). The main content area displays "Attributes" for the model, including Name (my_trained_reg_nb_model_new_1651613329), Version (1), and Created on (May 3, 2022 5:31 PM). To the right, there are sections for Tags (No tags) and Description (Click edit icon to...).

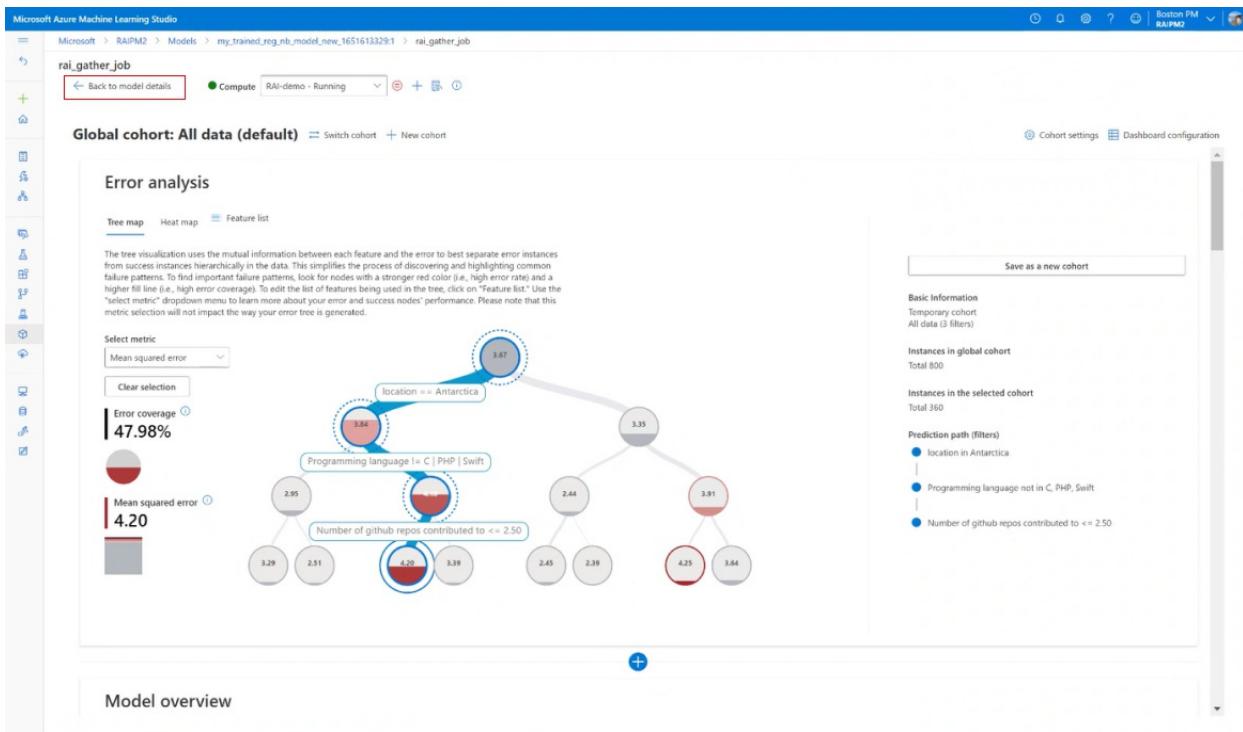
You can configure multiple dashboards and attach them to your registered model. Various combinations of components (interpretability, error analysis, causal analysis, and so on) can be attached to each Responsible AI dashboard. The following image displays a dashboard's customization and the components that were generated within it. In each dashboard, you can view or hide various components within the dashboard UI itself.



The screenshot shows the Responsible AI dashboard configuration page. At the top, there's a navigation bar with Details, Versions, Artifacts, Endpoints, Data, Explanations (preview), Fairness (preview), and Responsible AI (preview). The Responsible AI (preview) tab is selected and highlighted with a red box. Below the navigation bar, there's a note about generating Responsible AI insights from the SDK/CLI or through the UI wizard. A table lists "Showing 1-2 of 2 Responsible AI dashboards". The table has columns: Name, Explainer, Error analysis, Causal insights, Counterfactual, Train dataset, Test dataset, Status, Created on, Tags, and Created by. Two rows are shown:

Name	Explainer	Error analysis	Causal insights	Counterfactual	Train dataset	Test dataset	Status	Created on	Tags	Created by
rai-gather-job	Mimic explainer	✓	✓	✓			Completed	May 3, 2022 5:36 PM	[redacted]	Tong Yu
rai-gather-job-demo	Mimic explainer	✓					Completed	May 3, 2022 9:43 PM	[redacted]	Tong Yu

Select the name of the dashboard to open it into a full view in your browser. To return to your list of dashboards, you can select **Back to models details** at any time.



Full functionality with integrated compute resource

Some features of the Responsible AI dashboard require dynamic, on-the-fly, and real-time computation (for example, what-if analysis). Unless you connect a compute resource to the dashboard, you might find some functionality missing. When you connect to a compute resource, you enable full functionality of your Responsible AI dashboard for the following components:

- **Error analysis**

- Setting your global data cohort to any cohort of interest will update the error tree instead of disabling it.
- Selecting other error or performance metrics is supported.
- Selecting any subset of features for training the error tree map is supported.
- Changing the minimum number of samples required per leaf node and error tree depth is supported.
- Dynamically updating the heat map for up to two features is supported.

- **Feature importance**

- An individual conditional expectation (ICE) plot in the individual feature importance tab is supported.

- **Counterfactual what-if**

- Generating a new what-if counterfactual data point to understand the minimum change required for a desired outcome is supported.

- **Causal analysis**

- Selecting any individual data point, perturbing its treatment features, and seeing the expected causal outcome of causal what-if is supported (only for regression machine learning scenarios).

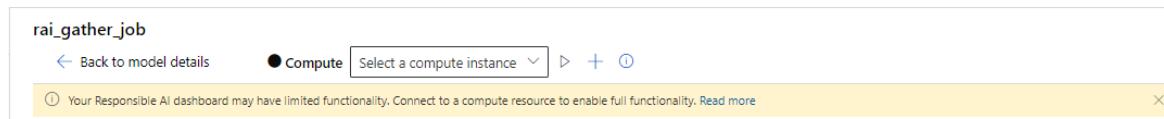
You can also find this information on the Responsible AI dashboard page by selecting the **Information** icon, as shown in the following image:



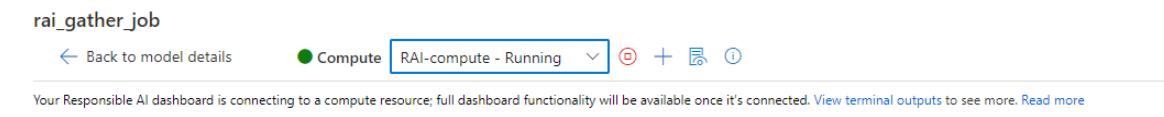
Enable full functionality of the Responsible AI dashboard

1. Select a running compute instance in the Compute dropdown list at the top of the dashboard. If you

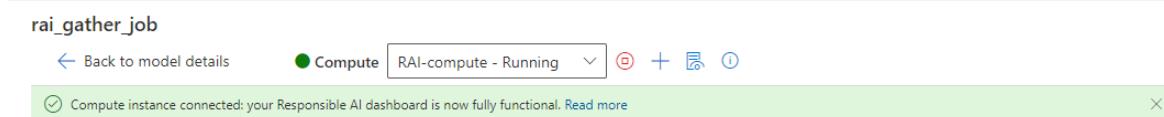
don't have a running compute, create a new compute instance by selecting the plus sign (+) next to the dropdown. Or you can select the **Start compute** button to start a stopped compute instance. Creating or starting a compute instance might take few minutes.



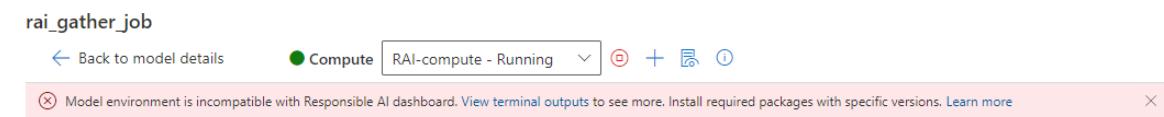
- When a compute is in a *Running* state, your Responsible AI dashboard starts to connect to the compute instance. To achieve this, a terminal process is created on the selected compute instance, and a Responsible AI endpoint is started on the terminal. Select **View terminal outputs** to view the current terminal process.



- When your Responsible AI dashboard is connected to the compute instance, you'll see a green message bar, and the dashboard is now fully functional.



- If the process takes a while and your Responsible AI dashboard is still not connected to the compute instance, or a red error message bar is displayed, it means there are issues with starting your Responsible AI endpoint. Select **View terminal outputs** and scroll down to the bottom to view the error message.



If you're having difficulty figuring out how to resolve the "failed to connect to compute instance" issue, select the **Smile** icon at the upper right. Submit feedback to us about any error or issue you encounter. You can include a screenshot and your email address in the feedback form.

UI overview of the Responsible AI dashboard

The Responsible AI dashboard includes a robust, rich set of visualizations and functionality to help you analyze your machine learning model or make data-driven business decisions:

- [Global controls](#)
- [Error analysis](#)
- [Model overview](#)
- [Data explorer](#)
- [Feature importance \(model explanations\)](#)
- [Counterfactual what-if](#)
- [Causal analysis](#)

Global controls

At the top of the dashboard, you can create cohorts (subgroups of data points that share specified characteristics) to focus your analysis of each component. The name of the cohort that's currently applied to the dashboard is always shown at the top left of your dashboard. The default view in your dashboard is your whole

dataset, titled **All data (default)**.

1. **Cohort settings**: Allows you to view and modify the details of each cohort in a side panel.
2. **Dashboard configuration**: Allows you to view and modify the layout of the overall dashboard in a side panel.
3. **Switch cohort**: Allows you to select a different cohort and view its statistics in a pop-up window.
4. **New cohort**: Allows you to create and add a new cohort to your dashboard.

Select **Cohort settings** to open a panel with a list of your cohorts, where you can create, edit, duplicate, or delete them.

Name	Details
All data	800 datapoints 0 filters
Programmers accepted	174 datapoints 1 filters
Programmers denied	626 datapoints 1 filters

Select **New cohort** at the top of the dashboard or in the Cohort settings to open a new panel with options to filter on the following:

1. **Index**: Filters by the position of the data point in the full dataset.
2. **Dataset**: Filters by the value of a particular feature in the dataset.
3. **Predicted Y**: Filters by the prediction made by the model.
4. **True Y**: Filters by the actual value of the target feature.
5. **Error (regression)**: Filters by error (or **Classification Outcome (classification)**): Filters by type and accuracy of classification).
6. **Categorical Values**: Filter by a list of values that should be included.
7. **Numerical Values**: Filter by a Boolean operation over the values (for example, select data points where age < 64).

Dataset cohort name: Rows < 600
Select filter: Index 1

Dataset cohort name: Programmers using tabs
Select filter: Dataset 2

Dataset cohort name: Programmers score < 7
Select filter: Predicted Y 3

Dataset cohort name: Programmers score < 7 (true)
Select filter: True Y 4

Programmers error rate
Select filter: Error 5

You can name your new dataset cohort, select **Add filter** to add each filter you want to use, and then do either of the following:

- Select **Save** to save the new cohort to your cohort list.
- Select **Save and switch** to save and immediately switch the global cohort of the dashboard to the newly created cohort.

Cohort settings

Global cohorts are subsets of data created by in the error analysis component. View, create,

New cohort

Name	Details
All data	800 datapoints 0 filters
Programmers accepted	174 datapoints 1 filters
Programmers denied	626 datapoints 1 filters

Cohort 4

Select filter

Index

Dataset

Predicted Y

True Y

Error

Select filter

Programming langua...

of unique values: 10

Included values:

C, C#, C++, GO, Java, Javascript, PHP, P...

Add filter

Filters

style includes tabs X

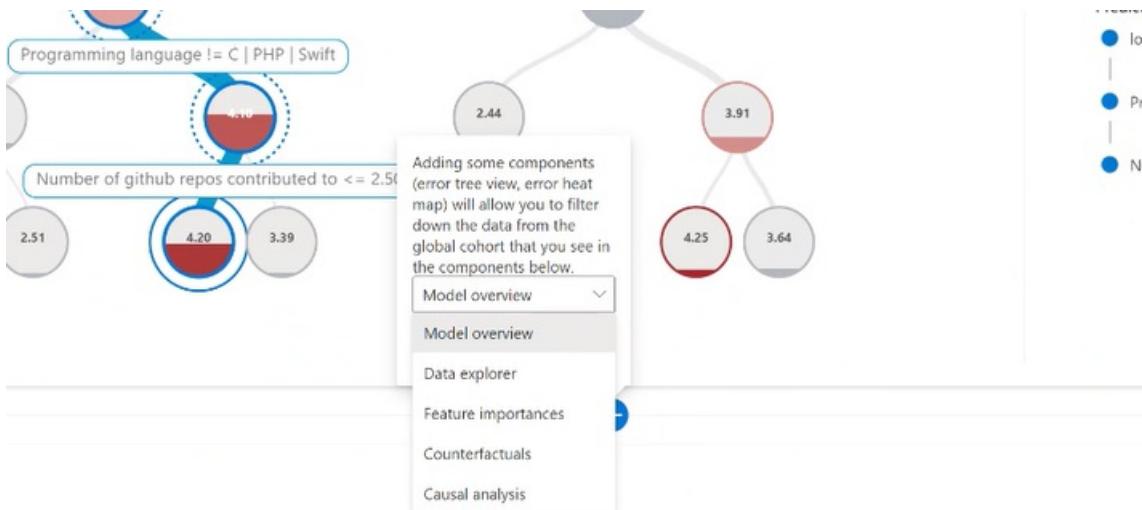
Clear all filters

Save Save and switch Cancel

Select **Dashboard configuration** to open a panel with a list of the components you've configured on your dashboard. You can hide components on your dashboard by selecting the **Trash** icon, as shown in the following image:

The screenshot shows the 'Dashboard configuration' page. At the top, there's a message: 'This list shows the layout of the dashboard. You can filter down data using the error analysis component, to be viewed in the components below.' Below this is a section titled 'Dashboard components' with four items: 'Error analysis' (with a gear icon), 'Model overview' (with a trash bin icon), 'Data explorer' (with a trash bin icon), and 'Feature importances' (with a trash bin icon). A button 'Save as a new cohort' is located at the bottom left.

You can add components back to your dashboard via the blue circular plus sign (+) icon in the divider between each component, as shown in the following image:



ng the distribution of your prediction values and the values of your
gate your model by looking at a comparative analysis of its
of your dataset. Select filters along y-value and x-value to cut across

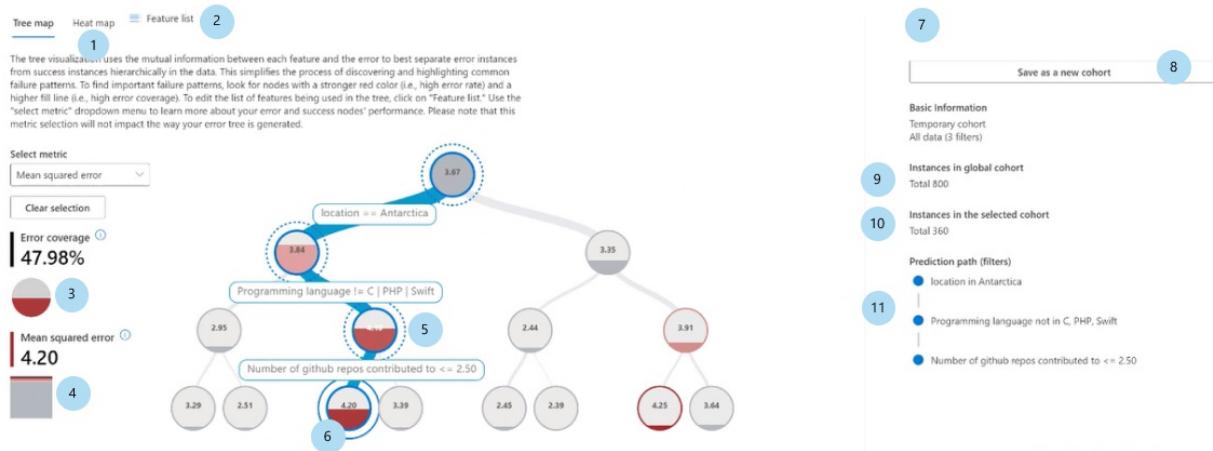
Error analysis

The next sections cover how to interpret and use error tree maps and heat maps.

Error tree map

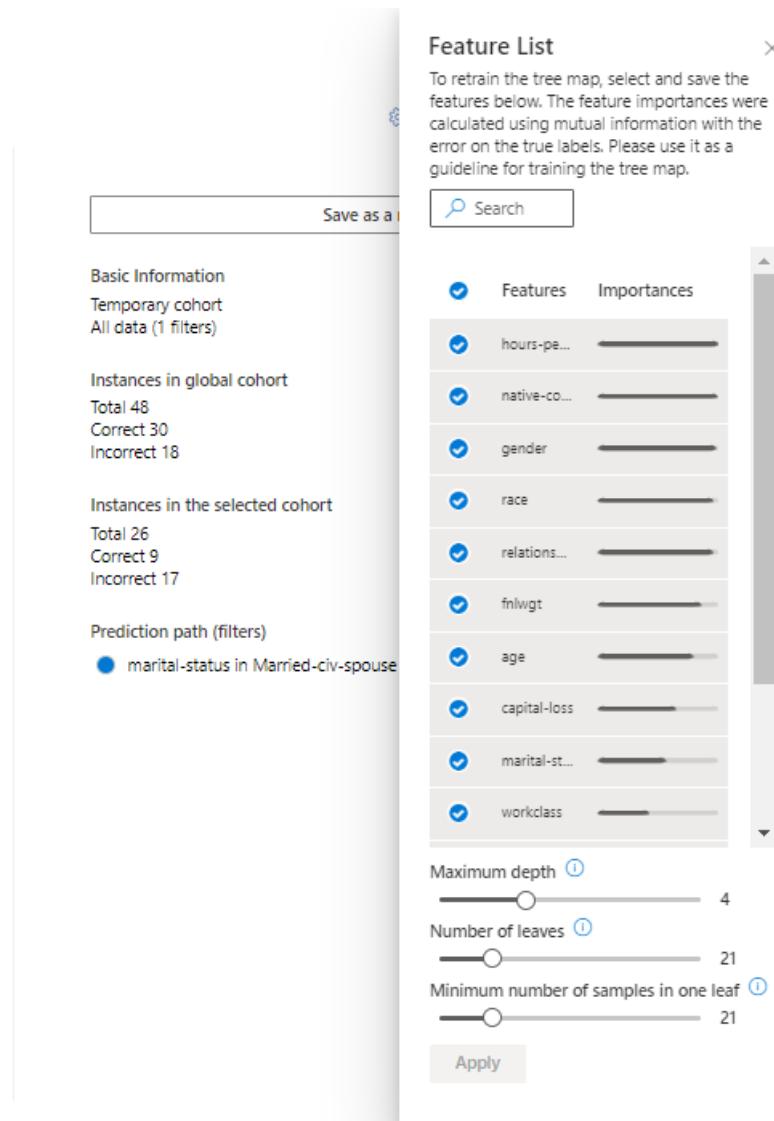
The first pane of the error analysis component is a tree map, which illustrates how model failure is distributed across various cohorts with a tree visualization. Select any node to see the prediction path on your features where an error was found.

Error analysis



1. **Heat map view:** Switches to heat map visualization of error distribution.
2. **Feature list:** Allows you to modify the features used in the heat map using a side panel.
3. **Error coverage:** Displays the percentage of all error in the dataset concentrated in the selected node.
4. **Error (regression) or Error rate (classification):** Displays the error or percentage of failures of all the data points in the selected node.
5. **Node:** Represents a cohort of the dataset, potentially with filters applied, and the number of errors out of the total number of data points in the cohort.
6. **Fill line:** Visualizes the distribution of data points into child cohorts based on filters, with the number of data points represented through line thickness.
7. **Selection information:** Contains information about the selected node in a side panel.
8. **Save as a new cohort:** Creates a new cohort with the specified filters.
9. **Instances in the base cohort:** Displays the total number of points in the entire dataset and the number of correctly and incorrectly predicted points.
10. **Instances in the selected cohort:** Displays the total number of points in the selected node and the number of correctly and incorrectly predicted points.
11. **Prediction path (filters):** Lists the filters placed over the full dataset to create this smaller cohort.

Select the **Feature list** button to open a side panel, from which you can retrain the error tree on specific features.



1. **Search features:** Allows you to find specific features in the dataset.
2. **Features:** Lists the name of the feature in the dataset.
3. **Importances:** A guideline for how related the feature might be to the error. Calculated via mutual

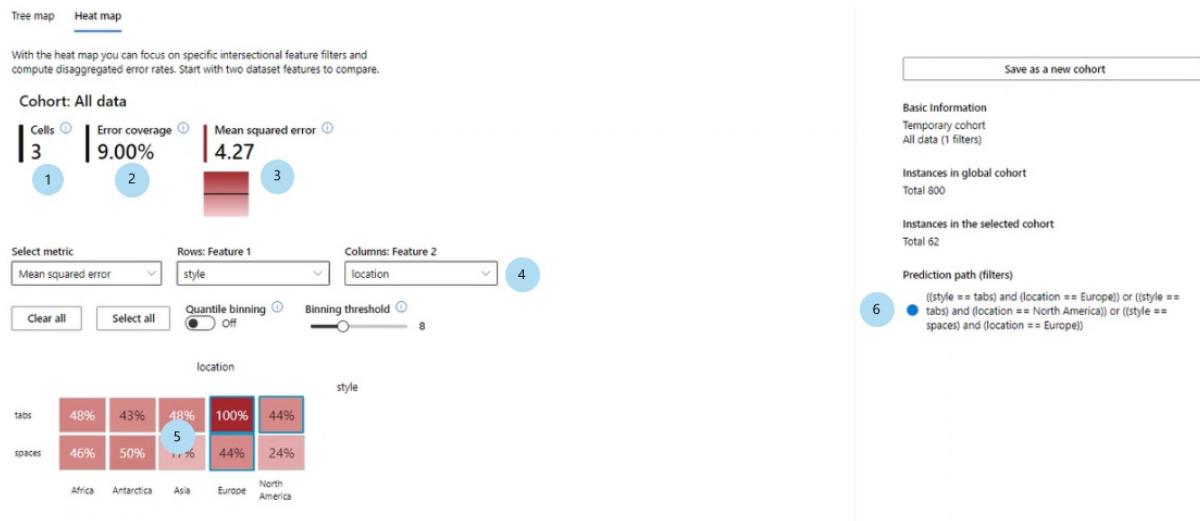
information score between the feature and the error on the labels. You can use this score to help you decide which features to choose in the error analysis.

4. **Check mark:** Allows you to add or remove the feature from the tree map.
5. **Maximum depth:** The maximum depth of the surrogate tree trained on errors.
6. **Number of leaves:** The number of leaves of the surrogate tree trained on errors.
7. **Minimum number of samples in one leaf:** The minimum amount of data required to create one leaf.

Error heat map

Select the **Heat map** tab to switch to a different view of the error in the dataset. You can select one or many heat map cells and create new cohorts. You can choose up to two features to create a heat map.

Error analysis



1. **Cells:** Displays the number of cells selected.
2. **Error coverage:** Displays the percentage of all errors concentrated in the selected cell(s).
3. **Error rate:** Displays the percentage of failures of all data points in the selected cell(s).
4. **Axis features:** Selects the intersection of features to display in the heat map.
5. **Cells:** Represents a cohort of the dataset, with filters applied, and the percentage of errors out of the total number of data points in the cohort. A blue outline indicates selected cells, and the darkness of red represents the concentration of failures.
6. **Prediction path (filters):** Lists the filters placed over the full dataset for each selected cohort.

Model overview

The model overview component provides a comprehensive set of performance and fairness metrics for evaluating your model, along with key performance disparity metrics along specified features and dataset cohorts.

Dataset cohorts

On the **Dataset cohorts** pane, you can investigate your model by comparing the model performance of various user-specified dataset cohorts (accessible via the **Cohort settings** icon at the top right of the dashboard).

NOTE

You can create new dataset cohorts from the UI experience or pass your pre-built cohorts to the dashboard via the SDK experience.

Model overview

Evaluate the performance of your model by exploring the distribution of your prediction values and the values of your model performance metrics. Use the "Dataset cohorts" tab to investigate your model by looking at a comparative analysis of its performance across different pre-built or newly-created dataset cohorts. Use the "Feature cohorts" to investigate your model by looking at a comparative analysis of its performance across sensitive/non-sensitive feature subcohorts. (e.g., performance across different genders, income levels).

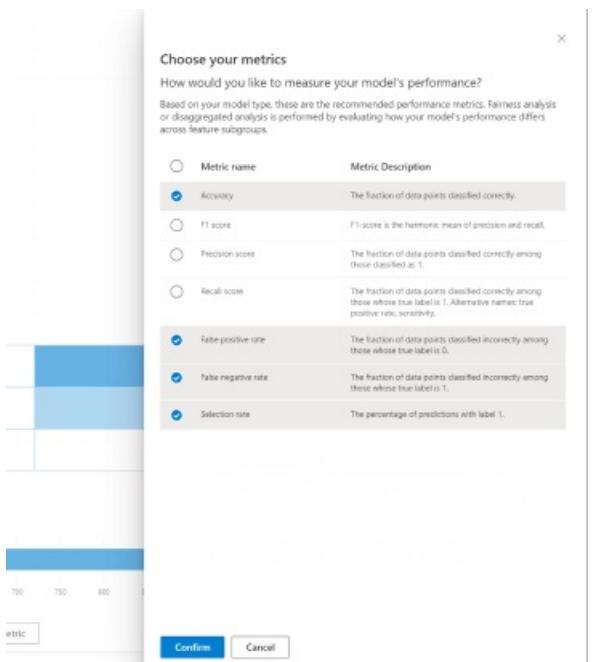


- 1. Help me choose metrics:** Select this icon to open a panel with more information about what model performance metrics are available to be shown in the table. Easily adjust which metrics to view by using the multi-select dropdown list to select and deselect performance metrics.
- 2. Show heat map:** Toggle on and off to show or hide heat map visualization in the table. The gradient of the heat map corresponds to the range normalized between the lowest value and the highest value in each column.
- 3. Table of metrics for each dataset cohort:** View columns of dataset cohorts, the sample size of each cohort, and the selected model performance metrics for each cohort.
- 4. Bar chart visualizing individual metric:** View mean absolute error across the cohorts for easy comparison.
- 5. Choose metric (x-axis):** Select this button to choose which metrics to view in the bar chart.
- 6. Choose cohorts (y-axis):** Select this button to choose which cohorts to view in the bar chart. **Feature cohort** selection might be disabled unless you first specify the features you want on the **Feature cohort** tab of the component.

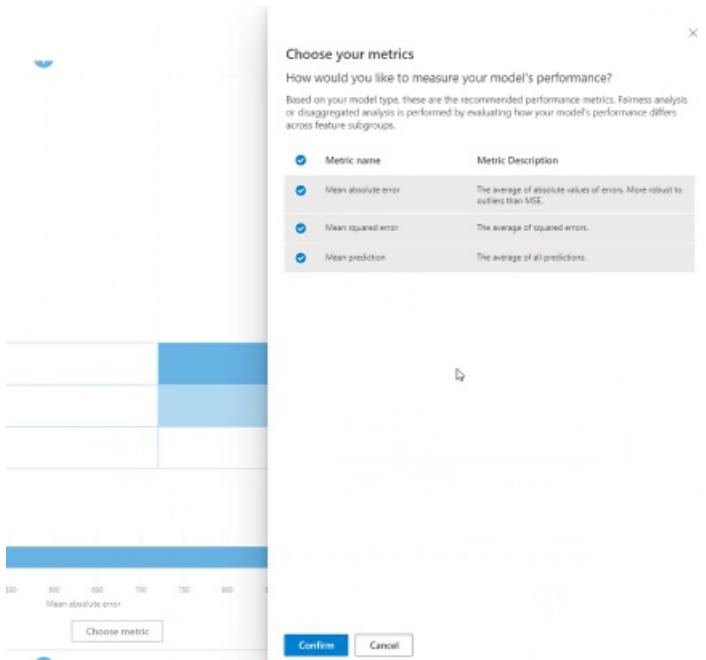
Select **Help me choose metrics** to open a panel with a list of model performance metrics and their definitions, which can help you select the right metrics to view.

MACHINE LEARNING SCENARIO	METRICS
Regression	Mean absolute error, Mean squared error, R-squared, Mean prediction.
Classification	Accuracy, Precision, Recall, F1 score, False positive rate, False negative rate, Selection rate.

Classification scenarios support accuracy scores, precision scores, recall, false positive rate, false negative rate, and selection rate (the percentage of predictions with label 1):



Regression scenarios support mean absolute error, mean squared error, and mean prediction:



Feature cohorts

On the **Feature cohorts** pane, you can investigate your model by comparing model performance across user-specified sensitive and non-sensitive features (for example, performance across various gender, race, and income level cohorts).

Model overview

Evaluate the performance of your model by exploring the distribution of your prediction values and the values of your model performance metrics. Use the "Dataset cohorts" tab to investigate your model by looking at a comparative analysis of its performance across different pre-built or newly-created dataset cohorts. Use the "Feature cohorts" to investigate your model by looking at a comparative analysis of its performance across sensitive/non-sensitive feature subcohorts. (e.g., performance across different genders, income levels).

The screenshot shows the "Feature cohorts" tab selected. At the top, there are dropdown menus for "Metric(s)" (containing "Mean absolute error, Mean squared error, Mean prediction") and "Feature(s)" (containing "style, YOE"). A "Show heatmap" toggle switch is turned on. Below these are two numbered callouts: 1 points to the "Help me choose metrics" icon, and 2 points to the "Help me choose features" icon. A third numbered callout, 3, points to the "Show heatmap" switch.

The main content area displays a table of metrics for four feature cohorts:

Cohorts	Sample size	Mean absolute error	Mean squared error	Mean prediction
(style in spaces) and (YOE < 11.00)	1	1.875	5.516	3.675
(style in spaces) and (YOE >= 11.00) and (YOE < 17.00)	223	307.581	3.274	3.630
(style in spaces) and (YOE >= 17.00)	155	235.938	5.605	3.631
(style in tabs) and (YOE < 11.00)	421	691.438	8.093	3.451

Below this is a section titled "Fairness metrics" with a single row:

Difference	Sample size	Mean absolute error	Mean squared error	Mean prediction
Ratio	420	659.563	0.797	3.635

Callouts 4 and 5 point to the first and second rows of the fairness metrics table respectively. Callout 6 points to the horizontal bar chart below:

The bar chart visualizes the Mean absolute error for each cohort. The x-axis ranges from 0 to 675. The bars are colored blue and labeled with their values: 1.875, 307.581, 235.938, and 691.438. Callout 7 points to the "Choose cohorts" button on the left, and callout 8 points to the "Choose metric" button at the bottom right of the chart.

- 1. Help me choose metrics:** Select this icon to open a panel with more information about what metrics are available to be shown in the table. Easily adjust which metrics to view by using the multi-select dropdown to select and deselect performance metrics.
- 2. Help me choose features:** Select this icon to open a panel with more information about what features are available to be shown in the table, with descriptors of each feature and their binning capability (see below). Easily adjust which features to view by using the multi-select dropdown to select and deselect them.

This screenshot shows the "Choose your features" dialog box. It lists three selected features: "style", "YOE", and "IDE".

- style:** Described as having 2 unique values. It has two sub-cohorts: "style in spaces" and "style in tabs".
- YOE:** Described as being continuous and split into 3 bins. It has three sub-cohorts: "YOE < 11.00", "(YOE >= 11.00) and (YOE < 17.00)", and "YOE >= 17.00". A dropdown menu allows changing the number of bins from 3 to 5.
- IDE:** Described as having 8 unique values. It has eight sub-cohorts: "IDE in Eclipse", "IDE in Emacs", "IDE in IntelliJ", "IDE in VSCode", "IDE in Vim", "IDE in Visual Studio", "IDE in XCode", and "IDE in pyCharm".

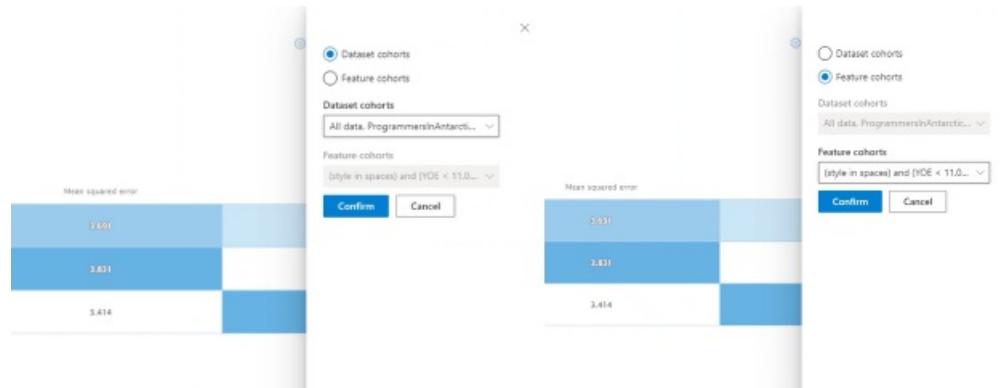
At the bottom, there are "Confirm" and "Cancel" buttons.

- 3. Show heat map:** Toggle on and off to see a heat map visualization. The gradient of the heat map corresponds to the range that's normalized between the lowest value and the highest value in each column.
- 4. Table of metrics for each feature cohort:** A table with columns for feature cohorts (sub-cohort of your selected feature), sample size of each cohort, and the selected model performance metrics for each

feature cohort.

5. **Fairness metrics/disparity metrics:** A table that corresponds to the metrics table and shows the maximum difference or maximum ratio in performance scores between any two feature cohorts.
6. **Bar chart visualizing individual metric:** View mean absolute error across the cohorts for easy comparison.
7. **Choose cohorts (y-axis):** Select this button to choose which cohorts to view in the bar chart.

Selecting **Choose cohorts** opens a panel with an option to either show a comparison of selected dataset cohorts or feature cohorts, depending on what you select in the multi-select dropdown list below it. Select **Confirm** to save the changes to the bar chart view.



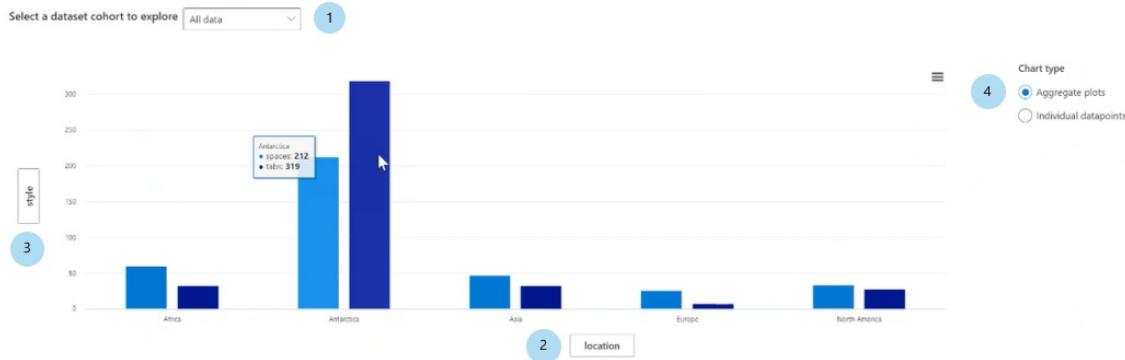
8. **Choose metric (x-axis):** Select this button to choose which metric to view in the bar chart.

Data explorer

With the data explorer component, you can analyze data statistics along the x-axis and y-axis by using filters such as predicted outcome, dataset features, and error groups. This component helps you understand overrepresentation and underrepresentation in your dataset.

Data explorer

Create dataset cohorts to analyze dataset statistics along filters such as predicted outcome, dataset features and error groups. Learn about the over/under presentation in your dataset.



1. **Select a dataset cohort to explore:** Specify which dataset cohort from your list of cohorts you want to view data statistics for.
2. **X-axis:** Displays the type of value being plotted horizontally. Modify the values by selecting the button to open a side panel.
3. **Y-axis:** Displays the type of value being plotted vertically. Modify the values by selecting the button to open a side panel.
4. **Chart type:** Specifies the chart type. Choose between aggregate plots (bar charts) or individual data points (scatter plot).

By selecting the **Individual data points** option under **Chart type**, you can shift to a disaggregated view

of the data with the availability of a color axis.

Data explorer

Create dataset cohorts to analyze dataset statistics along filters such as predicted outcome, dataset features and error groups. Learn about the over/under presentation in your dataset.

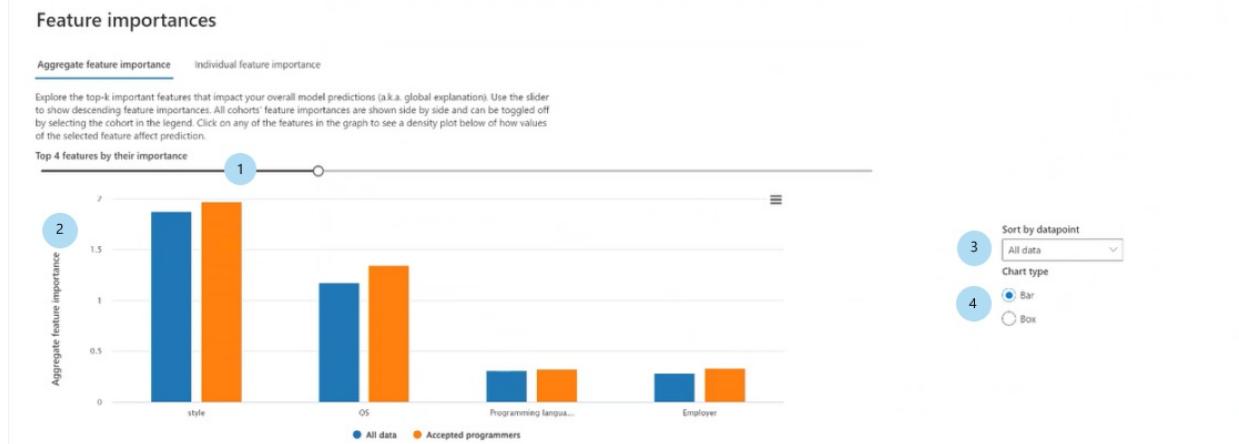
Select a dataset cohort to explore



Feature importances (model explanations)

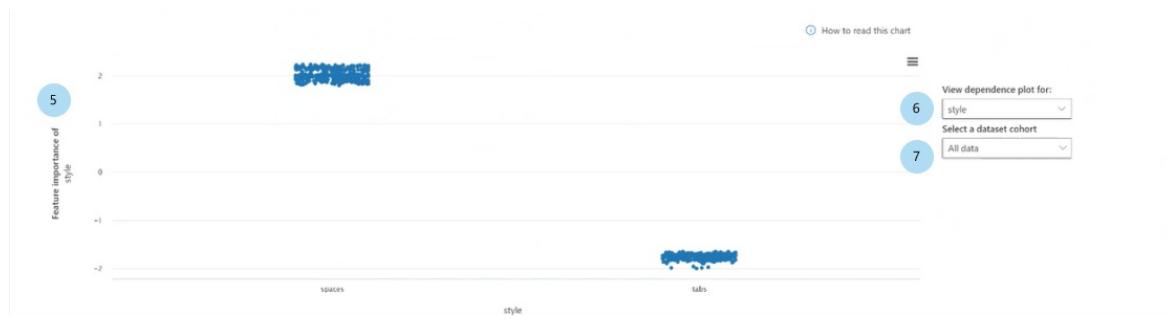
By using the model explanation component, you can see which features were most important in your model's predictions. You can view what features affected your model's prediction overall on the **Aggregate feature importance** pane or view feature importances for individual data points on the **Individual feature importance** pane.

Aggregate feature importances (global explanations)



- 1. Top k features:** Lists the most important global features for a prediction and allows you to change it by using a slider bar.
- 2. Aggregate feature importance:** Visualizes the weight of each feature in influencing model decisions across all predictions.
- 3. Sort by:** Allows you to select which cohort's importances to sort the aggregate feature importance graph by.
- 4. Chart type:** Allows you to select between a bar plot view of average importances for each feature and a box plot of importances for all data.

When you select one of the features in the bar plot, the dependence plot is populated, as shown in the following image. The dependence plot shows the relationship of the values of a feature to its corresponding feature importance values, which affect the model prediction.



5. Feature importance of [feature] (regression) or Feature importance of [feature] on [predicted class] (classification): Plots the importance of a particular feature across the predictions. For regression scenarios, the importance values are in terms of the output, so positive feature importance means it contributed positively toward the output. The opposite applies to negative feature importance. For classification scenarios, positive feature importances mean that feature value is contributing toward the predicted class denoted in the y-axis title. Negative feature importance means it's contributing against the predicted class.

6. View dependence plot for: Selects the feature whose importances you want to plot.

7. Select a dataset cohort: Selects the cohort whose importances you want to plot.

Individual feature importances (local explanations)

The following image illustrates how features influence the predictions that are made on specific data points. You can choose up to five data points to compare feature importances for.

Feature importances

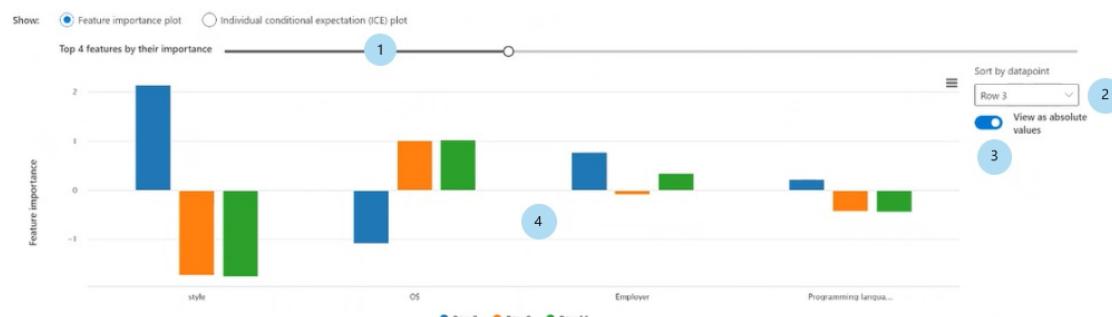
Aggregate feature importance Individual feature importance

Select a datapoint by clicking on a datapoint (up to 5 datapoints) in the table to view their local feature importance values (local explanation) and individual conditional expectation (ICE) plot below.

3/5 datapoints selected

Index	TrueY	PredictedY	style	VOE	IDE	Programming language	location	Number of github repos cont...
0	8	6.733062	spaces	16	Emacs	R	Antarctica	2
2	1	1.296875	tabs	7	XCode	C#	Antarctica	0
3	6	7.37109375	spaces	15	Visual Studio	R	Antarctica	0
4	5	4.1953125	tabs	7	Eclipse	Java	Antarctica	0
8	2	3.875	tabs	6	VSCode	Java	Antarctica	0
9	3	4.30859375	tabs	9	IntelliJ	Python	Antarctica	0
11	2	4.21875	tabs	7	XCode	Java	Antarctica	0
15	4	4.23026125	tabs	7	XCode	Python	Antarctica	0
20	0	5.40925	spaces	16	Visual Studio	C#	Antarctica	0
21	5	3.8828125	tabs	9	IntelliJ	Java	Antarctica	0

Point selection table: View your data points and select up to five points to display in the feature importance plot or the ICE plot below the table.

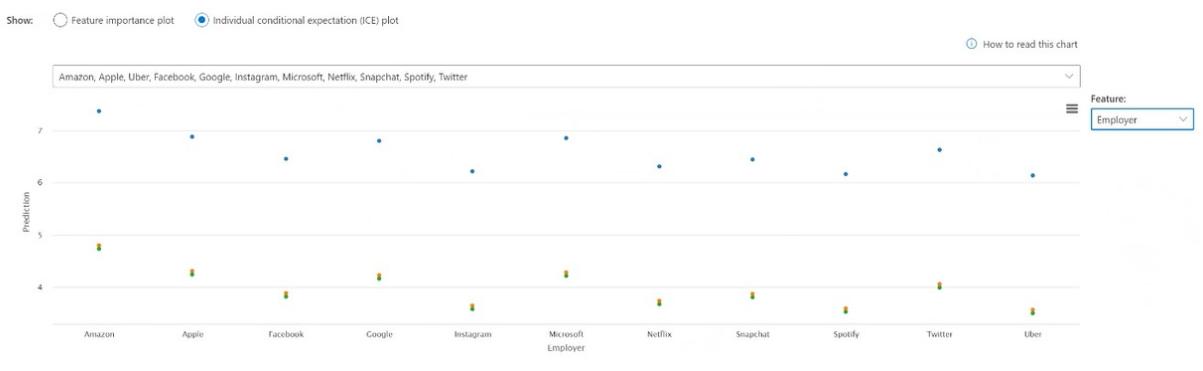


Feature importance plot: A bar plot of the importance of each feature for the model's prediction on the selected data points.

1. Top k features: Allows you to specify the number of features to show importances for by using a slider.

2. **Sort by:** Allows you to select the point (of those checked above) whose feature importances are displayed in descending order on the feature importance plot.
3. **View absolute values:** Toggle on to sort the bar plot by the absolute values. This allows you to see the most impactful features regardless of their positive or negative direction.
4. **Bar plot:** Displays the importance of each feature in the dataset for the model prediction of the selected data points.

Individual conditional expectation (ICE) plot: Switches to the ICE plot, which shows model predictions across a range of values of a particular feature.



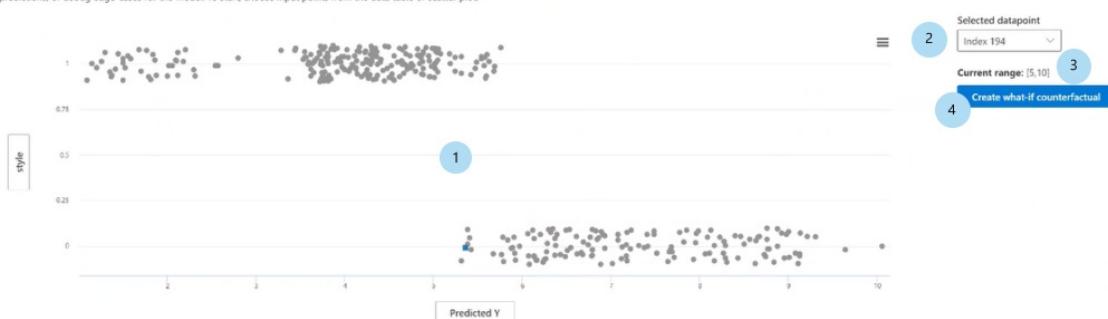
- **Min (numerical features):** Specifies the lower bound of the range of predictions in the ICE plot.
- **Max (numerical features):** Specifies the upper bound of the range of predictions in the ICE plot.
- **Steps (numerical features):** Specifies the number of points to show predictions for within the interval.
- **Feature values (categorical features):** Specifies which categorical feature values to show predictions for.
- **Feature:** Specifies the feature to make predictions for.

Counterfactual what-if

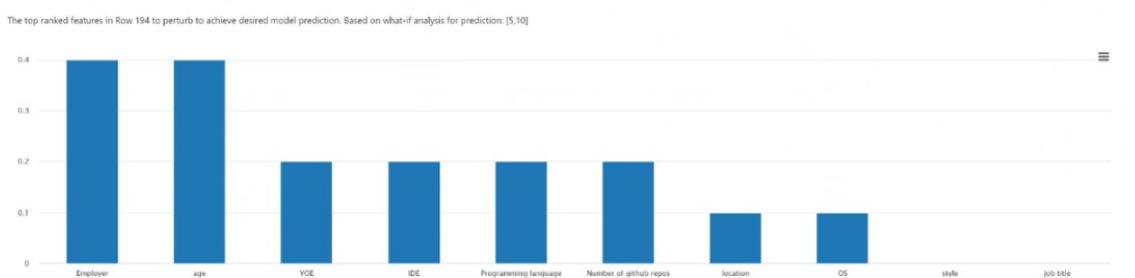
Counterfactual analysis provides a diverse set of *what-if* examples generated by changing the values of features minimally to produce the desired prediction class (classification) or range (regression).

Counterfactuals

What-if allows you to perturb features for any input and observe how the model's prediction changes. You can perturb features manually or specify the desired prediction (e.g., class label for a classifier) to see a list of closest data points to the original input that would lead to the desired prediction. Also known as prediction counterfactuals, you can use them for exploring the relationships learnt by the model; understanding important, necessary features for the model's predictions; or debug edge-cases for the model. To start, choose input points from the data table or scatter plot.



1. **Point selection:** Selects the point to create a counterfactual for and display in the top-ranking features plot below it.



Top ranked features plot: Displays, in descending order of average frequency, the features to perturb to create a diverse set of counterfactuals of the desired class. You must generate at least 10 diverse counterfactuals per data point to enable this chart, because there's a lack of accuracy with a lesser number of counterfactuals.

2. **Selected data point:** Performs the same action as the point selection in the table, except in a dropdown menu.
3. **Desired class for counterfactual(s):** Specifies the class or range to generate counterfactuals for.
4. **Create what-if counterfactual:** Opens a panel for counterfactual what-if data point creation.

Select the **Create what-if counterfactual** button to open a full window panel.

Row	Predicted value (score)	Employer	age	YOE	IDE	Programming language	Number of github repos	location	OS
Reference datapoint: Row 194 Set Value	0	Uber	29.3	17	VSCode	R	0	Antarctica	Linux
Counterfactual Ex 1 Set Value	5.14453125	-	22.4	-	-	-	-	Europe	-
Counterfactual Ex 2 Set Value	5.32421875	-	44.7	-	-	-	-	-	-
Counterfactual Ex 3 Set Value	5.24609375	-	-	14.1	Emacs	-	-	-	-
Counterfactual Ex 4 Set Value	5.32421875	-	-	12	-	-	-	-	-
Counterfactual Ex 5 Set Value	5.37109375	Amazon	-	-	-	C#	-	-	-
Counterfactual Ex 6 Set Value	5.97245625	Snapchat	-	-	-	-	-	MacOS	-
Counterfactual Ex 7 Set Value	5.32421875	-	36.5	-	-	-	1.1	-	-
Counterfactual Ex 8 Set Value	5.24609375	-	-	-	Emacs	-	5.3	-	-
Counterfactual Ex 9 Set Value	5.28515825	Microsoft	-	-	-	C	-	-	-
Counterfactual Ex 10 Set Value	5.34765625	Spotify	24.8	-	-	-	-	-	-
Create your own counterfactual: Copy of row 194 See prediction delta Now predicted value: 3.27	0	Spotify	29.3	17	Emacs	R	0	Antarctica	Linux
What-if counterfactual name: Row 194 spotify emacs Save as new datapoint	10	11	These counterfactuals are based on the model's prediction, and therefore may be capturing the same correlations that were picked up by the model. We do not advise users to make real-life decisions based on this tool. Use our toolkit for causal analysis for real-life decision-making scenarios.						

5. **Search features:** Finds features to observe and change values.
6. **Sort counterfactual by ranked features:** Sorts counterfactual examples in order of perturbation effect. (Also see **Top ranked features plot**, discussed earlier.)
7. **Counterfactual examples:** Lists feature values of example counterfactuals with the desired class or range. The first row is the original reference data point. Select **Set value** to set all the values of your own counterfactual data point in the bottom row with the values of the pre-generated counterfactual example.
8. **Predicted value or class:** Lists the model prediction of a counterfactual's class given those changed features.
9. **Create your own counterfactual:** Allows you to perturb your own features to modify the counterfactual. Features that have been changed from the original feature value are denoted by the title being bolded (for example, Employer and Programming language). Select **See prediction delta** to view the difference in the new prediction value from the original data point.
10. **What-if counterfactual name:** Allows you to name the counterfactual uniquely.
11. **Save as new data point:** Saves the counterfactual you've created.

Causal analysis

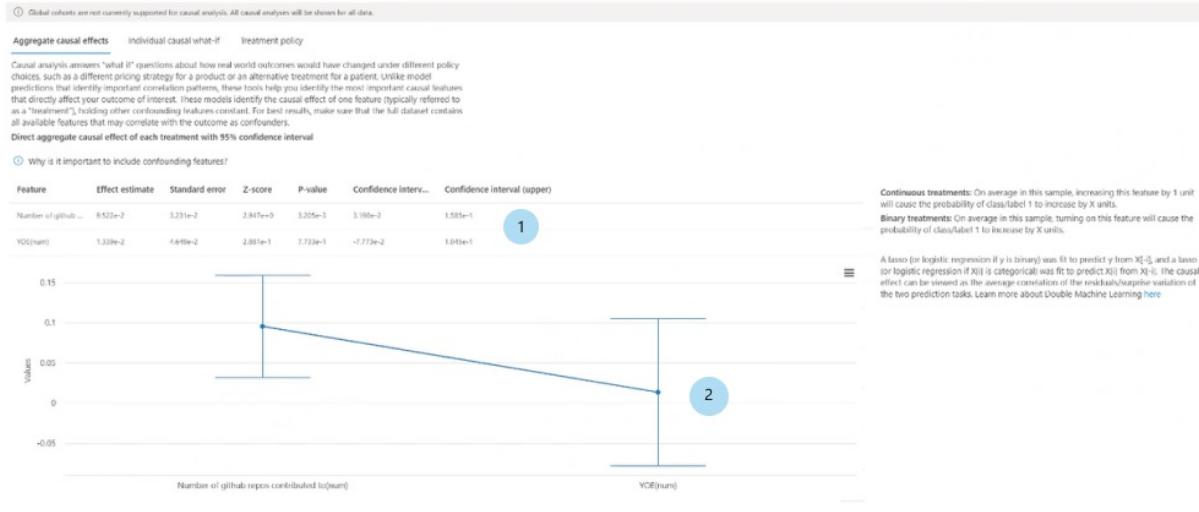
Aggregate causal effects

Select the **Aggregate causal effects** tab of the causal analysis component to display the average causal effects for pre-defined treatment features (the features that you want to treat to optimize your outcome).

NOTE

Global cohort functionality is not supported for the causal analysis component.

Causal analysis



1. **Direct aggregate causal effect table:** Displays the causal effect of each feature aggregated on the entire dataset and associated confidence statistics.

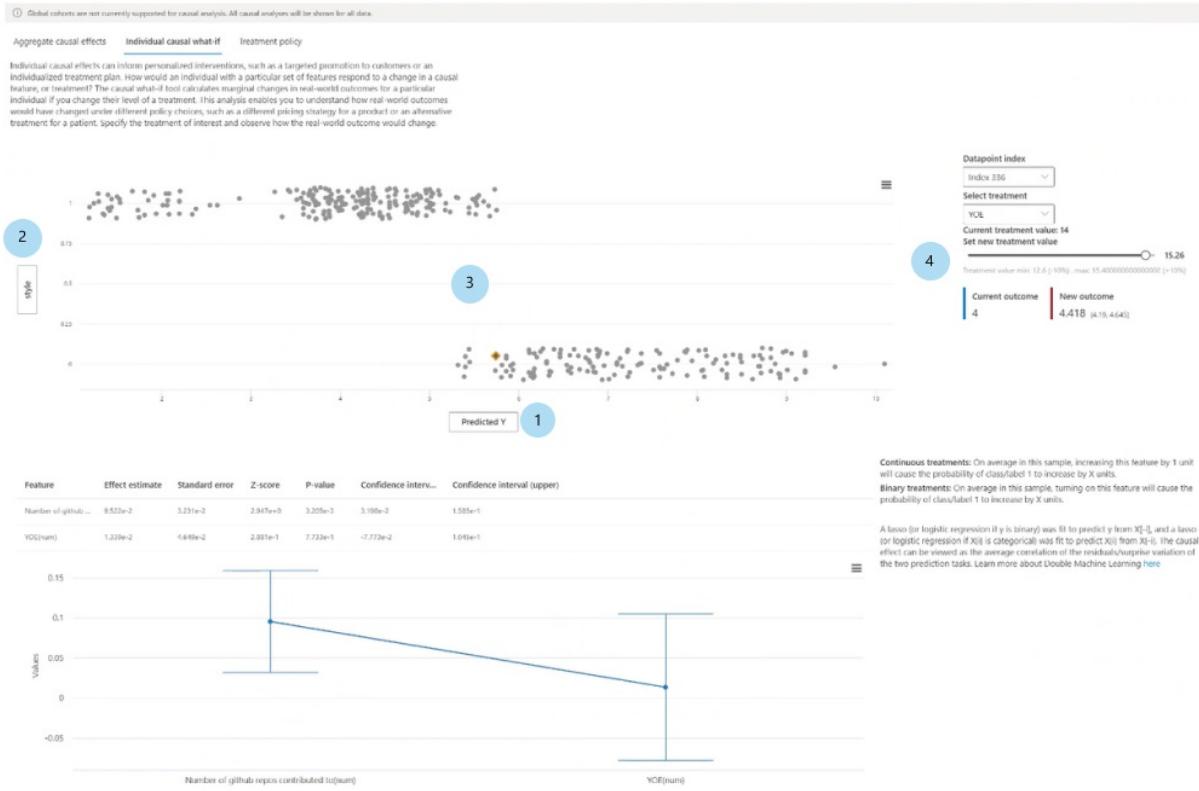
- **Continuous treatments:** On average in this sample, increasing this feature by one unit will cause the probability of class to increase by X units, where X is the causal effect.
- **Binary treatments:** On average in this sample, turning on this feature will cause the probability of class to increase by X units, where X is the causal effect.

2. **Direct aggregate causal effect whisker plot:** Visualizes the causal effects and confidence intervals of the points in the table.

Individual causal effects and causal what-if

To get a granular view of causal effects on an individual data point, switch to the **Individual causal what-if** tab.

Causal analysis



- X-axis:** Selects the feature to plot on the x-axis.
- Y-axis:** Selects the feature to plot on the y-axis.
- Individual causal scatter plot:** Visualizes points in the table as a scatter plot to select data points for analyzing causal what-if and viewing the individual causal effects below it.
- Set new treatment value:**
 - (numerical):** Shows a slider to change the value of the numerical feature as a real-world intervention.
 - (categorical):** Shows a dropdown list to select the value of the categorical feature.

Treatment policy

Select the **Treatment policy** tab to switch to a view to help determine real-world interventions and show treatments to apply to achieve a particular outcome.

Causal analysis

Global cohorts are not currently supported for causal analysis. All causal analyses will be shown for all data.

Aggregate causal effects Individual causal what-if Treatment policy

These tools help build policies for future interventions. You can identify what parts of your sample experience the largest responses to changes in causal features, or treatments, and construct rules to define which future populations should be targeted for particular interventions.

Set treatment feature Number of github repos co... 1

Interpretable recommended global treatment policy for sample size (n) = 800

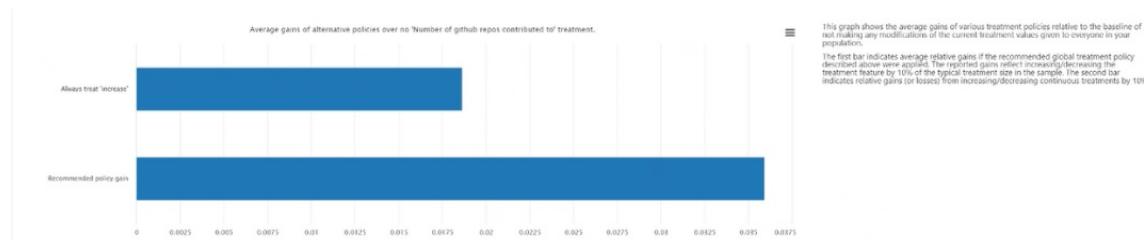
	Programming language != Javascript	Programming language == Javascript
Employer != Snapchat	$n = 658$ Recommended treatment = increase	$n = 64$ Recommended treatment = decrease
Employer == Snapchat	$n = 58$ Recommended treatment = decrease	

This table shows a recommended treatment policy that can be applied to the current data sample or other populations. The table provides a simple rule to segment observations into data cohorts based on the features with the largest impact on the target feature. For each cohort, the recommended treatment is shown. The table also specifies number of datapoints in the current data sample assigned to each segment.

The table can be read by taking a row and then taking a column of that specific row.

- Set treatment feature:** Selects a feature to change as a real-world intervention.
- Recommended global treatment policy:** Displays recommended interventions for data cohorts to improve the target feature value. The table can be read from left to right, where the segmentation of the dataset is first in rows and then in columns. For example, for 658 individuals whose employer isn't Snapchat and whose programming language isn't JavaScript, the recommended treatment policy is to increase the number of GitHub repos contributed to.

Average gains of alternative policies over always applying treatment: Plots the target feature value in a bar chart of the average gain in your outcome for the above recommended treatment policy versus always applying treatment.



Recommended individual treatment policy:

Recommended individual treatment policy

Data points with the largest estimated causal responses to treatment feature: Number of github repos contributed to 10 3

Recommended treatment	Current treatment	Effect of recommendation	CI lower	CI upper	index	style	YOE	IDE	Program...	location	Employer	C
decrease	0	1.050e-1	7.682e-2	3.133e-1	233	tabs	8	Vim	JavaScript	America	Snapchat	M
decrease	0	1.622e-1	3.248e-2	2.719e-1	689	spaces	34	Visual Studio	Swift	Antarctica	Snapchat	L
decrease	1	1.495e-1	1.599e-2	2.827e-1	95	spaces	17	Emacs	JavaScript	Antarctica	Snapchat	M
decrease	0	1.465e-1	5.716e-2	7.553e-1	473	spaces	16	Vim	C++	America	Snapchat	M
increase	0	1.433e-1	2.598e-2	2.516e-1	514	tabs	9	IntelliJ	Python	Africa	Instagram	M
increase	0	1.407e-1	3.391e-2	7.4e-1	583	tabs	9	pyCharm	Java	Africa	Spotify	M
decrease	5	1.406e-1	1.859e-2	2.625e-1	451	spaces	15	Vim	JavaScript	Europe	Microsoft	M
increase	5	1.404e-1	2.529e-2	2.556e-1	168	spaces	18	XCode	GO	Africa	Google	M
increase	0	1.383e-1	3.187e-2	2.466e-1	207	tabs	9	pyCharm	Python	Africa	Spotify	M
decrease	0	1.371e-1	1.053e-2	2.557e-1	415	spaces	15	VSCode	R	Antarctica	Snapchat	L

This list shows which datapoints in the current data sample have the largest causal response to the selected treatment, based on all features in the dataset. The columns report whether treatment is recommended for the observation, the current treatment, the effect of the treatment, the confidence interval for the effect, the index of the observation, the style of the observation, years of experience, the integrated development environment used, the programming language, the location, the employer, and the causal effect.

Use the spinner control to choose the top k samples you'd like to view.

- Show top k data point samples ordered by causal effects for recommended treatment feature:** Selects the number of data points to show in the table.

4. **Recommended individual treatment policy table:** Lists, in descending order of causal effect, the data points whose target features would be most improved by an intervention.

Next steps

- Summarize and share your Responsible AI insights with the [Responsible AI scorecard as a PDF export](#).
- Learn more about the [concepts and techniques behind the Responsible AI dashboard](#).
- View [sample YAML and Python notebooks](#) to generate a Responsible AI dashboard with YAML or Python.
- Explore the features of the Responsible AI dashboard through this [interactive AI lab web demo](#).
- Learn more about how you can use the Responsible AI dashboard and scorecard to debug data and models and inform better decision-making in this [tech community blog post](#).
- Learn about how the Responsible AI dashboard and scorecard were used by the UK National Health Service (NHS) in a [real-life customer story](#).

Share insights with a Responsible AI scorecard (preview)

9/22/2022 • 9 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)  Python SDK azure-ai-ml v2 (preview)

An Azure Machine Learning Responsible AI scorecard is a PDF report that's generated based on Responsible AI dashboard insights and customizations to accompany your machine learning models. You can easily configure, download, and share your PDF scorecard with your technical and non-technical stakeholders to educate them about your data and model health and compliance, and to help build trust. You can also use the scorecard in audit reviews to inform the stakeholders about the characteristics of your model.

Why a Responsible AI scorecard?

The Responsible AI dashboard is designed for machine learning professionals and data scientists to explore and evaluate model insights and inform their data-driven decisions. Though the dashboard can help you implement Responsible AI practically in your machine learning lifecycle, there are some needs left unaddressed:

- There often exists a gap between the technical Responsible AI tools (designed for machine learning professionals) and the ethical, regulatory, and business requirements that define the production environment.
- Although an end-to-end machine learning lifecycle keeps both technical and non-technical stakeholders in the loop, there's very little support to enable an effective multi-stakeholder alignment where technical experts get timely feedback and direction from the non-technical stakeholders.
- AI regulations make it essential to be able to share model and data insights with auditors and risk officers for auditability purposes.

One of the biggest benefits of using the Azure Machine Learning ecosystem is related to the ability to archive, for quick future reference, model and data insights in the Azure Machine Learning run history. As a part of that infrastructure and to accompany machine learning models and their corresponding Responsible AI dashboards, we're introducing the Responsible AI scorecard to empower machine learning professionals to generate and share their data and model health records easily.

Who should use a Responsible AI scorecard?

- If you're a data scientist or machine learning professional:

After training your model and generating its corresponding Responsible AI dashboards for assessment and decision-making purposes, you can extract those learnings via our PDF scorecard and share the report easily with your technical and non-technical stakeholders. Doing so helps build trust and gain their approval for deployment.

- If you're a product manager, a business leader, or an accountable stakeholder on an AI product:

You can pass your desired model performance and fairness target values, such as target accuracy or target error rate, to your data science team. The team can generate a scorecard with respect to your identified target values, assess whether your model meets them, and then advise as to whether the model should be deployed or further improved.

Generate a Responsible AI scorecard

The configuration stage requires you to use your domain expertise around the problem to set your desired target values on model performance and fairness metrics.

As with other Responsible AI dashboard components [configured in the YAML pipeline](#), you can add a component to generate the scorecard in the YAML pipeline.

In the following code, the *pdf_gen.json* file is the JSON configuration file for scorecard generation, and *cohorts.json* is the JSON definition file for pre-built cohorts.

```
scorecard_01:  
  
  type: command  
  component: azureml:rai_score_card@latest  
  inputs:  
    dashboard: ${parent.jobs.gather_01.outputs.dashboard}  
    pdf_generation_config:  
      type: uri_file  
      path: ./pdf_gen.json  
      mode: download  
  
    predefined_cohorts_json:  
      type: uri_file  
      path: ./cohorts.json  
      mode: download
```

Here's a sample JSON file for cohorts definition and scorecard-generation configuration:

Cohorts definition:

```
[  
  {  
    "name": "High Yoe",  
    "cohort_filter_list": [  
      {  
        "method": "greater",  
        "arg": [  
          5  
        ],  
        "column": "YOE"  
      }  
    ]  
  },  
  {  
    "name": "Low Yoe",  
    "cohort_filter_list": [  
      {  
        "method": "less",  
        "arg": [  
          6.5  
        ],  
        "column": "YOE"  
      }  
    ]  
  }]
```

Here's a scorecard-generation configuration file as a regression example:

```
{
  "Model": {
    "ModelName": "GPT-2 Access",
    "ModelType": "Regression",
    "ModelSummary": "This is a regression model to analyze how likely a programmer is given access to GPT-2"
  },
  "Metrics": {
    "mean_absolute_error": {
      "threshold": "<=20"
    },
    "mean_squared_error": {}
  },
  "FeatureImportance": {
    "top_n": 6
  },
  "DataExplorer": {
    "features": [
      "YOE",
      "age"
    ]
  },
  "Fairness": {
    "metric": ["mean_squared_error"],
    "sensitive_features": ["YOUR SENSITIVE ATTRIBUTE"],
    "fairness_evaluation_kind": "difference OR ratio"
  },
  "Cohorts": [
    "High Yoe",
    "Low Yoe"
  ]
}

```

Here's a scorecard-generation configuration file as a classification example:

```
{
  "Model": {
    "ModelName": "Housing Price Range Prediction",
    "ModelType": "Classification",
    "ModelSummary": "This model is a classifier that predicts whether the house will sell for more than the median price."
  },
  "Metrics": {
    "accuracy_score": {
      "threshold": ">=0.85"
    }
  },
  "FeatureImportance": {
    "top_n": 6
  },
  "DataExplorer": {
    "features": [
      "YearBuilt",
      "OverallQual",
      "GarageCars"
    ]
  },
  "Fairness": {
    "metric": ["accuracy_score", "selection_rate"],
    "sensitive_features": ["YOUR SENSITIVE ATTRIBUTE"],
    "fairness_evaluation_kind": "difference OR ratio"
  }
}
```

Definition of inputs for the Responsible AI scorecard component

This section lists and defines the parameters that are required to configure the Responsible AI scorecard component.

Model

MODELNAME	NAME OF MODEL
ModelType	Values in ['classification', 'regression'].
ModelSummary	Enter text that summarizes what the model is for.

NOTE

For multi-class classification, you should first use the One-vs-Rest strategy to choose your reference class, and then split your multi-class classification model into a binary classification problem for your selected reference class versus the rest of the classes.

Metrics

PERFORMANCE METRIC	DEFINITION	MODEL TYPE
accuracy_score	The fraction of data points that are classified correctly.	Classification
precision_score	The fraction of data points that are classified correctly among those classified as 1.	Classification
recall_score	The fraction of data points that are classified correctly among those whose true label is 1. Alternative names: true positive rate, sensitivity.	Classification
f1_score	The F1 score is the harmonic mean of precision and recall.	Classification
error_rate	The proportion of instances that are misclassified over the whole set of instances.	Classification
mean_absolute_error	The average of absolute values of errors. More robust to outliers than <code>mean_squared_error</code> .	Regression
mean_squared_error	The average of squared errors.	Regression
median_absolute_error	The median of squared errors.	Regression
r2_score	The fraction of variance in the labels explained by the model.	Regression

Threshold: The desired threshold for the selected metric. Allowed mathematical tokens are `>`, `<`, `>=`, and `<=`, followed by a real number. For example, `>= 0.75` means that the target for the selected metric is greater than or equal to 0.75.

Feature importance

`top_n`: The number of features to show, with a maximum of 10. Positive integers up to 10 are allowed.

Fairness

METRIC	DEFINITION
<code>metric</code>	The primary metric for evaluation fairness.
<code>sensitive_features</code>	A list of feature names from the input dataset to be designated as sensitive features for the fairness report.
<code>fairness_evaluation_kind</code>	Values in ['difference', 'ratio'].
<code>threshold</code>	The <i>desired target values</i> of the fairness evaluation. Allowed mathematical tokens are <code>></code> , <code><</code> , <code>>=</code> , and <code><=</code> , followed by a real number. For example, <code>metric="accuracy"</code> , <code>fairness_evaluation_kind="difference"</code> . <code><= 0.05</code> means that the target for the difference in accuracy is less than or equal to 0.05.

NOTE

Your choice of `fairness_evaluation_kind` (selecting 'difference' versus 'ratio') affects the scale of your target value. In your selection, be sure to choose a meaningful target value.

You can select from the following metrics, paired with `fairness_evaluation_kind`, to configure your fairness assessment component of the scorecard:

METRIC	FAIRNESS_EVALUATION_KIND	DEFINITION	MODEL TYPE
<code>accuracy_score</code>	difference	The maximum difference in accuracy score between any two groups.	Classification
<code>accuracy_score</code>	ratio	The minimum ratio in accuracy score between any two groups.	Classification
<code>precision_score</code>	difference	The maximum difference in precision score between any two groups.	Classification
<code>precision_score</code>	ratio	The maximum ratio in precision score between any two groups.	Classification
<code>recall_score</code>	difference	The maximum difference in recall score between any two groups.	Classification
<code>recall_score</code>	ratio	The maximum ratio in recall score between any two groups.	Classification

METRIC	FAIRNESS_EVALUATION_KIND	DEFINITION	MODEL TYPE
f1_score	difference	The maximum difference in f1 score between any two groups.	Classification
f1_score	ratio	The maximum ratio in f1 score between any two groups.	Classification
error_rate	difference	The maximum difference in error rate between any two groups.	Classification
error_rate	ratio	The maximum ratio in error rate between any two groups.	Classification
Selection_rate	difference	The maximum difference in selection rate between any two groups.	Classification
Selection_rate	ratio	The maximum ratio in selection rate between any two groups.	Classification
mean_absolute_error	difference	The maximum difference in mean absolute error between any two groups.	Regression
mean_absolute_error	ratio	The maximum ratio in mean absolute error between any two groups.	Regression
mean_squared_error	difference	The maximum difference in mean squared error between any two groups.	Regression
mean_squared_error	ratio	The maximum ratio in mean squared error between any two groups.	Regression
median_absolute_error	difference	The maximum difference in median absolute error between any two groups.	Regression
median_absolute_error	ratio	The maximum ratio in median absolute error between any two groups.	Regression
r2_score	difference	The maximum difference in R ² score between any two groups.	Regression
r2_Score	ratio	The maximum ratio in R ² score between any two groups.	Regression

View your Responsible AI scorecard

The Responsible AI scorecard is linked to a Responsible AI dashboard. To view your Responsible AI scorecard, go into your model registry and select the registered model that you've generated a Responsible AI dashboard for. After you've selected your model, select the **Responsible AI (preview)** tab to view a list of generated dashboards. Select which dashboard you want to export a Responsible AI scorecard PDF for by selecting **Responsible AI scorecard (preview)**.

This screenshot shows the Microsoft Responsible AI scorecard interface. At the top, there's a navigation bar with 'Microsoft > RAIPM2 > Models > component_registered_lr_01_1651118466:1'. Below the navigation is a sub-header 'component_registered_lr_01_1651118466:1'. A horizontal menu bar includes 'Details', 'Versions', 'Artifacts', 'Endpoints', 'Data', 'Explanations (preview)', 'Fairness (preview)', and 'Responsible AI (preview)'. The 'Responsible AI (preview)' tab is currently selected. A note below the menu states: 'Responsible AI insights can be generated from the SDK/CLI or through the UI wizard and viewed with your Responsible AI dashboard. To view your insights, click on the name of your dashboard. Different combinations of components (explainers, causal analysis, etc.) can be attached to each Responsible AI dashboard. The list below only shows whether or not a component was generated for your dashboard, but different components can be viewed or hidden within the dashboard itself. [Learn more](#)'.

Below the note, there are buttons for 'Refresh' and 'Responsible AI scorecard (preview)'. A dropdown menu shows 'Current view: Local' and 'Edit view'. On the right, there's a 'Page size: 25' dropdown. The main area displays a table titled 'Showing 1-1 of 1 Responsible AI dashboards'. The table has columns: Name, Explainer, Error analysis, Causal insights, Counterfactual, Status, Created on, and Tags. One row is shown: 'gather_01' (Explainer: Mimic explainer, Status: Completed, Created on: Apr 27, 2022 9:07 PM). There are three green checkmarks in the status column.

1. Select **Responsible AI scorecard (preview)** to display a list of all Responsible AI scorecards that are generated for this dashboard.

This screenshot shows the same Responsible AI scorecard interface as the previous one, but with a specific action highlighted. The 'View all' button in the 'Current view' dropdown is highlighted with a red box. The rest of the interface is identical to the first screenshot, including the table showing the single 'gather_01' dashboard entry.

2. In the list, select the scorecard you want to download, and then select **Download** to download the PDF to your machine.

The screenshot shows the Microsoft Responsible AI scorecards interface. On the left, there's a list of dashboards under the heading "Responsible AI scorecards". One dashboard, "gather_01", is selected. On the right, a modal dialog is open titled "Responsible AI scorecard name" with a search bar and three options: "Loan allocation model" (selected), "model scorecard", and "Loan allocation model (2)".

Read your Responsible AI scorecard

The Responsible AI scorecard is a PDF summary of key insights from your Responsible AI dashboard. The first summary segment of the scorecard gives you an overview of the machine learning model and the key target values you've set to help your stakeholders determine whether the model is ready to be deployed:

GPT-2 Access Predictor

Generated by Anna Garcia on April 14th, 2022

Model summary

Purpose

The linear regression model is predicting how likely a programmer should get access to a GPT2 model given their information, such as their preferred programming language, operating system, programming style (tabs or spaces), location, and so forth.

Type

Regression

How the model is evaluated

The model is evaluated on a test set with 800 datapoints

Target values

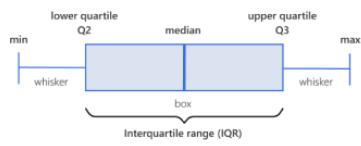
Here are your defined target minimum/maximum performance levels and/or target performance differences between groups/cohorts:

- Mean squared error (MSE): **below 2**
- Fairness metric: difference in MSE: **below 0.5**
- Showing the **top 10** important features

The data explorer segment shows you characteristics of your data, because any model story is incomplete without a correct understanding of your data:

Observe evidence of your model performance here:

How to read a box plot:

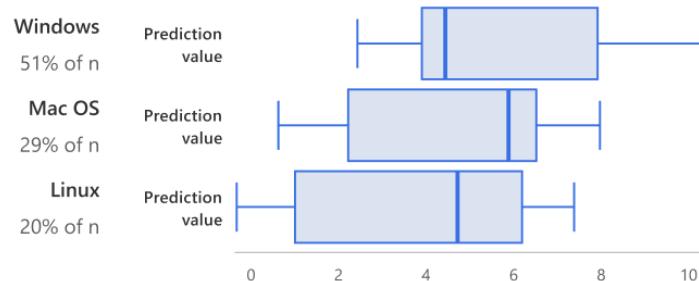


OS

For the "Windows" datapoints, 4.068 is the average of the squared difference between the actual value and the predicted value.

For the "Mac OS" datapoints, 2.926 is the average of the squared difference between the actual value and the predicted value.

For the "Linux" datapoints, 3.772 is the average of the squared difference between the actual value and the predicted value.

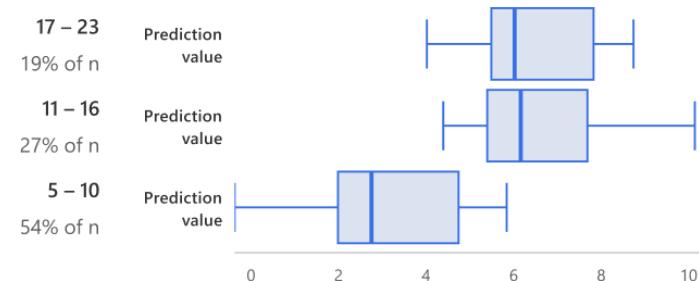


YOE

For the "17-23", 3.974 is the average of the squared difference between the actual value and the predicted value.

For the "11-16" datapoints, 3.258 is the average of the squared difference between the actual value and the predicted value.

For the "5-10" datapoints, 3.78 is the average of the squared difference between the actual value and the predicted value.



The model performance segment displays your model's most important metrics and characteristics of your predictions and how well they satisfy your desired target values:

Observe evidence of model error rates and performance here:

Mean Squared Error: 3.67

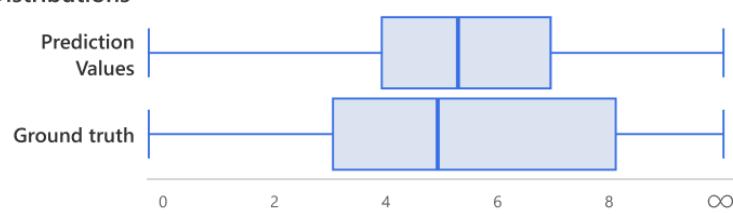
Does not meet target

3.67 is the average of the squared difference between the actual values and the predicted values.

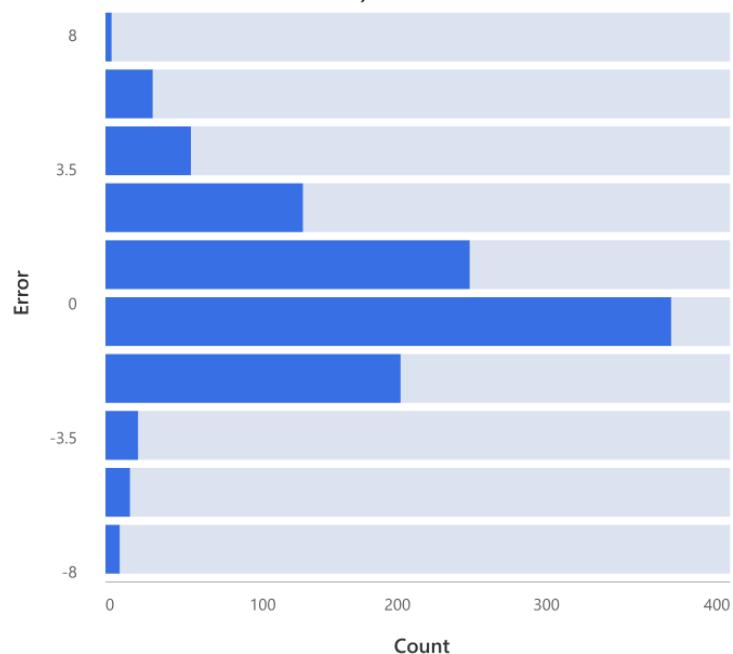
Mean Absolute Error: 1201.59

1201.59 is the average of the absolute error values.

Distributions



Histogram of your model errors (distance between a predicted value and the observed actual value)



Next, you can also view the top performing and worst performing data cohorts and subgroups that are automatically extracted for you to see the blind spots of your model:

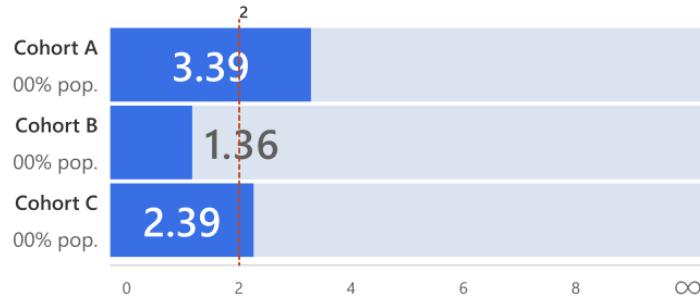
Cohorts

Observe evidence of model performance across your dataset cohorts:

Top performing cohorts:

- A: Location | Not Antarctica, Programming language | C, C#
- B: Location | Is Antarctica, Programming language | Javascript, OS | Windows
- C: Location | Is Antarctica, Programming language | PHP

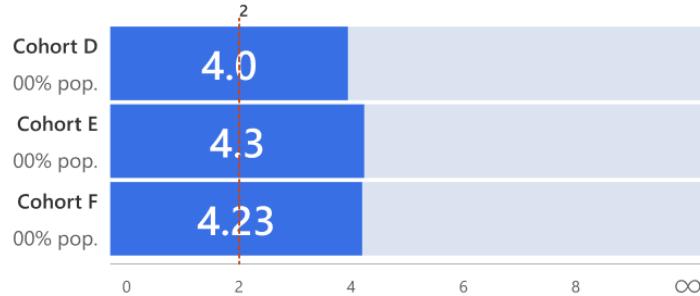
Top performing cohorts: MSE



Poorest performing cohorts:

- D: Location | Is Antarctica, Programming language | ≤ 2.5
- E: Location | Not Antarctica, Age | <32
- F: Location | Is Antarctica, Programming language | Not Javascript

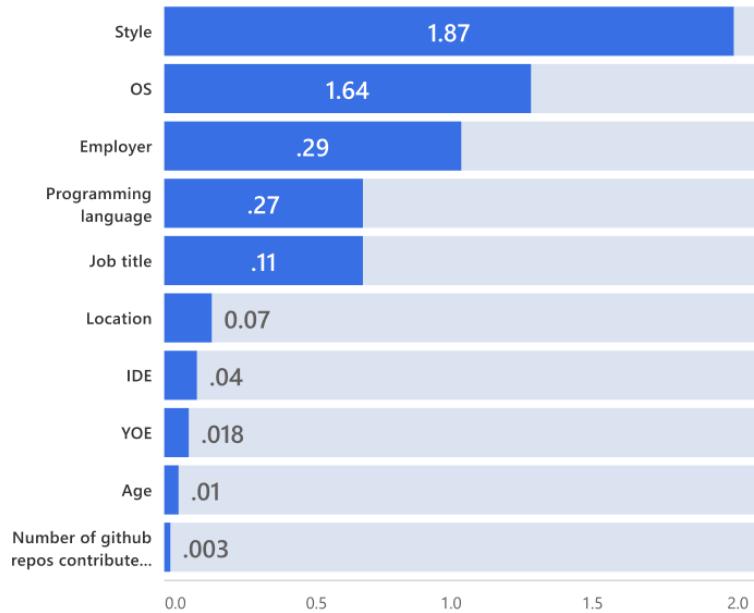
Poorest performing cohorts: MSE



You can see the top important factors that affect your model predictions, which is a requirement to build trust with how your model is performing its task:

Feature relevance (explainability)

Understand how your model is making its predictions by observing the top features that have impacted your model predictions the most.



You can further see your model fairness insights summarized and inspect how well your model is satisfying the fairness target values you've set for your desired sensitive groups:

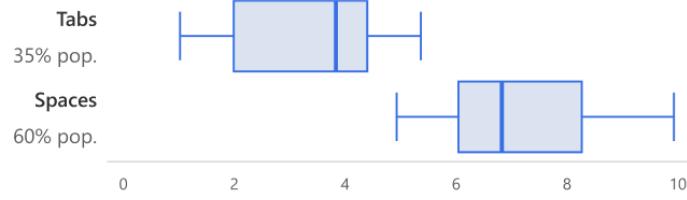
Understand your model's fairness issues using group-fairness metrics across sensitive features and cohorts. Pay particular attention to the subgroups who receive worse treatments (predictions) by your model.

Tabs has the highest MSE: 3.78

Spaces has the lowest MSE: 3.55

Difference in MSE: 0.23

Prediction distribution chart



Analysis across cohorts:

	Average Prediction	Average Ground Truth	Mean Squared Error	Mean Absolute Error
Tabs	3.4	3	3.78	658
Spaces	7.1	8	3.55	543
Difference	3.7	5	.23	115
Ratio	.47	.37	.93	.82

Finally, you can see your dataset's causal insights summarized, which can help you determine whether your identified factors or treatments have any causal effect on the real-world outcome:

Use causal analysis to learn whether your identified factors/treatments have any causal effect on the real-world outcome. Causal analysis answers your real-world "what if" questions about how changes of treatments would impact your real-world outcome.

Number of github repos contributed to

On average, increasing Bank promotion amount by 1 increases outcome by X.

Optimal policy to apply "Number of github repos contributed to" (rules to define how your future populations should be targeted for particular interventions):

When YOE is above 10 and Job Title is distinguished engineer, decrease "Number of github repos contributed to".

When YOE is below 10 and Job Title is not distinguished engineer, increase "Number of github repos contributed to".

Top data points responding the most to the treatment "Number of github repos contributed to":

What data points experience the largest impact (causal effect) to changes, while treating "bank promotion"

Index	Current treatment	Recommended treatment	Effect estimate
1182	0	increase	4.882e+1
1070	0	increase	4.688e+1
1386	5	increase	4.581e+1

Next steps

- See the how-to guide for generating a Responsible AI dashboard via [CLI v2 and SDK v2](#) or the [Azure Machine Learning studio UI](#).
- Learn more about the [concepts and techniques behind the Responsible AI dashboard](#).

- View [sample YAML and Python notebooks](#) to generate a Responsible AI dashboard with YAML or Python.
- Learn more about how you can use the Responsible AI dashboard and scorecard to debug data and models and inform better decision-making in this [tech community blog post](#).
- Learn about how the Responsible AI dashboard and scorecard were used by the UK National Health Service (NHS) in a [real-life customer story](#).
- Explore the features of the Responsible AI dashboard through this [interactive AI lab web demo](#).

How to migrate from v1 to v2

9/22/2022 • 10 minutes to read • [Edit Online](#)

Azure Machine Learning's v2 REST APIs, Azure CLI extension, and Python SDK (preview) introduce consistency and a set of new features to accelerate the production machine learning lifecycle. This article provides an overview of migrating from v1 to v2 with recommendations to help you decide on v1, v2, or both.

Prerequisites

- General familiarity with Azure ML and the v1 Python SDK.
- Understand [what is v2?](#)

Should I use v2?

You should use v2 if you're starting a new machine learning project. A new v2 project can reuse resources like workspaces and compute and assets like models and environments created using v1. You can also use v1 and v2 in tandem, for example using the v1 Python SDK within jobs that are submitted from the v2 CLI extension. However, see the [section below](#) for details on why separating v1 and v2 use is recommended.

We recommend assessing the effort needed to migrate a project from v1 to v2. First, you should ensure all the features needed from v1 are available in v2. Some notable feature gaps include:

- Spark support in jobs.
- Publishing jobs (pipelines in v1) as endpoints.
- AutoML jobs within pipeline jobs (AutoML step in a pipeline in v1).
- Model deployment to Azure Container Instance (ACI), replaced with managed online endpoints.
- An equivalent for ParallelRunStep in jobs.
- Support for SQL/database datastores.
- Built-in components in the designer.

You should then ensure the features you need in v2 meet your organization's requirements, such as being generally available. You and your team will need to assess on a case-by-case basis whether migrating to v2 is right for you.

IMPORTANT

New features in Azure ML will only be launched in v2.

How do I migrate to v2?

To migrate to v2, start by prototyping an existing v1 workflow into v2. Migrating will typically include:

- Optionally (and recommended in most cases), re-create resources and assets with v2.
- Refactor model training code to de-couple Azure ML code from ML model code (model training, model logging, and other model tracking code).
- Refactor Azure ML model deployment code and test with v2 endpoints.
- Refactor CI/CD code to use the v2 CLI (recommended), v2 Python SDK, or directly use REST.

Based on this prototype, you can estimate the effort involved for a full migration to v2. Consider the workflow

patterns (like [GitOps](#)) your organization wants to establish for use with v2 and factor this effort in.

Which v2 API should I use?

In v2 interfaces via REST API, CLI, and Python SDK (preview) are available. The interface you should use depends on your scenario and preferences.

API	NOTES
REST	Fewest dependencies and overhead. Use for building applications on Azure ML as a platform, directly in programming languages without an SDK provided, or per personal preference.
CLI	Recommended for automation with CI/CD or per personal preference. Allows quick iteration with YAML files and straightforward separation between Azure ML and ML model code.
Python SDK	Recommended for complicated scripting (for example, programmatically generating large pipeline jobs) or per personal preference. Allows quick iteration with YAML files or development solely in Python.

Can I use v1 and v2 together?

Generally, yes. Resources like workspace, compute, and datastore work across v1 and v2, with exceptions. A user can call the v1 Python SDK to change a workspace's description, then using the v2 CLI extension change it again. Jobs (experiments/runs/pipelines in v1) can be submitted to the same workspace from the v1 or v2 Python SDK. A workspace can have both v1 and v2 model deployment endpoints. You can also call v1 Python SDK code within jobs created via v2, though [this pattern isn't recommended](#).

We recommend creating a new workspace for using v2 to keep v1/v2 entities separate and avoid backward/forward compatibility considerations.

IMPORTANT

If your workspace uses a private endpoint, it will automatically have the `v1_legacy_mode` flag enabled, preventing usage of v2 APIs. See [how to configure network isolation with v2](#) for details.

Migrating resources and assets

This section gives an overview of migration recommendations for specific resources and assets in Azure ML. See the concept article for each entity for details on their usage in v2.

Workspace

Workspaces don't need to be migrated with v2. You can use the same workspace, regardless of whether you're using v1 or v2. We recommend creating a new workspace for using v2 to keep v1/v2 entities separate and avoid backward/forward compatibility considerations.

Do consider migrating the code for creating a workspace to v2. Typically Azure resources are managed via Azure Resource Manager (and Bicep) or similar resource provisioning tools. Alternatively, you can use the [CLI \(v2\) and YAML files](#).

IMPORTANT

If your workspace uses a private endpoint, it will automatically have the `v1_legacy_mode` flag enabled, preventing usage of v2 APIs. See [how to configure network isolation with v2](#) for details.

Connection (workspace connection in v1)

Workspace connections from v1 are persisted on the workspace, and fully available with v2.

We recommend migrating the code for creating connections to v2.

Datastore

Object storage datastore types created with v1 are fully available for use in v2. Database datastores are not supported; export to object storage (usually Azure Blob) is the recommended migration path.

We recommend migrating the code for [creating datastores](#) to v2.

Compute

Compute of type `AmlCompute` and `computeInstance` are fully available for use in v2.

We recommend migrating the code for creating compute to v2.

Endpoint and deployment (endpoint or web service in v1)

You can continue using your existing v1 model deployments. For new model deployments, we recommend migrating to v2. In v2, we offer managed endpoints or Kubernetes endpoints. The following table guides our recommendation:

ENDPOINT TYPE IN V2	MIGRATE FROM	NOTES
Local	ACI	Quick test of model deployment locally; not for production.
Managed online endpoint	ACI, AKS	Enterprise-grade managed model deployment infrastructure with near real-time responses and massive scaling for production.
Managed batch endpoint	ParallelRunStep in a pipeline for batch scoring	Enterprise-grade managed model deployment infrastructure with massively parallel batch processing for production.
Azure Kubernetes Service (AKS)	ACI, AKS	Manage your own AKS cluster(s) for model deployment, giving flexibility and granular control at the cost of IT overhead.
Azure Arc Kubernetes	N/A	Manage your own Kubernetes cluster(s) in other clouds or on-premises, giving flexibility and granular control at the cost of IT overhead.

Jobs (experiments, runs, pipelines in v1)

In v2, "experiments", "runs", and "pipelines" are consolidated into jobs. A job has a type. Most jobs are `command` jobs that run a command, like `python main.py`. What runs in a job is agnostic to any programming language, so you can run `bash` scripts, invoke `python` interpreters, run a bunch of `curl` commands, or anything else. Another common type of job is `pipeline`, which defines child jobs that may have input/output relationships,

forming a directed acyclic graph (DAG).

To migrate, you'll need to change your code for submitting jobs to v2. We recommend refactoring the control-plane code authoring a job into YAML file specification, which can then be submitted through the v2 CLI or Python SDK (preview). A simple `command` job looks like this:

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world"
environment:
  image: library/python:latest
compute: azurerm1:cpu-cluster
```

What you run *within* the job does not need to be migrated to v2. However, it is recommended to remove any code specific to Azure ML from your model training scripts. This separation allows for an easier transition between local and cloud and is considered best practice for mature MLOps. In practice, this means removing `azurerm1.*` lines of code. Model logging and tracking code should be replaced with MLflow. For more details, see [how to use MLflow in v2](#).

We recommend migrating the code for creating jobs to v2. You can see [how to train models with the CLI \(v2\)](#) and the [job YAML references](#) for authoring jobs in v2 YAMLs.

Data (datasets in v1)

Datasets are renamed to data assets. Interoperability between v1 datasets and v2 data assets is the most complex of any entity in Azure ML.

Data assets in v2 (or File Datasets in v1) are *references* to files in object storage. Thus, deleting a data asset (or v1 dataset) doesn't actually delete anything in underlying storage, only a reference. Therefore, it may be easier to avoid backward and forward compatibility considerations for data by re-creating v1 datasets as v2 data assets.

For details on data in v2, see the [data concept article](#).

We recommend migrating the code for [creating data assets](#) to v2.

Model

Models created from v1 can be used in v2. In v2, explicit model types are introduced. Similar to data assets, it may be easier to re-create a v1 model as a v2 model, setting the type appropriately.

We recommend migrating the code for creating models with [SDK](#) or [CLI](#) to v2.

Environment

Environments created from v1 can be used in v2. In v2, environments have new features like creation from a local Docker context.

We recommend migrating the code for creating environments to v2.

Scenarios across the machine learning lifecycle

There are a few scenarios that are common across the machine learning lifecycle using Azure ML. We'll look at a few and give general recommendations for migrating to v2.

Azure setup

Azure recommends Azure Resource Manager templates (often via Bicep for ease of use) to create resources. The same is a good approach for creating Azure ML resources as well.

If your team is only using Azure ML, you may consider provisioning the workspace and any other resources via YAML files and CLI instead.

Prototyping models

We recommend v2 for prototyping models. You may consider using the CLI for an interactive use of Azure ML, while your model training code is Python or any other programming language. Alternatively, you may adopt a full-stack approach with Python solely using the Azure ML SDK or a mixed approach with the Azure ML Python SDK and YAML files.

Production model training

We recommend v2 for production model training. Jobs consolidate the terminology and provide a set of consistency that allows for easier transition between types (for example, `command` to `sweep`) and a GitOps-friendly process for serializing jobs into YAML files.

With v2, you should separate your machine learning code from the control plane code. This separation allows for easier iteration and allows for easier transition between local and cloud.

Typically, converting to v2 will involve refactoring your code to use MLflow for tracking and model logging. See the [MLflow concept article](#) for details.

Production model deployment

We recommend v2 for production model deployment. Managed endpoints abstract the IT overhead and provide a performant solution for deploying and scoring models, both for online (near real-time) and batch (massively parallel) scenarios.

Kubernetes deployments are supported in v2 through AKS or Azure Arc, enabling Azure cloud and on-premises deployments managed by your organization.

Machine learning operations (MLOps)

A MLOps workflow typically involves CI/CD through an external tool. It's recommended refactor existing CI/CD workflows to use v2 APIs. Typically a CLI is used in CI/CD, though you can alternatively invoke Python or directly use REST.

The solution accelerator for MLOps with v2 is being developed at <https://github.com/Azure/mlops-v2> and can be used as reference or adopted for setup and automation of the machine learning lifecycle.

A note on GitOps with v2

A key paradigm with v2 is serializing machine learning entities as YAML files for source control with `git`, enabling better GitOps approaches than were possible with v1. For instance, you could enforce policy by which only a service principal used in CI/CD pipelines can create/update/delete some or all entities, ensuring changes go through a governed process like pull requests with required reviewers. Since the files in source control are YAML, they're easy to diff and track changes over time. You and your team may consider shifting to this paradigm as you migrate to v2.

You can obtain a YAML representation of any entity with the CLI via `az ml <entity> show --output yaml`. Note that this output will have system-generated properties, which can be ignored or deleted.

Next steps

- [Get started with the CLI \(v2\)](#)
- [Get started with the Python SDK \(v2\)](#)

Troubleshoot connection to a workspace with a private endpoint

9/22/2022 • 3 minutes to read • [Edit Online](#)

When connecting to a workspace that has been configured with a private endpoint, you may encounter a 403 or a messaging saying that access is forbidden. Use the information in this article to check for common configuration problems that can cause this error.

TIP

Before using the steps in this article, try the Azure Machine Learning workspace diagnostic API. It can help identify configuration problems with your workspace. For more information, see [How to use workspace diagnostics](#).

DNS configuration

The troubleshooting steps for DNS configuration differ based on whether you're using Azure DNS or a custom DNS. Use the following steps to determine which one you're using:

1. In the [Azure portal](#), select the private endpoint for your Azure Machine Learning workspace.
2. From the **Overview** page, select the **Network Interface** link.

The screenshot shows the Azure portal's 'Overview' page for a private endpoint named 'mymlworkspace-pe-0520'. The page includes a search bar, a delete button, and a refresh button. On the left, there's a navigation menu with links like 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Settings', 'Application security groups', 'DNS configuration', and 'Properties'. The 'Settings' section is currently selected. On the right, under the 'Essentials' heading, there are several details: Resource group ('docs-rg'), Location ('South Central US'), Subscription ('ML-docs'), Subscription ID ('abcdef1-2345-6789-0abc-def012345678'), Provisioning state ('Succeeded'), Virtual network/subnet ('ml-vnet-scentral/training'), Network interface ('mymlworkspace-pe-0520.nic.3fda0f62-b217-44ea-8742-...'), Private link resource ('mymlworkspace'), Target sub-resource ('amlworkspace'), Connection status ('Approved'), and Request/Response ('Auto-Approved'). The 'Network interface' link is highlighted with a red box.

3. Under **Settings**, select **IP Configurations** and then select the **Virtual network** link.

The screenshot shows the Azure portal interface for managing a network interface. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Settings, DNS servers, Network security group, Properties, Locks, and Monitoring. The 'Settings' section is expanded, and the 'IP configurations' option is selected, indicated by a red box. The main content area shows 'IP forwarding settings' with a switch from 'Disabled' to 'Enabled'. Below that is a 'Virtual network' dropdown set to 'ml-vnet-scentral', also highlighted with a red box. Under 'IP configurations', there's a table listing three endpoints: privateEndpoint1 (IPv4, Primary, Private IP 10.3.0.5, Public IP -), privateEndpoint2 (IPv4, Secondary, Private IP 10.3.0.6, Public IP -), and privateEndpoint3 (IPv4, Secondary, Private IP 10.3.0.7, Public IP -). A search bar for IP configurations is also present.

- From the **Settings** section on the left of the page, select the **DNS servers** entry.

The screenshot shows the Azure portal interface for managing a virtual network. The 'Virtual network' section is selected. On the left, there's a sidebar with options like Microsoft Defender for Cloud and Network manager. The 'DNS servers' option is selected, indicated by a red box. The main content area shows a 'DNS servers' section with two options: 'Default (Azure-provided)' (selected) and 'Custom', both enclosed in a red box.

- If this value is **Default (Azure-provided)** or **168.63.129.16**, then the VNet is using Azure DNS. Skip to the [Azure DNS troubleshooting](#) section.
- If there's a different IP address listed, then the VNet is using a custom DNS solution. Skip to the [Custom DNS troubleshooting](#) section.

Custom DNS troubleshooting

Use the following steps to verify if your custom DNS solution is correctly resolving names to IP addresses:

- From a virtual machine, laptop, desktop, or other compute resource that has a working connection to the private endpoint, open a web browser. In the browser, use the URL for your Azure region:

AZURE REGION	URL
Azure Government	https://portal.azure.us/?feature.privateendpointmanageddns=false
Azure China 21Vianet	https://portal.azure.cn/?feature.privateendpointmanageddns=false
All other regions	https://portal.azure.com/?feature.privateendpointmanageddns=false

- In the portal, select the private endpoint for the workspace. Make a list of FQDNs listed for the private endpoint.

3. Open a command prompt, PowerShell, or other command line and run the following command for each FQDN returned from the previous step. Each time you run the command, verify that the IP address returned matches the IP address listed in the portal for the FQDN:

```
nslookup <fqdn>
```

For example, running the command

`nslookup 29395bb6-8bdb-4737-bf06-848a6857793f.workspace.eastus.api.azureml.ms` would return a value similar to the following text:

```
Server: yourdnsserver
Address: yourdnsserver-IP-address

Name:   29395bb6-8bdb-4737-bf06-848a6857793f.workspace.eastus.api.azureml.ms
Address: 10.3.0.5
```

4. If the `nslookup` command returns an error, or returns a different IP address than displayed in the portal, then the custom DNS solution isn't configured correctly. For more information, see [How to use your workspace with a custom DNS server](#)

Azure DNS troubleshooting

When using Azure DNS for name resolution, use the following steps to verify that the Private DNS integration is configured correctly:

1. On the Private Endpoint, select **DNS configuration**. For each entry in the **Private DNS zone** column, there should also be an entry in the **DNS zone group** column.

Network Interface	IP addresses	FQDN
mymlworkspace-pe-0...	10.3.0.5	29395bb6-8bdb-4737-bf06-848a6857793f.works...
	10.3.0.6	29395bb6-8bdb-4737-bf06-848a6857793f.works...
	10.3.0.7	ml-mymlworkspace-southcentralus-29395bb6-8b...
		*.29395bb6-8bdb-4737-bf06-848a6857793f.infer...

IP address	Subscription	Private DNS zone	DNS zone group
6-8bdb-4737...	ML-docs	-	-
10.3.0.5		-	-
6-8bdb-4737...		-	-
10.3.0.5		-	-
ib6-8bdb-473...		-	-
10.3.0.7		-	-
	ML-docs	-	-
lworkspace-so...		-	-
10.3.0.6		-	-

- If there's a Private DNS zone entry, but no **DNS zone group** entry, delete and recreate the Private Endpoint. When recreating the private endpoint, enable **Private DNS zone integration**.
- If **DNS zone group** isn't empty, select the link for the **Private DNS zone** entry.

From the Private DNS zone, select **Virtual network links**. There should be a link to the VNet. If there isn't one, then delete and recreate the private endpoint. When recreating it, select a Private DNS Zone linked to the VNet or create a new one that is linked to it.

Link Name	Link status	Virtual network	Auto-Registration
4rwwm7pmtl624	Completed	ml-vnet-scentral	Disabled

2. Repeat the previous steps for the rest of the Private DNS zone entries.

Browser configuration (DNS over HTTPS)

Check if DNS over HTTP is enabled in your web browser. DNS over HTTP can prevent Azure DNS from responding with the IP address of the Private Endpoint.

- Mozilla Firefox: For more information, see [Disable DNS over HTTPS in Firefox](#).

- Microsoft Edge:
 1. Search for DNS in Microsoft Edge settings: image.png
 2. Disable **Use secure DNS to specify how to look up the network address for websites.**

Proxy configuration

If you use a proxy, it may prevent communication with a secured workspace. To test, use one of the following options:

- Temporarily disable the proxy setting and see if you can connect.
- Create a [Proxy auto-config \(PAC\)](#) file that allows direct access to the FQDNs listed on the private endpoint. It should also allow direct access to the FQDN for any compute instances.
- Configure your proxy server to forward DNS requests to Azure DNS.

Interactive debugging with Visual Studio Code

9/22/2022 • 16 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azureml v1

Learn how to interactively debug Azure Machine Learning experiments, pipelines, and deployments using Visual Studio Code (VS Code) and [debugpy](#).

Run and debug experiments locally

Use the Azure Machine Learning extension to validate, run, and debug your machine learning experiments before submitting them to the cloud.

Prerequisites

- Azure Machine Learning VS Code extension (preview). For more information, see [Set up Azure Machine Learning VS Code extension](#).

IMPORTANT

The Azure Machine Learning VS Code extension uses the CLI (v2) by default. The instructions in this guide use 1.0 CLI. To switch to the 1.0 CLI, set the `azureML.CLI Compatibility Mode` setting in Visual Studio Code to `1.0`.

For more information on modifying your settings in Visual Studio Code, see the [user and workspace settings documentation](#).

- Docker
 - Docker Desktop for Mac and Windows
 - Docker Engine for Linux.

NOTE

On Windows, make sure to [configure Docker to use Linux containers](#).

TIP

For Windows, although not required, it's highly recommended to [use Docker with Windows Subsystem for Linux \(WSL\) 2](#).

- Python 3

Debug experiment locally

IMPORTANT

Before running your experiment locally make sure that:

- Docker is running.
- The `azureML.CLI Compatibility Mode` setting in Visual Studio Code is set to `1.0` as specified in the prerequisites

1. In VS Code, open the Azure Machine Learning extension view.

2. Expand the subscription node containing your workspace. If you don't already have one, you can [create an Azure Machine Learning workspace](#) using the extension.
3. Expand your workspace node.
4. Right-click the **Experiments** node and select **Create experiment**. When the prompt appears, provide a name for your experiment.
5. Expand the **Experiments** node, right-click the experiment you want to run and select **Run Experiment**.
6. From the list of options, select **Locally**.
7. **First time use on Windows only.** When prompted to allow File Share, select **Yes**. When you enable file share, it allows Docker to mount the directory containing your script to the container. Additionally, it also allows Docker to store the logs and outputs from your run in a temporary directory on your system.
8. Select **Yes** to debug your experiment. Otherwise, select **No**. Selecting no will run your experiment locally without attaching to the debugger.
9. Select **Create new Run Configuration** to create your run configuration. The run configuration defines the script you want to run, dependencies, and datasets used. Alternatively, if you already have one, select it from the dropdown.
 - a. Choose your environment. You can choose from any of the [Azure Machine Learning curated](#) or create your own.
 - b. Provide the name of the script you want to run. The path is relative to the directory opened in VS Code.
 - c. Choose whether you want to use an Azure Machine Learning dataset or not. You can create [Azure Machine Learning datasets](#) using the extension.
 - d. Debugpy is required in order to attach the debugger to the container running your experiment. To add debugpy as a dependency, select **Add Debugpy**. Otherwise, select **Skip**. Not adding debugpy as a dependency runs your experiment without attaching to the debugger.
 - e. A configuration file containing your run configuration settings opens in the editor. If you're satisfied with the settings, select **Submit experiment**. Alternatively, you open the command palette (**View > Command Palette**) from the menu bar and enter the `Azure ML: Submit experiment` command into the text box.
10. Once your experiment is submitted, a Docker image containing your script and the configurations specified in your run configuration is created.

When the Docker image build process begins, the contents of the `60_control_log.txt` file stream to the output console in VS Code.

NOTE

The first time your Docker image is created can take several minutes.

11. Once your image is built, a prompt appears to start the debugger. Set your breakpoints in your script and select **Start debugger** when you're ready to start debugging. Doing so attaches the VS Code debugger to the container running your experiment. Alternatively, in the Azure Machine Learning extension, hover over the node for your current run and select the play icon to start the debugger.

IMPORTANT

You cannot have multiple debug sessions for a single experiment. You can however debug two or more experiments using multiple VS Code instances.

At this point, you should be able to step-through and debug your code using VS Code.

If at any point you want to cancel your run, right-click your run node and select **Cancel run**.

Similar to remote experiment runs, you can expand your run node to inspect the logs and outputs.

TIP

Docker images that use the same dependencies defined in your environment are reused between runs. However, if you run an experiment using a new or different environment, a new image is created. Since these images are saved to your local storage, it's recommended to remove old or unused Docker images. To remove images from your system, use the [Docker CLI](#) or the [VS Code Docker extension](#).

Debug and troubleshoot machine learning pipelines

In some cases, you may need to interactively debug the Python code used in your ML pipeline. By using VS Code and debugpy, you can attach to the code as it runs in the training environment.

Prerequisites

- An **Azure Machine Learning workspace** that is configured to use an **Azure Virtual Network**.
- An **Azure Machine Learning pipeline** that uses Python scripts as part of the pipeline steps. For example, a `PythonScriptStep`.
- An Azure Machine Learning Compute cluster, which is **in the virtual network** and is **used by the pipeline for training**.
- A **development environment** that is **in the virtual network**. The development environment might be one of the following:
 - An Azure Virtual Machine in the virtual network
 - A Compute instance of Notebook VM in the virtual network
 - A client machine that has private network connectivity to the virtual network, either by VPN or via ExpressRoute.

For more information on using an Azure Virtual Network with Azure Machine Learning, see [Virtual network isolation and privacy overview](#).

TIP

Although you can work with Azure Machine Learning resources that are not behind a virtual network, using a virtual network is recommended.

How it works

Your ML pipeline steps run Python scripts. These scripts are modified to perform the following actions:

1. Log the IP address of the host that they're running on. You use the IP address to connect the debugger to the script.
2. Start the debugpy debug component, and wait for a debugger to connect.
3. From your development environment, you monitor the logs created by the training process to find the IP address where the script is running.
4. You tell VS Code the IP address to connect the debugger to by using a `launch.json` file.
5. You attach the debugger and interactively step through the script.

Configure Python scripts

To enable debugging, make the following changes to the Python script(s) used by steps in your ML pipeline:

1. Add the following import statements:

```
import argparse
import os
import debugpy
import socket
from azureml.core import Run
```

2. Add the following arguments. These arguments allow you to enable the debugger as needed, and set the timeout for attaching the debugger:

```
parser.add_argument('--remote_debug', action='store_true')
parser.add_argument('--remote_debug_connection_timeout', type=int,
                    default=300,
                    help=f'Defines how much time the AzureML compute target '
                         f'will await a connection from a debugger client (VS CODE).')
parser.add_argument('--remote_debug_client_ip', type=str,
                    help=f'Defines IP Address of VS Code client')
parser.add_argument('--remote_debug_port', type=int,
                    default=5678,
                    help=f'Defines Port of VS Code client')
```

3. Add the following statements. These statements load the current run context so that you can log the IP address of the node that the code is running on:

```
global run
run = Run.get_context()
```

4. Add an `if` statement that starts debugpy and waits for a debugger to attach. If no debugger attaches before the timeout, the script continues as normal. Make sure to replace the `HOST` and `PORT` values in the `listen` function with your own.

```
if args.remote_debug:
    print(f'Timeout for debug connection: {args.remote_debug_connection_timeout}')
    # Log the IP and port
    try:
        ip = args.remote_debug_client_ip
    except:
        print("Need to supply IP address for VS Code client")
    print(f'ip_address: {ip}')
    debugpy.listen(address=(ip, args.remote_debug_port))
    # Wait for the timeout for debugger to attach
    debugpy.wait_for_client()
    print(f'Debugger attached = {debugpy.is_client_connected()}')
```

The following Python example shows a simple `train.py` file that enables debugging:

```

# Copyright (c) Microsoft. All rights reserved.
# Licensed under the MIT license.

import argparse
import os
import debugpy
import socket
from azureml.core import Run

print("In train.py")
print("As a data scientist, this is where I use my training code.")

parser = argparse.ArgumentParser("train")

parser.add_argument("--input_data", type=str, help="input data")
parser.add_argument("--output_train", type=str, help="output_train directory")

# Argument check for remote debugging
parser.add_argument('--remote_debug', action='store_true')
parser.add_argument('--remote_debug_connection_timeout', type=int,
                    default=300,
                    help=f'Defines how much time the AzureML compute target '
                         f'will await a connection from a debugger client (VS CODE).')
parser.add_argument('--remote_debug_client_ip', type=str,
                    help=f'Defines IP Address of VS Code client')
parser.add_argument('--remote_debug_port', type=int,
                    default=5678,
                    help=f'Defines Port of VS Code client')

# Get run object, so we can find and log the IP of the host instance
global run
run = Run.get_context()

args = parser.parse_args()

# Start debugger if remote_debug is enabled
if args.remote_debug:
    print(f'Timeout for debug connection: {args.remote_debug_connection_timeout}')
    # Log the IP and port
    ip = socket.gethostbyname(socket.gethostname())
    # try:
    #     ip = args.remote_debug_client_ip
    # except:
    #     print("Need to supply IP address for VS Code client")
    print(f'ip_address: {ip}')
    debugpy.listen(address=(ip, args.remote_debug_port))
    # Wait for the timeout for debugger to attach
    debugpy.wait_for_client()
    print(f'Debugger attached = {debugpy.is_client_connected()}')

print("Argument 1: %s" % args.input_data)
print("Argument 2: %s" % args.output_train)

if not (args.output_train is None):
    os.makedirs(args.output_train, exist_ok=True)
    print("%s created" % args.output_train)

```

Configure ML pipeline

To provide the Python packages needed to start debugpy and get the run context, create an environment and set `pip_packages=['debugpy', 'azureml-sdk==<SDK-VERSION>']`. Change the SDK version to match the one you're using. The following code snippet demonstrates how to create an environment:

```

# Use a RunConfiguration to specify some additional requirements for this step.
from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core.runconfig import DEFAULT_CPU_IMAGE

# create a new runconfig object
run_config = RunConfiguration()

# enable Docker
run_config.environment.docker.enabled = True

# set Docker base image to the default CPU-based image
run_config.environment.docker.base_image = DEFAULT_CPU_IMAGE

# use conda_dependencies.yml to create a conda environment in the Docker image for execution
run_config.environment.python.user_managed_dependencies = False

# specify CondaDependencies obj
run_config.environment.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'],
                                                               pip_packages=['debugpy',
                                                               'azureml-sdk==<SDK-VERSION>'])

```

In the [Configure Python scripts](#) section, new arguments were added to the scripts used by your ML pipeline steps. The following code snippet demonstrates how to use these arguments to enable debugging for the component and set a timeout. It also demonstrates how to use the environment created earlier by setting

```
runconfig=run_config :
```

```

# Use RunConfig from a pipeline step
step1 = PythonScriptStep(name="train_step",
                         script_name="train.py",
                         arguments=['--remote_debug', '--remote_debug_connection_timeout', 300, '--remote_debug_client_ip', '<VS-CODE-CLIENT-IP>', '--remote_debug_port', 5678],
                         compute_target=aml_compute,
                         source_directory=source_directory,
                         runconfig=run_config,
                         allow_reuse=False)

```

When the pipeline runs, each step creates a child run. If debugging is enabled, the modified script logs information similar to the following text in the `70_driver_log.txt` for the child run:

```
Timeout for debug connection: 300
ip_address: 10.3.0.5
```

Save the `ip_address` value. It's used in the next section.

TIP

You can also find the IP address from the run logs for the child run for this pipeline step. For more information on viewing this information, see [Monitor Azure ML experiment runs and metrics](#).

Configure development environment

1. To install debugpy on your VS Code development environment, use the following command:

```
python -m pip install --upgrade debugpy
```

For more information on using debugpy with VS Code, see [Remote Debugging](#).

2. To configure VS Code to communicate with the Azure Machine Learning compute that is running the debugger, create a new debug configuration:

- a. From VS Code, select the **Debug** menu and then select **Open configurations**. A file named `launch.json` opens.
- b. In the `launch.json` file, find the line that contains `"configurations": [`, and insert the following text after it. Change the `"host": "<IP-ADDRESS>"` entry to the IP address returned in your logs from the previous section. Change the `"localRoot": "${workspaceFolder}/code/step1"` entry to a local directory that contains a copy of the script being debugged:

```
{  
    "name": "Azure Machine Learning Compute: remote debug",  
    "type": "python",  
    "request": "attach",  
    "port": 5678,  
    "host": "<IP-ADDRESS>",  
    "redirectOutput": true,  
    "pathMappings": [  
        {  
            "localRoot": "${workspaceFolder}/code/step1",  
            "remoteRoot": "."  
        }  
    ]  
}
```

IMPORTANT

If there are already other entries in the configurations section, add a comma (,) after the code that you inserted.

TIP

The best practice, especially for pipelines is to keep the resources for scripts in separate directories so that code is relevant only for each of the steps. In this example the `localRoot` example value references `/code/step1`.

If you are debugging multiple scripts, in different directories, create a separate configuration section for each script.

- c. Save the `launch.json` file.

Connect the debugger

1. Open VS Code and open a local copy of the script.
2. Set breakpoints where you want the script to stop once you've attached.
3. While the child process is running the script, and the `Timeout for debug connection` is displayed in the logs, use the F5 key or select **Debug**. When prompted, select the **Azure Machine Learning Compute: remote debug** configuration. You can also select the debug icon from the side bar, the **Azure Machine Learning: remote debug** entry from the Debug dropdown menu, and then use the green arrow to attach the debugger.

At this point, VS Code connects to debugpy on the compute node and stops at the breakpoint you set previously. You can now step through the code as it runs, view variables, etc.

NOTE

If the log displays an entry stating `Debugger attached = False`, then the timeout has expired and the script continued without the debugger. Submit the pipeline again and connect the debugger after the `Timeout for debug connection` message, and before the timeout expires.

Debug and troubleshoot deployments

In some cases, you may need to interactively debug the Python code contained in your model deployment. For example, if the entry script is failing and the reason can't be determined by extra logging. By using VS Code and the debugpy, you can attach to the code running inside the Docker container.

TIP

Save time and catch bugs early by debugging managed online endpoints and deployments locally. For more information, see [Debug managed online endpoints locally in Visual Studio Code \(preview\)](#).

IMPORTANT

This method of debugging does not work when using `Model.deploy()` and `LocalWebservice.deploy_configuration` to deploy a model locally. Instead, you must create an image using the `Model.package()` method.

Local web service deployments require a working Docker installation on your local system. For more information on using Docker, see the [Docker Documentation](#). When working with compute instances, Docker is already installed.

Configure development environment

1. To install debugpy on your local VS Code development environment, use the following command:

```
python -m pip install --upgrade debugpy
```

For more information on using debugpy with VS Code, see [Remote Debugging](#).

2. To configure VS Code to communicate with the Docker image, create a new debug configuration:

- a. From VS Code, select the **Debug** menu in the **Run** extention and then select **Open configurations**. A file named `launch.json` opens.
- b. In the `launch.json` file, find the "**configurations**" item (the line that contains `"configurations": []`), and insert the following text after it.

```
{
    "name": "Azure Machine Learning Deployment: Docker Debug",
    "type": "python",
    "request": "attach",
    "connect": {
        "port": 5678,
        "host": "0.0.0.0",
    },
    "pathMappings": [
        {
            "localRoot": "${workspaceFolder}",
            "remoteRoot": "/var/azureml-app"
        }
    ]
}
```

After insertion, the `launch.json` file should be similar to the following:

```
{
// Use IntelliSense to learn about possible attributes.
// Hover to view descriptions of existing attributes.
// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
"version": "0.2.0",
"configurations": [
{
    "name": "Python: Current File",
    "type": "python",
    "request": "launch",
    "program": "${file}",
    "console": "integratedTerminal"
},
{
    "name": "Azure Machine Learning Deployment: Docker Debug",
    "type": "python",
    "request": "attach",
    "connect": {
        "port": 5678,
        "host": "0.0.0.0"
    },
    "pathMappings": [
        {
            "localRoot": "${workspaceFolder}",
            "remoteRoot": "/var/azureml-app"
        }
    ]
}
]
```

IMPORTANT

If there are already other entries in the configurations section, add a comma (,) after the code that you inserted.

This section attaches to the Docker container using port **5678**.

- c. Save the `launch.json` file.

Create an image that includes debugpy

1. Modify the conda environment for your deployment so that it includes debugpy. The following example demonstrates adding it using the `pip_packages` parameter:

```

from azureml.core.conda_dependencies import CondaDependencies

# Usually a good idea to choose specific version numbers
# so training is made on same packages as scoring
myenv = CondaDependencies.create(conda_packages=['numpy==1.15.4',
                                                'scikit-learn==0.19.1', 'pandas==0.23.4'],
                                 pip_packages = ['azureml-defaults==1.0.83', 'debugppy'])

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())

```

2. To start debugppy and wait for a connection when the service starts, add the following to the top of your `score.py` file:

```

import debugppy
# Allows other computers to attach to debugppy on this IP address and port.
debugppy.listen(('0.0.0.0', 5678))
# Wait 30 seconds for a debugger to attach. If none attaches, the script continues as normal.
debugppy.wait_for_client()
print("Debugger attached...")

```

3. Create an image based on the environment definition and pull the image to the local registry.

NOTE

This example assumes that `ws` points to your Azure Machine Learning workspace, and that `model` is the model being deployed. The `myenv.yml` file contains the conda dependencies created in step 1.

```

from azureml.core.conda_dependencies import CondaDependencies
from azureml.core.model import InferenceConfig
from azureml.core.environment import Environment

myenv = Environment.from_conda_specification(name="env", file_path="myenv.yml")
myenv.docker.base_image = None
myenv.docker.base_dockerfile = "FROM mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210615.v1"
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)
package = Model.package(ws, [model], inference_config)
package.wait_for_creation(show_output=True) # Or show_output=False to hide the Docker build logs.
package.pull()

```

Once the image has been created and downloaded (this process may take more than 10 minutes), the image path (includes repository, name, and tag, which in this case is also its digest) is finally displayed in a message similar to the following:

```
Status: Downloaded newer image for myregistry.azurecr.io/package@sha256:<image-digest>
```

4. To make it easier to work with the image locally, you can use the following command to add a tag for this image. Replace `myimagepath` in the following command with the location value from the previous step.

```
docker tag myimagepath debug:1
```

For the rest of the steps, you can refer to the local image as `debug:1` instead of the full image path value.

Debug the service

TIP

If you set a timeout for the debugpy connection in the `score.py` file, you must connect VS Code to the debug session before the timeout expires. Start VS Code, open the local copy of `score.py`, set a breakpoint, and have it ready to go before using the steps in this section.

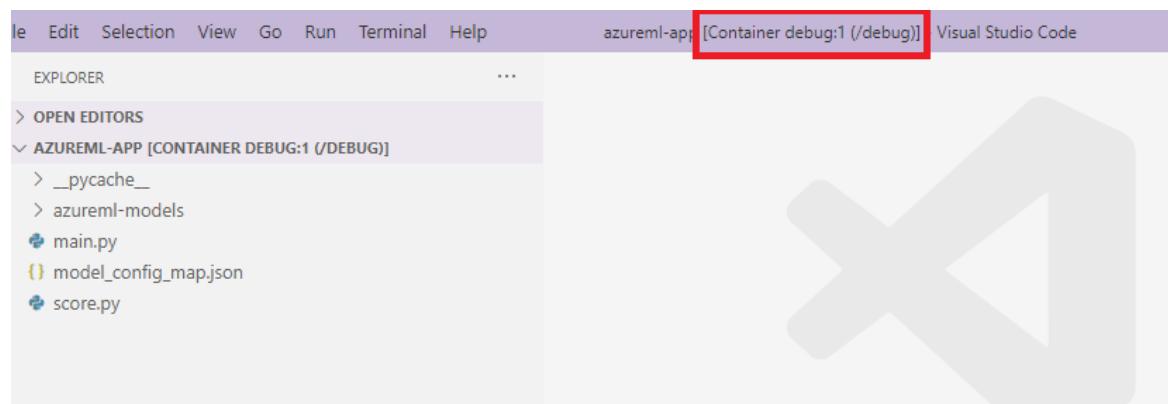
For more information on debugging and setting breakpoints, see [Debugging](#).

1. To start a Docker container using the image, use the following command:

```
docker run -it --name debug -p 8000:5001 -p 5678:5678 -v <my_local_path_to_score.py>:/var/azureml-app/score.py debug:1 /bin/bash
```

This command attaches your `score.py` locally to the one in the container. Therefore, any changes made in the editor are automatically reflected in the container

2. For a better experience, you can go into the container with a new VS code interface. Select the `Docker` extention from the VS Code side bar, find your local container created, in this documentation its `debug:1`. Right-click this container and select `"Attach Visual Studio Code"`, then a new VS Code interface will be opened automatically, and this interface shows the inside of your created container.



3. Inside the container, run the following command in the shell

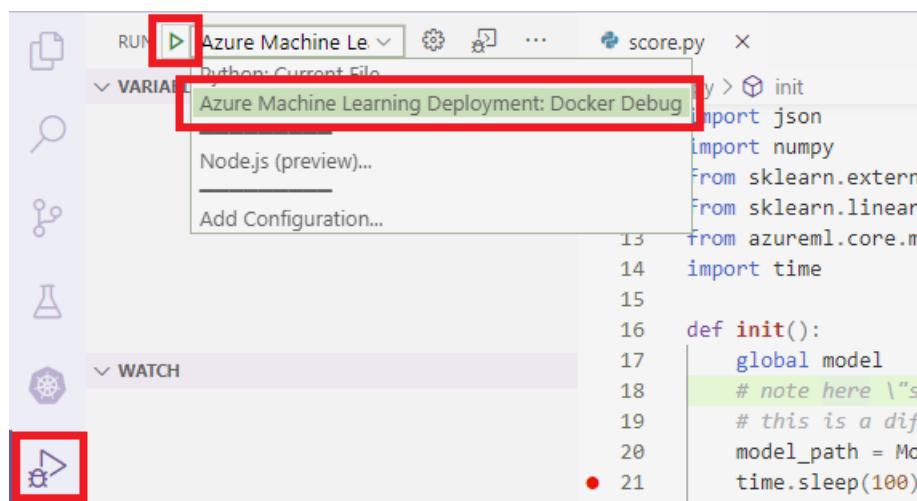
```
runsvdir /var/runit
```

Then you can see the following output in the shell inside your container:

```
PROBLEMS TERMINAL ... 1: runsvdir + □ □ □ ^ X

/usr/sbin/nginx: /azureml-envs/azureml_77aec60318de3a01d6bbfc3a161369ce/lib/lib
crypto.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_77aec60318de3a01d6bbfc3a161369ce/lib/lib
crypto.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_77aec60318de3a01d6bbfc3a161369ce/lib/lib
ssl.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_77aec60318de3a01d6bbfc3a161369ce/lib/lib
ssl.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_77aec60318de3a01d6bbfc3a161369ce/lib/lib
ssl.so.1.0.0: no version information available (required by /usr/sbin/nginx)
EdgeHubConnectionString and IOTEDGE_IOTHUBHOSTNAME are not set. Exiting...
2020-12-02T03:45:49,841665600+00:00 - iot-server/finish 1 0
2020-12-02T03:45:49,842827100+00:00 - Exit code 1 is normal. Not restarting iot
-server.
Starting gunicorn 19.9.0
Listening at: http://127.0.0.1:31311 (31287)
Using worker: sync
worker timeout is set to 300
Booting worker with pid: 31325
Initialized PySpark session.
```

- To attach VS Code to debug your code inside the container, open VS Code, and use the F5 key or select **Debug**. When prompted, select the **Azure Machine Learning Deployment: Docker Debug** configuration. You can also select the **Run** extension icon from the side bar, the **Azure Machine Learning Deployment: Docker Debug** entry from the Debug dropdown menu, and then use the green arrow to attach the debugger.



After you select the green arrow and attach the debugger, in the container VS Code interface you can see some new information:

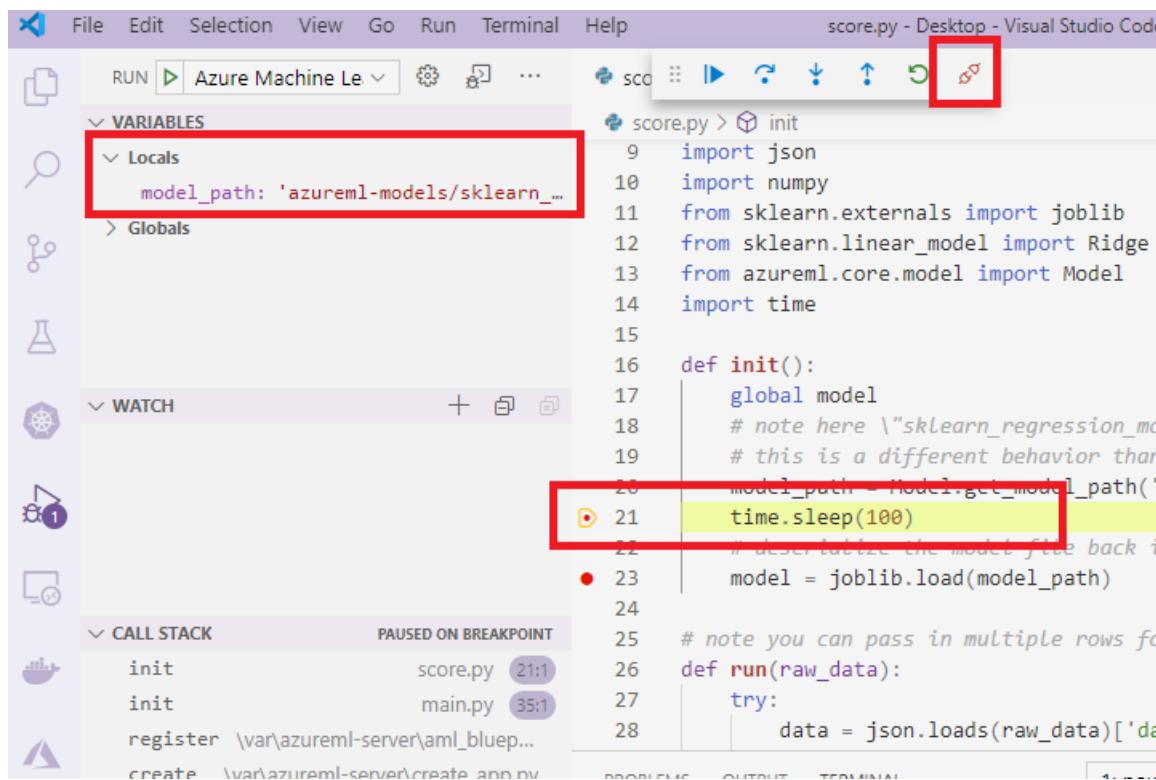
PROBLEMS TERMINAL ... 1: runsvdir + ⌂ ⌂ ⌂ ⌂

```
/usr/sbin/nginx: /azureml-envs/azureml_77aec60318de3a01d6bbfc3a161369ce/lib/libssl.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_77aec60318de3a01d6bbfc3a161369ce/lib/libssl.so.1.0.0: no version information available (required by /usr/sbin/nginx)
EdgeHubConnectionString and IOTEDGE_IOTHUBHOSTNAME are not set. Exiting...
2020-12-02T03:45:49,841665600+00:00 - iot-server/finish 1 0
2020-12-02T03:45:49,842827100+00:00 - Exit code 1 is normal. Not restarting iot-server.
Starting gunicorn 19.9.0
Listening at: http://127.0.0.1:31311 (31287)
Using worker: sync
worker timeout is set to 300
Booting worker with pid: 31325
Initialized PySpark session.
Debugger attached...
Initialising logger...
Starting up app insights client
Starting up request id generator
Starting up app insight hooks
Invoking user's init function

```

: virtualenv ⚡ 0 ⚡ 0 ⚡ 3 Ports Available ⌂ ⌂

Also, in your main VS Code interface, what you can see is following:



And now, the local `score.py` which is attached to the container has already stopped at the breakpoints where you set. At this point, VS Code connects to debugpy inside the Docker container and stops the Docker container at the breakpoint you set previously. You can now step through the code as it runs, view variables, etc.

For more information on using VS Code to debug Python, see [Debug your Python code](#).

Stop the container

To stop the container, use the following command:

```
docker stop debug
```

Next steps

Now that you've set up VS Code Remote, you can use a compute instance as remote compute from VS Code to interactively debug your code.

Learn more about troubleshooting:

- [Local model deployment](#)
- [Remote model deployment](#)
- [Machine learning pipelines](#)
- [ParallelRunStep](#)

Troubleshoot "cannot import name 'SerializationError'"

9/22/2022 • 2 minutes to read • [Edit Online](#)

When using Azure Machine Learning, you may receive one of the following errors:

- `cannot import name 'SerializationError'`
- `cannot import name 'SerializationError' from 'azure.core.exceptions'`

This error may occur when using an Azure Machine Learning environment. For example, when submitting a training job or using AutoML.

Cause

This problem is caused by a bug in the Azure Machine Learning SDK version 1.42.0.

Resolution

Update the affected environment to use SDK version 1.42.0.post1 or greater. For a local development environment or compute instance, use the following command:

```
pip install azureml-sdk[automl,explain,notebooks]>=1.42.0
```

For more information on updating an Azure Machine Learning environment (for training or deployment), see the following articles:

- [Manage environments in studio](#)
- [Create & use software environments \(SDK v1\)](#)
- [Create & manage environments \(CLI v2\)](#)

To verify the version of your installed SDK, use the following command:

```
pip show azureml-core
```

Next steps

For more information on updating an Azure Machine Learning environment (for training or deployment), see the following articles:

- [Manage environments in studio](#)
- [Create & use software environments \(SDK v1\)](#)
- [Create & manage environments \(CLI v2\)](#)

Troubleshoot

descriptors cannot not be created directly error

9/22/2022 • 2 minutes to read • [Edit Online](#)

When using Azure Machine Learning, you may receive the following error:

TypeError: Descriptors cannot not be created directly. If this call came from a _pb2.py file, your generated code is out of date and must be regenerated with protoc >= 3.19.0.” It is followed by the proposition to install the appropriate version of protobuf library.

If you cannot immediately regenerate your protos, some other possible workarounds are:

1. Downgrade the protobuf package to 3.20.x or lower.
2. Set PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=python (but this will use pure-Python parsing and will be much slower).

You may notice this error specifically when using AutoML.

Cause

This problem is caused by breaking changes introduced in protobuf 4.0.0. For more information, see <https://developers.google.com/protocol-buffers/docs/news/2022-05-06#python-updates>.

Resolution

For a local development environment or compute instance, install the Azure Machine Learning SDK version 1.42.0.post1 or greater.

```
pip install azureml-sdk[automl,explain,notebooks]>=1.42.0
```

For more information on updating an Azure Machine Learning environment (for training or deployment), see the following articles:

- [Manage environments in studio](#)
- [Create & use software environments \(SDK v1\)](#)
- [Create & manage environments \(CLI v2\)](#)

To verify the version of your installed SDK, use the following command:

```
pip show azureml-core
```

This command should return information similar to `Version: 1.42.0.post1`.

TIP

If you can't upgrade your Azure Machine Learning SDK installation, you can pin the protobuf version in your environment to 3.20.1. The following example is a `conda.yml` file that demonstrates how to pin the version:

```
name: model-env
channels:
  - conda-forge
dependencies:
  - python=3.8
  - numpy=1.21.2
  - pip=21.2.4
  - scikit-learn=0.24.2
  - scipy=1.7.1
  - pandas>=1.1,<1.2
  - pip:
    - inference-schema[numpy-support]==1.3.0
    - xlrd==2.0.1
    - mlflow== 1.26.0
    - azureml-mlflow==1.41.0
    - protobuf==3.20.1
```

Next steps

For more information on the breaking changes in protobuf 4.0.0, see <https://developers.google.com/protocol-buffers/docs/news/2022-05-06#python-updates>.

For more information on updating an Azure Machine Learning environment (for training or deployment), see the following articles:

- [Manage environments in studio](#)
- [Create & use software environments \(SDK v1\)](#)
- [Create & manage environments \(CLI v2\)](#)

Troubleshoot data access errors

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  Python SDK azure-ai-ml v2 (preview)

In this guide, learn how to identify and resolve known issues with data access with the [Azure Machine Learning SDK](#).

Error Codes

Data access error codes are hierarchical. Error codes are delimited by the full stop character `.` and are more specific the more segments there are.

ScriptExecution.DatabaseConnection

ScriptExecution.DatabaseConnection.NotFound

The database or server defined in the datastore couldn't be found or no longer exists. Check if the database still exists in Azure portal or linked to from the Azure Machine Learning studio datastore details page. If it doesn't exist, recreating it with the same name will enable the existing datastore to be used. If a new server name or database is used, the datastore will have to be deleted, and recreated to use the new name.

ScriptExecution.DatabaseConnection.Authentication

The authentication failed while trying to connect to the database. The authentication method is stored inside the datastore and supports SQL authentication, service principal, or no stored credential (identity based access). Enabling workspace MSI makes the authentication use the workspace MSI when previewing data in Azure Machine Learning studio. A SQL server user needs to be created for the service principal and workspace MSI (if applicable) and granted classic database permissions. More info can be found [here](#).

Contact your data admin to verify or add the correct permissions to the service principal or user identity.

Errors also include:

- ScriptExecution.DatabaseConnection.Authentication.AzureIdentityAccessTokenResolution.InvalidResource
 - The server under the subscription and resource group couldn't be found. Check that the subscription ID and resource group defined in the datastore matches that of the server and update the values if needed.

NOTE

Use the subscription ID and resource group of the server and not of the workspace. If the datastore is cross subscription or cross resource group server, these will be different.

- ScriptExecution.DatabaseConnection.Authentication.AzureIdentityAccessTokenResolution.FirewallSettingsResolutionFailure
 - The identity doesn't have permission to read firewall settings of the target server. Contact your data admin to the Reader role to the workspace MSI.

ScriptExecution.DatabaseQuery

ScriptExecution.DatabaseQuery.TimeoutExpired

The executed SQL query took too long and timed out. The timeout can be specified at time of data asset creation. If a new timeout is needed, a new asset must be created, or a new version of the current asset must be

created. In Azure Machine Learning studio SQL preview, there will have a fixed query timeout, but the defined value will always be honored for jobs.

ScriptExecution.StreamAccess

ScriptExecution.StreamAccess.Authentication

The authentication failed while trying to connect to the storage account. The authentication method is stored inside the datastore and depending on the datastore type, can support account key, SAS token, service principal or no stored credential (identity based access). Enabling workspace MSI makes the authentication use the workspace MSI when previewing data in Azure Machine Learning studio.

Contact your data admin to verify or add the correct permissions to the service principal or user identity.

IMPORTANT

If identity based access is used, the required RBAC role is Storage Blob Data Reader. If workspace MSI is used for Azure Machine Learning studio preview, the required RBAC roles are Storage Blob Data Reader and Reader.

Errors also include:

- ScriptExecution.StreamAccess.Authentication.AzureIdentityAccessTokenResolution.FirewallSettingsResolutionFailure
 - The identity doesn't have permission to read firewall settings of the target storage account. Contact your data admin to the Reader role to the workspace MSI.
- ScriptExecution.StreamAccess.Authentication.AzureIdentityAccessTokenResolution.PrivateEndpointResolutionFailure
 - The target storage account is using a virtual network but the logged in session isn't connecting to the workspace via a private endpoint. Add a private endpoint to the workspace and ensure that the virtual network or subnet of the private endpoint is allowed by the storage virtual network settings. Add the logged in session's public IP to the storage firewall allowlist.
- ScriptExecution.StreamAccess.Authentication.AzureIdentityAccessTokenResolution.NetworkIsolationViolated
 - The target storage account's firewall settings don't permit this data access. Check that your logged in session is within compatible network settings with the storage account. If Workspace MSI is used, check that it has Reader access to the storage account and to the private endpoints associated with the storage account.
- ScriptExecution.StreamAccess.Authentication.AzureIdentityAccessTokenResolution.InvalidResource
 - The storage account under the subscription and resource group couldn't be found. Check that the subscription ID and resource group defined in the datastore matches that of the storage account and update the values if needed.

NOTE

Use the subscription ID and resource group of the server and not of the workspace. If the datastore is cross subscription or cross resource group server, these will be different.

ScriptExecution.StreamAccess.NotFound

The specified file or folder path doesn't exist. Check that the provided path exists in Azure portal or if using a datastore, that the right datastore is used (including the datastore's account and container). If the storage account is an HNS enabled Blob storage, otherwise known as ADLS Gen2, or an `abfs[s]` URI, that storage ACLs may restrict particular folders or paths. This error will appear as a "NotFound" error instead of an "Authentication" error.

ScriptExecution.StreamAccess.Validation

There were validation errors in the request for data access.

Errors also include:

- ScriptExecution.StreamAccess.Validation.TextFile-InvalidEncoding
 - The defined encoding for delimited file parsing isn't applicable for the underlying data. Update the encoding of the MLTable to match the encoding of the file(s).
- ScriptExecution.StreamAccess.Validation.StorageRequest-InvalidUri
 - The requested URI isn't well formatted. We support `abfs[s]` , `wasb[s]` , `https` , and `azureml` URLs.

Next steps

- See more information on [data concepts in Azure Machine Learning](#)
- [Identity-based data access to storage services on Azure](#).
- [Read and write data in a job](#)

CLI (v2) YAML schemas

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The Azure Machine Learning CLI (v2), an extension to the Azure CLI, often uses and sometimes requires YAML files with specific schemas. This article lists reference docs and the source schema for YAML files. Examples are included inline in individual articles.

Workspace

REFERENCE	URI
Workspace	https://azureschemas.azureedge.net/latest/workspace.schema.json

Environment

REFERENCE	URI
Environment	https://azureschemas.azureedge.net/latest/environment.schema.json

Data

REFERENCE	URI
Dataset	https://azureschemas.azureedge.net/latest/data.schema.json

Model

REFERENCE	URI
Model	https://azureschemas.azureedge.net/latest/model.schema.json

Compute

REFERENCE	URI
Compute cluster (AmlCompute)	https://azureschemas.azureedge.net/latest/amlCompute.schema.json
Compute instance	https://azureschemas.azureedge.net/latest/computeInstance.schema.json
Attached Virtual Machine	https://azureschemas.azureedge.net/latest/vmCompute.schema.json
Attached Azure Arc-enabled Kubernetes (KubernetesCompute)	https://azureschemas.azureedge.net/latest/kubernetesCompute.schema.json

Job

REFERENCE	URI
Command	https://azureschemas.azureedge.net/latest/commandJob.schema.json
Sweep	https://azureschemas.azureedge.net/latest/sweepJob.schema.json
Pipeline	https://azureschemas.azureedge.net/latest/pipelineJob.schema.json

Datastore

REFERENCE	URI
Azure Blob	https://azureschemas.azureedge.net/latest/azureBlob.schema.json
Azure Files	https://azureschemas.azureedge.net/latest/azureFile.schema.json
Azure Data Lake Gen1	https://azureschemas.azureedge.net/latest/azureDataLakeGen1.schema.json
Azure Data Lake Gen2	https://azureschemas.azureedge.net/latest/azureDataLakeGen2.schema.json

Endpoint

REFERENCE	URI
Online (real-time)	https://azureschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
Batch	https://azureschemas.azureedge.net/latest/batchEndpoint.schema.json

Deployment

REFERENCE	URI
Managed online (real-time)	https://azureschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
Kubernetes online (real-time)	https://azureschemas.azureedge.net/latest/kubernetesOnlineDeployment.schema.json
Batch	https://azureschemas.azureedge.net/latest/batchDeployment.schema.json

Component

REFERENCE	URI

REFERENCE	URI
Command	https://azuremlschemas.azureedge.net/latest/commandComponent.schema.json

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) core YAML syntax

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

Every Azure Machine Learning entity has a schematized YAML representation. You can create a new entity from a YAML configuration file with a `.yml` or `.yaml` extension.

This article provides an overview of core syntax concepts you will encounter while configuring these YAML files.

Referencing an Azure ML entity

Azure ML provides a reference syntax (consisting of a shorthand and longhand format) for referencing an existing Azure ML entity when configuring a YAML file. For example, you can reference an existing registered environment in your workspace to use as the environment for a job.

Referencing an Azure ML asset

There are two options for referencing an Azure ML asset (environments, models, data, and components):

- Reference an explicit version of an asset:

- Shorthand syntax: `azureml:<asset_name>:<asset_version>`

- Longhand syntax, which includes the Azure Resource Manager (ARM) resource ID of the asset:

```
azureml:/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>/environments/<environment-name>/versions/<environment-version>
```

- Reference the latest version of an asset:

In some scenarios you may want to reference the latest version of an asset without having to explicitly look up and specify the actual version string itself. The latest version is defined as the latest (also known as most recently) created version of an asset under a given name.

You can reference the latest version using the following syntax: `azureml:<asset_name>@latest`. Azure ML will resolve the reference to an explicit asset version in the workspace.

Reference an Azure ML resource

To reference an Azure ML resource (such as compute), you can use either of the following syntaxes:

- Shorthand syntax: `azureml:<resource_name>`
- Longhand syntax, which includes the ARM resource ID of the resource:

```
azureml:/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>/computes/<compute-name>
```

Azure ML data reference URI

Azure ML offers a convenience data reference URI format to point to data in an Azure storage service. This can be used for scenarios where you need to specify a cloud storage location in your YAML file, such as creating an Azure ML model from file(s) in storage, or pointing to data to pass as input to a job.

To use this data URI format, the storage service you want to reference must first be registered as a datastore in your workspace. Azure ML will handle the data access using the credentials you provided during datastore creation.

The format consists of a datastore in the current workspace and the path on the datastore to the file or folder you want to point to:

```
azureml://datastores/<datastore-name>/paths/<path-on-datastore>/
```

For example:

- `azureml://datastores/workspaceblobstore/paths/example-data/`
- `azureml://datastores/workspaceblobstore/paths/example-data/iris.csv`

In addition to the Azure ML data reference URI, Azure ML also supports the following direct storage URI protocols: `https`, `wasbs`, `abfss`, and `adl`, as well as public `http` and `https` URLs.

Expression syntax for configuring Azure ML jobs and components

v2 job and component YAML files allow for the use of expressions to bind to contexts for different scenarios. The essential use case is using an expression for a value that might not be known at the time of authoring the configuration, but must be resolved at runtime.

Use the following syntax to tell Azure ML to evaluate an expression rather than treat it as a string:

```
${{ <expression> }}
```

The supported scenarios are covered below.

Parameterizing the `command` with the `inputs` and `outputs` contexts of a job

You can specify literal values, URI paths, and registered Azure ML data assets as inputs to a job. The `command` can then be parameterized with references to those input(s) using the `${{inputs.<input_name>}}` syntax. References to literal inputs will get resolved to the literal value at runtime, while references to data inputs will get resolved to the download path or mount path (depending on the `mode` specified).

Likewise, outputs to the job can also be referenced in the `command`. For each named output specified in the `outputs` dictionary, Azure ML will system-generate an output location on the default datastore where you can write files to. The output location for each named output is based on the following templated path:

`<default-datastore>/azureml/<job-name>/<output_name>/`. Parameterizing the `command` with the `${{outputs.<output_name>}}` syntax will resolve that reference to the system-generated path, so that your script can write files to that location from the job.

In the example below for a command job YAML file, the `command` is parameterized with two inputs, a literal input and a data input, and one output. At runtime, the `${{inputs.learning_rate}}` expression will resolve to `0.01`, and the `${{inputs.iris}}` expression will resolve to the download path of the `iris.csv` file. `${{outputs.model_dir}}` will resolve to the mount path of the system-generated output location corresponding to the `model_dir` output.

```

$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
code: ./src
command: python train.py --lr ${{inputs.learning_rate}} --training-data ${{inputs.iris}} --model-dir
${{outputs.model_dir}}
environment: azurerm:AzureML-Minimal@latest
compute: azurerm:cpu-cluster
inputs:
  learning_rate: 0.01
  iris:
    type: uri_file
    path: https://azurermexamples.blob.core.windows.net/datasets/iris.csv
    mode: download
outputs:
  model_dir:

```

Parameterizing the `command` with the `search_space` context of a sweep job

You will also use this expression syntax when performing hyperparameter tuning via a sweep job, since the actual values of the hyperparameters are not known during job authoring time. When you run a sweep job, Azure ML will select hyperparameter values for each trial based on the `search_space`. In order to access those values in your training script, you must pass them in via the script's command-line arguments. To do so, use the `${{search_space.<hyperparameter>}}` syntax in the `trial.command`.

In the example below for a sweep job YAML file, the `${{search_space.learning_rate}}` and `${{search_space.boosting}}` references in `trial.command` will resolve to the actual hyperparameter values selected for each trial when the trial job is submitted for execution.

```

$schema: https://azurermschemas.azureedge.net/latest/sweepJob.schema.json
type: sweep
sampling_algorithm:
  type: random
search_space:
  learning_rate:
    type: uniform
    min_value: 0.01
    max_value: 0.9
  boosting:
    type: choice
    values: ["gbdt", "dart"]
objective:
  goal: minimize
  primary_metric: test-multi_logloss
trial:
  code: ./src
  command: >-
    python train.py
    --training-data ${{inputs.iris}}
    --lr ${{search_space.learning_rate}}
    --boosting ${{search_space.boosting}}
  environment: azurerm:AzureML-Minimal@latest
inputs:
  iris:
    type: uri_file
    path: https://azurermexamples.blob.core.windows.net/datasets/iris.csv
    mode: download
compute: azurerm:cpu-cluster

```

Binding inputs and outputs between steps in a pipeline job

Expressions are also used for binding inputs and outputs between steps in a pipeline job. For example, you can bind the input of one job (job B) in a pipeline to the output of another job (job A). This usage will signal to Azure ML the dependency flow of the pipeline graph, and job B will get executed after job A, since the output of job A

is required as an input for job B.

For a pipeline job YAML file, the `inputs` and `outputs` sections of each child job are evaluated within the parent context (the top-level pipeline job). The `command`, on the other hand, will resolve to the current context (the child job).

There are two ways to bind inputs and outputs in a pipeline job:

Bind to the top-level inputs and outputs of the pipeline job

You can bind the inputs or outputs of a child job (a pipeline step) to the inputs/outputs of the top-level parent pipeline job using the following syntax: `${{parent.inputs.<input_name>}}` or `${{parent.outputs.<output_name>}}`. This reference resolves to the `parent` context; hence the top-level inputs/outputs.

In the example below, the input (`raw_data`) of the first `prep` step is bound to the top-level pipeline input via `${{parent.inputs.input_data}}`. The output (`model_dir`) of the final `train` step is bound to the top-level pipeline job output via `${{parent.outputs.trained_model}}`.

Bind to the inputs and outputs of another child job (step)

To bind the inputs/outputs of one step to the inputs/outputs of another step, use the following syntax:

`${{parent.jobs.<step_name>.inputs.<input_name>}}` or `${{parent.jobs.<step_name>.outputs.<outputs_name>}}`.

Again, this reference resolves to the parent context, so the expression must start with `parent.jobs.<step_name>`.

In the example below, the input (`training_data`) of the `train` step is bound to the output (`clean_data`) of the `prep` step via `${{parent.jobs.prep.outputs.clean_data}}`. The prepared data from the `prep` step will be used as the training data for the `train` step.

On the other hand, the context references within the `command` properties will resolve to the current context. For example, the `${{inputs.raw_data}}` reference in the `prep` step's `command` will resolve to the inputs of the current context, which is the `prep` child job. The lookup will be done on `prep.inputs`, so an input named `raw_data` must be defined there.

```

$schema: https://azurermschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
inputs:
  input_data:
    type: uri_folder
    path: https://azurermexamples.blob.core.windows.net/datasets/cifar10/
outputs:
  trained_model:
jobs:
  prep:
    type: command
    inputs:
      raw_data: ${parent.inputs.input_data}
    outputs:
      clean_data:
        code: src/prep
        environment: azureml:AzureML-Minimal@latest
        command: >-
          python prep.py
          --raw-data ${inputs.raw_data}
          --prep-data ${outputs.clean_data}
    compute: azureml:cpu-cluster
  train:
    type: command
    inputs:
      training_data: ${parent.jobs.prep.outputs.clean_data}
      num_epochs: 1000
    outputs:
      model_dir: ${parent.outputs.trained_model}
    code: src/train
    environment: azureml:AzureML-Minimal@latest
    command: >-
      python train.py
      --epochs ${inputs.num_epochs}
      --training-data ${inputs.training_data}
      --model-output ${outputs.model_dir}
    compute: azureml:gpu-cluster

```

Parameterizing the `command` with the `inputs` and `outputs` contexts of a component

Similar to the `command` for a job, the `command` for a component can also be parameterized with references to the `inputs` and `outputs` contexts. In this case the reference is to the component's inputs and outputs. When the component is run in a job, Azure ML will resolve those references to the job runtime input and output values specified for the respective component inputs and outputs. Below is an example of using the context syntax for a command component YAML specification.

```

$schema: https://azurermschemas.azureedge.net/latest/commandComponent.schema.json
name: train_data_component_cli
display_name: train_data
description: A example train component
tags:
  author: azureml-sdk-team
version: 5
type: command
inputs:
  training_data:
    type: uri_folder
  max_epochs:
    type: integer
    optional: true
  learning_rate:
    type: number
    default: 0.01
    optional: true
  learning_rate_schedule:
    type: string
    default: time-based
    optional: true
outputs:
  model_output:
    type: uri_folder
code: ./train_src
environment: azurerm:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:1
command: >-
  python train.py
  --training_data ${{inputs.training_data}}
  $[--max_epochs ${{inputs.max_epochs}}]
  $[--learning_rate ${{inputs.learning_rate}}]
  $[--learning_rate_schedule ${{inputs.learning_rate_schedule}}]
  --model_output ${{outputs.model_output}}

```

Define optional inputs in command line

When the input is set as `optional = true`, you need use `$[]` to embrace the command line with inputs. For example `$[--input1 ${{inputs.input1}}]`. The command line at runtime may have different inputs.

- If you are using only specify the required `training_data` and `model_output` parameters, the command line will look like:

```
python train.py --training_data some_input_path --learning_rate 0.01 --learning_rate_schedule time-based --model_output some_output_path
```

If no value is specified at runtime, `learning_rate` and `learning_rate_schedule` will use the default value.

- If all inputs/outputs provide values during runtime, the command line will look like:

```
python train.py --training_data some_input_path --max_epochs 10 --learning_rate 0.01 --learning_rate_schedule time-based --model_output some_output_path
```

Next steps

- [Install and use the CLI \(v2\)](#)
- [Train models with the CLI \(v2\)](#)
- [CLI \(v2\) YAML schemas](#)

CLI (v2) workspace YAML schema

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/workspace.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the workspace.		
<code>display_name</code>	string	Display name of the workspace in the studio UI. Can be non-unique within the resource group.		
<code>description</code>	string	Description of the workspace.		
<code>tags</code>	object	Dictionary of tags for the workspace.		
<code>location</code>	string	The location of the workspace. If omitted, defaults to the resource group location.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>resource_group</code>	string	Required. The resource group containing the workspace. If the resource group does not exist, a new one will be created.		
<code>hbi_workspace</code>	boolean	Whether the customer data is of high business impact (HBI), containing sensitive business information. For more information, see Data encryption at rest .		<code>false</code>
<code>storage_account</code>	string	The fully qualified resource ID of an existing Azure storage account to use as the default storage account for the workspace. A storage account with premium storage or hierarchical namespace cannot be used as the default storage account. If omitted, a new storage account will be created.		
<code>container_registry</code>	string	The fully qualified resource ID of an existing Azure container registry to use as the default container registry for the workspace. Azure ML uses Azure Container Registry (ACR) for managing container images used for training and deployment. If omitted, a new container registry will be created. Creation is lazy loaded, so the container registry gets created the first time it is needed for an operation for either training or deployment.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>key_vault</code>	string	The fully qualified resource ID of an existing Azure key vault to use as the default key vault for the workspace. If omitted, a new key vault will be created.		
<code>application_insights</code>	string	The fully qualified resource ID of an existing Azure application insights to use as the default application insights for the workspace. If omitted, a new application insights will be created.		
<code>customer_managed_key</code>	object	Azure Machine Learning stores metadata in an Azure Cosmos DB instance. By default the data is encrypted at rest with Microsoft-managed keys. To use your own customer-managed key for encryption, specify the customer-managed key information in this section. For more information, see Data encryption for Azure Cosmos DB .		
<code>customer_managed_key.key_vault</code>	string	The fully qualified resource ID of the key vault containing the customer-managed key. This key vault can be different than the default workspace key vault specified in <code>key_vault</code> .		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>customer_managed_key.string</code>		<p>The key URI of the customer-managed key to encrypt data at rest. The URI format is</p> <pre>https://<keyvault-dns-name>/keys/<key-name>/<key-version></pre>		
<code>image_build_compute</code>	string	<p>Name of the compute target to use for building environment Docker images when the container registry is behind a VNet. For more information, see Secure workspace resources behind VNets.</p>		
<code>public_network_access</code>	string	<p>Whether public endpoint access is allowed if the workspace will be using Private Link. For more information, see Enable public access when behind VNets.</p>	<code>enabled</code> , <code>disabled</code>	<code>disabled</code>

Remarks

The `az ml workspace` command can be used for managing Azure Machine Learning workspaces.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: basic

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: mlw-basic-prod
location: eastus
display_name: Basic workspace-example
description: This example shows a YAML configuration for a basic workspace. In case you use this configuration to deploy a new workspace, since no existing dependent resources are specified, these will be automatically created.
hbi_workspace: false
tags:
  purpose: demonstration
```

YAML: with existing resources

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: mlw-basicex-prod
location: eastus
display_name: Bring your own dependent resources-example
description: This configuration specifies a workspace configuration with existing dependent resources
storage_account:
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.Storage/storageAccounts
/<STORAGE_ACCOUNT>
container_registry:
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.ContainerRegistry/regis
tries/<CONTAINER_REGISTRY>
key_vault:
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.KeyVault/vaults/<KEY_VA
ULT>
application_insights:
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.insights/components/<AP
P_INSIGHTS>
tags:
purpose: demonstration
```

YAML: customer-managed key

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: mlw-cmkexample-prod
location: eastus
display_name: Customer managed key encryption-example
description: This configurations shows how to create a workspace that uses customer-managed keys for
encryption.
customer_managed_key:
key_vault:
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.KeyVault/vaults/<KEY_VA
ULT>
key_uri: https://<KEY_VAULT>.vault.azure.net/keys/<KEY_NAME>/<KEY_VERSION>
tags:
purpose: demonstration
```

YAML: private link

```
$schema: https://azurermschemas.azureedge.net/latest/workspace.schema.json
name: mlw-privatelink-prod
location: eastus
display_name: Private Link endpoint workspace-example
description: When using private link, you must set the image_build_compute property to a cluster name to use
for Docker image environment building. You can also specify whether the workspace should be accessible over
the internet.
image_build_compute: cpu-compute
public_network_access: Disabled
tags:
purpose: demonstration
```

YAML: high business impact

```
$schema: https://azuremlschemas.azureedge.net/latest/workspace.schema.json
name: mlw-hbiexample-prod
location: eastus
display_name: High business impact-example
description: This configuration shows how to configure a workspace with the hbi flag enabled. This flag specifies whether to reduce telemetry collection and enable additional encryption when high-business-impact data is used.
hbi_workspace: true
tags:
  purpose: demonstration
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) environment YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/environment.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the environment.		
<code>version</code>	string	Version of the environment. If omitted, Azure ML will autogenerated a version.		
<code>description</code>	string	Description of the environment.		
<code>tags</code>	object	Dictionary of tags for the environment.		
<code>image</code>	string	The Docker image to use for the environment. One of <code>image</code> or <code>build</code> is required .		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>conda_file</code>	string or object	<p>The standard conda YAML configuration file of the dependencies for a conda environment. See https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#creating-an-environment-file-manually.</p> <p>If specified, <code>image</code> must be specified as well. Azure ML will build the conda environment on top of the Docker image provided.</p>		
<code>build</code>	object	<p>The Docker build context configuration to use for the environment. One of <code>image</code> or <code>build</code> is required.</p>		
<code>build.path</code>	string	Local path to the directory to use as the build context.		
<code>build.dockerfile_path</code>	string	Relative path to the Dockerfile within the build context.		<code>Dockerfile</code>
<code>os_type</code>	string	The type of operating system.	<code>linux</code> , <code>windows</code>	<code>linux</code>
<code>inference_config</code>	object	Inference container configurations. Only applicable if the environment is used to build a serving container for online deployments. See Attributes of the <code>inference_config</code> key .		

Attributes of the `inference_config` key

KEY	TYPE	DESCRIPTION

KEY	TYPE	DESCRIPTION
<code>liveness_route</code>	object	The liveness route for the serving container.
<code>liveness_route.path</code>	string	The path to route liveness requests to.
<code>liveness_route.port</code>	integer	The port to route liveness requests to.
<code>readiness_route</code>	object	The readiness route for the serving container.
<code>readiness_route.path</code>	string	The path to route readiness requests to.
<code>readiness_route.port</code>	integer	The port to route readiness requests to.
<code>scoring_route</code>	object	The scoring route for the serving container.
<code>scoring_route.path</code>	string	The path to route scoring requests to.
<code>scoring_route.port</code>	integer	The port to route scoring requests to.

Remarks

The `az ml environment` command can be used for managing Azure Machine Learning environments.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: local Docker build context

```
$schema: https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: docker-context-example
build:
  path: docker-contexts/python-and-pip
```

YAML: Docker image

```
$schema: https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: docker-image-example
image: pytorch/pytorch:latest
description: Environment created from a Docker image.
```

YAML: Docker image plus conda file

```
$schema: https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: docker-image-plus-conda-example
image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04
conda_file: conda-yamls/pydata.yml
description: Environment created from a Docker image plus Conda environment.
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) data YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/data.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the data asset.		
<code>version</code>	string	Version of the dataset. If omitted, Azure ML will autogenerate a version.		
<code>description</code>	string	Description of the data asset.		
<code>tags</code>	object	Dictionary of tags for the data asset.		
<code>type</code>	string	The data asset type. Specify <code>uri_file</code> for data that points to a single file source, or <code>uri_folder</code> for data that points to a folder source.	<code>uri_file</code> , <code>uri_folder</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>path</code>	string	<p>Either a local path to the data source file or folder, or the URI of a cloud path to the data source file or folder. Please ensure that the source provided here is compatible with the <code>type</code> specified.</p> <p>Supported URI types are <code>azureml</code>, <code>https</code>, <code>wasbs</code>, <code>abfss</code>, and <code>adl</code>. See Core yaml syntax for more information on how to use the <code>azureml://</code> URI format.</p>		

Remarks

The `az ml data` commands can be used for managing Azure Machine Learning data assets.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: datastore file

```
$schema: https://azureschemas.azureedge.net/latest/data.schema.json
name: cloud-file-example
description: Data asset created from file in cloud.
type: uri_file
path: azureml://datastores/workspaceblobstore/paths/example-data/titanic.csv
```

YAML: datastore folder

```
$schema: https://azureschemas.azureedge.net/latest/data.schema.json
name: cloud-folder-example
description: Data asset created from folder in cloud.
type: uri_folder
path: azureml://datastores/workspaceblobstore/paths/example-data/
```

YAML: https file

```
$schema: https://azureschemas.azureedge.net/latest/data.schema.json
name: cloud-file-https-example
description: Data asset created from a file in cloud using https URL.
type: uri_file
path: https://mainstorage9c05dabf5c924.blob.core.windows.net/azureml-blobstore-54887b46-3cb0-485b-bb15-62e7b5578ee6/example-data/titanic.csv
```

YAML: https folder

```
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: cloud-folder-https-example
description: Dataset created from folder in cloud using https URL.
type: uri_folder
path: https://mainstorage9c05dabf5c924.blob.core.windows.net/azureml-blobstore-54887b46-3cb0-485b-bb15-
62e7b5578ee6/example-data/
```

YAML: wasbs file

```
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: cloud-file-wasbs-example
description: Data asset created from a file in cloud using wasbs URL.
type: uri_file
path: wasbs://mainstorage9c05dabf5c924.blob.core.windows.net/azureml-blobstore-54887b46-3cb0-485b-bb15-
62e7b5578ee6/example-data/titanic.csv
```

YAML: wasbs folder

```
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: cloud-folder-wasbs-example
description: Data asset created from folder in cloud using wasbs URL.
type: uri_folder
path: wasbs://mainstorage9c05dabf5c924.blob.core.windows.net/azureml-blobstore-54887b46-3cb0-485b-bb15-
62e7b5578ee6/example-data/
```

YAML: local file

```
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: local-file-example-titanic
description: Data asset created from local file.
type: uri_file
path: sample-data/titanic.csv
```

YAML: local folder

```
$schema: https://azurermschemas.azureedge.net/latest/data.schema.json
name: local-folder-example-titanic
description: Dataset created from local folder.
type: uri_folder
path: sample-data/
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) model YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/model.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>\$schema</code>	string	The YAML schema.	
<code>name</code>	string	Required. Name of the model.	
<code>version</code>	int	Version of the model. If omitted, Azure ML will autogenerate a version.	
<code>description</code>	string	Description of the model.	
<code>tags</code>	object	Dictionary of tags for the model.	
<code>path</code>	string	Either a local path to the model file(s), or the URI of a cloud path to the model file(s). This can point to either a file or a directory.	
<code>type</code>	string	Storage format type of the model. Applicable for no-code deployment scenarios.	<code>custom_model</code> , <code>mlflow_model</code> , <code>triton_model</code>
<code>flavors</code>	object	Flavors of the model. Each model storage format type may have one or more supported flavors. Applicable for no-code deployment scenarios.	

Remarks

The `az ml model` command can be used for managing Azure Machine Learning models.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: local file

```
$schema: https://azurermschemas.azureedge.net/latest/model.schema.json
name: local-file-example
path: mlflow-model/model.pkl
description: Model created from local file.
```

YAML: local folder in MLflow format

```
$schema: https://azurermschemas.azureedge.net/latest/model.schema.json
name: local-mlflow-example
path: mlflow-model
type: mlflow_model
description: Model created from local MLflow model directory.
```

- [Install and use the CLI \(v2\)](#)

CLI (v2) schedule YAML schema

9/22/2022 • 12 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/schedule.schema.json>.

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>\$schema</code>	string	The YAML schema.	
<code>name</code>	string	Required. Name of the schedule.	
<code>version</code>	string	Version of the schedule. If omitted, Azure ML will autogenerate a version.	
<code>description</code>	string	Description of the schedule.	
<code>tags</code>	object	Dictionary of tags for the schedule.	
<code>trigger</code>	object	The trigger configuration to define rule when to trigger job. One of <code>RecurrenceTrigger</code> or <code>CronTrigger</code> is required.	
<code>create_job</code>	object or string	Required. The definition of the job that will be triggered by a schedule. One of <code>string</code> or <code>JobDefinition</code> is required.	

Trigger configuration

Recurrence trigger

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	string	Required. Specifies the schedule type.	recurrence
<code>frequency</code>	string	Required. Specifies the unit of time that describes how often the schedule fires.	<code>minute</code> , <code>hour</code> , <code>day</code> , <code>week</code> , <code>month</code>
<code>interval</code>	integer	Required. Specifies the interval at which the schedule fires.	
<code>start_time</code>	string	Describes the start date and time with timezone. If <code>start_time</code> is omitted, the first job will run instantly and the future jobs will be triggered based on the schedule, saying <code>start_time</code> will be equal to the job created time. If the start time is in the past, the first job will run at the next calculated run time.	
<code>end_time</code>	string	Describes the end date and time with timezone. If <code>end_time</code> is omitted, the schedule will continue to run until it's explicitly disabled.	
<code>timezone</code>	string	Specifies the time zone of the recurrence. If omitted, by default is UTC.	See appendix for timezone values
<code>pattern</code>	object	Specifies the pattern of the recurrence. If pattern is omitted, the job(s) will be triggered according to the logic of <code>start_time</code> , <code>frequency</code> and <code>interval</code> .	

Recurrence schedule

Recurrence schedule defines the recurrence pattern, containing `hours`, `minutes`, and `weekdays`.

- When frequency is `day`, pattern can specify `hours` and `minutes`.
- When frequency is `week` and `month`, pattern can specify `hours`, `minutes` and `weekdays`.

KEY	TYPE	ALLOWED VALUES
<code>hours</code>	integer or array of integer	<code>0-23</code>
<code>minutes</code>	integer or array of integer	<code>0-59</code>

KEY	TYPE	ALLOWED VALUES
<code>week_days</code>	string or array of string	<code>monday</code> , <code>tuesday</code> , <code>wednesday</code> , <code>thursday</code> , <code>friday</code> , <code>saturday</code> , <code>sunday</code>

CronTrigger

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	string	Required. Specifies the schedule type.	<code>cron</code>
<code>expression</code>	string	Required. Specifies the cron expression to define how to trigger jobs. expression uses standard crontab expression to express a recurring schedule. A single expression is composed of five space-delimited fields: <code>MINUTES HOURS DAYS MONTHS DAYS-OF-WEEK</code>	
<code>start_time</code>	string	Describes the start date and time with timezone. If <code>start_time</code> is omitted, the first job will run instantly and the future jobs will be triggered based on the schedule, saying <code>start_time</code> will be equal to the job created time. If the start time is in the past, the first job will run at the next calculated run time.	
<code>end_time</code>	string	Describes the end date and time with timezone. If <code>end_time</code> is omitted, the schedule will continue to run until it's explicitly disabled.	
<code>timezone</code>	string	Specifies the time zone of the recurrence. If omitted, by default is UTC.	See appendix for timezone values

Job definition

Customer can directly use `create_job: azureml:<job_name>` or can use the following properties to define the job.

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	string	Required. Specifies the job type. Only pipeline job is supported.	<code>pipeline</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>job</code>	string	<p>Required. Define how to reference a job, it can be <code>azureml:<job_name></code> or a local pipeline job yaml such as</p> <pre>file:hello-pipeline.yml</pre> <p>.</p>	
<code>experiment_name</code>	string	Experiment name to organize the job under. Each job's run record will be organized under the corresponding experiment in the studio's "Experiments" tab. If omitted, we'll take schedule name as default value.	
<code>inputs</code>	object	Dictionary of inputs to the job. The key is a name for the input within the context of the job and the value is the input value.	
<code>outputs</code>	object	Dictionary of output configurations of the job. The key is a name for the output within the context of the job and the value is the output configuration.	
<code>settings</code>	object	Default settings for the pipeline job. See Attributes of the <code>settings</code> key for the set of configurable properties.	

Attributes of the `settings` key

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>default_datastore</code>	string	<p>Name of the datastore to use as the default datastore for the pipeline job. This value must be a reference to an existing datastore in the workspace using the <code>azureml:<datastore-name></code> syntax. Any outputs defined in the <code>outputs</code> property of the parent pipeline job or child step jobs will be stored in this datastore. If omitted, outputs will be stored in the workspace blob datastore.</p>	

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>default_compute</code>	string	Name of the compute target to use as the default compute for all steps in the pipeline. If compute is defined at the step level, it will override this default compute for that specific step. This value must be a reference to an existing compute in the workspace using the <code>azureml:<compute-name></code> syntax.	
<code>continue_on_step_failure</code>	boolean	Whether the execution of steps in the pipeline should continue if one step fails. The default value is <code>False</code> , which means that if one step fails, the pipeline execution will be stopped, canceling any running steps.	<code>False</code>

Job inputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	The type of job input. Specify <code>uri_file</code> for input data that points to a single file source, or <code>uri_folder</code> for input data that points to a folder source.	<code>uri_file</code> , <code>uri_folder</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
path	string	<p>The path to the data to use as input. This can be specified in a few ways:</p> <ul style="list-style-type: none"> - A local path to the data source file or folder, for example, <pre>path: ./iris.csv</pre> <p>The data will get uploaded during job submission.</p> <ul style="list-style-type: none"> - A URI of a cloud path to the file or folder to use as the input. Supported URI types are <code>azureml</code>, <code>https</code>, <code>wasbs</code>, <code>abfss</code>, <code>adl</code>. For more information on how to use the <code>azureml://</code> URI format, see Core yaml syntax. - An existing registered Azure ML data asset to use as the input. To reference a registered data asset, use the <code>azureml:<data_name>:<data_version></code> syntax or <code>azureml:<data_name>@latest</code> (to reference the latest version of that data asset), for example, <pre>path: azureml:cifar10- data:1</pre> <p>or</p> <pre>path: azureml:cifar10- data@latest</pre> <p>.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>mode</code>	string	<p>Mode of how the data should be delivered to the compute target.</p> <p>For read-only mount (<code>ro_mount</code>), the data will be consumed as a mount path. A folder will be mounted as a folder and a file will be mounted as a file. Azure ML will resolve the input to the mount path.</p> <p>For <code>download</code> mode the data will be downloaded to the compute target. Azure ML will resolve the input to the downloaded path.</p> <p>If you only want the URL of the storage location of the data artifact(s) rather than mounting or downloading the data itself, you can use the <code>direct</code> mode. This will pass in the URL of the storage location as the job input. In this case, you're fully responsible for handling credentials to access the storage.</p>	<code>ro_mount</code> , <code>download</code> , <code>direct</code>	<code>ro_mount</code>

Job outputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	The type of job output. For the default <code>uri_folder</code> type, the output will correspond to a folder.	<code>uri_folder</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
path	string	<p>The path to the data to use as input. This can be specified in a few ways:</p> <ul style="list-style-type: none"> - A local path to the data source file or folder, for example, <pre>path: ./iris.csv</pre> <p>The data will get uploaded during job submission.</p> <ul style="list-style-type: none"> - A URI of a cloud path to the file or folder to use as the input. Supported URI types are <code>azureml</code>, <code>https</code>, <code>wasbs</code>, <code>abfss</code>, <code>adl</code>. For more information on how to use the <code>azureml://</code> URI format, see Core yaml syntax. - An existing registered Azure ML data asset to use as the input. To reference a registered data asset, use the <code>azureml:<data_name>:<data_version></code> syntax or <code>azureml:<data_name>@latest</code> (to reference the latest version of that data asset), for example, <pre>path: azureml:cifar10- data:1</pre> <p>or</p> <pre>path: azureml:cifar10- data@latest</pre> <p>.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
mode	string	Mode of how output file(s) will get delivered to the destination storage. For read-write mount mode (<code>rw_mount</code>) the output directory will be a mounted directory. For upload mode the file(s) written will get uploaded at the end of the job.	<code>rw_mount</code> , <code>upload</code>	<code>rw_mount</code>

Remarks

The `az ml schedule` command can be used for managing Azure Machine Learning models.

Examples

Examples are available in the [examples GitHub repository](#). A couple are shown below.

YAML: Schedule with recurrence pattern

APPLIES TO:  Azure CLI ml extension v2 (current)

```
$schema: https://azuremlschemas.azureedge.net/latest/schedule.schema.json
name: simple_recurrence_job_schedule
display_name: Simple recurrence job schedule
description: a simple hourly recurrence job schedule

trigger:
  type: recurrence
  frequency: day #can be minute, hour, day, week, month
  interval: 1 #every day
  schedule:
    hours: [4,5,10,11,12]
    minutes: [0,30]
  start_time: "2022-07-10T10:00:00" # optional - default will be schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

create_job: ./simple-pipeline-job.yml
# create_job: azureml:simple-pipeline-job
```

YAML: Schedule with cron expression

APPLIES TO:  Azure CLI ml extension v2 (current)

```

$schema: https://azurermschemas.azureedge.net/latest/schedule.schema.json
name: simple_cron_job_schedule
display_name: Simple cron job schedule
description: a simple hourly cron job schedule

trigger:
  type: cron
  expression: "0 * * * *"
  start_time: "2022-07-10T10:00:00" # optional - default will be schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

# create_job: azurerm:simple-pipeline-job
create_job: ./simple-pipeline-job.yml

```

Appendix

Timezone

Current schedule supports the following timezones. The key can be used directly in the Python SDK, while the value can be used in the YAML job. The table is organized by UTC(Coordinated Universal Time).

UTC	KEY	VALUE
UTC -12:00	DATELINE_STANDARD_TIME	"Dateline Standard Time"
UTC -11:00	UTC_11	"UTC-11"
UTC - 10:00	ALEUTIAN_STANDARD_TIME	Aleutian Standard Time
UTC - 10:00	HAWAIIAN_STANDARD_TIME	"Hawaiian Standard Time"
UTC -09:30	MARQUESAS_STANDARD_TIME	"Marquesas Standard Time"
UTC -09:00	ALASKAN_STANDARD_TIME	"Alaskan Standard Time"
UTC -09:00	UTC_09	"UTC-09"
UTC -08:00	PACIFIC_STANDARD_TIME_MEXICO	"Pacific Standard Time (Mexico)"
UTC -08:00	UTC_08	"UTC-08"
UTC -08:00	PACIFIC_STANDARD_TIME	"Pacific Standard Time"
UTC -07:00	US_MOUNTAIN_STANDARD_TIME	"US Mountain Standard Time"
UTC -07:00	MOUNTAIN_STANDARD_TIME_MEXICO	"Mountain Standard Time (Mexico)"
UTC -07:00	MOUNTAIN_STANDARD_TIME	"Mountain Standard Time"
UTC -06:00	CENTRAL_AMERICA_STANDARD_TIME	"Central America Standard Time"
UTC -06:00	CENTRAL_STANDARD_TIME	"Central Standard Time"
UTC -06:00	EASTER_ISLAND_STANDARD_TIME	"Easter Island Standard Time"

UTC	KEY	VALUE
UTC -06:00	CENTRAL_STANDARD_TIME_MEXICO	"Central Standard Time (Mexico)"
UTC -06:00	CANADA_CENTRAL_STANDARD_TIME	"Canada Central Standard Time"
UTC -05:00	SA_PACIFIC_STANDARD_TIME	"SA Pacific Standard Time"
UTC -05:00	EASTERN_STANDARD_TIME_MEXICO	"Eastern Standard Time (Mexico)"
UTC -05:00	EASTERN_STANDARD_TIME	"Eastern Standard Time"
UTC -05:00	HAITI_STANDARD_TIME	"Haiti Standard Time"
UTC -05:00	CUBA_STANDARD_TIME	"Cuba Standard Time"
UTC -05:00	US_EASTERN_STANDARD_TIME	"US Eastern Standard Time"
UTC -05:00	TURKS_AND_CAICOS_STANDARD_TIME	"Turks And Caicos Standard Time"
UTC -04:00	PARAGUAY_STANDARD_TIME	"Paraguay Standard Time"
UTC -04:00	ATLANTIC_STANDARD_TIME	"Atlantic Standard Time"
UTC -04:00	VENEZUELA_STANDARD_TIME	"Venezuela Standard Time"
UTC -04:00	CENTRAL_BRAZILIAN_STANDARD_TIME	"Central Brazilian Standard Time"
UTC -04:00	SA_WESTERN_STANDARD_TIME	"SA Western Standard Time"
UTC -04:00	PACIFIC_SA_STANDARD_TIME	"Pacific SA Standard Time"
UTC -03:30	NEWFOUNDLAND_STANDARD_TIME	"Newfoundland Standard Time"
UTC -03:00	TOCANTINS_STANDARD_TIME	"Tocantins Standard Time"
UTC -03:00	E_SOUTH_AMERICAN_STANDARD_TIME	"E. South America Standard Time"
UTC -03:00	SA_EASTERN_STANDARD_TIME	"SA Eastern Standard Time"
UTC -03:00	ARGENTINA_STANDARD_TIME	"Argentina Standard Time"
UTC -03:00	GREENLAND_STANDARD_TIME	"Greenland Standard Time"
UTC -03:00	MONTEVIDEO_STANDARD_TIME	"Montevideo Standard Time"
UTC -03:00	SAINT_PIERRE_STANDARD_TIME	"Saint Pierre Standard Time"
UTC -03:00	BAHIA_STANDARD_TIME	"Bahia Standard Time"

UTC	KEY	VALUE
UTC -02:00	UTC_02	"UTC-02"
UTC -02:00	MID_ATLANTIC_STANDARD_TIME	"Mid-Atlantic Standard Time"
UTC -01:00	AZORES_STANDARD_TIME	"Azores Standard Time"
UTC -01:00	CAPE_VERDE_STANDARD_TIME	"Cape Verde Standard Time"
UTC	UTC	UTC
UTC +00:00	GMT_STANDARD_TIME	"GMT Standard Time"
UTC +00:00	GREENWICH_STANDARD_TIME	"Greenwich Standard Time"
UTC +01:00	MOROCCO_STANDARD_TIME	"Morocco Standard Time"
UTC +01:00	W_EUROPE_STANDARD_TIME	"W. Europe Standard Time"
UTC +01:00	CENTRAL_EUROPE_STANDARD_TIME	"Central Europe Standard Time"
UTC +01:00	ROMANCE_STANDARD_TIME	"Romance Standard Time"
UTC +01:00	CENTRAL_EUROPEAN_STANDARD_TIME	"Central European Standard Time"
UTC +01:00	W_CENTRAL_AFRICA_STANDARD_TIME	"W. Central Africa Standard Time"
UTC +02:00	NAMIBIA_STANDARD_TIME	"Namibia Standard Time"
UTC +02:00	JORDAN_STANDARD_TIME	"Jordan Standard Time"
UTC +02:00	GTB_STANDARD_TIME	"GTB Standard Time"
UTC +02:00	MIDDLE_EAST_STANDARD_TIME	"Middle East Standard Time"
UTC +02:00	EGYPT_STANDARD_TIME	"Egypt Standard Time"
UTC +02:00	E_EUROPE_STANDARD_TIME	"E. Europe Standard Time"
UTC +02:00	SYRIA_STANDARD_TIME	"Syria Standard Time"
UTC +02:00	WEST_BANK_STANDARD_TIME	"West Bank Standard Time"
UTC +02:00	SOUTH_AFRICA_STANDARD_TIME	"South Africa Standard Time"
UTC +02:00	FLE_STANDARD_TIME	"FLE Standard Time"
UTC +02:00	ISRAEL_STANDARD_TIME	"Israel Standard Time"

UTC	KEY	VALUE
UTC +02:00	KALININGRAD_STANDARD_TIME	"Kaliningrad Standard Time"
UTC +02:00	LIBYA_STANDARD_TIME	"Libya Standard Time"
UTC +03:00	TURKEY_STANDARD_TIME	"Turkey Standard Time"
UTC +03:00	ARABIC_STANDARD_TIME	"Arabic Standard Time"
UTC +03:00	ARAB_STANDARD_TIME	"Arab Standard Time"
UTC +03:00	BELARUS_STANDARD_TIME	"Belarus Standard Time"
UTC +03:00	RUSSIAN_STANDARD_TIME	"Russian Standard Time"
UTC +03:00	E_AFRICA_STANDARD_TIME	"E. Africa Standard Time"
UTC +03:30	IRAN_STANDARD_TIME	"Iran Standard Time"
UTC +04:00	ARABIAN_STANDARD_TIME	"Arabian Standard Time"
UTC +04:00	ASTRAKHAN_STANDARD_TIME	"Astrakhan Standard Time"
UTC +04:00	AZERBAIJAN_STANDARD_TIME	"Azerbaijan Standard Time"
UTC +04:00	RUSSIA_TIME_ZONE_3	"Russia Time Zone 3"
UTC +04:00	MAURITIUS_STANDARD_TIME	"Mauritius Standard Time"
UTC +04:00	GEORGIAN_STANDARD_TIME	"Georgian Standard Time"
UTC +04:00	CAUCASUS_STANDARD_TIME	"Caucasus Standard Time"
UTC +04:30	AFGHANISTAN_STANDARD_TIME	"Afghanistan Standard Time"
UTC +05:00	WEST_ASIA_STANDARD_TIME	"West Asia Standard Time"
UTC +05:00	EKATERINBURG_STANDARD_TIME	"Ekaterinburg Standard Time"
UTC +05:00	PAKISTAN_STANDARD_TIME	"Pakistan Standard Time"
UTC +05:30	INDIA_STANDARD_TIME	"India Standard Time"
UTC +05:30	SRI_LANKA_STANDARD_TIME	"Sri Lanka Standard Time"
UTC +05:45	NEPAL_STANDARD_TIME	"Nepal Standard Time"
UTC +06:00	CENTRAL_ASIA_STANDARD_TIME	"Central Asia Standard Time"
UTC +06:00	BANGLADESH_STANDARD_TIME	"Bangladesh Standard Time"

UTC	KEY	VALUE
UTC +06:30	MYANMAR_STANDARD_TIME	"Myanmar Standard Time"
UTC +07:00	N_CENTRAL_ASIA_STANDARD_TIME	"N. Central Asia Standard Time"
UTC +07:00	SE_ASIA_STANDARD_TIME	"SE Asia Standard Time"
UTC +07:00	ALTAI_STANDARD_TIME	"Altai Standard Time"
UTC +07:00	W_MONGOLIA_STANDARD_TIME	"W. Mongolia Standard Time"
UTC +07:00	NORTH_ASIA_STANDARD_TIME	"North Asia Standard Time"
UTC +07:00	TOMSK_STANDARD_TIME	"Tomsk Standard Time"
UTC +08:00	CHINA_STANDARD_TIME	"China Standard Time"
UTC +08:00	NORTH_ASIA_EAST_STANDARD_TIME	"North Asia East Standard Time"
UTC +08:00	SINGAPORE_STANDARD_TIME	"Singapore Standard Time"
UTC +08:00	W_AUSTRALIA_STANDARD_TIME	"W. Australia Standard Time"
UTC +08:00	TAIPEI_STANDARD_TIME	"Taipei Standard Time"
UTC +08:00	ULAANBAATAR_STANDARD_TIME	"Ulaanbaatar Standard Time"
UTC +08:45	AUS_CENTRAL_W_STANDARD_TIME	"Aus Central W. Standard Time"
UTC +09:00	NORTH_KOREA_STANDARD_TIME	"North Korea Standard Time"
UTC +09:00	TRANSBAIKAL_STANDARD_TIME	"Transbaikal Standard Time"
UTC +09:00	TOKYO_STANDARD_TIME	"Tokyo Standard Time"
UTC +09:00	KOREA_STANDARD_TIME	"Korea Standard Time"
UTC +09:00	YAKUTSK_STANDARD_TIME	"Yakutsk Standard Time"
UTC +09:30	CEN_AUSTRALIA_STANDARD_TIME	"Cen. Australia Standard Time"
UTC +09:30	AUS_CENTRAL_STANDARD_TIME	"AUS Central Standard Time"
UTC +10:00	E_AUSTRALIAN_STANDARD_TIME	"E. Australia Standard Time"
UTC +10:00	AUS_EASTERN_STANDARD_TIME	"AUS Eastern Standard Time"
UTC +10:00	WEST_PACIFIC_STANDARD_TIME	"West Pacific Standard Time"
UTC +10:00	TASMANIA_STANDARD_TIME	"Tasmania Standard Time"

UTC	KEY	VALUE
UTC +10:00	VLADIVOSTOK_STANDARD_TIME	"Vladivostok Standard Time"
UTC +10:30	LORD_HOWE_STANDARD_TIME	"Lord Howe Standard Time"
UTC +11:00	BOUGAINVILLE_STANDARD_TIME	"Bougainville Standard Time"
UTC +11:00	RUSSIA_TIME_ZONE_10	"Russia Time Zone 10"
UTC +11:00	MAGADAN_STANDARD_TIME	"Magadan Standard Time"
UTC +11:00	NORFOLK_STANDARD_TIME	"Norfolk Standard Time"
UTC +11:00	SAKHALIN_STANDARD_TIME	"Sakhalin Standard Time"
UTC +11:00	CENTRAL_PACIFIC_STANDARD_TIME	"Central Pacific Standard Time"
UTC +12:00	RUSSIA_TIME_ZONE_11	"Russia Time Zone 11"
UTC +12:00	NEW_ZEALAND_STANDARD_TIME	"New Zealand Standard Time"
UTC +12:00	UTC_12	"UTC+12"
UTC +12:00	FIJI_STANDARD_TIME	"Fiji Standard Time"
UTC +12:00	KAMCHATKA_STANDARD_TIME	"Kamchatka Standard Time"
UTC +12:45	CHATHAM_ISLANDS_STANDARD_TIME	"Chatham Islands Standard Time"
UTC +13:00	TONGA_STANDARD_TIME	"Tonga Standard Time"
UTC +13:00	SAMOA_STANDARD_TIME	"Samoa Standard Time"
UTC +14:00	LINE_ISLANDS_STANDARD_TIME	"Line Islands Standard Time"

CLI (v2) compute cluster (AmlCompute) YAML schema

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/amlCompute.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of compute.	<code>amlcompute</code>	
<code>name</code>	string	Required. Name of the compute.		
<code>description</code>	string	Description of the compute.		
<code>location</code>	string	The location for the compute. If omitted, defaults to the workspace location.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>size</code>	string	The VM size to use for the cluster. For more information, see Supported VM series and sizes . Note that not all sizes are available in all regions.	For the list of supported sizes in a given region, please use <code>az ml compute list-sizes</code>	<code>Standard_DS3_v2</code>
<code>tier</code>	string	The VM priority tier to use for the cluster. Low-priority VMs are pre-emptible but come at a reduced cost compared to dedicated VMs.	<code>dedicated</code> , <code>low_priority</code>	<code>dedicated</code>
<code>min_instances</code>	integer	The minimum number of nodes to use on the cluster. Setting the minimum number of nodes to <code>0</code> allows Azure ML to autoscale the cluster down to zero nodes when not in use. Any value larger than <code>0</code> will keep that number of nodes running, even if the cluster is not in use.		<code>0</code>
<code>max_instances</code>	integer	The maximum number of nodes to use on the cluster.		<code>1</code>
<code>idle_time_before_scale</code> <code>integer</code>	integer	Node idle time in seconds before scaling down the cluster.		<code>120</code>
<code>ssh_public_access_enabled</code> <code>boolean</code>	boolean	Whether to enable public SSH access on the nodes of the cluster.		<code>false</code>
<code>ssh_settings</code>	object	SSH settings for connecting to the cluster.		
<code>ssh_settings.admin_user</code> <code>string</code>	string	The name of the administrator user account that can be used to SSH into nodes.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>ssh_settings.admin_password</code>		The password of the administrator user account. One of <code>admin_password</code> or <code>ssh_key_value</code> is required.		
<code>ssh_settings.ssh_key_value</code>		The SSH public key of the administrator user account. One of <code>admin_password</code> or <code>ssh_key_value</code> is required.		
<code>network_settings</code>	object	Network security settings.		
<code>network_settings.vnet_name</code>		Name of the virtual network (VNet) when creating a new one or referencing an existing one.		
<code>network_settings.subnet_id</code>		Either the name of the subnet when creating a new VNet or referencing an existing one, or the fully qualified resource ID of a subnet in an existing VNet. Do not specify <code>network_settings.vnet_name</code> if the subnet ID is specified. The subnet ID can refer to a VNet/subnet in another resource group.		
<code>identity</code>	object	The managed identity configuration to assign to the compute. AmlCompute clusters support only one system-assigned identity or multiple user-assigned identities, not both concurrently.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>identity.type</code>	string	The type of managed identity to assign to the compute. If the type is <code>user_assigned</code> , the <code>identity.user_assigned_identities</code> property must also be specified.	<code>system_assigned</code> , <code>user_assigned</code>	
<code>identity.user_assigned_identities</code>		List of fully qualified resource IDs of the user-assigned identities.		

Remarks

The `az ml compute` commands can be used for managing Azure Machine Learning compute clusters (AmlCompute).

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: minimal

```
$schema: https://azureschemas.azureedge.net/latest/amlCompute.schema.json
name: minimal-example
type: amlcompute
```

YAML: basic

```
$schema: https://azureschemas.azureedge.net/latest/amlCompute.schema.json
name: basic-example
type: amlcompute
size: STANDARD_DS3_V2
min_instances: 0
max_instances: 2
idle_time_before_scale_down: 120
```

YAML: custom location

```
$schema: https://azureschemas.azureedge.net/latest/amlCompute.schema.json
name: location-example
type: amlcompute
size: STANDARD_DS3_V2
min_instances: 0
max_instances: 2
idle_time_before_scale_down: 120
location: westus
```

YAML: low priority

```
$schema: https://azuremlschemas.azureedge.net/latest/amlCompute.schema.json
name: low-pri-example
type: amlcompute
size: STANDARD_DS3_V2
min_instances: 0
max_instances: 2
idle_time_before_scale_down: 120
tier: low_priority
```

YAML: SSH username and password

```
$schema: https://azuremlschemas.azureedge.net/latest/amlCompute.schema.json
name: ssh-example
type: amlcompute
size: STANDARD_DS3_V2
min_instances: 0
max_instances: 2
idle_time_before_scale_down: 120
ssh_settings:
    admin_username: example-user
    admin_password: example-password
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) compute instance YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azuremlschemas.azureedge.net/latest/computeinstance.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of compute.	<code>computeinstance</code>	
<code>name</code>	string	Required. Name of the compute.		
<code>description</code>	string	Description of the compute.		
<code>size</code>	string	The VM size to use for the compute instance. For more information, see Supported VM series and sizes . Note that not all sizes are available in all regions.	For the list of supported sizes in a given region, please use the <code>az ml compute list-sizes</code> command.	<code>Standard_DS3_v2</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>create_on_behalf_of</code>	object	Settings for creating the compute instance on behalf of another user. Please ensure that the assigned user has correct RBAC permissions.		
<code>create_on_behalf_of.user_tenant_id</code>	string	The AAD Tenant ID of the assigned user.		
<code>create_on_behalf_of.user_object_id</code>	string	The AAD Object ID of the assigned user.		
<code>ssh_public_access_enabled</code>	boolean	Whether to enable public SSH access on the compute instance.		<code>false</code>
<code>ssh_settings</code>	object	SSH settings for connecting to the compute instance.		
<code>ssh_settings.ssh_key_value</code>	string	The SSH public key of the administrator user account.		
<code>network_settings</code>	object	Network security settings.		
<code>network_settings.vnet_name</code>	string	Name of the virtual network (VNet) when creating a new one or referencing an existing one.		
<code>network_settings.subnet_id</code>	string	Either the name of the subnet when creating a new VNet or referencing an existing one, or the fully qualified resource ID of a subnet in an existing VNet. Do not specify <code>network_settings.vnet_name</code> if the subnet ID is specified. The subnet ID can refer to a VNet/subnet in another resource group.		

Remarks

The `az ml compute` command can be used for managing Azure Machine Learning compute instances.

YAML: minimal

```
$schema: https://azuremlschemas.azureedge.net/latest/computeInstance.schema.json
name: minimal-example-i
type: computeinstance
```

YAML: basic

```
$schema: https://azuremlschemas.azureedge.net/latest/computeInstance.schema.json
name: basic-example-i
type: computeinstance
size: STANDARD_DS3_V2
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) attached Virtual Machine YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/vmCompute.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of compute.	<code>virtualmachine</code>	
<code>name</code>	string	Required. Name of the compute.		
<code>description</code>	string	Description of the compute.		
<code>resource_id</code>	string	Required. Fully qualified resource ID of the Azure Virtual Machine to attach to the workspace as a compute target.		
<code>ssh_settings</code>	object	SSH settings for connecting to the virtual machine.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>ssh_settings.admin_username</code>		The name of the administrator user account that can be used to SSH into the virtual machine.		
<code>ssh_settings.admin_password</code>		The password of the administrator user account. One of <code>admin_password</code> or <code>ssh_private_key_file</code> is required.		
<code>ssh_settings.ssh_private_key_file</code>		The local path to the SSH private key file of the administrator user account. One of <code>admin_password</code> or <code>ssh_private_key_file</code> is required.		
<code>ssh_settings.ssh_port</code>	integer	The SSH port on the virtual machine.		22

Remarks

The `az ml compute` command can be used for managing Virtual Machines (VM) attached to an Azure Machine Learning workspace.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: basic

```
$schema: https://azurermschemas.azureedge.net/latest/vmCompute.schema.json
name: vm-example
type: virtualmachine
resource_id:
/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP>/providers/Microsoft.Compute/virtualMachines
/<VM_NAME>
ssh_settings:
  admin_username: <admin_username>
  admin_password: <admin_password>
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) Attached Azure Arc-enabled Kubernetes cluster (KubernetesCompute) YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azuremlschemas.azureedge.net/latest/kubernetesCompute.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of compute.	<code>kubernetes</code>	
<code>name</code>	string	Required. Name of the compute.		
<code>description</code>	string	Description of the compute.		
<code>resource_id</code>	string	Fully qualified resource ID of the Azure Arc-enabled Kubernetes cluster to attach to the workspace as a compute target.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>namespace</code>	string	The Kubernetes namespace to use for the compute target. The namespace must be created in the Kubernetes cluster before the cluster can be attached to the workspace as a compute target. All Azure ML workloads running on this compute target will run under the namespace specified in this field.		
<code>identity</code>	object	The managed identity configuration to assign to the compute. KubernetesCompute clusters support only one system-assigned identity or multiple user-assigned identities, not both concurrently.		
<code>identity.type</code>	string	The type of managed identity to assign to the compute. If the type is <code>user_assigned</code> , the <code>identity.user_assigned_identities</code> property must also be specified.	<code>system_assigned</code> , <code>user_assigned</code>	
<code>identity.user_assigned_identities</code>		List of fully qualified resource IDs of the user-assigned identities.		

Remarks

The `az ml compute` commands can be used for managing Azure Arc-enabled Kubernetes clusters (KubernetesCompute) attached to an Azure Machine Learning workspace.

Next steps

- [Install and use the CLI \(v2\)](#)
- [Configure and attach Kubernetes clusters anywhere](#)

CLI (v2) command job YAML schema

9/22/2022 • 8 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/commandJob.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	const	The type of job.	<code>command</code>	<code>command</code>
<code>name</code>	string	Name of the job. Must be unique across all jobs in the workspace. If omitted, Azure ML will autogenerate a GUID for the name.		
<code>display_name</code>	string	Display name of the job in the studio UI. Can be non-unique within the workspace. If omitted, Azure ML will autogenerate a human-readable adjective-noun identifier for the display name.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>experiment_name</code>	string	Experiment name to organize the job under. Each job's run record will be organized under the corresponding experiment in the studio's "Experiments" tab. If omitted, Azure ML will default it to the name of the working directory where the job was created.		
<code>description</code>	string	Description of the job.		
<code>tags</code>	object	Dictionary of tags for the job.		
<code>command</code>	string	Required (if not using <code>component</code> field). The command to execute.		
<code>code</code>	string	Local path to the source code directory to be uploaded and used for the job.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>environment</code>	string or object	<p>Required (if not using <code>component</code> field). The environment to use for the job. This can be either a reference to an existing versioned environment in the workspace or an inline environment specification.</p> <p>To reference an existing environment use the</p> <pre>azureml: <environment_name>: <environment_version></pre> <p>syntax or</p> <pre>azureml: <environment_name>@latest</pre> <p>(to reference the latest version of an environment).</p> <p>To define an environment inline please follow the Environment schema. Exclude the <code>name</code> and <code>version</code> properties as they are not supported for inline environments.</p>		
<code>environment_variables</code>	object	Dictionary of environment variable key-value pairs to set on the process where the command is executed.		
<code>distribution</code>	object	The distribution configuration for distributed training scenarios. One of MpiConfiguration , PyTorchConfiguration , or TensorFlowConfiguration .		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>compute</code>	string	<p>Name of the compute target to execute the job on. This can be either a reference to an existing compute in the workspace (using the <code>azureml:<compute_name></code> syntax) or <code>local</code> to designate local execution. Note: jobs in pipeline didn't support <code>local</code> as <code>compute</code>.</p>		<code>local</code>
<code>resources.instance_count</code>	integer	The number of nodes to use for the job.		<code>1</code>
<code>resources.instance_type</code>	string	<p>The instance type to use for the job. Applicable for jobs running on Azure Arc-enabled Kubernetes compute (where the compute target specified in the <code>compute</code> field is of <code>type: kubernetes</code>). If omitted, this will default to the default instance type for the Kubernetes cluster. For more information, see Create and select Kubernetes instance types.</p>		
<code>limits.timeout</code>	integer	The maximum time in seconds the job is allowed to run. Once this limit is reached the system will cancel the job.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>inputs</code>	object	<p>Dictionary of inputs to the job. The key is a name for the input within the context of the job and the value is the input value.</p> <p>Inputs can be referenced in the <code>command</code> using the <code>\${{ inputs.<input_name> }}</code> expression.</p>		
<code>inputs.<input_name></code>	number, integer, boolean, string or object	<p>One of a literal value (of type number, integer, boolean, or string) or an object containing a job input data specification.</p>		
<code>outputs</code>	object	<p>Dictionary of output configurations of the job. The key is a name for the output within the context of the job and the value is the output configuration.</p> <p>Outputs can be referenced in the <code>command</code> using the <code>\${{ outputs.<output_name> }}</code> expression.</p>		
<code>outputs.<output_name></code>	object	<p>You can leave the object empty, in which case by default the output will be of type <code>uri_folder</code> and Azure ML will system-generate an output location for the output. File(s) to the output directory will be written via read-write mount. If you want to specify a different mode for the output, provide an object containing the job output specification.</p>		

Distribution configurations

MpiConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. Distribution type.	<code>mpi</code>
<code>process_count_per_instance</code>	integer	Required. The number of processes per node to launch for the job.	

PyTorchConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. Distribution type.	<code>pytorch</code>	
<code>process_count_per_instance</code>	integer	The number of processes per node to launch for the job.		<code>1</code>

TensorFlowConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. Distribution type.	<code>tensorflow</code>	
<code>worker_count</code>	integer	The number of workers to launch for the job.		Defaults to <code>resources.instance_count</code> .
<code>parameter_server_count</code>	integer	The number of parameter servers to launch for the job.		<code>0</code>

Job inputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	The type of job input. Specify <code>uri_file</code> for input data that points to a single file source, or <code>uri_folder</code> for input data that points to a folder source.	<code>uri_file</code> , <code>uri_folder</code> , <code>mlflow_model</code> , <code>custom_model</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
path	string	<p>The path to the data to use as input. This can be specified in a few ways:</p> <ul style="list-style-type: none"> - A local path to the data source file or folder, e.g. <pre>path: ./iris.csv</pre> <p>The data will get uploaded during job submission.</p> <ul style="list-style-type: none"> - A URI of a cloud path to the file or folder to use as the input. Supported URI types are <code>azureml</code>, <code>https</code>, <code>wasbs</code>, <code>abfss</code>, <code>adl</code>. See Core yaml syntax for more information on how to use the <code>azureml://</code> URI format. - An existing registered Azure ML data asset to use as the input. To reference a registered data asset use the <code>azureml:<data_name>:<data_version></code> syntax or <code>azureml:<data_name>@latest</code> (to reference the latest version of that data asset), e.g. <pre>path: azureml:cifar10- data:1</pre> <p>or</p> <pre>path: azureml:cifar10- data@latest</pre> <p>.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>mode</code>	string	<p>Mode of how the data should be delivered to the compute target.</p> <p>For read-only mount (<code>ro_mount</code>), the data will be consumed as a mount path. A folder will be mounted as a folder and a file will be mounted as a file. Azure ML will resolve the input to the mount path.</p> <p>For <code>download</code> mode the data will be downloaded to the compute target. Azure ML will resolve the input to the downloaded path.</p> <p>If you only want the URL of the storage location of the data artifact(s) rather than mounting or downloading the data itself, you can use the <code>direct</code> mode. This will pass in the URL of the storage location as the job input. Note that in this case you are fully responsible for handling credentials to access the storage.</p>	<code>ro_mount</code> , <code>download</code> , <code>direct</code>	<code>ro_mount</code>

Job outputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	The type of job output. For the default <code>uri_folder</code> type, the output will correspond to a folder.	<code>uri_folder</code> , <code>mlflow_model</code> , <code>custom_model</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
mode	string	Mode of how output file(s) will get delivered to the destination storage. For read-write mount mode (<code>rw_mount</code>) the output directory will be a mounted directory. For upload mode the file(s) written will get uploaded at the end of the job.	<code>rw_mount</code> , <code>upload</code>	<code>rw_mount</code>

Remarks

The `az ml job` command can be used for managing Azure Machine Learning jobs.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: hello world

```
$schema: https://azureschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world"
environment:
  image: library/python:latest
compute: azureml:cpu-cluster
```

YAML: display name, experiment name, description, and tags

```
$schema: https://azureschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world"
environment:
  image: library/python:latest
compute: azureml:cpu-cluster
tags:
  hello: world
display_name: hello-world-example
experiment_name: hello-world-example
description: |
  # Azure Machine Learning "hello world" job

  This is a "hello world" job running in the cloud via Azure Machine Learning!

  ## Description

  Markdown is supported in the studio for job descriptions! You can edit the description there or via CLI.
```

YAML: environment variables

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo $hello_env_var
environment:
  image: library/python:latest
compute: azurerm:cpu-cluster
environment_variables:
  hello_env_var: "hello world"
```

YAML: source code

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: ls
code: src
environment:
  image: library/python:latest
compute: azurerm:cpu-cluster
```

YAML: literal inputs

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  echo ${{inputs.hello_string}}
  echo ${{inputs.hello_number}}
environment:
  image: library/python:latest
inputs:
  hello_string: "hello world"
  hello_number: 42
compute: azurerm:cpu-cluster
```

YAML: write to default outputs

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world" > ./outputs/helloworld.txt
environment:
  image: library/python:latest
compute: azurerm:cpu-cluster
```

YAML: write to named data output

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: echo "hello world" > ${outputs.hello_output}/helloworld.txt
outputs:
  hello_output:
environment:
  image: python
compute: azurerm:cpu-cluster
```

YAML: datastore URI file input

```

$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  echo "--iris-csv: ${{inputs.iris_csv}}"
  python hello-iris.py --iris-csv ${{inputs.iris_csv}}
code: src
inputs:
  iris_csv:
    type: uri_file
    path: azurerm://datastores/workspaceblobstore/paths/example-data/iris.csv
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster

```

YAML: datastore URI folder input

```

$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  ls ${{inputs.data_dir}}
  echo "--iris-csv: ${{inputs.data_dir}}/iris.csv"
  python hello-iris.py --iris-csv ${{inputs.data_dir}}/iris.csv
code: src
inputs:
  data_dir:
    type: uri_folder
    path: azurerm://datastores/workspaceblobstore/paths/example-data/
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster

```

YAML: URI file input

```

$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  echo "--iris-csv: ${{inputs.iris_csv}}"
  python hello-iris.py --iris-csv ${{inputs.iris_csv}}
code: src
inputs:
  iris_csv:
    type: uri_file
    path: https://azurermexamples.blob.core.windows.net/datasets/iris.csv
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster

```

YAML: URI folder input

```

$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  ls ${{inputs.data_dir}}
  echo "--iris-csv: ${{inputs.data_dir}}/iris.csv"
  python hello-iris.py --iris-csv ${{inputs.data_dir}}/iris.csv
code: src
inputs:
  data_dir:
    type: uri_folder
    path: wasbs://datasets@azurermexamples.blob.core.windows.net/
environment: azurerm:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
compute: azurerm:cpu-cluster

```

YAML: Notebook via papermill

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: |
  pip install ipykernel papermill
  papermill hello-notebook.ipynb outputs/out.ipynb -k python
code: src
environment:
  image: library/python:latest
compute: azurerm1:cpu-cluster
```

YAML: basic Python model training

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
code: src
command: >-
  python main.py
  --iris-csv ${{inputs.iris_csv}}
  --C ${{inputs.C}}
  --kernel ${{inputs.kernel}}
  --coef0 ${{inputs.coef0}}
inputs:
  iris_csv:
    type: uri_file
    path: wasbs://datasets@azurermexamples.blob.core.windows.net/iris.csv
    C: 0.8
    kernel: "rbf"
    coef0: 0.1
environment: azurerm1:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
compute: azurerm1:cpu-cluster
display_name: sklearn-iris-example
experiment_name: sklearn-iris-example
description: Train a scikit-learn SVM on the Iris dataset.
```

YAML: basic R model training with local Docker build context

```
$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
command: >
  Rscript train.R
  --data_folder ${{inputs.iris}}
code: src
inputs:
  iris:
    type: uri_file
    path: https://azurermexamples.blob.core.windows.net/datasets/iris.csv
environment:
  build:
    path: docker-context
compute: azurerm1:cpu-cluster
display_name: r-iris-example
experiment_name: r-iris-example
description: Train an R model on the Iris dataset.
```

YAML: distributed PyTorch

```

$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
code: src
command: >-
  python train.py
  --epochs ${{inputs.epochs}}
  --learning-rate ${{inputs.learning_rate}}
  --data-dir ${{inputs.cifar}}
inputs:
  epochs: 1
  learning_rate: 0.2
  cifar:
    type: uri_folder
    path: azureml:cifar-10-example:1
environment: azureml:AzureML-pytorch-1.9-ubuntu18.04-py37-cuda11-gpu@latest
compute: azureml:gpu-cluster
distribution:
  type: pytorch
  process_count_per_instance: 1
resources:
  instance_count: 2
display_name: pytorch-cifar-distributed-example
experiment_name: pytorch-cifar-distributed-example
description: Train a basic convolutional neural network (CNN) with PyTorch on the CIFAR-10 dataset, distributed via PyTorch.

```

YAML: distributed TensorFlow

```

$schema: https://azurermschemas.azureedge.net/latest/commandJob.schema.json
code: src
command: >-
  python train.py
  --epochs ${{inputs.epochs}}
  --model-dir ${{inputs.model_dir}}
inputs:
  epochs: 1
  model_dir: outputs/keras-model
environment: azureml:AzureML-tensorflow-2.4-ubuntu18.04-py37-cuda11-gpu@latest
compute: azureml:gpu-cluster
resources:
  instance_count: 2
distribution:
  type: tensorflow
  worker_count: 2
display_name: tensorflow-mnist-distributed-example
experiment_name: tensorflow-mnist-distributed-example
description: Train a basic neural network with TensorFlow on the MNIST dataset, distributed via TensorFlow.

```

YAML: distributed MPI

```
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json
code: src
command: >-
    python train.py
    --epochs ${{inputs.epochs}}
inputs:
    epochs: 1
environment: azureml:AzureML-tensorflow-2.7-ubuntu20.04-py38-cuda11-gpu@latest
compute: azureml:gpu-cluster
resources:
    instance_count: 2
distribution:
    type: mpi
    process_count_per_instance: 2
display_name: tensorflow-mnist-distributed-horovod-example
experiment_name: tensorflow-mnist-distributed-horovod-example
description: Train a basic neural network with TensorFlow on the MNIST dataset, distributed via Horovod.
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) sweep job YAML schema

9/22/2022 • 10 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/sweepJob.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	const	Required. The type of job.	<code>sweep</code>	<code>sweep</code>
<code>name</code>	string	Name of the job. Must be unique across all jobs in the workspace. If omitted, Azure ML will autogenerated a GUID for the name.		
<code>display_name</code>	string	Display name of the job in the studio UI. Can be non-unique within the workspace. If omitted, Azure ML will autogenerated a human-readable adjective-noun identifier for the display name.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>experiment_name</code>	string	Experiment name to organize the job under. Each job's run record will be organized under the corresponding experiment in the studio's "Experiments" tab. If omitted, Azure ML will default it to the name of the working directory where the job was created.		
<code>description</code>	string	Description of the job.		
<code>tags</code>	object	Dictionary of tags for the job.		
<code>sampling_algorithm</code>	object	Required. The hyperparameter sampling algorithm to use over the <code>search_space</code> . One of RandomSamplingAlgorithm , GridSamplingAlgorithm , or BayesianSamplingAlgorithm .		
<code>search_space</code>	object	Required. Dictionary of the hyperparameter search space. The key is the name of the hyperparameter and the value is the parameter expression. Hyperparameters can be referenced in the <code>trial.command</code> using the <code>`\${{ search_space.<hyperparameter> }}</code> expression.		
<code>search_space.<hyperparameter></code>	object	See Parameter expressions for the set of possible expressions to use.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>objective.primary_metric</code>	string	Required. The name of the primary metric reported by each trial job. The metric must be logged in the user's training script using <code>mlflow.log_metric()</code> with the same corresponding metric name.		
<code>objective.goal</code>	string	Required. The optimization goal of the <code>objective.primary_metric</code> .	<code>maximize</code> , <code>minimize</code>	
<code>early_termination</code>	object	The early termination policy to use. A trial job is canceled when the criteria of the specified policy are met. If omitted, no early termination policy will be applied. One of BanditPolicy , MedianStoppingPolicy , or TruncationSelectionPolicy .		
<code>limits</code>	object	Limits for the sweep job. See Attributes of the limits key .		
<code>compute</code>	string	Required. Name of the compute target to execute the job on, using the <code>azureml:<compute_name></code> syntax.		
<code>trial</code>	object	Required. The job template for each trial. Each trial job will be provided with a different combination of hyperparameter values that the system samples from the <code>search_space</code> . See Attributes of the trial key .		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>inputs</code>	object	<p>Dictionary of inputs to the job. The key is a name for the input within the context of the job and the value is the input value.</p> <p>Inputs can be referenced in the <code>command</code> using the <code>\${{ inputs.<input_name> }}</code> expression.</p>		
<code>inputs.<input_name></code>	number, integer, boolean, string or object	<p>One of a literal value (of type number, integer, boolean, or string) or an object containing a job input data specification.</p>		
<code>outputs</code>	object	<p>Dictionary of output configurations of the job. The key is a name for the output within the context of the job and the value is the output configuration.</p> <p>Outputs can be referenced in the <code>command</code> using the <code>\${{ outputs.<output_name> }}</code> expression.</p>		
<code>outputs.<output_name></code>	object	<p>You can leave the object empty, in which case by default the output will be of type <code>uri_folder</code> and Azure ML will system-generate an output location for the output. File(s) to the output directory will be written via read-write mount. If you want to specify a different mode for the output, provide an object containing the job output specification.</p>		

Sampling algorithms

RandomSamplingAlgorithm

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. The type of sampling algorithm.	<code>random</code>	
<code>seed</code>	integer	A random seed to use for initializing the random number generation. If omitted, the default seed value will be null.		
<code>rule</code>	string	The type of random sampling to use. The default, <code>random</code> , will use simple uniform random sampling, while <code>sobol</code> will use the Sobol quasirandom sequence.	<code>random</code> , <code>sobol</code>	<code>random</code>

GridSamplingAlgorithm

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of sampling algorithm.	<code>grid</code>

BayesianSamplingAlgorithm

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of sampling algorithm.	<code>bayesian</code>

Early termination policies

BanditPolicy

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. The type of policy.	<code>bandit</code>	
<code>slack_factor</code>	number	The ratio used to calculate the allowed distance from the best performing trial. One of <code>slack_factor</code> or <code>slack_amount</code> is required.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>slack_amount</code>	number	The absolute distance allowed from the best performing trial. One of <code>slack_factor</code> or <code>slack_amount</code> is required.		
<code>evaluation_interval</code>	integer	The frequency for applying the policy.		<code>1</code>
<code>delay_evaluation</code>	integer	The number of intervals for which to delay the first policy evaluation. If specified, the policy applies on every multiple of <code>evaluation_interval</code> that is greater than or equal to <code>delay_evaluation</code> .		<code>0</code>

MedianStoppingPolicy

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. The type of policy.	<code>median_stopping</code>	
<code>evaluation_interval</code>	integer	The frequency for applying the policy.		<code>1</code>
<code>delay_evaluation</code>	integer	The number of intervals for which to delay the first policy evaluation. If specified, the policy applies on every multiple of <code>evaluation_interval</code> that is greater than or equal to <code>delay_evaluation</code> .		<code>0</code>

TruncationSelectionPolicy

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. The type of policy.	<code>truncation_selection</code>	
<code>truncation_percentage</code>	integer	Required. The percentage of trial jobs to cancel at each evaluation interval.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>evaluation_interval</code>	integer	The frequency for applying the policy.		<code>1</code>
<code>delay_evaluation</code>	integer	The number of intervals for which to delay the first policy evaluation. If specified, the policy applies on every multiple of <code>evaluation_interval</code> that is greater than or equal to <code>delay_evaluation</code> .		<code>0</code>

Parameter expressions

choice

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of expression.	<code>choice</code>
<code>values</code>	array	Required. The list of discrete values to choose from.	

randint

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of expression.	<code>randint</code>
<code>upper</code>	integer	Required. The exclusive upper bound for the range of integers.	

qlognormal, qnormal

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of expression.	<code>qlognormal</code> , <code>qnormal</code>
<code>mu</code>	number	Required. The mean of the normal distribution.	
<code>sigma</code>	number	Required. The standard deviation of the normal distribution.	
<code>q</code>	integer	Required. The smoothing factor.	

qloguniform, quniform

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of expression.	<code>qloguniform</code> , <code>quniform</code>
<code>min_value</code>	number	Required. The minimum value in the range (inclusive).	
<code>max_value</code>	number	Required. The maximum value in the range (inclusive).	
<code>q</code>	integer	Required. The smoothing factor.	

lognormal, normal

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of expression.	<code>lognormal</code> , <code>normal</code>
<code>mu</code>	number	Required. The mean of the normal distribution.	
<code>sigma</code>	number	Required. The standard deviation of the normal distribution.	

loguniform

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of expression.	<code>loguniform</code>
<code>min_value</code>	number	Required. The minimum value in the range will be <code>exp(min_value)</code> (inclusive).	
<code>max_value</code>	number	Required. The maximum value in the range will be <code>exp(max_value)</code> (inclusive).	

uniform

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. The type of expression.	<code>uniform</code>
<code>min_value</code>	number	Required. The minimum value in the range (inclusive).	

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>max_value</code>	number	Required. The maximum value in the range (inclusive).	

Attributes of the `limits` key

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>max_total_trials</code>	integer	The maximum time in seconds the job is allowed to run. Once this limit is reached the system will cancel the job.	<code>1000</code>
<code>max_concurrent_trials</code>	integer		Defaults to <code>max_total_trials</code> .
<code>timeout</code>	integer	The maximum time in seconds the entire sweep job is allowed to run. Once this limit is reached the system will cancel the sweep job, including all its trials.	<code>604800</code>
<code>trial_timeout</code>	integer	The maximum time in seconds each trial job is allowed to run. Once this limit is reached the system will cancel the trial.	

Attributes of the `trial` key

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>command</code>	string	Required. The command to execute.	
<code>code</code>	string	Local path to the source code directory to be uploaded and used for the job.	

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>environment</code>	string or object	<p>Required. The environment to use for the job. This can be either a reference to an existing versioned environment in the workspace or an inline environment specification.</p> <p>To reference an existing environment use the <code>azureml:<environment-name>:<environment-version></code> syntax.</p> <p>To define an environment inline please follow the Environment schema. Exclude the <code>name</code> and <code>version</code> properties as they are not supported for inline environments.</p>	
<code>environment_variables</code>	object	Dictionary of environment variable name-value pairs to set on the process where the command is executed.	
<code>distribution</code>	object	The distribution configuration for distributed training scenarios. One of MpiConfiguration , PyTorchConfiguration , or TensorFlowConfiguration .	
<code>resources.instance_count</code>	integer	The number of nodes to use for the job.	1

Distribution configurations

MpiConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. Distribution type.	<code>mpi</code>
<code>process_count_per_instance</code>	integer	Required. The number of processes per node to launch for the job.	

PyTorchConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. Distribution type.	<code>pytorch</code>	

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>process_count_per_instance</code>	integer	The number of processes per node to launch for the job.		1

TensorFlow Configuration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. Distribution type.	<code>tensorflow</code>	
<code>worker_count</code>	integer	The number of workers to launch for the job.		Defaults to <code>resources.instance_count</code> .
<code>parameter_server_count</code>	integer	The number of parameter servers to launch for the job.		0

Job inputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	The type of job input. Specify <code>uri_file</code> for input data that points to a single file source, or <code>uri_folder</code> for input data that points to a folder source. Learn more about data access.	<code>uri_file</code> , <code>uri_folder</code> , <code>mltable</code> , <code>mlflow_model</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
path	string	<p>The path to the data to use as input. This can be specified in a few ways:</p> <ul style="list-style-type: none"> - A local path to the data source file or folder, e.g. <pre>path: ./iris.csv</pre> <p>The data will get uploaded during job submission.</p> <ul style="list-style-type: none"> - A URI of a cloud path to the file or folder to use as the input. Supported URI types are <code>azureml</code>, <code>https</code>, <code>wasbs</code>, <code>abfss</code>, <code>adl</code>. See Core yaml syntax for more information on how to use the <code>azureml://</code> URI format. - An existing registered Azure ML data asset to use as the input. To reference a registered data asset use the <code>azureml:<data_name>:<data_version></code> syntax or <code>azureml:<data_name>@latest</code> (to reference the latest version of that data asset), e.g. <pre>path: azureml:cifar10- data:1</pre> <p>or</p> <pre>path: azureml:cifar10- data@latest</pre> <p>.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>mode</code>	string	<p>Mode of how the data should be delivered to the compute target.</p> <p>For read-only mount (<code>ro_mount</code>), the data will be consumed as a mount path. A folder will be mounted as a folder and a file will be mounted as a file. Azure ML will resolve the input to the mount path.</p> <p>For <code>download</code> mode the data will be downloaded to the compute target. Azure ML will resolve the input to the downloaded path.</p> <p>If you only want the URL of the storage location of the data artifact(s) rather than mounting or downloading the data itself, you can use the <code>direct</code> mode. This will pass in the URL of the storage location as the job input. Note that in this case you are fully responsible for handling credentials to access the storage.</p>	<code>ro_mount</code> , <code>download</code> , <code>direct</code>	<code>ro_mount</code>

Job outputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	The type of job output. For the default <code>uri_folder</code> type, the output will correspond to a folder.	<code>uri_file</code> , <code>uri_folder</code> , <code>mltable</code> , <code>mlflow_model</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>mode</code>	string	Mode of how output file(s) will get delivered to the destination storage. For read-write mount mode (<code>rw_mount</code>) the output directory will be a mounted directory. For upload mode the file(s) written will get uploaded at the end of the job.	<code>rw_mount</code> , <code>upload</code>	<code>rw_mount</code>

Remarks

The `az ml job` command can be used for managing Azure Machine Learning jobs.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: hello sweep

```
$schema: https://azureschemas.azureedge.net/latest/sweepJob.schema.json
type: sweep
trial:
  command: >-
    python hello-sweep.py
    --A ${{inputs.A}}
    --B ${{search_space.B}}
    --C ${{search_space.C}}
  code: src
  environment: azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
inputs:
  A: 0.5
compute: azureml:cpu-cluster
sampling_algorithm: random
search_space:
  B:
    type: choice
    values: ["hello", "world", "hello_world"]
  C:
    type: uniform
    min_value: 0.1
    max_value: 1.0
objective:
  goal: minimize
  primary_metric: random_metric
limits:
  max_total_trials: 4
  max_concurrent_trials: 2
  timeout: 3600
display_name: hello-sweep-example
experiment_name: hello-sweep-example
description: Hello sweep job example.
```

YAML: basic Python model hyperparameter tuning

```
$schema: https://azuremlschemas.azureedge.net/latest/sweepJob.schema.json
type: sweep
trial:
  code: src
  command: >-
    python main.py
    --iris-csv ${inputs.iris_csv}
    --C ${search_space.C}
    --kernel ${search_space.kernel}
    --coef0 ${search_space.coef0}
environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
inputs:
  iris_csv:
    type: uri_file
    path: wasbs://datasets@azuremlexamples.blob.core.windows.net/iris.csv
compute: azureml:cpu-cluster
sampling_algorithm: random
search_space:
  C:
    type: uniform
    min_value: 0.5
    max_value: 0.9
  kernel:
    type: choice
    values: ["rbf", "linear", "poly"]
  coef0:
    type: uniform
    min_value: 0.1
    max_value: 1
objective:
  goal: minimize
  primary_metric: training_f1_score
limits:
  max_total_trials: 20
  max_concurrent_trials: 10
  timeout: 7200
display_name: sklearn-iris-sweep-example
experiment_name: sklearn-iris-sweep-example
description: Sweep hyperparameters for training a scikit-learn SVM on the Iris dataset.
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) pipeline job YAML schema

9/22/2022 • 6 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	const	Required. The type of job.	<code>pipeline</code>	
<code>name</code>	string	Name of the job. Must be unique across all jobs in the workspace. If omitted, Azure ML will autogenerated a GUID for the name.		
<code>display_name</code>	string	Display name of the job in the studio UI. Can be non-unique within the workspace. If omitted, Azure ML will autogenerated a human-readable adjective-noun identifier for the display name.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>experiment_name</code>	string	Experiment name to organize the job under. Each job's run record will be organized under the corresponding experiment in the studio's "Experiments" tab. If omitted, Azure ML will default it to the name of the working directory where the job was created.		
<code>description</code>	string	Description of the job.		
<code>tags</code>	object	Dictionary of tags for the job.		
<code>settings</code>	object	Default settings for the pipeline job. See Attributes of the <code>settings</code> key for the set of configurable properties.		
<code>jobs</code>	object	<p>Required. Dictionary of the set of individual jobs to run as steps within the pipeline. These jobs are considered child jobs of the parent pipeline job.</p> <p>The key is the name of the step within the context of the pipeline job. This name is different from the unique job name of the child job. The value is the job specification, which can follow the command job schema or sweep job schema. Currently only command jobs and sweep jobs can be run in a pipeline. Later releases will have support for other job types.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>inputs</code>	object	<p>Dictionary of inputs to the pipeline job. The key is a name for the input within the context of the job and the value is the input value.</p> <p>These pipeline inputs can be referenced by the inputs of an individual step job in the pipeline using the</p> <pre><code>\${ parent.inputs. <input_name> }</code></pre> <p>expression. For more information on how to bind the inputs of a pipeline step to the inputs of the top-level pipeline job, see the Expression syntax for binding inputs and outputs between steps in a pipeline job.</p>		
<code>inputs.<input_name></code>	number, integer, boolean, string or object	One of a literal value (of type number, integer, boolean, or string) or an object containing a job input data specification .		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>outputs</code>	object	<p>Dictionary of output configurations of the pipeline job. The key is a name for the output within the context of the job and the value is the output configuration.</p> <p>These pipeline outputs can be referenced by the outputs of an individual step job in the pipeline using the <code>\${{ parents.outputs.<output_name> }}</code> expression. For more information on how to bind the inputs of a pipeline step to the inputs of the top-level pipeline job, see the Expression syntax for binding inputs and outputs between steps in a pipeline job.</p>		
<code>outputs.<output_name></code>	object	<p>You can leave the object empty, in which case by default the output will be of type <code>uri_folder</code> and Azure ML will system-generate an output location for the output based on the following templated path: <code>{settings.datastore}/azureml/{job-name}/{output-name}/</code></p> <p>. File(s) to the output directory will be written via read-write mount. If you want to specify a different mode for the output, provide an object containing the job output specification.</p>		

Attributes of the `settings` key

KEY	TYPE	DESCRIPTION	DEFAULT VALUE

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>default_datastore</code>	string	<p>Name of the datastore to use as the default datastore for the pipeline job. This value must be a reference to an existing datastore in the workspace using the <code>azureml:<datastore-name></code> syntax. Any outputs defined in the <code>outputs</code> property of the parent pipeline job or child step jobs will be stored in this datastore. If omitted, outputs will be stored in the workspace blob datastore.</p>	
<code>default_compute</code>	string	<p>Name of the compute target to use as the default compute for all steps in the pipeline. If compute is defined at the step level, it will override this default compute for that specific step. This value must be a reference to an existing compute in the workspace using the <code>azureml:<compute-name></code> syntax.</p>	
<code>continue_on_step_failure</code>	boolean	<p>Whether the execution of steps in the pipeline should continue if one step fails. The default value is <code>False</code>, which means that if one step fails, the pipeline execution will be stopped, canceling any running steps.</p>	<code>False</code>

Job inputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	<p>The type of job input. Specify <code>uri_file</code> for input data that points to a single file source, or <code>uri_folder</code> for input data that points to a folder source. Learn more about data access.</p>	<code>uri_file</code> , <code>uri_folder</code> , <code>mltable</code> , <code>mlflow_model</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
path	string	<p>The path to the data to use as input. This can be specified in a few ways:</p> <ul style="list-style-type: none"> - A local path to the data source file or folder, e.g. <pre>path: ./iris.csv</pre> <p>The data will get uploaded during job submission.</p> <ul style="list-style-type: none"> - A URI of a cloud path to the file or folder to use as the input. Supported URI types are <code>azureml</code>, <code>https</code>, <code>wasbs</code>, <code>abfss</code>, <code>adl</code>. See Core yaml syntax for more information on how to use the <code>azureml://</code> URI format. - An existing registered Azure ML data asset to use as the input. To reference a registered data asset use the <code>azureml:<data_name>:<data_version></code> syntax or <code>azureml:<data_name>@latest</code> (to reference the latest version of that data asset), e.g. <pre>path: azureml:cifar10- data:1</pre> <p>or</p> <pre>path: azureml:cifar10- data@latest</pre> <p>.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>mode</code>	string	<p>Mode of how the data should be delivered to the compute target.</p> <p>For read-only mount (<code>ro_mount</code>), the data will be consumed as a mount path. A folder will be mounted as a folder and a file will be mounted as a file. Azure ML will resolve the input to the mount path.</p> <p>For <code>download</code> mode the data will be downloaded to the compute target. Azure ML will resolve the input to the downloaded path.</p> <p>If you only want the URL of the storage location of the data artifact(s) rather than mounting or downloading the data itself, you can use the <code>direct</code> mode. This will pass in the URL of the storage location as the job input. Note that in this case you are fully responsible for handling credentials to access the storage.</p>	<code>ro_mount</code> , <code>download</code> , <code>direct</code>	<code>ro_mount</code>

Job outputs

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	The type of job output. For the default <code>uri_folder</code> type, the output will correspond to a folder.	<code>uri_file</code> , <code>uri_folder</code> , <code>mltable</code> , <code>mlflow_model</code>	<code>uri_folder</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
mode	string	Mode of how output file(s) will get delivered to the destination storage. For read-write mount mode (<code>rw_mount</code>) the output directory will be a mounted directory. For upload mode the file(s) written will get uploaded at the end of the job.	<code>rw_mount</code> , <code>upload</code>	<code>rw_mount</code>

Remarks

The `az ml job` commands can be used for managing Azure Machine Learning pipeline jobs.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: hello pipeline

```
$schema: https://azureschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline
jobs:
  hello_job:
    command: echo "hello"
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    compute: azureml:cpu-cluster
  world_job:
    command: echo "world"
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    compute: azureml:cpu-cluster
```

YAML: input/output dependency

```
$schema: https://azureschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline_io
jobs:
  hello_job:
    command: echo "hello" && echo "world" > ${{outputs.world_output}}/world.txt
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
    compute: azureml:cpu-cluster
    outputs:
      world_output:
  world_job:
    command: cat ${{inputs.world_input}}/world.txt
    environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:23
    compute: azureml:cpu-cluster
    inputs:
      world_input: ${parent.jobs.hello_job.outputs.world_output}
```

YAML: common pipeline job settings

```
$schema: https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline_settings

settings:
    default_datastore: azureml:workspaceblobstore
    default_compute: azureml:cpu-cluster
jobs:
    hello_job:
        command: echo 202204190 & echo "hello"
        environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:23
    world_job:
        command: echo 202204190 & echo "hello"
        environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:23
```

YAML: top-level input and overriding common pipeline job settings

```
$schema: https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: hello_pipeline_abc
compute: azureml:cpu-cluster

inputs:
    hello_string_top_level_input: "hello world"
jobs:
    a:
        command: echo hello ${{inputs.hello_string}}
        environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
        inputs:
            hello_string: ${{parent.inputs.hello_string_top_level_input}}
    b:
        command: echo "world" >> ${{outputs.world_output}}/world.txt
        environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
        outputs:
            world_output:
    c:
        command: echo ${{inputs.world_input}}/world.txt
        environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
        inputs:
            world_input: ${{parent.jobs.b.outputs.world_output}}
```

Next steps

- [Install and use the CLI \(v2\)](#)
- [Create ML pipelines using components](#)

CLI (v2) Azure Blob datastore YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/azureBlob.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of datastore.	<code>azure_blob</code>	
<code>name</code>	string	Required. Name of the datastore.		
<code>description</code>	string	Description of the datastore.		
<code>tags</code>	object	Dictionary of tags for the datastore.		
<code>account_name</code>	string	Required. Name of the Azure storage account.		
<code>container_name</code>	string	Required. Name of the container.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>endpoint</code>	string	Endpoint suffix of the storage service, which is used for creating the storage account endpoint URL by combining the storage account name and <code>endpoint</code> . Example storage account URL: <code>https://<storage-account-name>.blob.core.windows.net</code>		<code>core.windows.net</code>
<code>protocol</code>	string	Protocol to use to connect to the container.	<code>https</code> , <code>wasbs</code>	<code>https</code>
<code>credentials</code>	object	Credential-based authentication credentials for connecting to the Azure storage account. You can provide either an account key or a shared access signature (SAS) token. Credential secrets are stored in the workspace key vault.		
<code>credentials.account_key</code>	string	The account key for accessing the storage account. One of <code>credentials.account_key</code> or <code>credentials.sas_token</code> is required if <code>credentials</code> is specified.		
<code>credentials.sas_token</code>	string	The SAS token for accessing the storage account. One of <code>credentials.account_key</code> or <code>credentials.sas_token</code> is required if <code>credentials</code> is specified.		

Remarks

The `az ml datastore` command can be used for managing Azure Machine Learning datastores.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: identity-based access

```
$schema: https://azurermschemas.azureedge.net/latest/azureBlob.schema.json
name: blob_credless_example
type: azure_blob
description: Credential-less datastore pointing to a blob container.
account_name: mytestblobstore
container_name: data-container
```

YAML: account key

YAML: wasbs protocol

```
$schema: https://azuremlschemas.azureedge.net/latest/azureBlob.schema.json
name: blob_protocol_example
type: azure_blob
description: Datastore pointing to a blob container using wasbs protocol.
account_name: mytestblobstore
protocol: wasbs
container_name: data-container
credentials:
    account_key: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

YAML: sas token

Next steps

- Install and use the CLI (y?)

CLI (v2) Azure Files datastore YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

The source JSON schema can be found at <https://azuremlschemas.azureedge.net/latest/azureFile.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of datastore.	<code>azure_file</code>	
<code>name</code>	string	Required. Name of the datastore.		
<code>description</code>	string	Description of the datastore.		
<code>tags</code>	object	Dictionary of tags for the datastore.		
<code>account_name</code>	string	Required. Name of the Azure storage account.		
<code>file_share_name</code>	string	Required. Name of the file share.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>endpoint</code>	string	Endpoint suffix of the storage service, which is used for creating the storage account endpoint URL by combining the storage account name and <code>endpoint</code> . Example storage account URL: <code>https://<storage-account-name>.file.core.windows.net</code>		<code>core.windows.net</code>
<code>protocol</code>	string	Protocol to use to connect to the file share.	<code>https</code>	<code>https</code>
<code>credentials</code>	object	Credential-based authentication credentials for connecting to the Azure storage account. You can provide either an account key or a shared access signature (SAS) token. Credential secrets are stored in the workspace key vault.		
<code>credentials.account_key</code>	string	The account key for accessing the storage account. One of <code>credentials.account_key</code> or <code>credentials.sas_token</code> is required if <code>credentials</code> is specified.		
<code>credentials.sas_token</code>	string	The SAS token for accessing the storage account. One of <code>credentials.account_key</code> or <code>credentials.sas_token</code> is required if <code>credentials</code> is specified.		

Remarks

The `az ml datastore` command can be used for managing Azure Machine Learning datastores.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: account key

```
$schema: https://azurermschemas.azureedge.net/latest/azureFile.schema.json
name: file_example
type: azure_file
description: Datastore pointing to an Azure File Share.
account_name: mytestfilestore
file_share_name: my-share
credentials:
    account_key: XxXxXxXXXXXXxXxXxxXxxXXXXXXXXxXxxXxxXXXXXXXXxxxXxXxXXXXxXxXxxXxxXxxXXXXXxXxxX
```

YAML: sas token

Next steps

- Install and use the CLI (v2)

CLI (v2) Azure Data Lake Gen1 YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azuremschemas.azureedge.net/latest/azureDataLakeGen1.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of datastore.	<code>azure_data_lake_gen1</code>	
<code>name</code>	string	Required. Name of the datastore.		
<code>description</code>	string	Description of the datastore.		
<code>tags</code>	object	Dictionary of tags for the datastore.		
<code>store_name</code>	string	Required. Name of the Azure Data Lake Storage Gen1 account.		
<code>credentials</code>	object	Service principal credentials for connecting to the Azure storage account. Credential secrets are stored in the workspace key vault.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>credentials.tenant_id</code> string		The tenant ID of the service principal. Required if <code>credentials</code> is specified.		
<code>credentials.client_id</code> string		The client ID of the service principal. Required if <code>credentials</code> is specified.		
<code>credentials.client_secret</code> string		The client secret of the service principal. Required if <code>credentials</code> is specified.		
<code>credentials.resource_url</code> string		The resource URL that determines what operations will be performed on the Azure Data Lake Storage Gen1 account.		<code>https://datalake.azure.net/</code>
<code>credentials.authority_url</code> string		The authority URL used to authenticate the user.		<code>https://login.microsoftonline.com/</code>

Remarks

The `az ml datastore` command can be used for managing Azure Machine Learning datastores.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: identity-based access

```
$schema: https://azurermschemas.azureedge.net/latest/azureDataLakeGen1.schema.json
name: alds_gen1_credless_example
type: azure_data_lake_gen1
description: Credential-less datastore pointing to an Azure Data Lake Storage Gen1.
store_name: mytestdatalakegen1
```

YAML: tenant ID, client ID, client secret

```
$schema: https://azurermschemas.azureedge.net/latest/azureDataLakeGen1.schema.json
name: adls_gen1_example
type: azure_data_lake_gen1
description: Datastore pointing to an Azure Data Lake Storage Gen1.
store_name: mytestdatalakegen1
credentials:
  tenant_id: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
  client_id: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
  client_secret: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) Azure Data Lake Gen2 YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/azureDataLakeGen2.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	string	Required. The type of datastore.	<code>azure_data_lake_gen2</code>	
<code>name</code>	string	Required. Name of the datastore.		
<code>description</code>	string	Description of the datastore.		
<code>tags</code>	object	Dictionary of tags for the datastore.		
<code>account_name</code>	string	Required. Name of the Azure storage account.		
<code>filesystem</code>	string	Required. Name of the file system. The parent directory that contains the files and folders. This is equivalent to a container in Azure Blob storage.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>endpoint</code>	string	Endpoint suffix of the storage service, which is used for creating the storage account endpoint URL by combining the storage account name and <code>endpoint</code> . Example storage account URL: <code>https://<storage-account-name>.dfs.core.windows.net</code>		<code>core.windows.net</code>
<code>protocol</code>	string	Protocol to use to connect to the file system.	<code>https</code> , <code>abfss</code>	<code>https</code>
<code>credentials</code>	object	Service principal credentials for connecting to the Azure storage account. Credential secrets are stored in the workspace key vault.		
<code>credentials.tenant_id</code> string		The tenant ID of the service principal. Required if <code>credentials</code> is specified.		
<code>credentials.client_id</code> string		The client ID of the service principal. Required if <code>credentials</code> is specified.		
<code>credentials.client_secret</code> string		The client secret of the service principal. Required if <code>credentials</code> is specified.		
<code>credentials.resource_uri</code> string		The resource URL that determines what operations will be performed on the Azure Data Lake Storage Gen2 account.		<code>https://storage.azure.com/</code>
<code>credentials.authority_string</code>		The authority URL used to authenticate the user.		<code>https://login.microsoftonline.com/</code>

Remarks

The `az ml datastore` command can be used for managing Azure Machine Learning datastores.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: identity-based access

```
$schema: https://azurermschemas.azureedge.net/latest/azureDataLakeGen2.schema.json
name: adls_gen2_credless_example
type: azure_data_lake_gen2
description: Credential-less datastore pointing to an Azure Data Lake Storage Gen2.
account_name: mytestdatalakegen2
filesystem: my-gen2-container
```

YAML: tenant ID, client ID, client secret

```
$schema: https://azurermschemas.azureedge.net/latest/azureDataLakeGen2.schema.json
name: adls_gen2_example
type: azure_data_lake_gen2
description: Datastore pointing to an Azure Data Lake Storage Gen2.
account_name: mytestdatalakegen2
filesystem: my-gen2-container
credentials:
  tenant_id: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
  client_id: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
  client_secret: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) online endpoint YAML schema

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

NOTE

A fully specified sample YAML for online endpoints is available for [reference](#)

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the endpoint. Needs to be unique at the Azure region level. Naming rules are defined under managed online endpoint limits .		
<code>description</code>	string	Description of the endpoint.		
<code>tags</code>	object	Dictionary of tags for the endpoint.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>auth_mode</code>	string	The authentication method for the endpoint. Key-based authentication and Azure ML token-based authentication are supported. Key-based authentication doesn't expire but Azure ML token-based authentication does.	<code>key</code> , <code>aml_token</code>	<code>key</code>
<code>compute</code>	string	Name of the compute target to run the endpoint deployments on. This field is only applicable for endpoint deployments to Azure Arc-enabled Kubernetes clusters (the compute target specified in this field must have <code>type: kubernetes</code>). Don't specify this field if you're doing managed online inference.		
<code>identity</code>	object	The managed identity configuration for accessing Azure resources for endpoint provisioning and inference.		
<code>identity.type</code>	string	The type of managed identity. If the type is <code>user_assigned</code> , the <code>identity.user_assigned_identities</code> property must also be specified.	<code>system_assigned</code> , <code>user_assigned</code>	
<code>identity.user_assigned_identities</code>		List of fully qualified resource IDs of the user-assigned identities.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
traffic	object	<p>Traffic represents the percentage of requests to be served by different deployments. It's represented by a dictionary of key-value pairs, where keys represent the deployment name and value represent the percentage of traffic to that deployment. For example,</p> <pre>blue: 90 green: 10</pre> <p>means 90% requests are sent to the deployment named <code>blue</code> and 10% is sent to deployment <code>green</code>. Total traffic has to either be 0 or sum up to 100. See Safe rollout for online endpoints to see the traffic configuration in action.</p> <p>Note: you can't set this field during online endpoint creation, as the deployments under that endpoint must be created before traffic can be set. You can update the traffic for an online endpoint after the deployments have been created using</p> <pre>az ml online- endpoint update</pre> <p>; for example,</p> <pre>az ml online- endpoint update - -name <endpoint_name> - -traffic "blue=90 green=10"</pre> <p>.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>public_network_access</code>	string	This flag controls the visibility of the managed endpoint. When <code>disabled</code> , inbound scoring requests are received using the private endpoint of the Azure Machine Learning workspace and the endpoint can't be reached from public networks. This flag is applicable only for managed endpoints	<code>enabled</code> , <code>disabled</code>	<code>enabled</code>
<code>mirror_traffic</code>	string	Percentage of live traffic to mirror to a deployment. Mirroring traffic doesn't change the results returned to clients. The mirrored percentage of traffic is copied and submitted to the specified deployment so you can gather metrics and logging without impacting clients. For example, to check if latency is within acceptable bounds and that there are no HTTP errors. It's represented by a dictionary with a single key-value pair, where the key represents the deployment name and the value represents the percentage of traffic to mirror to the deployment. For more information, see Test a deployment with mirrored traffic .		

Remarks

The `az ml online-endpoint` commands can be used for managing Azure Machine Learning online endpoints.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: basic

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: my-endpoint
auth_mode: key
```

YAML: system-assigned identity

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: my-sai-endpoint
auth_mode: key
```

YAML: user-assigned identity

```
$schema: https://azurermschemas.azureedge.net/latest/managedOnlineEndpoint.schema.json
name: my-uai-endpoint
auth_mode: key
identity:
  type: user_assigned
  user_assigned_identities:
    - resource_id: user_identity_ARM_id_place_holder
```

Next steps

- [Install and use the CLI \(v2\)](#)
- Learn how to [deploy a model with a managed online endpoint](#)
- [Troubleshooting managed online endpoints deployment and scoring \(preview\)](#)

CLI (v2) batch endpoint YAML schema

9/22/2022 • 2 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurerm.schemas.azureedge.net/latest/batchEndpoint.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurerm.schemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the endpoint. Needs to be unique at the Azure region level.		
<code>description</code>	string	Description of the endpoint.		
<code>tags</code>	object	Dictionary of tags for the endpoint.		
<code>auth_mode</code>	string	The authentication method for the endpoint. Currently only Azure Active Directory (Azure AD) token-based authentication is supported.	<code>aad_token</code>	<code>aad_token</code>
<code>defaults</code>	object	Default settings for the endpoint.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
defaults.deployment_name	string	Name of the deployment that will serve as the default deployment for the endpoint.		

Remarks

The `az ml batch-endpoint` commands can be used for managing Azure Machine Learning endpoints.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: basic

```
$schema: https://azureschemas.azureedge.net/latest/batchEndpoint.schema.json
name: mybatchedp
description: my sample batch endpoint
auth_mode: aad_token
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) managed online deployment YAML schema

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/managedOnlineDeployment.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the deployment. Naming rules are defined here .		
<code>description</code>	string	Description of the deployment.		
<code>tags</code>	object	Dictionary of tags for the deployment.		
<code>endpoint_name</code>	string	Required. Name of the endpoint to create the deployment under.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>model</code>	string or object	<p>The model to use for the deployment. This value can be either a reference to an existing versioned model in the workspace or an inline model specification.</p> <p>To reference an existing model, use the <code>azureml:<model-name>:<model-version></code> syntax.</p> <p>To define a model inline, follow the Model schema.</p> <p>As a best practice for production scenarios, you should create the model separately and reference it here.</p> <p>This field is optional for custom container deployment scenarios.</p>		
<code>model_mount_path</code>	string	<p>The path to mount the model in a custom container. Applicable only for custom container deployment scenarios. If the <code>model</code> field is specified, it is mounted on this path in the container.</p>		
<code>code_configuration</code>	object	<p>Configuration for the scoring code logic.</p> <p>This field is optional for custom container deployment scenarios.</p>		
<code>code_configuration.code</code>	string	Local path to the source code directory for scoring the model.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>code_configuration.scoring_script</code>	string	Relative path to the scoring file in the source code directory.		
<code>environment_variables</code>	object	Dictionary of environment variable key-value pairs to set in the deployment container. You can access these environment variables from your scoring scripts.		
<code>environment</code>	string or object	<p>Required. The environment to use for the deployment. This value can be either a reference to an existing versioned environment in the workspace or an inline environment specification.</p> <p>To reference an existing environment, use the</p> <pre>azureml: <environment-name>: <environment-version></pre> <p>syntax.</p> <p>To define an environment inline, follow the Environment schema.</p> <p>As a best practice for production scenarios, you should create the environment separately and reference it here.</p>		
<code>instance_type</code>	string	Required. The VM size to use for the deployment. For the list of supported sizes, see Managed online endpoints SKU list .		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>instance_count</code>	integer	<p>Required. The number of instances to use for the deployment. Specify the value based on the workload you expect. For high availability, Microsoft recommends you set it to at least <code>3</code>.</p> <p><code>instance_count</code> can be updated after deployment creation using <code>az ml online-deployment update</code> command.</p> <p>We reserve an extra 20% for performing upgrades. For more information, see managed online endpoint quotas.</p>		
<code>app_insights_enabled</code>	boolean	Whether to enable integration with the Azure Application Insights instance associated with your workspace.		<code>false</code>
<code>scale_settings</code>	object	<p>The scale settings for the deployment. Currently only the <code>default</code> scale type is supported, so you do not need to specify this property.</p> <p>With this <code>default</code> scale type, you can either manually scale the instance count up and down after deployment creation by updating the <code>instance_count</code> property, or create an autoscaling policy.</p>		
<code>scale_settings.type</code>	string	The scale type.	<code>default</code>	<code>default</code>

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>request_settings</code>	object	Scoring request settings for the deployment. See RequestSettings for the set of configurable properties.		
<code>liveness_probe</code>	object	Liveness probe settings for monitoring the health of the container regularly. See ProbeSettings for the set of configurable properties.		
<code>readiness_probe</code>	object	Readiness probe settings for validating if the container is ready to serve traffic. See ProbeSettings for the set of configurable properties.		
<code>egress_public_network_strings</code>		This flag secures the deployment by restricting communication between the deployment and the Azure resources used by it. Set to <code>disabled</code> to ensure that the download of the model, code, and images needed by your deployment are secured with a private endpoint. This flag is applicable only for managed online endpoints.	<code>enabled</code> , <code>disabled</code>	<code>enabled</code>

RequestSettings

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>request_timeout_ms</code>	integer	The scoring timeout in milliseconds.	<code>5000</code>

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>max_concurrent_requests_per_instance</code>		<p>The maximum number of concurrent requests per instance allowed for the deployment.</p> <p>Do not change this setting from the default value unless instructed by Microsoft Technical Support or a member of the Azure ML team.</p>	<code>1</code>
<code>max_queue_wait_ms</code>	integer	<p>The maximum amount of time in milliseconds a request will stay in the queue.</p>	<code>500</code>

ProbeSettings

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>period</code>	integer	How often (in seconds) to perform the probe.	<code>10</code>
<code>initial_delay</code>	integer	The number of seconds after the container has started before the probe is initiated. Minimum value is <code>1</code> .	<code>10</code>
<code>timeout</code>	integer	The number of seconds after which the probe times out. Minimum value is <code>1</code> .	<code>2</code>
<code>success_threshold</code>	integer	The minimum consecutive successes for the probe to be considered successful after having failed. Minimum value is <code>1</code> .	<code>1</code>
<code>failure_threshold</code>	integer	When a probe fails, the system will try <code>failure_threshold</code> times before giving up. Giving up in the case of a liveness probe means the container will be restarted. In the case of a readiness probe the container will be marked Unready. Minimum value is <code>1</code> .	<code>30</code>

Remarks

The `az ml online-deployment` commands can be used for managing Azure Machine Learning managed online deployments.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: basic

```
$schema: https://azureschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: blue
endpoint_name: my-endpoint
model:
  path: ../../model-1/model/
code_configuration:
  code: ../../model-1/onlinescoring/
  scoring_script: score.py
environment:
  conda_file: ../../model-1/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1
instance_type: Standard_DS2_v2
instance_count: 1
```

```
$schema: https://azureschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: green
endpoint_name: my-endpoint
model:
  path: ../../model-2/model/
code_configuration:
  code: ../../model-2/onlinescoring/
  scoring_script: score.py
environment:
  conda_file: ../../model-2/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1
instance_type: Standard_DS2_v2
instance_count: 1
```

YAML: system-assigned identity

```
$schema: https://azureschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
name: blue
model:
  path: ../../model-1/model/
code_configuration:
  code: ../../model-1/onlinescoring/
  scoring_script: score_managedidentity.py
environment:
  conda_file: ../../model-1/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1
instance_type: Standard_DS2_v2
instance_count: 1
environment_variables:
  STORAGE_ACCOUNT_NAME: "storage_place_holder"
  STORAGE_CONTAINER_NAME: "container_place_holder"
  FILE_NAME: "file_place_holder"
```

YAML: user-assigned identity

```
$schema: https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.schema.json
model:
  path: ../../model-1/model/
code_configuration:
  code: ../../model-1/onlinescorer/
  scoring_script: score_managedidentity.py
environment:
  conda_file: ../../model-1/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:20210727.v1
instance_type: Standard_DS2_v2
instance_count: 1
environment_variables:
  STORAGE_ACCOUNT_NAME: "storage_place_holder"
  STORAGE_CONTAINER_NAME: "container_place_holder"
  FILE_NAME: "file_place_holder"
  UAI_CLIENT_ID: "uai_client_id_place_holder"
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) Azure Arc-enabled Kubernetes online deployment YAML schema

9/22/2022 • 5 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azuremlschemas.azureedge.net/latest/kubernetesOnlineDeployment.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the deployment. Naming rules are defined here .		
<code>description</code>	string	Description of the deployment.		
<code>tags</code>	object	Dictionary of tags for the deployment.		
<code>endpoint_name</code>	string	Required. Name of the endpoint to create the deployment under.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>model</code>	string or object	<p>The model to use for the deployment. This value can be either a reference to an existing versioned model in the workspace or an inline model specification.</p> <p>To reference an existing model, use the <code>azureml:<model-name>:<model-version></code> syntax.</p> <p>To define a model inline, follow the Model schema.</p> <p>As a best practice for production scenarios, you should create the model separately and reference it here.</p> <p>This field is optional for custom container deployment scenarios.</p>		
<code>model_mount_path</code>	string	<p>The path to mount the model in a custom container. Applicable only for custom container deployment scenarios. If the <code>model</code> field is specified, it's mounted on this path in the container.</p>		
<code>code_configuration</code>	object	<p>Configuration for the scoring code logic.</p> <p>This field is optional for custom container deployment scenarios.</p>		
<code>code_configuration.code</code>	string	Local path to the source code directory for scoring the model.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>code_configuration.scoring_script</code>	string	Relative path to the scoring file in the source code directory.		
<code>environment_variables</code>	object	Dictionary of environment variable key-value pairs to set in the deployment container. You can access these environment variables from your scoring scripts.		
<code>environment</code>	string or object	<p>Required. The environment to use for the deployment. This value can be either a reference to an existing versioned environment in the workspace or an inline environment specification.</p> <p>To reference an existing environment, use the</p> <pre>azureml: <environment-name>: <environment-version></pre> <p>syntax.</p> <p>To define an environment inline, follow the Environment schema.</p> <p>As a best practice for production scenarios, you should create the environment separately and reference it here.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>instance_type</code>	string	<p>The instance type used to place the inference workload. If omitted, the inference workload will be placed on the default instance type of the Kubernetes cluster specified in the endpoint's <code>compute</code> field. If specified, the inference workload will be placed on that selected instance type.</p> <p>The set of instance types for a Kubernetes cluster is configured via the Kubernetes cluster custom resource definition (CRD), hence they aren't part of the Azure ML YAML schema for attaching Kubernetes compute. For more information, see Create and select Kubernetes instance types.</p>		
<code>instance_count</code>	integer	<p>The number of instances to use for the deployment. Specify the value based on the workload you expect. This field is only required if you're using the <code>default</code> scale type (</p> <pre>scale_settings.type: default</pre> <p>).</p> <p><code>instance_count</code> can be updated after deployment creation using</p> <pre>az ml online-deployment update</pre> <p>command.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>app_insights_enabled</code>	boolean	Whether to enable integration with the Azure Application Insights instance associated with your workspace.		<code>false</code>
<code>scale_settings</code>	object	<p>The scale settings for the deployment. The two types of scale settings supported are the <code>default</code> scale type and the <code>target_utilization</code> scale type.</p> <p>With the <code>default</code> scale type (<code>scale_settings.type: default</code>), you can manually scale the instance count up and down after deployment creation by updating the <code>instance_count</code> property.</p> <p>To configure the <code>target_utilization</code> scale type (<code>scale_settings.type: target_utilization</code>), see TargetUtilizationScale Settings for the set of configurable properties.</p>		
<code>scale_settings.type</code>	string	The scale type.	<code>default</code> , <code>target_utilization</code>	<code>target_utilization</code>
<code>request_settings</code>	object	Scoring request settings for the deployment. See RequestSettings for the set of configurable properties.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>liveness_probe</code>	object	Liveness probe settings for monitoring the health of the container regularly. See ProbeSettings for the set of configurable properties.		
<code>readiness_probe</code>	object	Readiness probe settings for validating if the container is ready to serve traffic. See ProbeSettings for the set of configurable properties.		
<code>resources</code>	object	Container resource requirements.		
<code>resources.requests</code>	object	Resource requests for the container. See ContainerResourceRequests for the set of configurable properties.		
<code>resources.limits</code>	object	Resource limits for the container. See ContainerResourceLimits for the set of configurable properties.		

RequestSettings

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>request_timeout_ms</code>	integer	The scoring timeout in milliseconds.	5000
<code>max_concurrent_requests_per_interface</code>		<p>The maximum number of concurrent requests per instance allowed for the deployment.</p> <p>Do not change this setting from the default value unless instructed by Microsoft Technical Support or a member of the Azure ML team.</p>	1

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>max_queue_wait_ms</code>	integer	The maximum amount of time in milliseconds a request will stay in the queue.	500

ProbeSettings

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>period</code>	integer	How often (in seconds) to perform the probe.	10
<code>initial_delay</code>	integer	The number of seconds after the container has started before the probe is initiated. Minimum value is 1.	10
<code>timeout</code>	integer	The number of seconds after which the probe times out. Minimum value is 1.	2
<code>success_threshold</code>	integer	The minimum consecutive successes for the probe to be considered successful after having failed. Minimum value is 1.	1
<code>failure_threshold</code>	integer	When a probe fails, the system will try <code>failure_threshold</code> times before giving up. Giving up in the case of a liveness probe means the container will be restarted. In the case of a readiness probe the container will be marked Unready. Minimum value is 1.	30

TargetUtilizationScaleSettings

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>type</code>	const	The scale type	<code>target_utilization</code>
<code>min_instances</code>	integer	The minimum number of instances to use.	1
<code>max_instances</code>	integer	The maximum number of instances to scale to.	1
<code>target_utilization_percentage</code>	integer	The target CPU usage for the autoscaler.	70

KEY	TYPE	DESCRIPTION	DEFAULT VALUE
<code>polling_interval</code>	integer	How often the autoscaler should attempt to scale the deployment, in seconds.	1

ContainerResourceRequests

KEY	TYPE	DESCRIPTION
<code>cpu</code>	string	The number of CPU cores requested for the container.
<code>memory</code>	string	The memory size requested for the container
<code>nvidia.com/gpu</code>	string	The number of Nvidia GPU cards requested for the container

ContainerResourceLimits

KEY	TYPE	DESCRIPTION
<code>cpu</code>	string	The limit for the number of CPU cores for the container.
<code>memory</code>	string	The limit for the memory size for the container.
<code>nvidia.com/gpu</code>	string	The limit for the number of Nvidia GPU cards for the container

Remarks

The `az ml online-deployment` commands can be used for managing Azure Machine Learning Kubernetes online deployments.

Examples

Examples are available in the [examples GitHub repository](#).

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) batch deployment YAML schema

9/22/2022 • 3 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azuremlschemas.azureedge.net/latest/batchDeployment.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azuremlschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>name</code>	string	Required. Name of the deployment.		
<code>description</code>	string	Description of the deployment.		
<code>tags</code>	object	Dictionary of tags for the deployment.		
<code>endpoint_name</code>	string	Required. Name of the endpoint to create the deployment under.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>model</code>	string or object	<p>Required. The model to use for the deployment. This value can be either a reference to an existing versioned model in the workspace or an inline model specification.</p> <p>To reference an existing model, use the <code>azureml:<model-name>:<model-version></code> syntax.</p> <p>To define a model inline, follow the Model schema.</p> <p>As a best practice for production scenarios, you should create the model separately and reference it here.</p>		
<code>code_configuration</code>	object	<p>Configuration for the scoring code logic.</p> <p>This property is not required if your model is in MLflow format.</p>		
<code>code_configuration.code</code>	string	The local directory that contains all the Python source code to score the model.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>code_configuration.scoring_script</code>	String	<p>The Python file in the above directory. This file must have an <code>init()</code> function and a <code>run()</code> function. Use the <code>init()</code> function for any costly or common preparation (for example, load the model in memory). <code>init()</code> will be called only once at beginning of process. Use <code>run(mini_batch)</code> to score each entry; the value of <code>mini_batch</code> is a list of file paths. The <code>run()</code> function should return a pandas DataFrame or an array. Each returned element indicates one successful run of input element in the <code>mini_batch</code>. For more information on how to author scoring script, see Understanding the scoring script.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>environment</code>	string or object	<p>The environment to use for the deployment. This value can be either a reference to an existing versioned environment in the workspace or an inline environment specification.</p> <p>This property is not required if your model is in MLflow format.</p> <p>To reference an existing environment, use the</p> <pre>azureml: <environment-name>: <environment-version></pre> <p>syntax.</p> <p>To define an environment inline, follow the Environment schema.</p> <p>As a best practice for production scenarios, you should create the environment separately and reference it here.</p>		
<code>compute</code>	string	<p>Required. Name of the compute target to execute the batch scoring jobs on. This value should be a reference to an existing compute in the workspace using the</p> <pre>azureml:<compute-name></pre> <p>syntax.</p>		
<code>resources.instance_count</code>	integer	The number of nodes to use for each batch scoring job.		1
<code>max_concurrency_per_instance</code>	integer	The maximum number of parallel <code>scoring_script</code> runs per instance.		1

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>error_threshold</code>	integer	The number of file failures that should be ignored. If the error count for the entire input goes above this value, the batch scoring job will be terminated. <code>error_threshold</code> is for the entire input and not for individual mini batches. If omitted, any number of file failures will be allowed without terminating the job.		-1
<code>logging_level</code>	string	The log verbosity level.	<code>warning</code> , <code>info</code> , <code>debug</code>	<code>info</code>
<code>mini_batch_size</code>	integer	The number of files the <code>code_configuration.scoring_script</code> can process in one <code>run()</code> call.		10
<code>retry_settings</code>	object	Retry settings for scoring each mini batch.		
<code>retry_settings.max_retries</code>	integer	The maximum number of retries for a failed or timed-out mini batch.		3
<code>retry_settings.timeout</code>	integer	The timeout in seconds for scoring a mini batch.		30
<code>output_action</code>	string	Indicates how the output should be organized in the output file.	<code>append_row</code> , <code>summary_only</code>	<code>append_row</code>
<code>output_file_name</code>	string	Name of the batch scoring output file.		<code>predictions.csv</code>
<code>environment_variables</code>	object	Dictionary of environment variable key-value pairs to set for each batch scoring job.		

Remarks

The `az ml batch-deployment` commands can be used for managing Azure Machine Learning batch deployments.

Examples

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: basic (MLflow)

```
$schema: https://azuremlschemas.azureedge.net/latest/batchDeployment.schema.json
name: mlflowdp
endpoint_name: mybatchedp
model:
  path: ./autolog_nyc_taxi
compute: azureml:batch-cluster
```

YAML: custom model and scoring code

```
$schema: https://azuremlschemas.azureedge.net/latest/batchDeployment.schema.json
name: nonmlflowdp
endpoint_name: mybatchedp
model:
  path: ./mnist/model/
code_configuration:
  code: ./mnist/code/
  scoring_script: digit_identification.py
environment:
  conda_file: ./mnist/environment/conda.yml
  image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04:latest
compute: azureml:batch-cluster
resources:
  instance_count: 1
  max_concurrency_per_instance: 2
  mini_batch_size: 10
  output_action: append_row
  output_file_name: predictions.csv
retry_settings:
  max_retries: 3
  timeout: 30
error_threshold: -1
logging_level: info
```

Next steps

- [Install and use the CLI \(v2\)](#)

CLI (v2) command component YAML schema

9/22/2022 • 4 minutes to read • [Edit Online](#)

APPLIES TO:  Azure CLI ml extension v2 (current)

The source JSON schema can be found at

<https://azurermschemas.azureedge.net/latest/commandComponent.schema.json>.

NOTE

The YAML syntax detailed in this document is based on the JSON schema for the latest version of the ML CLI v2 extension. This syntax is guaranteed only to work with the latest version of the ML CLI v2 extension. You can find the schemas for older extension versions at <https://azurermschemasprod.azureedge.net/>.

YAML syntax

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>\$schema</code>	string	The YAML schema. If you use the Azure Machine Learning VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.		
<code>type</code>	const	The type of component.	<code>command</code>	<code>command</code>
<code>name</code>	string	Required. Name of the component. Must start with lowercase letter. Allowed characters are lowercase letters, numbers, and underscore(_). Maximum length is 255 characters.		
<code>version</code>	string	Version of the component. If omitted, Azure ML will autogenerated a version.		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>display_name</code>	string	Display name of the component in the studio UI. Can be non-unique within the workspace.		
<code>description</code>	string	Description of the component.		
<code>tags</code>	object	Dictionary of tags for the component.		
<code>command</code>	string	Required. The command to execute.		
<code>code</code>	string	Local path to the source code directory to be uploaded and used for the component.		
<code>environment</code>	string or object	<p>Required. The environment to use for the component. This value can be either a reference to an existing versioned environment in the workspace or an inline environment specification.</p> <p>To reference an existing environment, use the</p> <pre>azureml: <environment-name>: <environment-version></pre> <p>syntax.</p> <p>To define an environment inline, follow the Environment schema. Exclude the <code>name</code> and <code>version</code> properties as they are not supported for inline environments.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>distribution</code>	object	The distribution configuration for distributed training scenarios. One of MpiConfiguration , PyTorchConfiguration , or TensorFlowConfiguration .		
<code>resources.instance_count</code>	integer	The number of nodes to use for the job.		<code>1</code>
<code>inputs</code>	object	<p>Dictionary of component inputs. The key is a name for the input within the context of the component and the value is the component input definition.</p> <p>Inputs can be referenced in the <code>command</code> using the <code>\${{ inputs.<input_name> }}</code> expression.</p>		
<code>inputs.<input_name></code>	object	The component input definition. See Component input for the set of configurable properties.		
<code>outputs</code>	object	<p>Dictionary of component outputs. The key is a name for the output within the context of the component and the value is the component output definition.</p> <p>Outputs can be referenced in the <code>command</code> using the <code>\${{ outputs.<output_name> }}</code> expression.</p>		

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>outputs.<output_name></code>	object	The component output definition. See Component output for the set of configurable properties.		

Distribution configurations

MpiConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES
<code>type</code>	const	Required. Distribution type.	<code>mpi</code>
<code>process_count_per_instance</code>	integer	Required. The number of processes per node to launch for the job.	

PyTorchConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. Distribution type.	<code>pytorch</code>	
<code>process_count_per_instance</code>	integer	The number of processes per node to launch for the job.		<code>1</code>

TensorFlowConfiguration

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	const	Required. Distribution type.	<code>tensorflow</code>	
<code>worker_count</code>	integer	The number of workers to launch for the job.		Defaults to <code>resources.instance_count</code> .
<code>parameter_server_count</code>	integer	The number of parameter servers to launch for the job.		<code>0</code>

Component input

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	Required. The type of component input. Learn more about data access	<code>number</code> , <code>integer</code> , <code>boolean</code> , <code>string</code> , <code>uri_file</code> , <code>uri_folder</code> , <code>mltable</code> , <code>mlflow_model</code>	

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>description</code>	string	Description of the input.		
<code>default</code>	number, integer, boolean, or string	The default value for the input.		
<code>optional</code>	boolean	Whether the input is required. If set to <code>true</code> , you need use the command includes optional inputs with <code>\$[[]]</code>		<code>false</code>
<code>min</code>	integer or number	The minimum accepted value for the input. This field can only be specified if <code>type</code> field is <code>number</code> or <code>integer</code> .		
<code>max</code>	integer or number	The maximum accepted value for the input. This field can only be specified if <code>type</code> field is <code>number</code> or <code>integer</code> .		
<code>enum</code>	array	The list of allowed values for the input. Only applicable if <code>type</code> field is <code>string</code> .		

Component output

KEY	TYPE	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUE
<code>type</code>	string	Required. The type of component output.	<code>uri_file</code> , <code>uri_folder</code> , <code>mltable</code> , <code>mlflow_model</code>	
<code>description</code>	string	Description of the output.		

Remarks

The `az ml component` commands can be used for managing Azure Machine Learning components.

Examples

Command component examples are available in the examples GitHub repository. Select examples for are shown

below.

Examples are available in the [examples GitHub repository](#). Several are shown below.

YAML: Hello world command component

```
$schema: https://azurermschemas.azureedge.net/latest/commandComponent.schema.json
type: command

name: hello_python_world
display_name: Hello_Python_World
version: 1

code: ./src

environment:
  image: python

command: >-
  python hello.py
```

YAML: Component with different input types

```
$schema: https://azurermschemas.azureedge.net/latest/commandComponent.schema.json
name: train_data_component_cli
display_name: train_data
description: A example train component
tags:
  author: azureml-sdk-team
version: 5
type: command
inputs:
  training_data:
    type: uri_folder
  max_epochs:
    type: integer
    optional: true
  learning_rate:
    type: number
    default: 0.01
    optional: true
  learning_rate_schedule:
    type: string
    default: time-based
    optional: true
outputs:
  model_output:
    type: uri_folder
code: ./train_src
environment: azurerm:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:1
command: >-
  python train.py
  --training_data ${{inputs.training_data}}
  $[--max_epochs ${{inputs.max_epochs}}]
  $[--learning_rate ${{inputs.learning_rate}}]
  $[--learning_rate_schedule ${{inputs.learning_rate_schedule}}]
  --model_output ${{outputs.model_output}}
```

Define optional inputs in command line

When the input is set as `optional = true`, you need use `$[]` to embrace the command line with inputs. For example `$[--input1 ${{inputs.input1}}]`. The command line at runtime may have different inputs.

- If you are using only specify the required `training_data` and `model_output` parameters, the command line will look like:

```
python train.py --training_data some_input_path --learning_rate 0.01 --learning_rate_schedule time-based --model_output some_output_path
```

If no value is specified at runtime, `learning_rate` and `learning_rate_schedule` will use the default value.

- If all inputs/outputs provide values during runtime, the command line will look like:

```
python train.py --training_data some_input_path --max_epochs 10 --learning_rate 0.01 --learning_rate_schedule time-based --model_output some_output_path
```

Next steps

- [Install and use the CLI \(v2\)](#)
- [Create ML pipelines using components \(CLI v2\)](#)

Data schemas to train computer vision models with automated machine learning

9/22/2022 • 10 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

IMPORTANT

This feature is currently in public preview. This preview version is provided without a service-level agreement. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Learn how to format your JSONL files for data consumption in automated ML experiments for computer vision tasks during training and inference.

Data schema for training

Azure Machine Learning AutoML for Images requires input image data to be prepared in [JSONL](#) (JSON Lines) format. This section describes input data formats or schema for image classification multi-class, image classification multi-label, object detection, and instance segmentation. We'll also provide a sample of final training or validation JSON Lines file.

Image classification (binary/multi-class)

Input data format/schema in each JSON Line:

```
{  
    "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/<path_to_image>",  
    "image_details": {  
        "format": "image_format",  
        "width": "image_width",  
        "height": "image_height"  
    },  
    "label": "class_name",  
}
```

KEY	DESCRIPTION	EXAMPLE
<code>image_url</code>	Image location in AzureML datastore. <code>my-subscription-id</code> needs to be replaced by the Azure subscription where images are located. More information about Azure subscriptions can be found here . Similarly <code>my-resource-group</code> , <code>my-workspace</code> , <code>my-datastore</code> should be replaced by <code>resource group name</code> , <code>workspace name</code> and <code>datastore name</code> respectively. <code>path_to_image</code> should be the full path to image on datastore. <code>Required, String</code>	"azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg"

KEY	DESCRIPTION	EXAMPLE
<code>image_details</code>	Image details Optional, Dictionary	<code>"image_details": {"format": "jpg", "width": "400px", "height": "258px"}</code>
<code>format</code>	Image type (all the available Image formats in Pillow library are supported) Optional, String from {"jpg", "jpeg", "png", "jpe", "jfif", "bmp", "tif", "tiff"}	<code>"jpg" or "jpeg" or "png" or "jpe" or "jfif" or "bmp" or "tif" or "tiff"</code>
<code>width</code>	Width of the image Optional, String or Positive Integer	<code>"400px" or 400</code>
<code>height</code>	Height of the image Optional, String or Positive Integer	<code>"200px" or 200</code>
<code>label</code>	Class/label of the image Required, String	<code>"cat"</code>

Example of a JSONL file for multi-class image classification:

```
{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg", "image_details": {"format": "jpg", "width": "400px", "height": "258px"}, "label": "can"}
{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_02.jpg", "image_details": {"format": "jpg", "width": "397px", "height": "296px"}, "label": "milk_bottle"}
.
.
.
{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_n.jpg", "image_details": {"format": "jpg", "width": "1024px", "height": "768px"}, "label": "water_bottle"}
```



Image classification multi-label

The following is an example of input data format/schema in each JSON Line for image classification.

```
{
  "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/<path_to_image>",
  "image_details": {
    "format": "image_format",
    "width": "image_width",
    "height": "image_height"
  },
  "label": [
    "class_name_1",
    "class_name_2",
    "class_name_3",
    "...",
    "class_name_n"
  ]
}
```

KEY	DESCRIPTION	EXAMPLE
<code>image_url</code>	Image location in AzureML datastore. <code>my-subscription-id</code> needs to be replaced by the Azure subscription where images are located. More information about Azure subscriptions can be found here . Similarly <code>my-resource-group</code> , <code>my-workspace</code> , <code>my-datastore</code> should be replaced by resource group name , workspace name and datastore name respectively. <code>path_to_image</code> should be the full path to image on datastore. <code>Required, String</code>	"azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg"
<code>image_details</code>	Image details <code>Optional, Dictionary</code>	"image_details": {"format": "jpg", "width": "400px", "height": "258px"}
<code>format</code>	Image type (all the Image formats available in Pillow library are supported) <code>Optional, String from {"jpg", "jpeg", "png", "jpe", "jfif", "bmp", "tif", "tiff"}</code>	"jpg" or "jpeg" or "png" or "jpe" or "jfif" or "bmp" or "tif" or "tiff"
<code>width</code>	Width of the image <code>Optional, String or Positive Integer</code>	"400px" or 400
<code>height</code>	Height of the image <code>Optional, String or Positive Integer</code>	"200px" or 200
<code>label</code>	List of classes/labels in the image <code>Required, List of Strings</code>	["cat", "dog"]

Example of a JSONL file for Image Classification Multi-label:

```
{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg", "image_details": {"format": "jpg", "width": "400px", "height": "258px"}, "label": ["can"]}

{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_02.jpg", "image_details": {"format": "jpg", "width": "397px", "height": "296px"}, "label": ["can", "milk_bottle"]}

.

.

.

{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_n.jpg", "image_details": {"format": "jpg", "width": "1024px", "height": "768px"}, "label": ["carton", "milk_bottle", "water_bottle"]}
```



Object detection

The following is an example JSONL file for object detection.

```
{
  "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/<path_to_image>",
  "image_details": {
    "format": "image_format",
    "width": "image_width",
    "height": "image_height"
  },
  "label": [
    {
      "label": "class_name_1",
      "topX": "xmin/width",
      "topY": "ymin/height",
      "bottomX": "xmax/width",
      "bottomY": "ymax/height",
      "isCrowd": "isCrowd"
    },
    {
      "label": "class_name_2",
      "topX": "xmin/width",
      "topY": "ymin/height",
      "bottomX": "xmax/width",
      "bottomY": "ymax/height",
      "isCrowd": "isCrowd"
    },
    ...
  ]
}
```

Here,

- `xmin` = x coordinate of top-left corner of bounding box
- `ymin` = y coordinate of top-left corner of bounding box
- `xmax` = x coordinate of bottom-right corner of bounding box
- `ymax` = y coordinate of bottom-right corner of bounding box

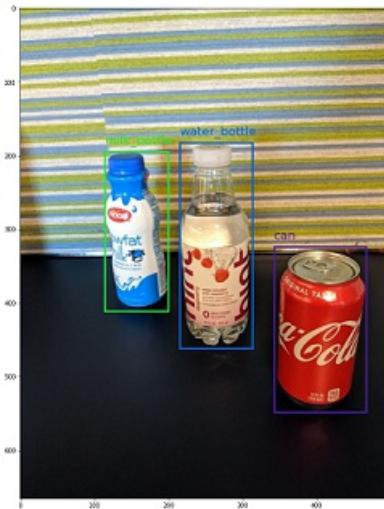
KEY	DESCRIPTION	EXAMPLE
<code>image_url</code>	<p>Image location in AzureML datastore.</p> <p><code>my-subscription-id</code> needs to be replaced by the Azure subscription where images are located. More information about Azure subscriptions can be found here. Similarly <code>my-resource-group</code>, <code>my-workspace</code>, <code>my-datastore</code> should be replaced by resource group name, workspace name and datastore name respectively.</p> <p><code>path_to_image</code> should be the full path to image on datastore.</p> <p><code>Required, String</code></p>	<code>"azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg"</code>
<code>image_details</code>	<p>Image details</p> <p><code>Optional, Dictionary</code></p>	<code>"image_details": {"format": "jpg", "width": "400px", "height": "258px"}</code>
<code>format</code>	<p>Image type (all the Image formats available in <code>Pillow</code> library are supported. But for YOLO only image formats allowed by <code>opencv</code> are supported)</p> <p><code>Optional, String from {"jpg", "jpeg", "png", "jpe", "jfif", "bmp", "tif", "tiff"}</code></p>	<code>"jpg" or "jpeg" or "png" or "jpe" or "jfif" or "bmp" or "tif" or "tiff"</code>
<code>width</code>	<p>Width of the image</p> <p><code>Optional, String or Positive Integer</code></p>	<code>"499px" or 499</code>
<code>height</code>	<p>Height of the image</p> <p><code>Optional, String or Positive Integer</code></p>	<code>"665px" or 665</code>
<code>label</code> (outer key)	<p>List of bounding boxes, where each box is a dictionary of <code>label</code>, <code>topX</code>, <code>topY</code>, <code>bottomX</code>, <code>bottomY</code>, <code>isCrowd</code> their top-left and bottom-right coordinates</p> <p><code>Required, List of dictionaries</code></p>	<code>[{"label": "cat", "topX": 0.260, "topY": 0.406, "bottomX": 0.735, "bottomY": 0.701, "isCrowd": 0}]</code>
<code>label</code> (inner key)	<p>Class/label of the object in the bounding box</p> <p><code>Required, String</code></p>	<code>"cat"</code>
<code>topX</code>	<p>Ratio of x coordinate of top-left corner of the bounding box and width of the image</p> <p><code>Required, Float in the range [0,1]</code></p>	<code>0.260</code>

KEY	DESCRIPTION	EXAMPLE
topY	Ratio of y coordinate of top-left corner of the bounding box and height of the image Required, Float in the range [0,1]	0.406
bottomX	Ratio of x coordinate of bottom-right corner of the bounding box and width of the image Required, Float in the range [0,1]	0.735
bottomY	Ratio of y coordinate of bottom-right corner of the bounding box and height of the image Required, Float in the range [0,1]	0.701
isCrowd	Indicates whether the bounding box is around the crowd of objects. If this special flag is set, we skip this particular bounding box when calculating the metric. Optional, Bool	0

Example of a JSONL file for object detection:

```
{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg", "image_details": {"format": "jpg", "width": "499px", "height": "666px"}, "label": [{"label": "can", "topX": 0.260, "topY": 0.406, "bottomX": 0.735, "bottomY": 0.701, "isCrowd": 0}]}
{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_02.jpg", "image_details": {"format": "jpg", "width": "499px", "height": "666px"}, "label": [{"label": "carton", "topX": 0.172, "topY": 0.153, "bottomX": 0.432, "bottomY": 0.659, "isCrowd": 0}, {"label": "milk_bottle", "topX": 0.300, "topY": 0.566, "bottomX": 0.891, "bottomY": 0.735, "isCrowd": 0}]}
.
.
.

{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_n.jpg", "image_details": {"format": "jpg", "width": "499px", "height": "666px"}, "label": [{"label": "carton", "topX": 0.0180, "topY": 0.297, "bottomX": 0.380, "bottomY": 0.836, "isCrowd": 0}, {"label": "milk_bottle", "topX": 0.454, "topY": 0.348, "bottomX": 0.613, "bottomY": 0.683, "isCrowd": 0}, {"label": "water_bottle", "topX": 0.667, "topY": 0.279, "bottomX": 0.841, "bottomY": 0.615, "isCrowd": 0}]}
```



Instance segmentation

For instance segmentation, automated ML only support polygon as input and output, no masks.

The following is an example JSONL file for instance segmentation.

```
{
  "image_url": "azureml://subscriptions/<my-subscription-id>/resourcegroups/<my-resource-group>/workspaces/<my-workspace>/datastores/<my-datastore>/paths/<path_to_image>",
  "image_details": {
    "format": "image_format",
    "width": "image_width",
    "height": "image_height"
  },
  "label": [
    {
      "label": "class_name",
      "isCrowd": "isCrowd",
      "polygon": [[x1, y1, x2, y2, x3, y3, ..., xn, yn]]
    }
  ]
}
```

KEY	DESCRIPTION	EXAMPLE
<code>image_url</code>	<p>Image location in AzureML datastore.</p> <p><code>my-subscription-id</code> needs to be replaced by the Azure subscription where images are located. More information about Azure subscriptions can be found here. Similarly <code>my-resource-group</code>, <code>my-workspace</code>, <code>my-datastore</code> should be replaced by resource group name, workspace name and datastore name respectively.</p> <p><code>path_to_image</code> should be the full path to image on datastore.</p> <p><code>Required, String</code></p>	<code>"azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg"</code>
<code>image_details</code>	<p>Image details</p> <p><code>Optional, Dictionary</code></p>	<code>"image_details": {"format": "jpg", "width": "400px", "height": "258px"}'</code>

KEY	DESCRIPTION	EXAMPLE
format	Image type Optional, String from {"jpg", "jpeg", "png", "jpe", "jfif", "bmp", "tif", "tiff" }	"jpg" or "jpeg" or "png" or "jpe" or "jfif" or "bmp" or "tif" or "tiff"
width	Width of the image Optional, String or Positive Integer	"499px" or 499
height	Height of the image Optional, String or Positive Integer	"665px" or 665
label (outer key)	List of masks, where each mask is a dictionary of label, isCrowd, polygon coordinates Required, List of dictionaries	[{"label": "can", "isCrowd": 0, "polygon": [[0.577, 0.689, 0.562, 0.681, 0.559, 0.686]]}]
label (inner key)	Class/label of the object in the mask Required, String	"cat"
isCrowd	Indicates whether the mask is around the crowd of objects Optional, Bool	0
polygon	Polygon coordinates for the object Required, List of list for multiple segments of the same instance. Float values in the range [0,1]	[[0.577, 0.689, 0.567, 0.689, 0.559, 0.686]]

Example of a JSONL file for Instance Segmentation:

```

{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_01.jpg", "image_details": {"format": "jpg", "width": "499px", "height": "666px"}, "label": [{"label": "can", "isCrowd": 0, "polygon": [[0.577, 0.689, 0.567, 0.689, 0.559, 0.686, 0.380, 0.593, 0.304, 0.555, 0.294, 0.545, 0.290, 0.534, 0.274, 0.512, 0.2705, 0.496, 0.270, 0.478, 0.284, 0.453, 0.308, 0.432, 0.326, 0.423, 0.356, 0.415, 0.418, 0.417, 0.635, 0.493, 0.683, 0.507, 0.701, 0.518, 0.709, 0.528, 0.713, 0.545, 0.719, 0.554, 0.719, 0.579, 0.713, 0.597, 0.697, 0.621, 0.695, 0.629, 0.631, 0.678, 0.619, 0.683, 0.595, 0.683, 0.577, 0.689]]}]}
{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_02.jpg", "image_details": {"format": "jpg", "width": "499px", "height": "666px"}, "label": [{"label": "carton", "isCrowd": 0, "polygon": [[0.240, 0.65, 0.234, 0.654, 0.230, 0.647, 0.210, 0.512, 0.202, 0.403, 0.182, 0.267, 0.184, 0.243, 0.180, 0.166, 0.186, 0.159, 0.198, 0.156, 0.396, 0.162, 0.408, 0.169, 0.406, 0.217, 0.414, 0.249, 0.422, 0.262, 0.422, 0.569, 0.342, 0.569, 0.334, 0.572, 0.320, 0.585, 0.308, 0.624, 0.306, 0.648, 0.240, 0.657]]}], {"label": "milk_bottle", "isCrowd": 0, "polygon": [[0.675, 0.732, 0.635, 0.731, 0.621, 0.725, 0.573, 0.717, 0.516, 0.717, 0.505, 0.720, 0.462, 0.722, 0.438, 0.719, 0.396, 0.719, 0.358, 0.714, 0.334, 0.714, 0.322, 0.711, 0.312, 0.701, 0.306, 0.687, 0.304, 0.663, 0.308, 0.630, 0.320, 0.596, 0.32, 0.588, 0.326, 0.579]]}]}
.
.
.

{"image_url": "azureml://subscriptions/my-subscription-id/resourcegroups/my-resource-group/workspaces/my-workspace/datastores/my-datastore/paths/image_data/Image_n.jpg", "image_details": {"format": "jpg", "width": "499px", "height": "666px"}, "label": [{"label": "water_bottle", "isCrowd": 0, "polygon": [[0.334, 0.626, 0.304, 0.621, 0.254, 0.603, 0.164, 0.605, 0.158, 0.602, 0.146, 0.602, 0.142, 0.608, 0.094, 0.612, 0.084, 0.599, 0.080, 0.585, 0.080, 0.539, 0.082, 0.536, 0.092, 0.533, 0.126, 0.530, 0.132, 0.533, 0.144, 0.533, 0.162, 0.525, 0.172, 0.525, 0.186, 0.521, 0.196, 0.521 ]]}, {"label": "milk_bottle", "isCrowd": 0, "polygon": [[0.392, 0.773, 0.380, 0.732, 0.379, 0.767, 0.367, 0.755, 0.362, 0.735, 0.362, 0.735, 0.362, 0.714, 0.352, 0.644, 0.352, 0.611, 0.362, 0.597, 0.40, 0.593, 0.444, 0.494, 0.588, 0.515, 0.585, 0.621, 0.588, 0.671, 0.582, 0.713, 0.572, 0.753 ]]}]}

```



Data format for inference

In this section, we document the input data format required to make predictions when using a deployed model. Any aforementioned image format is accepted with content type `application/octet-stream`.

Input format

The following is the input format needed to generate predictions on any task using task-specific model endpoint. After we [deploy the model](#), we can use the following code snippet to get predictions for all tasks.

```

# input image for inference
sample_image = './test_image.jpg'
# load image data
data = open(sample_image, 'rb').read()
# set the content type
headers = {'Content-Type': 'application/octet-stream'}
# if authentication is enabled, set the authorization header
headers['Authorization'] = f'Bearer {key}'
# make the request and display the response
response = requests.post(scoring_uri, data, headers=headers)

```

Output format

Predictions made on model endpoints follow different structure depending on the task type. This section explores the output data formats for multi-class, multi-label image classification, object detection, and instance segmentation tasks.

Image classification

Endpoint for image classification returns all the labels in the dataset and their probability scores for the input image in the following format.

```
{
  "filename": "/tmp/tmpppjr4et28",
  "probs": [
    2.098e-06,
    4.783e-08,
    0.999,
    8.637e-06
  ],
  "labels": [
    "can",
    "carton",
    "milk_bottle",
    "water_bottle"
  ]
}
```

Image classification multi-label

For image classification multi-label, model endpoint returns labels and their probabilities.

```
{
  "filename": "/tmp/tmppsdzxlmlm",
  "probs": [
    0.997,
    0.960,
    0.982,
    0.025
  ],
  "labels": [
    "can",
    "carton",
    "milk_bottle",
    "water_bottle"
  ]
}
```

Object detection

Object detection model returns multiple boxes with their scaled top-left and bottom-right coordinates along with box label and confidence score.

```
{  
    "filename": "/tmp/tmpdkg2wkdy",  
    "boxes": [  
        {  
            "box": {  
                "topX": 0.224,  
                "topY": 0.285,  
                "bottomX": 0.399,  
                "bottomY": 0.620  
            },  
            "label": "milk_bottle",  
            "score": 0.937  
        },  
        {  
            "box": {  
                "topX": 0.664,  
                "topY": 0.484,  
                "bottomX": 0.959,  
                "bottomY": 0.812  
            },  
            "label": "can",  
            "score": 0.891  
        },  
        {  
            "box": {  
                "topX": 0.423,  
                "topY": 0.253,  
                "bottomX": 0.632,  
                "bottomY": 0.725  
            },  
            "label": "water_bottle",  
            "score": 0.876  
        }  
    ]  
}
```

Instance segmentation

In instance segmentation, output consists of multiple boxes with their scaled top-left and bottom-right coordinates, labels, confidence scores, and polygons (not masks). Here, the polygon values are in the same format that we discussed in the schema section.

```
{
  "filename": "/tmp/tmpi8604s0h",
  "boxes": [
    {
      "box": {
        "topX": 0.679,
        "topY": 0.491,
        "bottomX": 0.926,
        "bottomY": 0.810
      },
      "label": "can",
      "score": 0.992,
      "polygon": [
        [
          0.82, 0.811, 0.771, 0.810, 0.758, 0.805, 0.741, 0.797, 0.735, 0.791, 0.718, 0.785, 0.715,
          0.778, 0.706, 0.775, 0.696, 0.758, 0.695, 0.717, 0.698, 0.567, 0.705, 0.552, 0.706, 0.540, 0.725, 0.520,
          0.735, 0.505, 0.745, 0.502, 0.755, 0.493
        ]
      ]
    },
    {
      "box": {
        "topX": 0.220,
        "topY": 0.298,
        "bottomX": 0.397,
        "bottomY": 0.601
      },
      "label": "milk_bottle",
      "score": 0.989,
      "polygon": [
        [
          0.365, 0.602, 0.273, 0.602, 0.26, 0.595, 0.263, 0.588, 0.251, 0.546, 0.248, 0.501, 0.25,
          0.485, 0.246, 0.478, 0.245, 0.463, 0.233, 0.442, 0.231, 0.43, 0.226, 0.423, 0.226, 0.408, 0.234, 0.385,
          0.241, 0.371, 0.238, 0.345, 0.234, 0.335, 0.233, 0.325, 0.24, 0.305, 0.586, 0.38, 0.592, 0.375, 0.598, 0.365
        ]
      ]
    },
    {
      "box": {
        "topX": 0.433,
        "topY": 0.280,
        "bottomX": 0.621,
        "bottomY": 0.679
      },
      "label": "water_bottle",
      "score": 0.988,
      "polygon": [
        [
          0.576, 0.680, 0.501, 0.680, 0.475, 0.675, 0.460, 0.625, 0.445, 0.630, 0.443, 0.572, 0.440,
          0.560, 0.435, 0.515, 0.431, 0.501, 0.431, 0.433, 0.433, 0.426, 0.445, 0.417, 0.456, 0.407, 0.465, 0.381,
          0.468, 0.327, 0.471, 0.318
        ]
      ]
    }
  ]
}
```

NOTE

The images used in this article are from the Fridge Objects dataset, copyright © Microsoft Corporation and available at [computervision-recipes/01_training_introduction.ipynb](https://github.com/microsoft/computervision-recipes/blob/main/01_training/introduction.ipynb) under the [MIT License](#).

Next steps

- Learn how to Prepare data for training computer vision models with automated ML.
- Set up computer vision tasks in AutoML
- Tutorial: Train an object detection model (preview) with AutoML and Python.

Hyperparameters for computer vision tasks in automated machine learning

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (preview)

Learn which hyperparameters are available specifically for computer vision tasks in automated ML experiments.

With support for computer vision tasks, you can control the model algorithm and sweep hyperparameters. These model algorithms and hyperparameters are passed in as the parameter space for the sweep. While many of the hyperparameters exposed are model-agnostic, there are instances where hyperparameters are model-specific or task-specific.

Model-specific hyperparameters

This table summarizes hyperparameters specific to the `yolov5` algorithm.

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>validation_metric_type</code>	Metric computation method to use for validation metrics. Must be <code>none</code> , <code>coco</code> , <code>voc</code> , or <code>coco_voc</code> .	<code>voc</code>
<code>validation_iou_threshold</code>	IOU threshold for box matching when computing validation metrics. Must be a float in the range [0.1, 1].	0.5
<code>image_size</code>	Image size for train and validation. Must be a positive integer. <i>Note: training run may get into CUDA OOM if the size is too big.</i>	640
<code>model_size</code>	Model size. Must be <code>small</code> , <code>medium</code> , <code>large</code> , or <code>extra_large</code> . <i>Note: training run may get into CUDA OOM if the model size is too big.</i>	<code>medium</code>
<code>multi_scale</code>	Enable multi-scale image by varying image size by +/- 50% Must be 0 or 1. <i>Note: training run may get into CUDA OOM if no sufficient GPU memory.</i>	0

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>box_score_threshold</code>	During inference, only return proposals with a score greater than <code>box_score_threshold</code> . The score is the multiplication of the objectness score and classification probability. Must be a float in the range [0, 1].	0.1
<code>nms_iou_threshold</code>	IOU threshold used during inference in non-maximum suppression post processing. Must be a float in the range [0, 1].	0.5

This table summarizes hyperparameters specific to the `maskrcnn_*` for instance segmentation during inference.

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>mask_pixel_score_threshold</code>	Score cutoff for considering a pixel as part of the mask of an object.	0.5
<code>max_number_of_polygon_points</code>	Maximum number of (x, y) coordinate pairs in polygon after converting from a mask.	100
<code>export_as_image</code>	Export masks as images.	False
<code>image_type</code>	Type of image to export mask as (options are jpg, png, bmp).	JPG

Model agnostic hyperparameters

The following table describes the hyperparameters that are model agnostic.

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>number_of_epochs</code>	Number of training epochs. Must be a positive integer.	15 (except <code>yolov5</code> : 30)
<code>training_batch_size</code>	Training batch size. Must be a positive integer.	Multi-class/multi-label: 78 (except <code>vit-variants</code> : <code>vits16r224</code> : 128 <code>vitb16r224</code> : 48 <code>vitl16r224</code> : 10) Object detection: 2 (except <code>yolov5</code> : 16) Instance segmentation: 2 <i>Note: The defaults are largest batch size that can be used on 12 GiB GPU memory.</i>

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>validation_batch_size</code>	Validation batch size. Must be a positive integer.	Multi-class/multi-label: 78 (except <i>vit-variants</i> : <code>vits16r224</code> : 128 <code>vitb16r224</code> : 48 <code>vitl16r224</code> : 10) Object detection: 1 (except <code>yolov5</code> : 16) Instance segmentation: 1 <i>Note: The defaults are largest batch size that can be used on 12 GiB GPU memory.</i>
<code>gradient_accumulation_step</code>	Gradient accumulation means running a configured number of <code>gradient_accumulation_step</code> without updating the model weights while accumulating the gradients of those steps, and then using the accumulated gradients to compute the weight updates. Must be a positive integer.	1
<code>early_stopping</code>	Enable early stopping logic during training. Must be 0 or 1.	1
<code>early_stopping_patience</code>	Minimum number of epochs or validation evaluations with no primary metric improvement before the run is stopped. Must be a positive integer.	5
<code>early_stopping_delay</code>	Minimum number of epochs or validation evaluations to wait before primary metric improvement is tracked for early stopping. Must be a positive integer.	5
<code>learning_rate</code>	Initial learning rate. Must be a float in the range [0, 1].	Multi-class: 0.01 (except <i>vit-variants</i> : <code>vits16r224</code> : 0.0125 <code>vitb16r224</code> : 0.0125 <code>vitl16r224</code> : 0.001) Multi-label: 0.035 (except <i>vit-variants</i> : <code>vits16r224</code> : 0.025 <code>vitb16r224</code> : 0.025 <code>vitl16r224</code> : 0.002) Object detection: 0.005 (except <code>yolov5</code> : 0.01) Instance segmentation: 0.005

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>learning_rate_scheduler</code>	Type of learning rate scheduler. Must be <code>warmup_cosine</code> or <code>step</code> .	<code>warmup_cosine</code>
<code>step_lr_gamma</code>	Value of gamma when learning rate scheduler is <code>step</code> . Must be a float in the range [0, 1].	0.5
<code>step_lr_step_size</code>	Value of step size when learning rate scheduler is <code>step</code> . Must be a positive integer.	5
<code>warmup_cosine_lr_cycles</code>	Value of cosine cycle when learning rate scheduler is <code>warmup_cosine</code> . Must be a float in the range [0, 1].	0.45
<code>warmup_cosine_lr_warmup_epochs</code>	Value of warmup epochs when learning rate scheduler is <code>warmup_cosine</code> . Must be a positive integer.	2
<code>optimizer</code>	Type of optimizer. Must be either <code>sgd</code> , <code>adam</code> , <code>adamw</code> .	<code>sgd</code>
<code>momentum</code>	Value of momentum when optimizer is <code>sgd</code> . Must be a float in the range [0, 1].	0.9
<code>weight_decay</code>	Value of weight decay when optimizer is <code>sgd</code> , <code>adam</code> , or <code>adamw</code> . Must be a float in the range [0, 1].	1e-4
<code>nesterov</code>	Enable <code>nesterov</code> when optimizer is <code>sgd</code> . Must be 0 or 1.	1
<code>beta1</code>	Value of <code>beta1</code> when optimizer is <code>adam</code> or <code>adamw</code> . Must be a float in the range [0, 1].	0.9
<code>beta2</code>	Value of <code>beta2</code> when optimizer is <code>adam</code> or <code>adamw</code> . Must be a float in the range [0, 1].	0.999
<code>ams_gradient</code>	Enable <code>ams_gradient</code> when optimizer is <code>adam</code> or <code>adamw</code> . Must be 0 or 1.	0
<code>evaluation_frequency</code>	Frequency to evaluate validation dataset to get metric scores. Must be a positive integer.	1
<code>checkpoint_frequency</code>	Frequency to store model checkpoints. Must be a positive integer.	Checkpoint at epoch with best primary metric on validation.

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>checkpoint_run_id</code>	The run ID of the experiment that has a pretrained checkpoint for incremental training.	no default
<code>layers_to_freeze</code>	<p>How many layers to freeze for your model. For instance, passing 2 as value for <code>seresnext</code> means freezing layer0 and layer1 referring to the below supported model layer info.</p> <p>Must be a positive integer.</p> <pre>'resnet': [('conv1.', 'bn1.'), 'layer1.', 'layer2.', 'layer3.', 'layer4.'], 'mobilenetv2': ['features.0.', 'features.1.', 'features.2.', 'features.3.', 'features.4.', 'features.5.', 'features.6.', 'features.7.', 'features.8.', 'features.9.', 'features.10.', 'features.11.', 'features.12.', 'features.13.', 'features.14.', 'features.15.', 'features.16.', 'features.17.', 'features.18.'], 'seresnext': ['layer0.', 'layer1.', 'layer2.', 'layer3.', 'layer4.'], 'vit': ['patch_embed', 'blocks.0.', 'blocks.1.', 'blocks.2.', 'blocks.3.', 'blocks.4.', 'blocks.5.', 'blocks.6.', 'blocks.7.', 'blocks.8.', 'blocks.9.', 'blocks.10.', 'blocks.11.'], 'yolov5_backbone': ['model.0.', 'model.1.', 'model.2.', 'model.3.', 'model.4.', 'model.5.', 'model.6.', 'model.7.', 'model.8.', 'model.9.'], 'resnet_backbone': ['backbone.body.conv1.', 'backbone.body.layer1.', 'backbone.body.layer2.', 'backbone.body.layer3.', 'backbone.body.layer4.']}</pre>	no default

Image classification (multi-class and multi-label) specific hyperparameters

The following table summarizes hyperparameters for image classification (multi-class and multi-label) tasks.

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>weighted_loss</code>	<ul style="list-style-type: none"> • 0 for no weighted loss. • 1 for weighted loss with sqrt. (class_weights) • 2 for weighted loss with class_weights. • Must be 0 or 1 or 2. 	0

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>validation_resize_size</code>	<ul style="list-style-type: none"> Image size to which to resize before cropping for validation dataset. Must be a positive integer. <p><i>Notes:</i></p> <ul style="list-style-type: none"> <code>seresnext</code> doesn't take an arbitrary size. Training run may get into CUDA OOM if the size is too big 	256
<code>validation_crop_size</code>	<ul style="list-style-type: none"> Image crop size that's input to your neural network for validation dataset. Must be a positive integer. <p><i>Notes:</i></p> <ul style="list-style-type: none"> <code>seresnext</code> doesn't take an arbitrary size. <i>ViT-variants should have the same <code>validation_crop_size</code> and <code>training_crop_size</code>.</i> Training run may get into CUDA OOM if the size is too big 	224
<code>training_crop_size</code>	<ul style="list-style-type: none"> Image crop size that's input to your neural network for train dataset. Must be a positive integer. <p><i>Notes:</i></p> <ul style="list-style-type: none"> <code>seresnext</code> doesn't take an arbitrary size. <i>ViT-variants should have the same <code>validation_crop_size</code> and <code>training_crop_size</code>.</i> Training run may get into CUDA OOM if the size is too big 	224

Object detection and instance segmentation task specific hyperparameters

The following hyperparameters are for object detection and instance segmentation tasks.

WARNING

These parameters are not supported with the `yolov5` algorithm. See the [model specific hyperparameters](#) section for `yolov5` supported hyperparameters.

PARAMETER NAME	DESCRIPTION	DEFAULT

PARAMETER NAME	DESCRIPTION	DEFAULT
<code>validation_metric_type</code>	Metric computation method to use for validation metrics. Must be <code>none</code> , <code>coco</code> , <code>voc</code> , or <code>coco_voc</code> .	<code>voc</code>
<code>validation_iou_threshold</code>	IOU threshold for box matching when computing validation metrics. Must be a float in the range [0.1, 1].	0.5
<code>min_size</code>	Minimum size of the image to be rescaled before feeding it to the backbone. Must be a positive integer. <i>Note: training run may get into CUDA OOM if the size is too big.</i>	600
<code>max_size</code>	Maximum size of the image to be rescaled before feeding it to the backbone. Must be a positive integer. <i>Note: training run may get into CUDA OOM if the size is too big.</i>	1333
<code>box_score_threshold</code>	During inference, only return proposals with a classification score greater than <code>box_score_threshold</code> . Must be a float in the range [0, 1].	0.3
<code>nms_iou_threshold</code>	IOU (intersection over union) threshold used in non-maximum suppression (NMS) for the prediction head. Used during inference. Must be a float in the range [0, 1].	0.5
<code>box_detections_per_image</code>	Maximum number of detections per image, for all classes. Must be a positive integer.	100
<code>tile_grid_size</code>	The grid size to use for tiling each image. <i>Note: tile_grid_size must not be None to enable small object detection logic</i> A tuple of two integers passed as a string. Example: --tile_grid_size "(3, 2)"	No Default
<code>tile_overlap_ratio</code>	Overlap ratio between adjacent tiles in each dimension. Must be float in the range of [0, 1)	0.25
<code>tile_predictions_nms_threshold</code>	The IOU threshold to use to perform NMS while merging predictions from tiles and image. Used in validation/inference. Must be float in the range of [0, 1]	0.25

Next steps

- Learn how to [Set up AutoML to train computer vision models with Python \(preview\)](#).
- [Tutorial: Train an object detection model \(preview\) with AutoML and Python.](#)

Reference for configuring Kubernetes cluster for Azure Machine Learning

9/22/2022 • 5 minutes to read • [Edit Online](#)

This article contains reference information that may be useful when [configuring Kubernetes with Azure Machine Learning](#).

Supported Kubernetes version and region

- Kubernetes clusters installing AzureML extension have a version support window of "N-2", that is aligned with [Azure Kubernetes Service \(AKS\) version support policy](#), where 'N' is the latest GA minor version of Azure Kubernetes Service.
 - For example, if AKS introduces 1.20.a today, versions 1.20.a, 1.20.b, 1.19.c, 1.19.d, 1.18.e, and 1.18.f are supported.
 - If customers are running an unsupported Kubernetes version, they'll be asked to upgrade when requesting support for the cluster. Clusters running unsupported Kubernetes releases aren't covered by the AzureML extension support policies.
- AzureML extension region availability:
 - AzureML extension can be deployed to AKS or Azure Arc-enabled Kubernetes in supported regions listed in [Azure Arc enabled Kubernetes region support](#).

Prerequisites for ARO or OCP clusters

Disable Security Enhanced Linux (SELinux)

[AzureML dataset](#) (used in AzureML training jobs) isn't supported on machines with SELinux enabled. Therefore, you need to disable `selinux` on all workers in order to use AzureML dataset.

Privileged setup for ARO and OCP

For AzureML extension deployment on ARO or OCP cluster, grant privileged access to AzureML service accounts, run `oc edit scc privileged` command, and add following service accounts under "users":

- `system:serviceaccount:azure-arc:azure-arc-kube-aad-proxy-sa`
- `system:serviceaccount:azureml:{EXTENSION-NAME}-kube-state-metrics`
- `system:serviceaccount:azureml:prom-admission`
- `system:serviceaccount:azureml:default`
- `system:serviceaccount:azureml:prom-operator`
- `system:serviceaccount:azureml:load-amlarc-selinux-policy-sa`
- `system:serviceaccount:azureml:azureml-fe-v2`
- `system:serviceaccount:azureml:prom-prometheus`
- `system:serviceaccount:{KUBERNETES-COMPUTE-NAMESPACE}:default`
- `system:serviceaccount:azureml:azureml-ingress-nginx`
- `system:serviceaccount:azureml:azureml-ingress-nginx-admission`

NOTE

- `{EXTENSION-NAME}` : is the extension name specified with the `az k8s-extension create --name` CLI command.
- `{KUBERNETES-COMPUTE-NAMESPACE}` : is the namespace of the Kubernetes compute specified when attaching the compute to the Azure Machine Learning workspace. Skip configuring `system:serviceaccount:{KUBERNETES-COMPUTE-NAMESPACE}:default` if `KUBERNETES-COMPUTE-NAMESPACE` is `default`.

AzureML extension components

For Arc-connected cluster, AzureML extension deployment will create [Azure Relay](#) in Azure cloud, used to route traffic between Azure services and the Kubernetes cluster. For AKS cluster without Arc connected, Azure Relay resource won't be created.

Upon AzureML extension deployment completes, it will create following resources in Kubernetes cluster, depending on each AzureML extension deployment scenario:

RESOURCE NAME	RESOURCE TYPE	TRAINING	INFERENCE	TRAINING AND INFERENCE	DESCRIPTION	COMMUNICATION WITH CLOUD
relayserver	Kubernetes deployment	✓	✓	✓	relayserver is only needed in arc-connected cluster, and won't be installed in AKS cluster. Relayserver works with Azure Relay to communicate with the cloud services.	Receive the request of job creation, model deployment from cloud service; sync the job status with cloud service.
gateway	Kubernetes deployment	✓	✓	✓	The gateway is used to communicate and send data back and forth.	Send nodes and cluster resource information to cloud services.
aml-operator	Kubernetes deployment	✓	N/A	✓	Manage the lifecycle of training jobs.	Token exchange with the cloud token service for authentication and authorization of Azure Container Registry.

Resource Name	Resource Type	Training	Inference	Training and Inference	Description	Communication with Cloud
metrics-controller-manager	Kubernetes deployment	✓	✓	✓	Manage the configuration for Prometheus	N/A
{EXTENSION-NAME}-kube-state-metrics	Kubernetes deployment	✓	✓	✓	Export the cluster-related metrics to Prometheus.	N/A
{EXTENSION-NAME}-prometheus-operator	Kubernetes deployment	Optional	Optional	Optional	Provide Kubernetes native deployment and management of Prometheus and related monitoring components.	N/A
amlarc-identity-controller	Kubernetes deployment	N/A	✓	✓	Request and renew Azure Blob/Azure Container Registry token through managed identity.	Token exchange with the cloud token service for authentication and authorization of Azure Container Registry and Azure Blob used by inference/model deployment.
amlarc-identity-proxy	Kubernetes deployment	N/A	✓	✓	Request and renew Azure Blob/Azure Container Registry token through managed identity.	Token exchange with the cloud token service for authentication and authorization of Azure Container Registry and Azure Blob used by inference/model deployment.

Resource Name	Resource Type	Training	Inference	Training and Inference	Description	Communication with Cloud
azureml-fe-v2	Kubernetes deployment	N/A	✓	✓	The front-end component that routes incoming inference requests to deployed services.	Send service logs to Azure Blob.
inference-operator-controller-manager	Kubernetes deployment	N/A	✓	✓	Manage the lifecycle of inference endpoints.	N/A
volcano-admission	Kubernetes deployment	Optional	N/A	Optional	Volcano admission webhook.	N/A
volcano-controllers	Kubernetes deployment	Optional	N/A	Optional	Manage the lifecycle of Azure Machine Learning training job pods.	N/A
volcano-scheduler	Kubernetes deployment	Optional	N/A	Optional	Used to perform in-cluster job scheduling.	N/A
fluent-bit	Kubernetes daemonset	✓	✓	✓	Gather the components' system log.	Upload the components' system log to cloud.
{EXTENSION-NAME}-dcgm-exporter	Kubernetes daemonset	Optional	Optional	Optional	dcmg-exporter exposes GPU metrics for Prometheus.	N/A
nvidia-device-plugin-daemonset	Kubernetes daemonset	Optional	Optional	Optional	nvidia-device-plugin-daemonset exposes GPUs on each node of your cluster	N/A
prometheus-prom-prometheus	Kubernetes statefulset	✓	✓	✓	Gather and send job metrics to cloud.	Send job metrics like cpu/gpu/memory utilization to cloud.

IMPORTANT

- Azure Relay resource is under the same resource group as the Arc cluster resource. It is used to communicate with the Kubernetes cluster and modifying them will break attached compute targets.
- By default, the kubernetes deployment resources are randomly deployed to 1 or more nodes of the cluster, and daemonset resources are deployed to ALL nodes. If you want to restrict the extension deployment to specific nodes, use `nodeSelector` configuration setting described as below.

NOTE

- {EXTENSION-NAME}: is the extension name specified with `az k8s-extension create --name` CLI command.

AzureML jobs connect with custom data storage

[Persistent Volume \(PV\)](#) and [Persistent Volume Claim \(PVC\)](#) are Kubernetes concept, allowing user to provide and consume various storage resources.

1. Create PV, take NFS as example,

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: ""
  nfs:
    path: /share/nfs
    server: 20.98.110.84
    readOnly: false
```

2. Create PVC in the same Kubernetes namespace with ML workloads. In `metadata`, you **must** add label `ml.azure.com/pvc: "true"` to be recognized by AzureML, and add annotation `ml.azure.com/mountpath: <mount path>` to set the mount path.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
  namespace: default
  labels:
    ml.azure.com/pvc: "true"
  annotations:
    ml.azure.com/mountpath: "/mnt/nfs"
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

IMPORTANT

Only the job pods in the same Kubernetes namespace with the PVC(s) will be mounted the volume. Data scientist is able to access the `mount path` specified in the PVC annotation in the job.

Sample YAML definition of Kubernetes secret for TLS/SSL

To enable HTTPS endpoint for real-time inference, you need to provide both PEM-encoded TLS/SSL certificate and key. The best practice is to save the certificate and key in a Kubernetes secret in the `azureml` namespace.

The sample YAML definition of the TLS/SSL secret is as follows,

```
apiVersion: v1
data:
  cert.pem: <PEM-encoded SSL certificate>
  key.pem: <PEM-encoded SSL key>
kind: Secret
metadata:
  name: <secret name>
  namespace: azureml
type: Opaque
```

Monitoring Azure machine learning data reference

9/22/2022 • 14 minutes to read • [Edit Online](#)

Learn about the data and resources collected by Azure Monitor from your Azure Machine Learning workspace. See [Monitoring Azure Machine Learning](#) for details on collecting and analyzing monitoring data.

Metrics

This section lists all the automatically collected platform metrics collected for Azure Machine Learning. The resource provider for these metrics is [Microsoft.MachineLearningServices/workspaces](#).

Model

METRIC	UNIT	DESCRIPTION
Model Register Succeeded	Count	Number of model registrations that succeeded in this workspace
Model Register Failed	Count	Number of model registrations that failed in this workspace
Model Deploy Started	Count	Number of model deployments started in this workspace
Model Deploy Succeeded	Count	Number of model deployments that succeeded in this workspace
Model Deploy Failed	Count	Number of model deployments that failed in this workspace

Quota

Quota information is for Azure Machine Learning compute only.

METRIC	UNIT	DESCRIPTION
Total Nodes	Count	Number of total nodes. This total includes some of Active Nodes, Idle Nodes, Unusable Nodes, Preempted Nodes, Leaving Nodes
Active Nodes	Count	Number of Active nodes. The nodes that are actively running a job.
Idle Nodes	Count	Number of idle nodes. Idle nodes are the nodes that are not running any jobs but can accept new job if available.
Unusable Nodes	Count	Number of unusable nodes. Unusable nodes are not functional due to some unresolvable issue. Azure will recycle these nodes.

METRIC	UNIT	DESCRIPTION
Preempted Nodes	Count	Number of preempted nodes. These nodes are the low-priority nodes that are taken away from the available node pool.
Leaving Nodes	Count	Number of leaving nodes. Leaving nodes are the nodes that just finished processing a job and will go to Idle state.
Total Cores	Count	Number of total cores
Active Cores	Count	Number of active cores
Idle Cores	Count	Number of idle cores
Unusable Cores	Count	Number of unusable cores
Preempted Cores	Count	Number of preempted cores
Leaving Cores	Count	Number of leaving cores
Quota Utilization Percentage	Count	Percent of quota utilized

Resource

METRIC	UNIT	DESCRIPTION
CpuUtilization	Count	Percentage of utilization on a CPU node. Utilization is reported at one-minute intervals.
GpuUtilization	Count	Percentage of utilization on a GPU node. Utilization is reported at one-minute intervals.
GpuMemoryUtilization	Count	Percentage of memory utilization on a GPU node. Utilization is reported at one-minute intervals.
GpuEnergyJoules	Count	Interval energy in Joules on a GPU node. Energy is reported at one-minute intervals.

Run

Information on training runs for the workspace.

METRIC	UNIT	DESCRIPTION
Cancelled Runs	Count	Number of runs canceled for this workspace. Count is updated when a run is successfully canceled.

METRIC	UNIT	DESCRIPTION
Cancel Requested Runs	Count	Number of runs where cancel was requested for this workspace. Count is updated when cancellation request has been received for a run.
Completed Runs	Count	Number of runs completed successfully for this workspace. Count is updated when a run has completed and output has been collected.
Failed Runs	Count	Number of runs failed for this workspace. Count is updated when a run fails.
Finalizing Runs	Count	Number of runs entered finalizing state for this workspace. Count is updated when a run has completed but output collection still in progress.
Not Responding Runs	Count	Number of runs not responding for this workspace. Count is updated when a run enters Not Responding state.
Not Started Runs	Count	Number of runs in Not Started state for this workspace. Count is updated when a request is received to create a run but run information has not yet been populated.
Preparing Runs	Count	Number of runs that are preparing for this workspace. Count is updated when a run enters Preparing state while the run environment is being prepared.
Provisioning Runs	Count	Number of runs that are provisioning for this workspace. Count is updated when a run is waiting on compute target creation or provisioning.
Queued Runs	Count	Number of runs that are queued for this workspace. Count is updated when a run is queued in compute target. Can occur when waiting for required compute nodes to be ready.
Started Runs	Count	Number of runs running for this workspace. Count is updated when run starts running on required resources.
Starting Runs	Count	Number of runs started for this workspace. Count is updated after request to create run and run info, such as the Run ID, has been populated

METRIC	UNIT	DESCRIPTION
Errors	Count	Number of run errors in this workspace. Count is updated whenever run encounters an error.
Warnings	Count	Number of run warnings in this workspace. Count is updated whenever a run encounters a warning.

Metric dimensions

For more information on what metric dimensions are, see [Multi-dimensional metrics](#).

Azure Machine Learning has the following dimensions associated with its metrics.

DIMENSION	DESCRIPTION
Cluster Name	The name of the compute cluster resource. Available for all quota metrics.
Vm Family Name	The name of the VM family used by the cluster. Available for quota utilization percentage.
Vm Priority	The priority of the VM. Available for quota utilization percentage.
CreatedTime	Only available for CpuUtilization and GpuUtilization.
DeviceId	ID of the device (GPU). Only available for GpuUtilization.
NodeId	ID of the node created where job is running. Only available for CpuUtilization and GpuUtilization.
RunId	ID of the run/job. Only available for CpuUtilization and GpuUtilization.
ComputeType	The compute type that the run used. Only available for Completed runs, Failed runs, and Started runs.
PipelineStepType	The type of PipelineStep used in the run. Only available for Completed runs, Failed runs, and Started runs.
PublishedPipelineId	The ID of the published pipeline used in the run. Only available for Completed runs, Failed runs, and Started runs.
RunType	The type of run. Only available for Completed runs, Failed runs, and Started runs.

The valid values for the RunType dimension are:

VALUE	DESCRIPTION
Experiment	Non-pipeline runs.

VALUE	DESCRIPTION
PipelineRun	A pipeline run, which is the parent of a StepRun.
StepRun	A run for a pipeline step.
ReusedStepRun	A run for a pipeline step that reuses a previous run.

Activity log

The following table lists the operations related to Azure Machine Learning that may be created in the Activity log.

OPERATION	DESCRIPTION
Creates or updates a Machine Learning workspace	A workspace was created or updated
CheckComputeNameAvailability	Check if a compute name is already in use
Creates or updates the compute resources	A compute resource was created or updated
Deletes the compute resources	A compute resource was deleted
List secrets	An operation listed secrets for a Machine Learning workspace

Resource logs

This section lists the types of resource logs you can collect for Azure Machine Learning workspace.

Resource Provider and Type: [Microsoft.MachineLearningServices/workspace](#).

CATEGORY	DISPLAY NAME
AmlComputeClusterEvent	AmlComputeClusterEvent
AmlComputeClusterNodeEvent (deprecated)	AmlComputeClusterNodeEvent
AmlComputeCpuGpuUtilization	AmlComputeCpuGpuUtilization
AmlComputeJobEvent	AmlComputeJobEvent
AmlRunStatusChangedEvent	AmlRunStatusChangedEvent
ModelsChangeEvent	ModelsChangeEvent
ModelsReadEvent	ModelsReadEvent
ModelsActionEvent	ModelsActionEvent
DeploymentReadEvent	DeploymentReadEvent

CATEGORY	DISPLAY NAME
DeploymentEventACI	DeploymentEventACI
DeploymentEventAKS	DeploymentEventAKS
InferencingOperationAKS	InferencingOperationAKS
InferencingOperationACI	InferencingOperationACI
EnvironmentChangeEvent	EnvironmentChangeEvent
EnvironmentReadEvent	EnvironmentReadEvent
DataLabelChangeEvent	DataLabelChangeEvent
DataLabelReadEvent	DataLabelReadEvent
ComputeInstanceEvent	ComputeInstanceEvent
DataStoreChangeEvent	DataStoreChangeEvent
DataStoreReadEvent	DataStoreReadEvent
DataSetChangeEvent	DataSetChangeEvent
DataSetReadEvent	DataSetReadEvent
PipelineChangeEvent	PipelineChangeEvent
PipelineReadEvent	PipelineReadEvent
RunEvent	RunEvent
RunReadEvent	RunReadEvent

Schemas

The following schemas are in use by Azure Machine Learning

AmlComputeJobEvent table

PROPERTY	DESCRIPTION
TimeGenerated	Time when the log entry was generated
OperationName	Name of the operation associated with the log event
Category	Name of the log event
JobId	ID of the Job submitted

PROPERTY	DESCRIPTION
ExperimentId	ID of the Experiment
ExperimentName	Name of the Experiment
CustomerSubscriptionId	SubscriptionId where Experiment and Job was submitted
WorkspaceName	Name of the machine learning workspace
ClusterName	Name of the Cluster
ProvisioningState	State of the Job submission
ResourceGroupName	Name of the resource group
JobName	Name of the Job
ClusterId	ID of the cluster
EventType	Type of the Job event. For example, JobSubmitted, JobRunning, JobFailed, JobSucceeded.
ExecutionState	State of the job (the Run). For example, Queued, Running, Succeeded, Failed
ErrorDetails	Details of job error
CreationApiVersion	Api version used to create the job
ClusterResourceGroupName	Resource group name of the cluster
TFWorkerCount	Count of TF workers
TFParameterServerCount	Count of TF parameter server
ToolType	Type of tool used
RunInContainer	Flag describing if job should be run inside a container
JobErrorMessage	detailed message of Job error
NodeId	ID of the node created where job is running

AmlComputeClusterEvent table

PROPERTY	DESCRIPTION
TimeGenerated	Time when the log entry was generated
OperationName	Name of the operation associated with the log event
Category	Name of the log event

PROPERTY	DESCRIPTION
ProvisioningState	Provisioning state of the cluster
ClusterName	Name of the cluster
ClusterType	Type of the cluster
CreatedBy	User who created the cluster
CoreCount	Count of the cores in the cluster
VmSize	Vm size of the cluster
VmPriority	Priority of the nodes created inside a cluster Dedicated/LowPriority
ScalingType	Type of cluster scaling manual/auto
InitialNodeCount	Initial node count of the cluster
MinimumNodeCount	Minimum node count of the cluster
MaximumNodeCount	Maximum node count of the cluster
NodeDeallocationOption	How the node should be deallocated
Publisher	Publisher of the cluster type
Offer	Offer with which the cluster is created
Sku	Sku of the Node/VM created inside cluster
Version	Version of the image used while Node/VM is created
SubnetId	SubnetId of the cluster
AllocationState	Cluster allocation state
CurrentNodeCount	Current node count of the cluster
TargetNodeCount	Target node count of the cluster while scaling up/down
EventType	Type of event during cluster creation.
NodeIdleTimeSecondsBeforeScaleDown	Idle time in seconds before cluster is scaled down
PreemptedNodeCount	Preempted node count of the cluster
IsResizeGrow	Flag indicating that cluster is scaling up

PROPERTY	DESCRIPTION
VmFamilyName	Name of the VM family of the nodes that can be created inside cluster
LeavingNodeCount	Leaving node count of the cluster
UnusableNodeCount	Unusable node count of the cluster
IdleNodeCount	Idle node count of the cluster
RunningNodeCount	Running node count of the cluster
PreparingNodeCount	Preparing node count of the cluster
QuotaAllocated	Allocated quota to the cluster
QuotaUtilized	Utilized quota of the cluster
AllocationStateTransitionTime	Transition time from one state to another
ClusterErrorCodes	Error code received during cluster creation or scaling
CreationApiVersion	Api version used while creating the cluster

AmlComputeClusterNodeEvent table

PROPERTY	DESCRIPTION
TimeGenerated	Time when the log entry was generated
OperationName	Name of the operation associated with the log event
Category	Name of the log event
ClusterName	Name of the cluster
NodeId	ID of the cluster node created
VmSize	Vm size of the node
VmFamilyName	Vm family to which the node belongs
VmPriority	Priority of the node created Dedicated/LowPriority
Publisher	Publisher of the vm image. For example, microsoft-dsvm
Offer	Offer associated with the VM creation
Sku	Sku of the Node/VM created
Version	Version of the image used while Node/VM is created

PROPERTY	DESCRIPTION
ClusterCreationTime	Time when cluster was created
ResizeStartTime	Time when cluster scale up/down started
ResizeEndTime	Time when cluster scale up/down ended
NodeAllocationTime	Time when Node was allocated
NodeBootTime	Time when Node was booted up
StartTaskStartTime	Time when task was assigned to a node and started
StartTaskEndTime	Time when task assigned to a node ended
TotalE2ETimeInSeconds	Total time node was active

NOTE

Effective February 2022, the AmlComputeClusterNodeEvent table will be deprecated. We recommend that you instead use the AmlComputeClusterEvent table.

AmlComputeInstanceEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlComputeInstanceEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
CorrelationId	A GUID used to group together a set of related events, when applicable.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
AmlComputeInstanceName	"The name of the compute instance associated with the log entry.

AmlDataLabelEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlDataLabelEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
CorrelationId	A GUID used to group together a set of related events, when applicable.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
AmlProjectId	The unique identifier of the AzureML project.
AmlProjectName	The name of the AzureML project.
AmlLabelNames	The label class names which are created for the project.
AmlDataStoreName	The name of the data store where the project's data is stored.

AmlDataSetEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlDataSetEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
AmlWorkspaceId	A GUID and unique ID of the AzureML workspace.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.

PROPERTY	DESCRIPTION
AmlDatasetId	The ID of the AzureML Data Set.
AmlDatasetName	The name of the AzureML Data Set.

AmlDataStoreEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlDataStoreEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
AmlWorkspaceId	A GUID and unique ID of the AzureML workspace.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
AmlDatastoreName	The name of the AzureML Data Store.

AmlDeploymentEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlDeploymentEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
AmlServiceName	The name of the AzureML Service.

AmlInferencingEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlInferencingEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
AmlServiceName	The name of the AzureML Service.

AmlModelsEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlModelsEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
ResultSignature	The HTTP status code of the event. Typical values include 200, 201, 202 etc.
AmlModelName	The name of the AzureML Model.

AmlPipelineEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlPipelineEvent

PROPERTY	DESCRIPTION
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
AmlWorkspaceId	A GUID and unique ID of the AzureML workspace.
AmlWorkspaceName	The name of the AzureML workspace.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
AmlModuleId	A GUID and unique ID of the module.
AmlModelName	The name of the AzureML Model.
AmlPipelineId	The ID of the AzureML pipeline.
AmlParentPipelineId	The ID of the parent AzureML pipeline (in the case of cloning).
AmlPipelineDraftId	The ID of the AzureML pipeline draft.
AmlPipelineDraftName	The name of the AzureML pipeline draft.
AmlPipelineEndpointId	The ID of the AzureML pipeline endpoint.
AmlPipelineEndpointName	The name of the AzureML pipeline endpoint.

AmlRunEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlRunEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
ResultType	The status of the event. Typical values include Started, In Progress, Succeeded, Failed, Active, and Resolved.
OperationName	The name of the operation associated with the log entry

PROPERTY	DESCRIPTION
AmlWorkspaceId	A GUID and unique ID of the AzureML workspace.
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
RunId	The unique ID of the run.

AmlEnvironmentEvent table

PROPERTY	DESCRIPTION
Type	Name of the log event, AmlEnvironmentEvent
TimeGenerated	Time (UTC) when the log entry was generated
Level	The severity level of the event. Must be one of Informational, Warning, Error, or Critical.
OperationName	The name of the operation associated with the log entry
Identity	The identity of the user or application that performed the operation.
AadTenantId	The AAD tenant ID the operation was submitted for.
AmlEnvironmentName	The name of the AzureML environment configuration.
AmlEnvironmentVersion	The name of the AzureML environment configuration version.

See also

- See [Monitoring Azure Machine Learning](#) for a description of monitoring Azure Machine Learning.
- See [Monitoring Azure resources with Azure Monitor](#) for details on monitoring Azure resources.

Azure Policy built-in policy definitions for Azure Machine Learning

9/22/2022 • 4 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy](#) built-in policy definitions for Azure Machine Learning. Common use cases for Azure Policy include implementing governance for resource consistency, regulatory compliance, security, cost, and management. Policy definitions for these common use cases are already available in your Azure environment as built-ins to help you get started. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the [GitHub](#) column to view the source on the [Azure Policy GitHub repo](#).

Built-in policy definitions

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
Audit Azure Machine Learning Compute Cluster and Instance is behind virtual network	Azure Virtual Network deployment provides enhanced security and isolation for your Azure Machine Learning Compute Clusters and Instances, as well as subnets, access control policies, and other features to further restrict access. When an Azure Machine Learning Compute instance is configured with a virtual network, it is not publicly addressable and can only be accessed from virtual machines and applications within the virtual network.	Audit, Disabled	1.0.0
Azure Machine Learning Compute Instance should have idle shutdown.	Having an idle shutdown schedule reduces cost by shutting down computes that are idle after a pre-determined period of activity.	Audit, Deny, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Azure Machine Learning workspaces should be encrypted with a customer-managed key	<p>Manage encryption at rest of Azure Machine Learning workspace data with customer-managed keys. By default, customer data is encrypted with service-managed keys, but customer-managed keys are commonly required to meet regulatory compliance standards. Customer-managed keys enable the data to be encrypted with an Azure Key Vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management. Learn more at https://aka.ms/azureml-workspaces-cmk.</p>	Audit, Deny, Disabled	1.0.3
Azure Machine Learning workspaces should disable public network access	<p>Disabling public network access improves security by ensuring that the machine learning workspaces aren't exposed on the public internet. You can limit exposure of your workspaces by creating private endpoints instead. Learn more at: https://aka.ms/privateendpoints.</p>	Audit, Deny, Disabled	2.0.0
Azure Machine Learning workspaces should enable V1LegacyMode to support network isolation backward compatibility	<p>Azure ML is making a transition to a new V2 API platform on Azure Resource Manager and you can control API platform version using V1LegacyMode parameter. Enabling the V1LegacyMode parameter will enable you to keep your workspaces in the same network isolation as V1, though you won't have use of the new V2 features. We recommend turning on V1 Legacy Mode only when you want to keep the AzureML control plane data inside your private networks. Learn more at: https://aka.ms/V1LegacyMode.</p>	Audit, Deny, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Azure Machine Learning workspaces should use private link	<p>Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network.</p> <p>Mapping private endpoints to Azure Machine Learning workspaces reduces data leakage risks. Learn more about private links at: https://docs.microsoft.com/azure/machine-learning/how-to-configure-private-link.</p>	Audit, Deny, Disabled	1.1.0
Azure Machine Learning workspaces should use user-assigned managed identity	<p>Manage access to Azure ML workspace and associated resources, Azure Container Registry, KeyVault, Storage, and App Insights using user-assigned managed identity. By default, system-assigned managed identity is used by Azure ML workspace to access the associated resources. User-assigned managed identity allows you to create the identity as an Azure resource and maintain the life cycle of that identity.</p> <p>Learn more at https://docs.microsoft.com/azure/machine-learning/how-to-use-managed-identities?tabs=python.</p>	Audit, Deny, Disabled	1.0.0
Configure Azure Machine Learning workspace to use private DNS zones	<p>Use private DNS zones to override the DNS resolution for a private endpoint. A private DNS zone links to your virtual network to resolve to Azure Machine Learning workspaces. Learn more at: https://docs.microsoft.com/azure/machine-learning/how-to-network-security-overview.</p>	DeployIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Configure Azure Machine Learning workspaces to disable public network access	Disable public network access for Azure Machine Learning workspaces so that your workspaces aren't accessible over the public internet. This will help protect the workspaces against data leakage risks. You can limit exposure of your machine learning workspaces by creating private endpoints instead. Learn more at: https://aka.ms/privateendpoints .	Modify, Disabled	1.0.0
Configure Azure Machine Learning workspaces with private endpoints	Private endpoints connect your virtual network to Azure services without a public IP address at the source or destination. By mapping private endpoints to your Azure Machine Learning workspace, you can reduce data leakage risks. Learn more about private links at: https://docs.microsoft.com/azure/machine-learning/how-to-configure-private-link .	DeployIfNotExists, Disabled	1.0.0
Configure Machine Learning computes to disable local authentication methods	Disable location authentication methods so that your Machine Learning computes require Azure Active Directory identities exclusively for authentication. Learn more at: https://aka.ms/azure-ml-aad-policy .	Modify, Disabled	1.0.0
Machine Learning computes should have local authentication methods disabled	Disabling local authentication methods improves security by ensuring that Machine Learning computes require Azure Active Directory identities exclusively for authentication. Learn more at: https://aka.ms/azure-ml-aad-policy .	Audit, Deny, Disabled	1.0.0

Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).

Azure Machine Learning Curated Environments

9/22/2022 • 2 minutes to read • [Edit Online](#)

This article lists the curated environments with latest framework versions in Azure Machine Learning. Curated environments are provided by Azure Machine Learning and are available in your workspace by default. They are backed by cached Docker images that use the latest version of the Azure Machine Learning SDK, reducing the run preparation cost and allowing for faster deployment time. Use these environments to quickly get started with various machine learning frameworks.

NOTE

Use the [Python SDK](#), [CLI](#), or Azure Machine Learning [studio](#) to get the full list of environments and their dependencies. For more information, see the [environments](#) article.

Why should I use curated environments?

- Reduces training and deployment latency.
- Improves training and deployment success rate.
- Avoid unnecessary image builds.
- Only have required dependencies and access right in the image/container.

IMPORTANT

To view more information about curated environment packages and versions, visit the Environments tab in the Azure Machine Learning [studio](#).

Curated environments

IMPORTANT

Items marked (preview) in this article are currently in public preview. The preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Azure Container for PyTorch (ACPT) (preview)

Name: AzureML-ACPT-pytorch-1.11-py38-cuda11.5-gpu

Description: The Azure Curated Environment for PyTorch is our latest PyTorch curated environment. It is optimized for large, distributed deep learning workloads and comes pre-packaged with the best of Microsoft technologies for accelerated training, e.g., OnnxRuntime Training (ORT), DeepSpeed, MSCCL, etc.

The following configurations are supported:

ENVIRONMENT NAME	OS	GPU VERSION	PYTHON VERSION	PYTORCH VERSION	ORT-TRAINING VERSION	DEEPSPEED VERSION	TORCH-ORT VERSION
AzureML-ACPT-pytorch-1.11-py38-cuda11.5-gpu	Ubuntu 20.04	cu115	3.8	1.11.0	1.11.1	0.7.1	1.11.0
AzureML-ACPT-pytorch-1.11-py38-cuda11.3-gpu	Ubuntu 20.04	cu113	3.8	1.11.0	1.11.1	0.7.1	1.11.0

PyTorch

Name: AzureML-pytorch-1.10-ubuntu18.04-py38-cuda11-gpu

Description: An environment for deep learning with PyTorch containing the AzureML Python SDK and other Python packages.

- GPU: Cuda11
- OS: Ubuntu18.04
- PyTorch: 1.10

Other available PyTorch environments:

- AzureML-pytorch-1.9-ubuntu18.04-py37-cuda11-gpu
- AzureML-pytorch-1.8-ubuntu18.04-py37-cuda11-gpu
- AzureML-pytorch-1.7-ubuntu18.04-py37-cuda11-gpu

LightGBM

Name: AzureML-lightgbm-3.2-ubuntu18.04-py37-cpu

Description: An environment for machine learning with Scikit-learn, LightGBM, XGBoost, Dask containing the AzureML Python SDK and other packages.

- OS: Ubuntu18.04
- Dask: 2021.6
- LightGBM: 3.2
- Scikit-learn: 0.24
- XGBoost: 1.4

Sklearn

Name: AzureML-sklearn-1.0-ubuntu20.04-py38-cpu

Description: An environment for tasks such as regression, clustering, and classification with Scikit-learn.

Contains the AzureML Python SDK and other Python packages.

- OS: Ubuntu20.04
- Scikit-learn: 1.0

Other available Sklearn environments:

- AzureML-sklearn-0.24-ubuntu18.04-py37-cpu

TensorFlow

Name: AzureML-tensorflow-2.4-ubuntu18.04-py37-cuda11-gpu

Description: An environment for deep learning with TensorFlow containing the AzureML Python SDK and other Python packages.

- GPU: Cuda11
- Horovod: 2.4.1
- OS: Ubuntu18.04
- TensorFlow: 2.4

Automated ML (AutoML)

Azure ML pipeline training workflows that use AutoML automatically selects a curated environment based on the compute type and whether DNN is enabled. AutoML provides the following curated environments:

NAME	COMPUTE TYPE	DNN ENABLED
AzureML-AutoML	CPU	No
AzureML-AutoML-DNN	CPU	Yes
AzureML-AutoML-GPU	GPU	No
AzureML-AutoML-DNN-GPU	GPU	Yes

For more information on AutoML and Azure ML pipelines, see [use automated ML in an Azure Machine Learning pipeline in Python](#).

Support

Version updates for supported environments, including the base images they reference, are released every two weeks to address vulnerabilities no older than 30 days. Based on usage, some environments may be deprecated (hidden from the product but usable) to support more common machine learning scenarios.

Azure Machine Learning Python SDK release notes

9/22/2022 • 137 minutes to read • [Edit Online](#)

In this article, learn about Azure Machine Learning Python SDK releases. For the full SDK reference content, visit the Azure Machine Learning's [main SDK for Python](#) reference page.

RSS feed: Get notified when this page is updated by copying and pasting the following URL into your feed reader:

`https://learn.microsoft.com/api/search/rss?search=%22Azure+machine+learning+release+notes%22&locale=en-us`

2022-08-29

Azure Machine Learning SDK for Python v1.45.0

- **azureml-automl-runtime**
 - Fixed a bug where the sample_weight column was not properly validated.
 - Added rolling_forecast() public method to the forecasting pipeline wrappers for all supported forecasting models. This method replaces the deprecated rolling_evaluation() method.
 - Fixed an issue where AutoML Regression tasks may fall back to train-valid split for model evaluation, when CV would have been a more appropriate choice.
- **azureml-core**
 - New cloud configuration suffix added, "aml_discovery_endpoint".
 - Updated the vendored azure-storage package from version 2 to version 12.
- **azureml-milflow**
 - New cloud configuration suffix added, "aml_discovery_endpoint".
- **azureml-responsibleai**
 - update azureml-responsibleai package and curated images to raiwidgets and responsibleai 0.21.0
- **azureml-sdk**
 - The azureml-sdk package now allow Python 3.9.

2022-08-01

Azure Machine Learning SDK for Python v1.44.0

- **azureml-automl-dnn-nlp**
 - Weighted accuracy and Matthews correlation coefficient (MCC) will no longer be a metric displayed on calculated metrics for NLP Multilabel classification.
- **azureml-automl-dnn-vision**
 - Raise user error when invalid annotation format is provided
- **azureml-cli-common**
 - Updated the v1 CLI description
- **azureml-contrib-automl-dnn-forecasting**
 - Fixed the "Failed to calculate TCN metrics." issues caused for TCNForecaster when different timeseries in the validation dataset have different lengths.
 - Added auto timeseries ID detection for DNN forecasting models like TCNForecaster.
 - Fixed a bug with the Forecast TCN model where validation data could be corrupted in some circumstances when the user provided the validation set.
- **azureml-core**

- Allow setting a timeout_seconds parameter when downloading artifacts from a Run
- Warning message added - Azure ML CLI v1 is getting retired on 30 Sep 2025. Users are recommended to adopt CLI v2.
- Fix submission to non-AmlComputes throwing exceptions.
- Added docker context support for environments
- **azureml-interpret**
 - Increase numpy version for AutoML packages
- **azureml-pipeline-core**
 - Fix regenerate_outputs=True not taking effect when submit pipeline.
- **azureml-train-automl-runtime**
 - Increase numpy version for AutoML packages
 - Enable code generation for vision and nlp
 - Original columns on which grains are created are added as part of predictions.csv

2022-07-21

Announcing end of support for Python 3.6 in AzureML SDK v1 packages

- **Feature deprecation**
 - Deprecate Python 3.6 as a supported runtime for SDK v1 packages
 - On December 05, 2022, AzureML will deprecate Python 3.6 as a supported runtime, formally ending our Python 3.6 support for SDK v1 packages.
 - From the deprecation date of December 05, 2022, AzureML will no longer apply security patches and other updates to the Python 3.6 runtime used by AzureML SDK v1 packages.
 - The existing AzureML SDK v1 packages with Python 3.6 still will continue to run. However, AzureML strongly recommends that you migrate your scripts and dependencies to a supported Python runtime version so that you continue to receive security patches and remain eligible for technical support.
 - We recommend using Python 3.8 version as a runtime for AzureML SDK v1 packages.
 - In addition, AzureML SDK v1 packages using Python 3.6 will no longer be eligible for technical support.
 - If you have any questions, contact us through AML Support.

2022-06-27

- **azureml-automl-dnn-nlp**
 - Remove duplicate labels column from multi-label predictions
- **azureml-contrib-automl-pipeline-steps**
 - Many Models now provide the capability to generate prediction output in csv format as well. - Many Models prediction will now include column names in the output file in case of csv file format.
- **azureml-core**
 - ADAL authentication is now deprecated and all authentication classes now use MSAL authentication. Please install `azure-cli>=2.30.0` to utilize MSAL based authentication when using `AzureCliAuthentication` class.
 - Added fix to force environment registration when `Environment.build(workspace)`. The fix solves confusion of the latest environment built instead of the asked one when environment is cloned or inherited from another instance.
 - SDK warning message to restart Compute Instance before May 31, 2022, if it was created before September 19, 2021
- **azureml-interpret**

- Updated azureml-interpret package to interpret-community 0.26.*
- In the azureml-interpret package, add ability to get raw and engineered feature names from scoring explainer. Also, add example to the scoring notebook to get feature names from the scoring explainer and add documentation about raw and engineered feature names.
- **azureml-mlflow**
 - azureml-core as a dependency of azureml-mlflow has been removed. - MLflow projects and local deployments will require azureml-core and needs to be installed separately.
 - Adding support for creating endpoints and deploying to them via the MLflow client plugin.
- **azureml-responsibleai**
 - Updated azureml-responsibleai package and environment images to latest responsibleai and raiwidgets 0.19.0 release
- **azureml-train-automl-client**
 - Now OutputDatasetConfig is supported as the input of the MM/HTS pipeline builder. The mappings are: 1) OutputTabularDatasetConfig -> treated as unpartitioned tabular dataset. 2) OutputFileDatasetConfig -> treated as file dataset.
- **azureml-train-automl-runtime**
 - Added data validation that requires the number of minority class samples in the dataset to be at least as much as the number of CV folds requested.
 - Automatic cross-validation parameter configuration is now available for AutoML forecasting tasks. Users can now specify "auto" for n_cross_validations and cv_step_size or leave them empty, and AutoML will provide those configurations base on your data. However, currently this feature is not supported when TCN is enabled.
 - Forecasting Parameters in Many Models and Hierarchical Time Series can now be passed via object rather than using individual parameters in dictionary.
 - Enabled forecasting model endpoints with quantiles support to be consumed in Power BI.
 - Updated AutoML scipy dependency upper bound to 1.5.3 from 1.5.2

2022-04-25

Azure Machine Learning SDK for Python v1.41.0

Breaking change warning

This breaking change comes from the June release of `azureml-inference-server-http`. In the `azureml-inference-server-http` June release (v0.9.0), Python 3.6 support will be dropped. Since `azureml-defaults` depends on `azureml-inference-server-http`, this change will be propagated to `azureml-defaults`. If you are not using `azureml-defaults` for inference, feel free to use `azureml-core` or any other AzureML SDK packages directly instead of install `azureml-defaults`.

- **azureml-automl-dnn-nlp**
 - Turning on long range text feature by default.
- **azureml-automl-dnn-vision**
 - Changing the ObjectAnnotation Class type from object to "dataobject".
- **azureml-core**
 - This release updates the Keyvault class used by customers to enable them to provide the keyvault content type when creating a secret using the SDK. This release also updates the SDK to include a new function that enables customers to retrieve the value of the content type from a specific secret.
- **azureml-interpret**
 - updated azureml-interpret package to interpret-community 0.25.0
- **azureml-pipeline-core**
 - Do not print run detail anymore if `pipeline_run.wait_for_completion` with `show_output=False`

- **azureml-train-automl-runtime**
 - Fixes a bug that would cause code generation to fail when the azureml-contrib-automl-dnn-forecasting package is present in the training environment.
 - Fix error when using a test dataset without a label column with AutoML Model Testing.

2022-03-28

Azure Machine Learning SDK for Python v1.40.0

- **azureml-automl-dnn-nlp**
 - We're making the Long Range Text feature optional and only if the customers explicitly opt in for it, using the kwarg "enable_long_range_text"
 - Adding data validation layer for multi-class classification scenario which leverages the same base class as multilabel for common validations, and a derived class for additional task specific data validation checks.
- **azureml-automl-dnn-vision**
 - Fixing KeyError while computing class weights.
- **azureml-contrib-reinforcementlearning**
 - SDK warning message for upcoming deprecation of RL service
- **azureml-core**
 - ○ Return logs for runs that went through our new runtime when calling any of the get logs function on the run object, including `run.get_details`, `run.get_all_logs`, etc.
 - Added experimental method Datastore.register_onpremises_hdfs to allow users to create datastores pointing to on-premises HDFS resources.
 - Updating the CLI documentation in the help command
- **azureml-interpret**
 - For `azureml-interpret` package, remove shap pin with packaging update. Remove numba and numpy pin after CE env update.
- **azureml-mlflow**
 - Bugfix for MLflow deployment client `run_local` failing when config object wasn't provided.
- **azureml-pipeline-steps**
 - Remove broken link of deprecated pipeline `EstimatorStep`
- **azureml-responsibleai**
 - update `azureml-responsibleai` package to raiwidgets and responsibleai 0.17.0 release
- **azureml-train-automl-runtime**
 - Code generation for automated ML now supports ForecastTCN models (experimental).
 - Models created via code generation will now have all metrics calculated by default (except normalized mean absolute error, normalized median absolute error, normalized RMSE, and normalized RMSLE in the case of forecasting models). The list of metrics to be calculated can be changed by editing the return value of `get_metrics_names()`. Cross validation will now be used by default for forecasting models created via code generation..
- **azureml-training-tabular**
 - The list of metrics to be calculated can be changed by editing the return value of `get_metrics_names()`. Cross validation will now be used by default for forecasting models created via code generation.
 - Converting decimal type y-test into float to allow for metrics computation to proceed without errors.

2022-02-28

Azure Machine Learning SDK for Python v1.39.0

- **azureml-automl-core**

- Fix incorrect form displayed in PBI for integration with AutoML regression models
- Adding min-label-classes check for both classification tasks (multi-class and multi-label). It will throw an error for the customer's run if the unique number of classes in the input training dataset is fewer than 2. It is meaningless to run classification on fewer than two classes.
- **azureml-automl-runtime**
 - Converting decimal type y-test into float to allow for metrics computation to proceed without errors.
 - AutoML training now supports numpy version 1.8.
- **azureml-contrib-automl-dnn-forecasting**
 - Fixed a bug in the TCNForecaster model where not all training data would be used when cross-validation settings were provided.
 - TCNForecaster wrapper's forecast method that was corrupting inference-time predictions. Also fixed an issue where the forecast method would not use the most recent context data in train-valid scenarios.
- **azureml-interpret**
 - For azureml-interpret package, remove shap pin with packaging update. Remove numba and numpy pin after CE env update.
- **azureml-responsibleai**
 - azureml-responsibleai package to raiwidgets and responsibleai 0.17.0 release
- **azureml-synapse**
 - Fix the issue that magic widget is disappeared.
- **azureml-train-automl-runtime**
 - Updating AutoML dependencies to support Python 3.8. This change will break compatibility with models trained with SDK 1.37 or below due to newer Pandas interfaces being saved in the model.
 - AutoML training now supports numpy version 1.19
 - Fix AutoML reset index logic for ensemble models in automl_setup_model_explanations API
 - In AutoML, use lightgbm surrogate model instead of linear surrogate model for sparse case after latest lightgbm version upgrade
 - All internal intermediate artifacts that are produced by AutoML are now stored transparently on the parent run (instead of being sent to the default workspace blob store). Users should be able to see the artifacts that AutoML generates under the 'outputs/' directory on the parent run.

2022-01-24

Azure Machine Learning SDK for Python v1.38.0

- **azureml-automl-core**
 - Tabnet Regressor and Tabnet Classifier support in AutoML
 - Saving data transformer in parent run outputs, which can be reused to produce same featurized dataset which was used during the experiment run
 - Supporting getting primary metrics for Forecasting task in get_primary_metrics API.
 - Renamed second optional parameter in v2 scoring scripts as GlobalParameters
- **azureml-automl-dnn-vision**
 - Added the scoring metrics in the metrics UI
- **azureml-automl-runtime**
 - Bug fix for cases where the algorithm name for NimbusML models may show up as empty strings, either on the ML Studio, or on the console outputs.
- **azureml-core**
 - Added parameter blobfuse_enabled in
azureml.core.webservice.aks.AksWebservice.deploy_configuration. When this parameter is true,

models and scoring files will be downloaded with blobfuse instead of the blob storage API.

- **azureml-interpret**
 - Updated azureml-interpret to interpret-community 0.24.0
 - In azureml-interpret update scoring explainer to support latest version of lightgbm with sparse TreeExplainer
 - Update azureml-interpret to interpret-community 0.23.*
- **azureml-pipeline-core**
 - Add note in pipelinedata, recommend user to use pipeline output dataset instead.
- **azureml-pipeline-steps**
 - Add `environment_variables` to ParallelRunConfig, runtime environment variables can be passed by this parameter and will be set on the process where the user script is executed.
- **azureml-train-automl-client**
 - Tabnet Regressor and Tabnet Classifier support in AutoML
- **azureml-train-automl-runtime**
 - Saving data transformer in parent run outputs, which can be reused to produce same featurized dataset which was used during the experiment run
- **azureml-train-core**
 - Enable support for early termination for Bayesian Optimization in Hyperdrive
 - Bayesian and GridParameterSampling objects can now pass on properties

2021-12-13

Azure Machine Learning SDK for Python v1.37.0

- **Breaking changes**
 - **azureml-core**
 - Starting in version 1.37.0, AzureML SDK uses MSAL as the underlying authentication library. MSAL uses Azure Active Directory (Azure AD) v2.0 authentication flow to provide more functionality and increases security for token cache. For more details, see [Overview of the Microsoft Authentication Library \(MSAL\)](#).
 - Update AML SDK dependencies to the latest version of Azure Resource Management Client Library for Python (`azure-mgmt-resource`) >= 15.0.0, < 20.0.0) & adopt track2 SDK.
 - Starting in version 1.37.0, `azure-ml-cli` extension should be compatible with the latest version of Azure CLI >= 2.30.0.
 - When using Azure CLI in a pipeline, like as Azure DevOps, ensure all tasks/stages are using versions of Azure CLI above v2.30.0 for MSAL-based Azure CLI. Azure CLI 2.30.0 is not backward compatible with prior versions and throws an error when using incompatible versions. To use Azure CLI credentials with AzureML SDK, Azure CLI should be installed as pip package.
- **Bug fixes and improvements**
 - **azureml-core**
 - Removed instance types from the attach workflow for Kubernetes compute. Instance types can now directly be set up in the Kubernetes cluster. For more details, please visit [aka.ms/amlarc/doc](#).
 - **azureml-interpret**
 - updated `azureml-interpret` to `interpret-community` 0.22.*
 - **azureml-pipeline-steps**
 - Fixed a bug where the experiment "placeholder" might be created on submission of a Pipeline with an `AutoMLStep`.

- **azureml-responsibleai**
 - update azureml-responsibleai and compute instance environment to responsibleai and raiwidgets 0.15.0 release
 - update azureml-responsibleai package to latest responsibleai 0.14.0.
- **azureml-tensorboard**
 - You can now use `Tensorboard(runs, use_display_name=True)` to mount the TensorBoard logs to folders named after the `run.display_name/run.id` instead of `run.id`.
- **azureml-train-automl-client**
 - Fixed a bug where the experiment "placeholder" might be created on submission of a Pipeline with an AutoMLStep.
 - Update AutoMLConfig test_data and test_size docs to reflect preview status.
- **azureml-train-automl-runtime**
 - Added new feature that allows users to pass time series grains with one unique value.
 - In certain scenarios, an AutoML model can predict NaNs. The rows that correspond to these NaN predictions will be removed from test datasets and predictions before computing metrics in test runs.

2021-11-08

Azure Machine Learning SDK for Python v1.36.0

- Bug fixes and improvements
 - **azureml-automl-dnn-vision**
 - Cleaned up minor typos on some error messages.
 - **azureml-contrib-reinforcementlearning**
 - Submitting Reinforcement Learning runs that use simulators is no longer supported.
 - **azureml-core**
 - Added support for partitioned premium blob.
 - Specifying non-public clouds for Managed Identity authentication is no longer supported.
 - User can migrate AKS web service to online endpoint and deployment which is managed by CLI (v2).
 - The instance type for training jobs on Kubernetes compute targets can now be set via a RunConfiguration property: `run_config.kubernetescompute.instance_type`.
 - **azureml-defaults**
 - Removed redundant dependencies like gunicorn and werkzeug
 - **azureml-interpret**
 - `azureml-interpret` package updated to 0.21.* version of interpret-community
 - **azureml-pipeline-steps**
 - Deprecate MpiStep in favor of using CommandStep for running ML training (including distributed training) in pipelines.
 - **azureml-train-automl-rutime**
 - Update the AutoML model test predictions output format docs.
 - Added docstring descriptions for Naive, SeasonalNaive, Average, and SeasonalAverage forecasting model.
 - Featurization summary is now stored as an artifact on the run (check for a file named 'featurization_summary.json' under the outputs folder)
 - Enable categorical indicators support for Tabnet Learner.
 - Add downsample parameter to `automl_setup_model_explanations` to allow users to get explanations on all data without downsampling by setting this parameter to be false.

2021-10-11

Azure Machine Learning SDK for Python v1.35.0

- Bug fixes and improvements
 - **azureml-automl-core**
 - Enable binary metrics calculation
 - **azureml-contrib-fairness**
 - Improve error message on failed dashboard download
 - **azureml-core**
 - Bug in specifying non-public clouds for Managed Identity authentication has been resolved.
 - Dataset.File.upload_directory() and Dataset.Tabular.register_pandas_dataframe() experimental flags are now removed.
 - Experimental flags are now removed in partition_by() method of TabularDataset class.
 - **azureml-pipeline-steps**
 - Experimental flags are now removed for the `partition_keys` parameter of the ParallelRunConfig class.
 - **azureml-interpret**
 - azureml-interpret package updated to intepret-community 0.20.*
 - **azureml-mlflow**
 - Made it possible to log artifacts and images with MLflow using subdirectories
 - **azureml-responsibleai**
 - Improve error message on failed dashboard download
 - **azureml-train-automl-client**
 - Added support for computer vision tasks such as Image Classification, Object Detection and Instance Segmentation. Detailed documentation can be found at: [How to automatically train image models](#)
 - Enable binary metrics calculation
 - **azureml-train-automl-runtime**
 - Add TCNForecaster support to model test runs.
 - Update the model test predictions.csv output format. The output columns now include the original target values and the features which were passed in to the test run. This can be turned off by setting `test_include_predictions_only=True` in `AutoMLConfig` or by setting `include_predictions_only=True` in `ModelProxy.test()`. If the user has requested to only include predictions then the output format looks like (forecasting is the same as regression):
Classification => [predicted values] [probabilities] Regression => [predicted values] else (default): Classification => [original test data labels] [predicted values] [probabilities] [features]
Regression => [original test data labels] [predicted values] [features] The `[predicted values]` column name = `[label column name] + "_predicted"`. The `[probabilities]` column names = `[class name] + "_predicted_proba"`. If no target column was passed in as input to the test run, then `[original test data labels]` will not be in the output.

2021-09-07

Azure Machine Learning SDK for Python v1.34.0

- Bug fixes and improvements
 - **azureml-automl-core**
 - Added support for re-fitting a previously trained forecasting pipeline.
 - Added ability to get predictions on the training data (in-sample prediction) for forecasting.

- **azureml-automl-runtime**
 - Add support to return predicted probabilities from a deployed endpoint of an AutoML classifier model.
 - Added a forecasting option for users to specify that all predictions should be integers.
 - Removed the target column name from being part of model explanation feature names for local experiments with training_data_label_column_name
 - as dataset inputs.
 - Added support for re-fitting a previously trained forecasting pipeline.
 - Added ability to get predictions on the training data (in-sample prediction) for forecasting.
- **azureml-core**
 - Added support to set stream column type, mount and download stream columns in tabular dataset.
 - New optional fields added to Kubernetes.attach_configuration(identity_type=None, identity_ids=None) which allow attaching KubernetesCompute with either SystemAssigned or UserAssigned identity. New identity fields will be included when calling print(compute_target) or compute_target.serialize(): identity_type, identity_id, principal_id, and tenant_id/client_id.
- **azureml-datatransform**
 - Added support to set stream column type for tabular dataset. added support to mount and download stream columns in tabular dataset.
- **azureml-defaults**
 - The dependency `azureml-inference-server-http==0.3.1` has been added to `azureml-defaults`.
- **azureml-mlflow**
 - Allow pagination of list_experiments API by adding `max_results` and `page_token` optional params. For documentation, see MLflow official docs.
- **azureml-sdk**
 - Replaced dependency on deprecated package(azureml-train) inside azureml-sdk.
 - Add azureml-responsibleai to azureml-sdk extras
- **azureml-train-automl-client**
 - Expose the `test_data` and `test_size` parameters in `AutoMLConfig`. These parameters can be used to automatically start a test run after the model
 - training phase has been completed. The test run will compute predictions using the best model and will generate metrics given these predictions.

2021-08-24

Azure Machine Learning Experimentation User Interface

- **Run Delete**
 - Run Delete is a new functionality that allows users to delete one or multiple runs from their workspace.
 - This functionality can help users reduce storage costs and manage storage capacity by regularly deleting runs and experiments from the UI directly.
- **Batch Cancel Run**
 - Batch Cancel Run is new functionality that allows users to select one or multiple runs to cancel from their run list.
 - This functionality can help users cancel multiple queued runs and free up space on their cluster.

2021-08-18

Azure Machine Learning Experimentation User Interface

- **Run Display Name**
 - The Run Display Name is a new, editable and optional display name that can be assigned to a run.
 - This name can help with more effectively tracking, organizing and discovering the runs.
 - The Run Display Name is defaulted to an adjective_noun_guid format (Example: awesome_watch_2i3uns).
 - This default name can be edited to a more customizable name. This can be edited from the Run details page in the Azure Machine Learning studio user interface.

2021-08-02

Azure Machine Learning SDK for Python v1.33.0

- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Improved error handling around XGBoost model retrieval.
 - Added possibility to convert the predictions from float to integers for forecasting and regression tasks.
 - Updated default value for enable_early_stopping in AutoMLConfig to True.
 - **azureml-automl-runtime**
 - Added possibility to convert the predictions from float to integers for forecasting and regression tasks.
 - Updated default value for enable_early_stopping in AutoMLConfig to True.
 - **azureml-contrib-automl-pipeline-steps**
 - Hierarchical timeseries (HTS) is enabled for forecasting tasks through pipelines.
 - Add Tabular dataset support for inferencing
 - Custom path can be specified for the inference data
 - **azureml-contrib-reinforcementlearning**
 - Some properties in `azureml.core.environment.DockerSection` are deprecated, such as `shm_size` property used by Ray workers in reinforcement learning jobs. This property can now be specified in `azureml.contrib.train.rl.WorkerConfiguration` instead.
 - **azureml-core**
 - Fixed a hyperlink in `ScriptRunConfig.distributed_job_config` documentation
 - Azure Machine Learning compute clusters can now be created in a location different from the location of the workspace. This is useful for maximizing idle capacity allocation and managing quota utilization across different locations without having to create more workspaces just to use quota and create a compute cluster in a particular location. For more information, see [Create an Azure Machine Learning compute cluster](#).
 - Added `display_name` as a mutable name field of Run object.
 - Dataset `from_files` now supports skipping of data extensions for large input data
 - **azureml-datatprep**
 - Fixed a bug where `to_dask_dataframe` would fail because of a race condition.
 - Dataset `from_files` now supports skipping of data extensions for large input data
 - **azureml-defaults**
 - We are removing the dependency `azureml-model-management-sdk==1.0.1b6.post1` from `azureml-defaults`.
 - **azureml-interpret**
 - updated `azureml-interpret` to `interpret-community 0.19.*`
 - **azureml-pipeline-core**
 - Hierarchical timeseries (HTS) is enabled for forecasting tasks through pipelines.

- **azureml-train-automl-client**
 - Switch to using blob store for caching in Automated ML.
 - Hierarchical timeseries (HTS) is enabled for forecasting tasks through pipelines.
 - Improved error handling around XGBoost model retrieval.
 - Updated default value for enable_early_stopping in AutoMLConfig to True.
- **azureml-train-automl-runtime**
 - Switch to using blob store for caching in Automated ML.
 - Hierarchical timeseries (HTS) is enabled for forecasting tasks through pipelines.
 - Updated default value for enable_early_stopping in AutoMLConfig to True.

2021-07-06

Azure Machine Learning SDK for Python v1.32.0

- Bug fixes and improvements
 - **azureml-core**
 - Expose diagnose workspace health in SDK/CLI
 - **azureml-defaults**
 - Added `opencensus-ext-azure==1.0.8` dependency to azureml-defaults
 - **azureml-pipeline-core**
 - Updated the AutoMLStep to use prebuilt images when the environment for job submission matches the default environment
 - **azureml-responsibleai**
 - New error analysis client added to upload, download and list error analysis reports
 - Ensure `raiwidgets` and `responsibleai` packages are version synchronized
 - **azureml-train-automl-runtime**
 - Set the time allocated to dynamically search across various featurization strategies to a maximum of one-fourth of the overall experiment timeout

2021-06-21

Azure Machine Learning SDK for Python v1.31.0

- Bug fixes and improvements
 - **azureml-core**
 - Improved documentation for platform property on Environment class
 - Changed default AML Compute node scale down time from 120 seconds to 1800 seconds
 - Updated default troubleshooting link displayed on the portal for troubleshooting failed runs to: <https://aka.ms/azureml-run-troubleshooting>
 - **azureml-automl-runtime**
 - Data Cleaning: Samples with target values in [None, "", "nan", np.nan] will be dropped prior to featurization and/or model training
 - **azureml-interpret**
 - Prevent flush task queue error on remote AzureML runs that use ExplanationClient by increasing timeout
 - **azureml-pipeline-core**
 - Add jar parameter to synapse step
 - **azureml-train-automl-runtime**
 - Fix high cardinality guardrails to be more aligned with docs

2021-06-07

Azure Machine Learning SDK for Python v1.30.0

- Bug fixes and improvements

- **azureml-core**

- Pin dependency `ruamel-yaml` to < 0.17.5 as a breaking change was released in 0.17.5.

- `aml_k8s_config` property is being replaced with `namespace`, `default_instance_type`, and `instance_types` parameters for `KubernetesCompute` attach.

- Workspace sync keys was changed to a long running operation.

- **azureml-automl-runtime**

- Fixed problems where runs with big data may fail with `Elements of y_test cannot be NaN`.

- **azureml-mlflow**

- MLFlow deployment plugin bugfix for models with no signature.

- **azureml-pipeline-steps**

- ParallelRunConfig: update doc for process_count_per_node.

- **azureml-train-automl-runtime**

- Support for custom defined quantiles during MM inference

- Support for forecast_quantiles during batch inference.

- **azureml-contrib-automl-pipeline-steps**

- Support for custom defined quantiles during MM inference

- Support for forecast_quantiles during batch inference.

2021-05-25

Announcing the CLI (v2) (preview) for Azure Machine Learning

The `ml` extension to the Azure CLI is the next-generation interface for Azure Machine Learning. It enables you to train and deploy models from the command line, with features that accelerate scaling data science up and out while tracking the model lifecycle. [Install and get started](#).

Azure Machine Learning SDK for Python v1.29.0

- Bug fixes and improvements

- **Breaking changes**

- Dropped support for Python 3.5.

- **azureml-automl-runtime**

- Fixed a bug where the STLFeaturizer failed if the time-series length was shorter than the seasonality. This error manifested as an IndexError. The case is handled now without error, though the seasonal component of the STL will just consist of zeros in this case.

- **azureml-contrib-automl-dnn-vision**

- Added a method for batch inferencing with file paths.

- **azureml-contrib-gbdt**

- The `azureml-contrib-gbdt` package has been deprecated and might not receive future updates and will be removed from the distribution altogether.

- **azureml-core**

- Corrected explanation of parameter `create_if_not_exists` in `Datastore.register_azure_blob_container`.

- Added sample code to `DatasetConsumptionConfig` class.

- Added support for step as an alternative axis for scalar metric values in `run.log()`

- **azureml-dataprep**

- Limit partition size accepted in `_with_partition_size()` to 2GB
- **azureml-interpret**
 - update `azureml-interpret` to the latest `interpret-core` package version
 - Dropped support for SHAP `DenseData`, which has been deprecated in SHAP 0.36.0.
 - Enable `ExplanationClient` to upload to a user specified datastore.
- **azureml-mlflow**
 - Move `azureml-mlflow` to `mlflow-skinny` to reduce the dependency footprint while maintaining full plugin support
- **azureml-pipeline-core**
 - PipelineParameter code sample is updated in the reference doc to use correct parameter.

2021-05-10

Azure Machine Learning SDK for Python v1.28.0

- Bug fixes and improvements
 - **azureml-automl-runtime**
 - Improved AutoML Scoring script to make it consistent with designer
 - Patch bug where forecasting with the Prophet model would throw a "missing column" error if trained on an earlier version of the SDK.
 - Added the ARIMAX model to the public-facing, forecasting-supported model lists of the AutoML SDK. Here, ARIMAX is a regression with ARIMA errors and a special case of the transfer function models developed by Box and Jenkins. For a discussion of how the two approaches are different, see [The ARIMAX model muddle](#). Unlike the rest of the multivariate models that use auto-generated, time-dependent features (hour of the day, day of the year, and so on) in AutoML, this model uses only features that are provided by the user, and it makes interpreting coefficients easy.
 - **azureml-contrib-dataset**
 - Updated documentation description with indication that libfuse should be installed while using mount.
 - **azureml-core**
 - Default CPU curated image is now `mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04`. Default GPU image is now `mcr.microsoft.com/azureml/openmpi3.1.2-cuda10.2-cudnn8-ubuntu18.04`
 - `Run.fail()` is now deprecated, use `Run.tag()` to mark run as failed or use `Run.cancel()` to mark the run as canceled.
 - Updated documentation with a note that libfuse should be installed when mounting a file dataset.
 - Add experimental `register_dask_dataframe()` support to tabular dataset.
 - Support DatabricksStep with Azure Blob/ADL-S as inputs/outputs and expose parameter `permit_cluster_restart` to let customer decide whether AML can restart cluster when i/o access configuration need to be added into cluster
 - **azureml-dataset-runtime**
 - `azureml-dataset-runtime` now supports versions of pyarrow < 4.0.0
 - **azureml-mlflow**
 - Added support for deploying to AzureML via our MLFlow plugin.
 - **azureml-pipeline-steps**
 - Support DatabricksStep with Azure Blob/ADL-S as inputs/outputs and expose parameter `permit_cluster_restart` to let customer decide whether AML can restart cluster when i/o access configuration need to be added into cluster

- **azureml-synapse**
 - Enable audience in msi authentication
- **azureml-train-automl-client**
 - Added changed link for compute target doc

2021-04-19

Azure Machine Learning SDK for Python v1.27.0

- **Bug fixes and improvements**
 - **azureml-core**
 - Added the ability to override the default timeout value for artifact uploading via the "AZUREML_ARTIFACTS_DEFAULT_TIMEOUT" environment variable.
 - Fixed a bug where docker settings in Environment object on ScriptRunConfig are not respected.
 - Allow partitioning a dataset when copying it to a destination.
 - Added a custom mode to the OutputDatasetConfig to enable passing created Datasets in pipelines through a link function. These support enhancements made to enable Tabular Partitioning for PRS.
 - Added a new KubernetesCompute compute type to azureml-core.
 - **azureml-pipeline-core**
 - Adding a custom mode to the OutputDatasetConfig and enabling a user to pass through created Datasets in pipelines through a link function. File path destinations support placeholders. These support the enhancements made to enable Tabular Partitioning for PRS.
 - Addition of new KubernetesCompute compute type to azureml-core.
 - **azureml-pipeline-steps**
 - Addition of new KubernetesCompute compute type to azureml-core.
 - **azureml-synapse**
 - Update spark UI url in widget of azureml synapse
 - **azureml-train-automl-client**
 - The STL featurizer for the forecasting task now uses a more robust seasonality detection based on the frequency of the time series.
 - **azureml-train-core**
 - Fixed bug where docker settings in Environment object are not respected.
 - Addition of new KubernetesCompute compute type to azureml-core.

2021-04-05

Azure Machine Learning SDK for Python v1.26.0

- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Fixed an issue where Naive models would be recommended in AutoMLStep runs and fail with lag or rolling window features. These models will not be recommended when target lags or target rolling window size are set.
 - Changed console output when submitting an AutoML run to show a portal link to the run.
 - **azureml-core**
 - Added HDFS mode in documentation.
 - Added support to understand File Dataset partitions based on glob structure.
 - Added support for update container registry associated with AzureML Workspace.
 - Deprecated Environment attributes under the DockerSection - "enabled", "shared_volume" and

"arguments" are a part of DockerConfiguration in RunConfiguration now.

- Updated Pipeline CLI clone documentation
- Updated portal URIs to include tenant for authentication
- Removed experiment name from run URIs to avoid redirects
- Updated experiment URO to use experiment ID.
- Bug fixes for attaching remote compute with AzureML CLI.
- Updated portal URIs to include tenant for authentication.
- Updated experiment URI to use experiment ID.
- **azureml-interpret**
 - `azureml-interpret` updated to use `interpret-community` 0.17.0
- **azureml-opendatasets**
 - Input start date and end date type validation and error indication if it's not datetime type.
- **azureml-parallel-run**
 - [Experimental feature] Add `partition_keys` parameter to `ParallelRunConfig`, if specified, the input dataset(s) would be partitioned into mini-batches by the keys specified by it. It requires all input datasets to be partitioned dataset.
- **azureml-pipeline-steps**
 - Bugfix - supporting `path_on_compute` while passing dataset configuration as download.
 - Deprecate `RScriptStep` in favor of using `CommandStep` for running R scripts in pipelines.
 - Deprecate `EstimatorStep` in favor of using `CommandStep` for running ML training (including distributed training) in pipelines.
- **azureml-sdk**
 - Update `python_requires` to < 3.9 for `azureml-sdk`
- **azureml-train-automl-client**
 - Changed console output when submitting an AutoML run to show a portal link to the run.
- **azureml-train-core**
 - Deprecated DockerSection's 'enabled', 'shared_volume', and 'arguments' attributes in favor of using DockerConfiguration with `ScriptRunConfig`.
 - Use Azure Open Datasets for MNIST dataset
 - Hyperdrive error messages have been updated.

2021-03-22

Azure Machine Learning SDK for Python v1.25.0

• Bug fixes and improvements

- **azureml-automl-core**
 - Changed console output when submitting an AutoML run to show a portal link to the run.
- **azureml-core**
 - Starts to support updating container registry for workspace in SDK and CLI
 - Deprecated DockerSection's 'enabled', 'shared_volume', and 'arguments' attributes in favor of using DockerConfiguration with `ScriptRunConfig`.
 - Updated Pipeline CLI clone documentation
 - Updated portal URIs to include tenant for authentication
 - Removed experiment name from run URIs to avoid redirects
 - Updated experiment URO to use experiment ID.
 - Bug fixes for attaching remote compute using `az` CLI
 - Updated portal URIs to include tenant for authentication.

- Added support to understand File Dataset partitions based on glob structure.
- **azureml-interpret**
 - azureml-interpret updated to use interpret-community 0.17.0
- **azureml-opendatasets**
 - Input start date and end date type validation and error indication if it's not datetime type.
- **azureml-pipeline-core**
 - Bugfix - supporting path_on_compute while passing dataset configuration as download.
- **azureml-pipeline-steps**
 - Bugfix - supporting path_on_compute while passing dataset configuration as download.
 - Deprecate RScriptStep in favor of using CommandStep for running R scripts in pipelines.
 - Deprecate EstimatorStep in favor of using CommandStep for running ML training (including distributed training) in pipelines.
- **azureml-train-automl-runtime**
 - Changed console output when submitting an AutoML run to show a portal link to the run.
- **azureml-train-core**
 - Deprecated DockerSection's 'enabled', 'shared_volume', and 'arguments' attributes in favor of using DockerConfiguration with ScriptRunConfig.
 - Use Azure Open Datasets for MNIST dataset
 - Hyperdrive error messages have been updated.

2021-03-31

Azure Machine Learning studio Notebooks Experience (March Update)

- **New features**
 - Render CSV/TSV. Users will be able to render and TSV/CSV file in a grid format for easier data analysis.
 - SSO Authentication for Compute Instance. Users can now easily authenticate any new compute instances directly in the Notebook UI, making it easier to authenticate and use Azure SDKs directly in AzureML.
 - Compute Instance Metrics. Users will be able to view compute metrics like CPU usage and memory via terminal.
 - File Details. Users can now see file details including the last modified time, and file size by clicking the 3 dots beside a file.
- **Bug fixes and improvements**
 - Improved page load times.
 - Improved performance.
 - Improved speed and kernel reliability.
 - Gain vertical real estate by permanently moving Notebook file pane up
 - Links are now clickable in Terminal
 - Improved Intellisense performance

2021-03-08

Azure Machine Learning SDK for Python v1.24.0

- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Removed backwards compatible imports from `azureml.automl.core.shared`. Module not found

errors in the `azureml.automl.core.shared` namespace can be resolved by importing from `azureml.automl.runtime.shared`.

- **azureml-contrib-automl-dnn-vision**
 - Exposed object detection yolo model.
- **azureml-contrib-dataset**
 - Added functionality to filter Tabular Datasets by column values and File Datasets by metadata.
- **azureml-contrib-fairness**
 - Include JSON schema in wheel for `azureml-contrib-fairness`
- **azureml-contrib-mir**
 - With setting `show_output` to True when deploy models, inference configuration and deployment configuration will be replayed before sending the request to server.
- **azureml-core**
 - Added functionality to filter Tabular Datasets by column values and File Datasets by metadata.
 - Previously, it was possible for users to create provisioning configurations for ComputeTarget's that did not satisfy the password strength requirements for the `admin_user_password` field (i.e., that they must contain at least 3 of the following: 1 lowercase letter, 1 uppercase letter, 1 digit, and 1 special character from the following set: `\`~!@#$%^&*()=+_[]{}|;.:/'",<>?`). If the user created a configuration with a weak password and ran a job using that configuration, the job would fail at runtime. Now, the call to `AmlCompute.provisioning_configuration` will throw a `ComputeTargetException` with an accompanying error message explaining the password strength requirements.
 - Additionally, it was also possible in some cases to specify a configuration with a negative number of maximum nodes. It is no longer possible to do this. Now, `AmlCompute.provisioning_configuration` will throw a `ComputeTargetException` if the `max_nodes` argument is a negative integer.
 - With setting `show_output` to True when deploy models, inference configuration and deployment configuration will be displayed.
 - With setting `show_output` to True when wait for the completion of model deployment, the progress of deployment operation will be displayed.
 - Allow customer specified AzureML auth config directory through environment variable: `AZUREML_AUTH_CONFIG_DIR`
 - Previously, it was possible to create a provisioning configuration with the minimum node count less than the maximum node count. The job would run but fail at runtime. This bug has now been fixed. If you now try to create a provisioning configuration with `min_nodes < max_nodes` the SDK will raise a `ComputeTargetException`.
- **azureml-interpret**
 - fix explanation dashboard not showing aggregate feature importances for sparse engineered explanations
 - optimized memory usage of ExplanationClient in `azureml-interpret` package
- **azureml-train-automl-client**
 - Fixed `show_output=False` to return control to the user when running using spark.

2021-02-28

Azure Machine Learning studio Notebooks Experience (February Update)

- New features

- **Native Terminal (GA).** Users will now have access to an integrated terminal as well as Git operation via the integrated terminal.

- Notebook Snippets (preview). Common Azure ML code excerpts are now available at your fingertips. Navigate to the code snippets panel, accessible via the toolbar, or activate the in-code snippets menu using Ctrl + Space.
- [Keyboard Shortcuts](#). Full parity with keyboard shortcuts available in Jupyter.
- Indicate Cell parameters. Shows users which cells in a notebook are parameter cells and can run parameterized notebooks via [Papermill](#) on the Compute Instance.
- Terminal and Kernel session manager: Users will be able to manage all kernels and terminal sessions running on their compute.
- Sharing Button. Users can now share any file in the Notebook file explorer by right-clicking the file and using the share button.

- **Bug fixes and improvements**

- Improved page load times
- Improved performance
- Improved speed and kernel reliability
- Added spinning wheel to show progress for all ongoing [Compute Instance operations](#).
- Right click in File Explorer. Right-clicking any file will now open file operations.

2021-02-16

Azure Machine Learning SDK for Python v1.23.0

- **Bug fixes and improvements**

- **azureml-core**
 - [Experimental feature] Add support to link synapse workspace into AML as a linked service
 - [Experimental feature] Add support to attach synapse spark pool into AML as a compute
 - [Experimental feature] Add support for identity based data access. Users can register datastore or datasets without providing credentials. In such case, users' Azure AD token or managed identity of compute target will be used for authentication. To learn more, see [Connect to storage by using identity-based data access](#).
- **azureml-pipeline-steps**
 - [Experimental feature] Add support for [SynapseSparkStep](#)
- **azureml-synapse**
 - [Experimental feature] Add support of spark magic to run interactive session in synapse spark pool.

- **Bug fixes and improvements**

- **azureml-automl-runtime**
 - In this update, we added holt winters exponential smoothing to forecasting toolbox of AutoML SDK. Given a time series, the best model is selected by [AICc \(Corrected Akaike's Information Criterion\)](#) and returned.
 - AutoML will now generate two log files instead of one. Log statements will go to one or the other depending on which process the log statement was generated in.
 - Remove unnecessary in-sample prediction during model training with cross-validations. This may decrease model training time in some cases, especially for time-series forecasting models.
- **azureml-contrib-fairness**
 - Add a JSON schema for the dashboardDictionary uploads.
- **azureml-contrib-interpret**
 - [azureml-contrib-interpret](#) README is updated to reflect that package will be removed in next update after being deprecated since October, use [azureml-interpret](#) package instead
- **azureml-core**

- Previously, it was possible to create a provisioning configuration with the minimum node count less than the maximum node count. This has now been fixed. If you now try to create a provisioning configuration with `min_nodes < max_nodes` the SDK will raise a `ComputeTargetException`.
- Fixes bug in `wait_for_completion` in `AmlCompute` which caused the function to return control flow before the operation was actually complete
- `Run.fail()` is now deprecated, use `Run.tag()` to mark run as failed or use `Run.cancel()` to mark the run as canceled.
- Show error message 'Environment name expected str, {} found' when provided environment name is not a string.
- **azureml-train-automl-client**
 - Fixed a bug that prevented AutoML experiments performed on Azure Databricks clusters from being canceled.

2021-02-09

Azure Machine Learning SDK for Python v1.22.0

- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Fixed bug where an extra pip dependency was added to the conda yml file for vision models.
 - **azureml-automl-runtime**
 - Fixed a bug where classical forecasting models (e.g. AutoArima) could receive training data wherein rows with imputed target values were not present. This violated the data contract of these models. * Fixed various bugs with lag-by-occurrence behavior in the time-series lagging operator. Previously, the lag-by-occurrence operation did not mark all imputed rows correctly and so would not always generate the correct occurrence lag values. Also fixed some compatibility issues between the lag operator and the rolling window operator with lag-by-occurrence behavior. This previously resulted in the rolling window operator dropping some rows from the training data that it should otherwise use.
 - **azureml-core**
 - Adding support for Token Authentication by audience.
 - Add `process_count` to [PyTorchConfiguration](#) to support multi-process multi-node PyTorch jobs.
 - **azureml-pipeline-steps**
 - [CommandStep](#) now GA and no longer experimental.
 - [ParallelRunConfig](#): add argument `allowed_failed_count` and `allowed_failed_percent` to check error threshold on mini batch level. Error threshold has 3 flavors now:
 - `error_threshold` - the number of allowed failed mini batch items;
 - `allowed_failed_count` - the number of allowed failed mini batches;
 - `allowed_failed_percent`- the percent of allowed failed mini batches.
A job will stop if exceeds any of them. `error_threshold` is required to keep it backward compatibility. Set the value to -1 to ignore it.
 - Fixed whitespace handling in AutoMLStep name.
 - ScriptRunConfig is now supported by HyperDriveStep
 - **azureml-train-core**
 - HyperDrive runs invoked from a ScriptRun will now be considered a child run.
 - Add `process_count` to [PyTorchConfiguration](#) to support multi-process multi-node PyTorch jobs.

- **azureml-widgets**
 - Add widget ParallelRunStepDetails to visualize status of a ParallelRunStep.
 - Allows hyperdrive users to see an additional axis on the parallel coordinates chart that shows the metric value corresponding to each set of hyperparameters for each child run.

2021-01-31

Azure Machine Learning studio Notebooks Experience (January Update)

- **New features**

- Native Markdown Editor in AzureML. Users can now render and edit markdown files natively in AzureML Studio.
- [Run Button for Scripts \(.py, .R and .sh\)](#). Users can easily now run Python, R and Bash script in AzureML
- [Variable Explorer](#). Explore the contents of variables and data frames in a pop-up panel. Users can easily check data type, size, and contents.
- [Table of Content](#). Navigate to sections of your notebook, indicated by Markdown headers.
- Export your Notebook as Latex/HTML/Py. Create easy-to-share notebook files by exporting to LaTex, HTML, or .py
- Intellicode. ML-powered results provides an enhanced [intelligent autocomplete experience](#).

- **Bug fixes and improvements**

- Improved page load times
- Improved performance
- Improved speed and kernel reliability

2021-01-25

Azure Machine Learning SDK for Python v1.21.0

- **Bug fixes and improvements**

- **azure-cli-ml**
 - Fixed CLI help text when using AmlCompute with UserAssigned Identity
- **azureml-contrib-automl-dnn-vision**
 - Deploy and download buttons will become visible for AutoML vision runs, and models can be deployed or downloaded similar to other AutoML runs. There are two new files (scoring_file_v_1_0_0.py and conda_env_v_1_0_0.yml) which contain a script to run inferencing and a yml file to recreate the conda environment. The 'model.pth' file has also been renamed to use the '.pt' extension.
- **azureml-core**
 - MSI support for azure-cli-ml
 - User Assigned Managed Identity Support.
 - With this change, the customers should be able to provide a user assigned identity that can be used to fetch the key from the customer key vault for encryption at rest.
 - fix row_count=0 for the profile of very large files - fix error in double conversion for delimited values with white space padding
 - Remove experimental flag for Output dataset GA
 - Update documentation on how to fetch specific version of a Model
 - Allow updating workspace for mixed mode access in case of private link
 - Fix to remove additional registration on datastore for resume run feature
 - Added CLI/SDK support for updating primary user assigned identity of workspace
- **azureml-interpret**

- updated azureml-interpret to interpret-community 0.16.0
- memory optimizations for explanation client in azureml-interpret
- **azureml-train-automl-runtime**
 - Enabled streaming for ADB runs
- **azureml-train-core**
 - Fix to remove additional registration on datastore for resume run feature
- **azureml-widgets**
 - Customers should not see changes to existing run data visualization using the widget, and now will have support if they optionally use conditional hyperparameters.
 - The user run widget now includes a detailed explanation for why a run is in the queued state.

2021-01-11

Azure Machine Learning SDK for Python v1.20.0

- **Bug fixes and improvements**
 - **azure-cli-ml**
 - framework_version added in OptimizationConfig. It will be used when model is registered with framework MULTI.
 - **azureml-contrib-optimization**
 - framework_version added in OptimizationConfig. It will be used when model is registered with framework MULTI.
 - **azureml-pipeline-steps**
 - Introducing CommandStep which would take command to process. Command can include executables, shell commands, scripts, etc.
 - **azureml-core**
 - Now workspace creation supports user assigned identity. Adding the uai support from SDK/CLI
 - Fixed issue on service.reload() to pick up changes on score.py in local deployment.
 - `run.get_details()` has an extra field named "submittedBy" which displays the author's name for this run.
 - Edited Model.register method documentation to mention how to register model from run directly
 - Fixed IOT-Server connection status change handling issue.

2020-12-31

Azure Machine Learning studio Notebooks Experience (December Update)

- **New features**
 - User Filename search. Users are now able to search all the files saved in a workspace.
 - Markdown Side by Side support per Notebook Cell. In a notebook cell, users can now have the option to view rendered markdown and markdown syntax side-by-side.
 - Cell Status Bar. The status bar indicates what state a code cell is in, whether a cell run was successful, and how long it took to run.
- **Bug fixes and improvements**
 - Improved page load times
 - Improved performance
 - Improved speed and kernel reliability

2020-12-07

Azure Machine Learning SDK for Python v1.19.0

- Bug fixes and improvements
 - **azureml-automl-core**
 - Added experimental support for test data to AutoMLStep.
 - Added the initial core implementation of test set ingestion feature.
 - Moved references to sklearn.externals.joblib to depend directly on joblib.
 - introduce a new AutoML task type of "image-instance-segmentation".
 - **azureml-automl-runtime**
 - Added the initial core implementation of test set ingestion feature.
 - When all the strings in a text column have a length of exactly 1 character, the TfIdf word-gram featurizer won't work because its tokenizer ignores the strings with fewer than 2 characters.
The current code change will allow AutoML to handle this use case.
 - introduce a new AutoML task type of "image-instance-segmentation".
 - **azureml-contrib-automl-dnn-nlp**
 - Initial PR for new dnn-nlp package
 - **azureml-contrib-automl-dnn-vision**
 - introduce a new AutoML task type of "image-instance-segmentation".
 - **azureml-contrib-automl-pipeline-steps**
 - This new package is responsible for creating steps required for many models train/inference scenario. - It also moves the train/inference code into azureml.train.automl.runtime package so any future fixes would be automatically available through curated environment releases.
 - **azureml-contrib-dataset**
 - introduce a new AutoML task type of "image-instance-segmentation".
 - **azureml-core**
 - Added the initial core implementation of test set ingestion feature.
 - Fixing the xref warnings for documentation in azureml-core package
 - Doc string fixes for Command support feature in SDK
 - Adding command property to RunConfiguration. The feature enables users to run an actual command or executables on the compute through AzureML SDK.
 - Users can delete an empty experiment given the ID of that experiment.
 - **azureml-dataprep**
 - Added dataset support for Spark built with Scala 2.12. This adds to the existing 2.11 support.
 - **azureml-mlflow**
 - AzureML-MLflow adds safe guards in remote scripts to avoid early termination of submitted runs.
 - **azureml-pipeline-core**
 - Fixed a bug in setting a default pipeline for pipeline endpoint created via UI
 - **azureml-pipeline-steps**
 - Added experimental support for test data to AutoMLStep.
 - **azureml-tensorboard**
 - Fixing the xref warnings for documentation in azureml-core package
 - **azureml-train-automl-client**
 - Added experimental support for test data to AutoMLStep.
 - Added the initial core implementation of test set ingestion feature.
 - introduce a new AutoML task type of "image-instance-segmentation".

- **azureml-train-automl-runtime**
 - Added the initial core implementation of test set ingestion feature.
 - Fix the computation of the raw explanations for the best AutoML model if the AutoML models are trained using validation_size setting.
 - Moved references to sklearn.externals.joblib to depend directly on joblib.
- **azureml-train-core**
 - HyperDriveRun.get_children_sorted_by_primary_metric() should complete faster now
 - Improved error handling in HyperDrive SDK.
 - Deprecated all estimator classes in favor of using ScriptRunConfig to configure experiment runs. Deprecated classes include:
 - MMLBase
 - Estimator
 - PyTorch
 - TensorFlow
 - Chainer
 - SKLearn
 - Deprecated the use of Nccl and Gloo as valid input types for Estimator classes in favor of using PyTorchConfiguration with ScriptRunConfig.
 - Deprecated the use of Mpi as a valid input type for Estimator classes in favor of using MpiConfiguration with ScriptRunConfig.
 - Adding command property to runconfiguration. The feature enables users to run an actual command or executables on the compute through AzureML SDK.
 - Deprecated all estimator classes in favor of using ScriptRunConfig to configure experiment runs. Deprecated classes include: + MMLBaseEstimator + Estimator + PyTorch + TensorFlow + Chainer + SKLearn
 - Deprecated the use of Nccl and Gloo as a valid type of input for Estimator classes in favor of using PyTorchConfiguration with ScriptRunConfig.
 - Deprecated the use of Mpi as a valid type of input for Estimator classes in favor of using MpiConfiguration with ScriptRunConfig.

2020-11-30

Azure Machine Learning studio Notebooks Experience (November Update)

- **New features**

- Native Terminal. Users will now have access to an integrated terminal as well as Git operation via the [integrated terminal](#).
- Duplicate Folder
- Costing for Compute Drop Down
- Offline Compute Pylance

- **Bug fixes and improvements**

- Improved page load times
- Improved performance
- Improved speed and kernel reliability
- Large File Upload. You can now upload file >95mb

2020-11-09

Azure Machine Learning SDK for Python v1.18.0

- Bug fixes and improvements
 - **azureml-automl-core**
 - Improved handling of short time series by allowing padding them with gaussian noise.
 - **azureml-automl-runtime**
 - Throw ConfigException if a DateTime column has OutOfBoundsDatetime value
 - Improved handling of short time series by allowing padding them with gaussian noise.
 - Making sure that each text column can leverage char-gram transform with the n-gram range based on the length of the strings in that text column
 - Providing raw feature explanations for best mode for AutoML experiments running on user's local compute
 - **azureml-core**
 - Pin the package: pyjwt to avoid pulling in breaking versions in upcoming releases.
 - Creating an experiment will return the active or last archived experiment with that same given name if such experiment exists or a new experiment.
 - Calling `get_experiment` by name will return the active or last archived experiment with that given name.
 - Users cannot rename an experiment while reactivating it.
 - Improved error message to include potential fixes when a dataset is incorrectly passed to an experiment (e.g. `ScriptRunConfig`).
 - Improved documentation for `OutputDatasetConfig.register_on_complete` to include the behavior of what will happen when the name already exists.
 - Specifying dataset input and output names that have the potential to collide with common environment variables will now result in a warning
 - Repurposed `grant_workspace_access` parameter when registering datastores. Set it to `True` to access data behind virtual network from Machine Learning studio. [Learn more](#)
 - Linked service API is refined. Instead of providing resource ID, we have 3 separate parameters `sub_id`, `rg`, and `name` defined in configuration.
 - In order to enable customers to self-resolve token corruption issues, enable workspace token synchronization to be a public method.
 - This change allows an empty string to be used as a value for a `script_param`
 - **azureml-train-automl-client**
 - Improved handling of short time series by allowing padding them with gaussian noise.
 - **azureml-train-automl-runtime**
 - Throw ConfigException if a DateTime column has OutOfBoundsDatetime value
 - Added support for providing raw feature explanations for best model for AutoML experiments running on user's local compute
 - Improved handling of short time series by allowing padding them with gaussian noise.
 - **azureml-train-core**
 - This change allows an empty string to be used as a value for a `script_param`
 - **azureml-train-restclients-hyperdrive**
 - README has been changed to offer more context
 - **azureml-widgets**
 - Add string support to charts/parallel-coordinates library for widget.

2020-11-05

Data Labeling for image instance segmentation (polygon annotation) (preview)

The image instance segmentation (polygon annotations) project type in data labeling is available now, so users can draw and annotate with polygons around the contour of the objects in the images. Users will be able assign a class and a polygon to each object which of interest within an image.

Learn more about [image instance segmentation labeling](#).

2020-10-26

Azure Machine Learning SDK for Python v1.17.0

- **new examples**
 - A new community-driven repository of examples is available at <https://github.com/Azure/azureml-examples>
- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Fixed an issue where get_output may raise an XGBoostError.
 - **azureml-automl-runtime**
 - Time/calendar based features created by AutoML will now have the prefix.
 - Fixed an IndexError occurring during training of StackEnsemble for classification datasets with large number of classes and subsampling enabled.
 - Fixed an issue where VotingRegressor predictions may be inaccurate after refitting the model.
 - **azureml-core**
 - Additional detail added about relationship between AKS deployment configuration and Azure Kubernetes Service concepts.
 - Environment client labels support. User can label Environments and reference them by label.
 - **azureml-databricks**
 - Better error message when using currently unsupported Spark with Scala 2.12.
 - **azureml-explain-model**
 - The azureml-explain-model package is officially deprecated
 - **azureml-milflow**
 - Resolved a bug in milflow.projects.run against azureml backend where Finalizing state was not handled properly.
 - **azureml-pipeline-core**
 - Add support to create, list and get pipeline schedule based one pipeline endpoint.
 - Improved the documentation of PipelineData.as_dataset with an invalid usage example - Using PipelineData.as_dataset improperly will now result in a ValueError being thrown
 - Changed the HyperDriveStep pipelines notebook to register the best model within a PipelineStep directly after the HyperDriveStep run.
 - **azureml-pipeline-steps**
 - Changed the HyperDriveStep pipelines notebook to register the best model within a PipelineStep directly after the HyperDriveStep run.
 - **azureml-train-automl-client**
 - Fixed an issue where get_output may raise an XGBoostError.

Azure Machine Learning studio Notebooks Experience (October Update)

- **New features**
 - [Full virtual network support](#)
 - [Focus Mode](#)
 - Save notebooks Ctrl-S

- Line Numbers
- Bug fixes and improvements
 - Improvement in speed and kernel reliability
 - Jupyter Widget UI updates

2020-10-12

Azure Machine Learning SDK for Python v1.16.0

- Bug fixes and improvements
 - **azure-cli-ml**
 - AKSWebservice and AKSEndpoints now support pod-level CPU and Memory resource limits.
These optional limits can be used by setting `--cpu-cores-limit` and `--memory-gb-limit` flags in applicable CLI calls
 - **azureml-core**
 - Pin major versions of direct dependencies of azureml-core
 - AKSWebservice and AKSEndpoints now support pod-level CPU and Memory resource limits.
More information on [Kubernetes Resources and Limits](#)
 - Updated `run.log_table` to allow individual rows to be logged.
 - Added static method `Run.get(workspace, run_id)` to retrieve a run only using a workspace
 - Added instance method `Workspace.get_run(run_id)` to retrieve a run within the workspace
 - Introducing command property in run configuration which will enable users to submit command instead of script & arguments.
 - **azureml-interpret**
 - fixed explanation client `is_raw` flag behavior in `azureml-interpret`
 - **azureml-sdk**
 - `azureml-sdk` officially support Python 3.8.
 - **azureml-train-core**
 - Adding TensorFlow 2.3 curated environment
 - Introducing command property in run configuration which will enable users to submit command instead of script & arguments.
 - **azureml-widgets**
 - Redesigned interface for script run widget.

2020-09-28

Azure Machine Learning SDK for Python v1.15.0

- Bug fixes and improvements
 - **azureml-contrib-interpret**
 - LIME explainer moved from `azureml-contrib-interpret` to `interpret-community` package and image explainer removed from `azureml-contrib-interpret` package
 - visualization dashboard removed from `azureml-contrib-interpret` package, explanation client moved to `azureml-interpret` package and deprecated in `azureml-contrib-interpret` package and notebooks updated to reflect improved API
 - fix pypi package descriptions for `azureml-interpret`, `azureml-explain-model`, `azureml-contrib-interpret` and `azureml-tensorboard`
 - **azureml-contrib-notebook**
 - Pin `nbcov` dependency to < 6 so that `papermill` 1.x continues to work.
 - **azureml-core**

- Added parameters to the TensorflowConfiguration and Mp IConfiguration constructor to enable a more streamlined initialization of the class attributes without requiring the user to set each individual attribute. Added a PyTorchConfiguration class for configuring distributed PyTorch jobs in ScriptRunConfig.
- Pin the version of azure-mgmt-resource to fix the authentication error.
- Support Triton No Code Deploy
- outputs directories specified in Run.start_logging() will now be tracked when using run in interactive scenarios. The tracked files will be visible on ML Studio upon calling Run.complete()
- File encoding can be now specified during dataset creation with `Dataset.Tabular.from_delimited_files` and `Dataset.Tabular.from_json_lines_files` by passing the `encoding` argument. The supported encodings are 'utf8', 'iso88591', 'latin1', 'ascii', 'utf16', 'utf32', 'utf8bom' and 'windows1252'.
- Bug fix when environment object is not passed to ScriptRunConfig constructor.
- Updated Run.cancel() to allow cancel of a local run from another machine.
- **azureml-dataprep**
 - Fixed dataset mount timeout issues.
- **azureml-explain-model**
 - fix pypi package descriptions for azureml-interpret, azureml-explain-model, azureml-contrib-interpret and azureml-tensorboard
- **azureml-interpret**
 - visualization dashboard removed from azureml-contrib-interpret package, explanation client moved to azureml-interpret package and deprecated in azureml-contrib-interpret package and notebooks updated to reflect improved API
 - azureml-interpret package updated to depend on interpret-community 0.15.0
 - fix pypi package descriptions for azureml-interpret, azureml-explain-model, azureml-contrib-interpret and azureml-tensorboard
- **azureml-pipeline-core**
 - Fixed pipeline issue with `outputFileDatasetConfig` where the system may stop responding when `register_on_complete` is called with the `name` parameter set to a pre-existing dataset name.
- **azureml-pipeline-steps**
 - Removed stale databricks notebooks.
- **azureml-tensorboard**
 - fix pypi package descriptions for azureml-interpret, azureml-explain-model, azureml-contrib-interpret and azureml-tensorboard
- **azureml-train-automl-runtime**
 - visualization dashboard removed from azureml-contrib-interpret package, explanation client moved to azureml-interpret package and deprecated in azureml-contrib-interpret package and notebooks updated to reflect improved API
- **azureml-widgets**
 - visualization dashboard removed from azureml-contrib-interpret package, explanation client moved to azureml-interpret package and deprecated in azureml-contrib-interpret package and notebooks updated to reflect improved API

2020-09-21

Azure Machine Learning SDK for Python v1.14.0

- Bug fixes and improvements
 - `azure-cli-ml`

- Grid Profiling removed from the SDK and is not longer supported.
- **azureml-accel-models**
 - `azureml-accel-models` package now supports TensorFlow 2.x
- **azureml-automl-core**
 - Added error handling in `get_output` for cases when local versions of pandas/sklearn don't match the ones used during training
- **azureml-automl-runtime**
 - Fixed a bug where AutoArima iterations would fail with a `PredictionException` and the message: "Silent failure occurred during prediction."
- **azureml-cli-common**
 - Grid Profiling removed from the SDK and is not longer supported.
- **azureml-contrib-server**
 - Update description of the package for pypi overview page.
- **azureml-core**
 - Grid Profiling removed from the SDK and is no longer supported.
 - Reduce number of error messages when workspace retrieval fails.
 - Don't show warning when fetching metadata fails
 - New Kusto Step and Kusto Compute Target.
 - Update document for sku parameter. Remove sku in workspace update functionality in CLI and SDK.
 - Update description of the package for pypi overview page.
 - Updated documentation for AzureML Environments.
 - Expose service managed resources settings for AML workspace in SDK.
- **azureml-datatransform**
 - Enable execute permission on files for Dataset mount.
- **azureml-mlflow**
 - Updated AzureML MLflow documentation and notebook samples
 - New support for MLflow projects with AzureML backend
 - MLflow model registry support
 - Added Azure RBAC support for AzureML-MLflow operations
- **azureml-pipeline-core**
 - Improved the documentation of the `PipelineOutputFileDataset.parse_*` methods.
 - New Kusto Step and Kusto Compute Target.
 - Provided Swaggerurl property for pipeline-endpoint entity via that user can see the schema definition for published pipeline endpoint.
- **azureml-pipeline-steps**
 - New Kusto Step and Kusto Compute Target.
- **azureml-telemetry**
 - Update description of the package for pypi overview page.
- **azureml-train**
 - Update description of the package for pypi overview page.
- **azureml-train-automl-client**

- Added error handling in get_output for cases when local versions of pandas/sklearn don't match the ones used during training
- **azureml-train-core**
 - Update description of the package for pypi overview page.

2020-08-31

Azure Machine Learning SDK for Python v1.13.0

- **Preview features**
 - **azureml-core** With the new output datasets capability, you can write back to cloud storage including Blob, ADLS Gen 1, ADLS Gen 2, and FileShare. You can configure where to output data, how to output data (via mount or upload), whether to register the output data for future reuse and sharing and pass intermediate data between pipeline steps seamlessly. This enables reproducibility, sharing, prevents duplication of data, and results in cost efficiency and productivity gains. [Learn how to use it](#)
- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Added validated_{platform}_requirements.txt file for pinning all pip dependencies for AutoML.
 - This release supports models greater than 4 Gb.
 - Upgraded AutoML dependencies: `scikit-learn` (now 0.22.1), `pandas` (now 0.25.1), `numpy` (now 1.18.2).
 - **azureml-automl-runtime**
 - Set horovod for text DNN to always use fp16 compression.
 - This release supports models greater than 4 Gb.
 - Fixed issue where AutoML fails with ImportError: cannot import name `RollingOriginValidator`.
 - Upgraded AutoML dependencies: `scikit-learn` (now 0.22.1), `pandas` (now 0.25.1), `numpy` (now 1.18.2).
 - **azureml-contrib-automl-dnn-forecasting**
 - Upgraded AutoML dependencies: `scikit-learn` (now 0.22.1), `pandas` (now 0.25.1), `numpy` (now 1.18.2).
 - **azureml-contrib-fairness**
 - Provide a short description for azureml-contrib-fairness.
 - **azureml-contrib-pipeline-steps**
 - Added message indicating this package is deprecated and user should use azureml-pipeline-steps instead.
 - **azureml-core**
 - Added list key command for workspace.
 - Add tags parameter in Workspace SDK and CLI.
 - Fixed the bug where submitting a child run with Dataset will fail due to `TypeError: can't pickle _thread.RLock objects`.
 - Adding page_count default/documentation for Model list().
 - Modify CLI&SDK to take adbworkspace parameter and Add workspace adb lin/unlink runner.
 - Fix bug in Dataset.update that caused newest Dataset version to be updated not the version of the Dataset update was called on.
 - Fix bug in Dataset.get_by_name that would show the tags for the newest Dataset version even when a specific older version was retrieved.
 - **azureml-interpret**
 - Added probability outputs to shap scoring explainers in azureml-interpret based on

`shap_values_output` parameter from original explainer.

- **azureml-pipeline-core**
 - Improved `PipelineOutputAbstractDataset.register`'s documentation.
- **azureml-train-automl-client**
 - Upgraded AutoML dependencies: `scikit-learn` (now 0.22.1), `pandas` (now 0.25.1), `numpy` (now 1.18.2).
- **azureml-train-automl-runtime**
 - Upgraded AutoML dependencies: `scikit-learn` (now 0.22.1), `pandas` (now 0.25.1), `numpy` (now 1.18.2).
- **azureml-train-core**
 - Users must now provide a valid `hyperparameter_sampling` arg when creating a `HyperDriveConfig`. In addition, the documentation for `HyperDriveRunConfig` has been edited to inform users of the deprecation of `HyperDriveRunConfig`.
 - Reverting PyTorch Default Version to 1.4.
 - Adding PyTorch 1.6 & TensorFlow 2.2 images and curated environment.

Azure Machine Learning studio Notebooks Experience (August Update)

- **New features**
 - New Getting started landing Page
- **Preview features**
 - Gather feature in Notebooks. With the [Gather](#) feature, users can now easily clean up notebooks with, Gather uses an automated dependency analysis of your notebook, ensuring the essential code is kept, but removing any irrelevant pieces.
- **Bug fixes and improvements**
 - Improvement in speed and reliability
 - Dark mode bugs fixed
 - Output Scroll Bugs fixed
 - Sample Search now searches all the content of all the files in the Azure Machine Learning sample notebooks repo
 - Multi-line R cells can now run
 - "I trust contents of this file" is now auto checked after first time
 - Improved Conflict resolution dialog, with new "Make a copy" option

2020-08-17

Azure Machine Learning SDK for Python v1.12.0

- **Bug fixes and improvements**
 - **azure-cli-ml**
 - Add `image_name` and `image_label` parameters to `Model.package()` to enable renaming the built package image.
 - **azureml-automl-core**
 - AutoML raises a new error code from `dataprep` when content is modified while being read.
 - **azureml-automl-runtime**
 - Added alerts for the user when data contains missing values but featurization is turned off.
 - Fixed child run failures when data contains nan and featurization is turned off.
 - AutoML raises a new error code from `dataprep` when content is modified while being read.
 - Updated normalization for forecasting metrics to occur by grain.

- Improved calculation of forecast quantiles when lookback features are disabled.
- Fixed bool sparse matrix handling when computing explanations after AutoML.
- **azureml-core**
 - A new method `run.get_detailed_status()` now shows the detailed explanation of current run status. It is currently only showing explanation for `Queued` status.
 - Add `image_name` and `image_label` parameters to `Model.package()` to enable renaming the built package image.
 - New method `set_pip_requirements()` to set the entire pip section in `CondaDependencies` at once.
 - Enable registering credential-less ADLS Gen2 datastore.
 - Improved error message when trying to download or mount an incorrect dataset type.
 - Update time series dataset filter sample notebook with more examples of `partition_timestamp` that provides filter optimization.
 - Change the sdk and CLI to accept `subscriptionId`, `resourceGroup`, `workspaceName`, `peConnectionName` as parameters instead of `ArmResourceId` when deleting private endpoint connection.
 - Experimental Decorator shows class name for easier identification.
 - Descriptions for the Assets inside of Models are no longer automatically generated based on a Run.
- **azureml-datadrift**
 - Mark `create_from_model` API in `DataDriftDetector` as to be deprecated.
- **azureml-dataprep**
 - Improved error message when trying to download or mount an incorrect dataset type.
- **azureml-pipeline-core**
 - Fixed bug when deserializing pipeline graph that contains registered datasets.
- **azureml-pipeline-steps**
 - `RScriptStep` supports `RSection` from `azureml.core.environment`.
 - Removed the `passthru_automl_config` parameter from the `AutoMLStep` public API and converted it to an internal only parameter.
- **azureml-train-automl-client**
 - Removed local asynchronous, managed environment runs from AutoML. All local runs will run in the environment the run was launched from.
 - Fixed snapshot issues when submitting AutoML runs with no user-provided scripts.
 - Fixed child run failures when data contains nan and featurization is turned off.
- **azureml-train-automl-runtime**
 - AutoML raises a new error code from dataprep when content is modified while being read.
 - Fixed snapshot issues when submitting AutoML runs with no user-provided scripts.
 - Fixed child run failures when data contains nan and featurization is turned off.
- **azureml-train-core**
 - Added support for specifying pip options (for example `--extra-index-url`) in the pip requirements file passed to an `Estimator` through `pip_requirements_file` parameter.

2020-08-03

Azure Machine Learning SDK for Python v1.11.0

- **Bug fixes and improvements**
 - **azure-cli-ml**
 - Fix model framework and model framework not passed in run object in CLI model registration

- path
- Fix CLI amlcompute identity show command to show tenant ID and principal ID
- **azureml-train-automl-client**
 - Added `get_best_child()` to `AutoMLRun` for fetching the best child run for an AutoML Run without downloading the associated model.
 - Added `ModelProxy` object that allow predict or forecast to be run on a remote training environment without downloading the model locally.
 - Unhandled exceptions in AutoML now point to a known issues HTTP page, where more information about the errors can be found.
- **azureml-core**
 - Model names can be 255 characters long.
 - `Environment.get_image_details()` return object type changed. `DockerImageDetails` class replaced `dict`, image details are available from the new class properties. Changes are backward compatible.
 - Fix bug for `Environment.from_pip_requirements()` to preserve dependencies structure
 - Fixed a bug where `log_list` would fail if an int and double were included in the same list.
 - While enabling private link on an existing workspace, please note that if there are compute targets associated with the workspace, those targets will not work if they are not behind the same virtual network as the workspace private endpoint.
 - Made `as_named_input` optional when using datasets in experiments and added `as_mount` and `as_download` to `FileDataset`. The input name will automatically generated if `as_mount` or `as_download` is called.
- **azureml-automl-core**
 - Unhandled exceptions in AutoML now point to a known issues HTTP page, where more information about the errors can be found.
 - Added `get_best_child()` to `AutoMLRun` for fetching the best child run for an AutoML Run without downloading the associated model.
 - Added `ModelProxy` object that allows predict or forecast to be run on a remote training environment without downloading the model locally.
- **azureml-pipeline-steps**
 - Added `enable_default_model_output` and `enable_default_metrics_output` flags to `AutoMLStep`. These flags can be used to enable/disable the default outputs.

2020-07-20

Azure Machine Learning SDK for Python v1.10.0

- Bug fixes and improvements
 - **azureml-automl-core**
 - When using AutoML, if a path is passed into the `AutoMLConfig` object and it does not already exist, it will be automatically created.
 - Users can now specify a time series frequency for forecasting tasks by using the `freq` parameter.
 - **azureml-automl-runtime**
 - When using AutoML, if a path is passed into the `AutoMLConfig` object and it does not already exist, it will be automatically created.
 - Users can now specify a time series frequency for forecasting tasks by using the `freq` parameter.
 - AutoML Forecasting now supports rolling evaluation, which applies to the use case that the length of a test or validation set is longer than the input horizon, and known `y_pred` value is

used as forecasting context.

- **azureml-core**

- Warning messages will be printed if no files were downloaded from the datastore in a run.
- Added documentation for `skip_validation` to the `Datastore.register_azure_sql_database` method .
- Users are required to upgrade to sdk v1.10.0 or above to create an auto approved private endpoint. This includes the Notebook resource that is usable behind the VNet.
- Expose NotebookInfo in the response of get workspace.
- Changes to have calls to list compute targets and getting compute target succeed on a remote run. Sdk functions to get compute target and list workspace compute targets will now work in remote runs.
- Add deprecation messages to the class descriptions for `azureml.core.image` classes.
- Throw exception and clean up workspace and dependent resources if workspace private endpoint creation fails.
- Support workspace sku upgrade in workspace update method.

- **azureml-datadrift**

- Update matplotlib version from 3.0.2 to 3.2.1 to support Python 3.8.

- **azureml-dataprep**

- Added support of web url data sources with `Range` or `Head` request.
- Improved stability for file dataset mount and download.

- **azureml-train-automl-client**

- Fixed issues related to removal of `RequirementParseError` from `setuptools`.
- Use docker instead of conda for local runs submitted using "compute_target='local'"
- The iteration duration printed to the console has been corrected. Previously, the iteration duration was sometimes printed as run end time minus run creation time. It has been corrected to equal run end time minus run start time.
- When using AutoML, if a path is passed into the `AutoMLConfig` object and it does not already exist, it will be automatically created.
- Users can now specify a time series frequency for forecasting tasks by using the `freq` parameter.

- **azureml-train-automl-runtime**

- Improved console output when best model explanations fail.
- Renamed input parameter to "blocked_models" to remove a sensitive term.
 - Renamed input parameter to "allowed_models" to remove a sensitive term.
- Users can now specify a time series frequency for forecasting tasks by using the `freq` parameter.

2020-07-06

Azure Machine Learning SDK for Python v1.9.0

- **Bug fixes and improvements**

- **azureml-automl-core**

- Replaced `get_model_path()` with `AZUREML_MODEL_DIR` environment variable in AutoML autogenerated scoring script. Also added telemetry to track failures during `init()`.
 - Removed the ability to specify `enable_cache` as part of `AutoMLConfig`
 - Fixed a bug where runs may fail with service errors during specific forecasting runs
 - Improved error handling around specific models during `get_output`
 - Fixed call to `fitted_model.fit(X, y)` for classification with y transformer

- Enabled customized forward fill imputer for forecasting tasks
- A new ForecastingParameters class will be used instead of forecasting parameters in a dict format
- Improved target lag autodetection
- Added limited availability of multi-noded, multi-gpu distributed featurization with BERT
- **azureml-automl-runtime**
 - Prophet now does additive seasonality modeling instead of multiplicative.
 - Fixed the issue when short grains, having frequencies different from ones of the long grains will result in failed runs.
- **azureml-contrib-automl-dnn-vision**
 - Collect system/gpu stats and log averages for training and scoring
- **azureml-contrib-mir**
 - Added support for enable-app-insights flag in ManagedInferencing
- **azureml-core**
 - A validate parameter to these APIs by allowing validation to be skipped when the data source is not accessible from the current compute.
 - TabularDataset.time_before(end_time, include_boundary=True, validate=True)
 - TabularDataset.time_after(start_time, include_boundary=True, validate=True)
 - TabularDataset.time_recent(time_delta, include_boundary=True, validate=True)
 - TabularDataset.time_between(start_time, end_time, include_boundary=True, validate=True)
 - Added framework filtering support for model list, and added NCD AutoML sample in notebook back
 - For Datastore.register_azure_blob_container and Datastore.register_azure_file_share (only options that support SAS token), we have updated the doc strings for the `sas_token` field to include minimum permissions requirements for typical read and write scenarios.
 - Deprecating `_with_auth` param in `ws.get_mlflow_tracking_uri()`
- **azureml-mlflow**
 - Add support for deploying local file:// models with AzureML-MLflow
 - Deprecating `_with_auth` param in `ws.get_mlflow_tracking_uri()`
- **azureml-opendatasets**
 - Recently published Covid-19 tracking datasets are now available with the SDK
- **azureml-pipeline-core**
 - Log out warning when "azureml-defaults" is not included as part of pip-dependency
 - Improve Note rendering.
 - Added support for quoted line breaks when parsing delimited files to PipelineOutputFileDataset.
 - The PipelineDataset class is deprecated. For more information, see <https://aka.ms/dataset-deprecation>. Learn how to use dataset with pipeline, see <https://aka.ms/pipeline-with-dataset>.
- **azureml-pipeline-steps**
 - Doc updates to `azureml-pipeline-steps`.
 - Added support in ParallelRunConfig's `load_yaml()` for users to define Environments inline with the rest of the config or in a separate file
- **azureml-train-automl-client**
 - Removed the ability to specify `enable_cache` as part of AutoMLConfig
- **azureml-train-automl-runtime**
 - Added limited availability of multi-noded, multi-gpu distributed featurization with BERT.

- Added error handling for incompatible packages in ADB based automated machine learning runs.
- **azureml-widgets**
 - Doc updates to `azureml-widgets`.

2020-06-22

Azure Machine Learning SDK for Python v1.8.0

● Preview features

- **azureml-contrib-fairness** The `azureml-contrib-fairness` package provides integration between the open-source fairness assessment and unfairness mitigation package [Fairlearn](#) and Azure Machine Learning studio. In particular, the package enables model fairness evaluation dashboards to be uploaded as part of an AzureML Run and appear in Azure Machine Learning studio

● Bug fixes and improvements

- **azure-cli-ml**
 - Support getting logs of init container.
 - Added new CLI commands to manage ComputeInstance
- **azureml-automl-core**
 - Users are now able to enable stack ensemble iteration for Time series tasks with a warning that it could potentially overfit.
 - Added a new type of user exception that is raised if the cache store contents have been tampered with
- **azureml-automl-runtime**
 - Class Balancing Sweeping will no longer be enabled if user disables featurization.
- **azureml-contrib-notebook**
 - Doc improvements to `azureml-contrib-notebook` package.
- **azureml-contrib-pipeline-steps**
 - Doc improvements to `azureml-contrib--pipeline-steps` package.
- **azureml-core**
 - Add `set_connection`, `get_connection`, `list_connections`, `delete_connection` functions for customer to operate on workspace connection resource
 - Documentation updates to `azureml-coore/azureml.exceptions` package.
 - Documentation updates to `azureml-core` package.
 - Doc updates to `ComputeInstance` class.
 - Doc improvements to `azureml-core/azureml.core.compute` package.
 - Doc improvements for webservice-related classes in `azureml-core`.
 - Support user-selected datastore to store profiling data
 - Added `expand` and `page_count` property for model list API
 - Fixed bug where removing the `overwrite` property will cause the submitted run to fail with deserialization error.
 - Fixed inconsistent folder structure when downloading or mounting a `FileDataset` referencing to a single file.
 - Loading a dataset of parquet files `to_spark_dataframe` is now faster and supports all parquet and Spark SQL datatypes.
 - Support getting logs of init container.
 - AutoML runs are now marked as child run of Parallel Run Step.
- **azureml-datadrift**

- Doc improvements to azureml-contrib-notebook package.
- **azureml-dataprep**
 - Loading a dataset of parquet files to_spark_dataframe is now faster and supports all parquet and Spark SQL datatypes.
 - Better memory handling for OutOfMemory issue for to_pandas_dataframe.
- **azureml-interpret**
 - Upgraded azureml-interpret to use interpret-community version 0.12.*
- **azureml-mlflow**
 - Doc improvements to azureml-mlflow.
 - Adds support for AML model registry with MLFlow.
- **azureml-opendatasets**
 - Added support for Python 3.8
- **azureml-pipeline-core**
 - Updated `PipelineDataset`'s documentation to make it clear it is an internal class.
 - ParallelRunStep updates to accept multiple values for one argument, for example: "--group_column_names", "Col1", "Col2", "Col3"
 - Removed the passthru_automl_config requirement for intermediate data usage with AutoMLStep in Pipelines.
- **azureml-pipeline-steps**
 - Doc improvements to azureml-pipeline-steps package.
 - Removed the passthru_automl_config requirement for intermediate data usage with AutoMLStep in Pipelines.
- **azureml-telemetry**
 - Doc improvements to azureml-telemetry.
- **azureml-train-automl-client**
 - Fixed a bug where `experiment.submit()` called twice on an `AutoMLConfig` object resulted in different behavior.
 - Users are now able to enable stack ensemble iteration for Time series tasks with a warning that it could potentially overfit.
 - Changed AutoML run behavior to raise UserErrorException if service throws user error
 - Fixes a bug that caused azureml_automl.log to not get generated or be missing logs when performing an AutoML experiment on a remote compute target.
 - For Classification data sets with imbalanced classes, we will apply Weight Balancing, if the feature sweeper determines that for subsampled data, Weight Balancing improves the performance of the classification task by a certain threshold.
 - AutoML runs are now marked as child run of Parallel Run Step.
- **azureml-train-automl-runtime**
 - Changed AutoML run behavior to raise UserErrorException if service throws user error
 - AutoML runs are now marked as child run of Parallel Run Step.

2020-06-08

Azure Machine Learning SDK for Python v1.7.0

- **Bug fixes and improvements**
 - **azure-cli-ml**
 - Completed the removal of model profiling from mir contrib by cleaning up CLI commands and package dependencies, Model profiling is available in core.
 - Upgrades the min Azure CLI version to 2.3.0

- **azureml-automl-core**
 - Better exception message on featurization step `fit_transform()` due to custom transformer parameters.
 - Add support for multiple languages for deep learning transformer models such as BERT in automated ML.
 - Remove deprecated `lag_length` parameter from documentation.
 - The forecasting parameters documentation was improved. The `lag_length` parameter was deprecated.
- **azureml-automl-runtime**
 - Fixed the error raised when one of categorical columns is empty in forecast/test time.
 - Fix the run failures happening when the lookback features are enabled and the data contain short grains.
 - Fixed the issue with duplicated time index error message when lags or rolling windows were set to 'auto'.
 - Fixed the issue with Prophet and Arima models on data sets, containing the lookback features.
 - Added support of dates before 1677-09-21 or after 2262-04-11 in columns other than date time in the forecasting tasks. Improved error messages.
 - The forecasting parameters documentation was improved. The `lag_length` parameter was deprecated.
 - Better exception message on featurization step `fit_transform()` due to custom transformer parameters.
 - Add support for multiple languages for deep learning transformer models such as BERT in automated ML.
 - Cache operations that result in some OSErrors will raise user error.
 - Added checks to ensure training and validation data have the same number and set of columns
 - Fixed issue with the autogenerated AutoML scoring script when the data contains quotation marks
 - Enabling explanations for AutoML Prophet and ensembled models that contain Prophet model.
 - A recent customer issue revealed a live-site bug wherein we log messages along Class-Balancing-Sweeping even when the Class Balancing logic isn't properly enabled. Removing those logs/messages with this PR.
- **azureml-cli-common**
 - Completed the removal of model profiling from mir contrib by cleaning up CLI commands and package dependencies, Model profiling is available in core.
- **azureml-contrib-reinforcementlearning**
 - Load testing tool
- **azureml-core**
 - Documentation changes on `Script_run_config.py`
 - Fixes a bug with printing the output of `run submit-pipeline` CLI
 - Documentation improvements to `azureml-core/azureml.data`
 - Fixes issue retrieving storage account using `hdfs getconf` command
 - Improved `register_azure_blob_container` and `register_azure_file_share` documentation
- **azureml-datadrift**
 - Improved implementation for disabling and enabling dataset drift monitors
- **azureml-interpret**
 - In explanation client, remove NaNs or Infs prior to json serialization on upload from artifacts
 - Update to latest version of `interpret-community` to improve out of memory errors for global explanations with many features and classes

- Add true_ys optional parameter to explanation upload to enable additional features in the studio UI
- Improve download_model_explanations() and list_model_explanations() performance
- Small tweaks to notebooks, to aid with debugging
- **azureml-opendatasets**
 - azureml-opendatasets needs azureml-databricks version 1.4.0 or higher. Added warning if lower version is detected
- **azureml-pipeline-core**
 - This change allows user to provide an optional runconfig to the moduleVersion when calling module.Publish_python_script.
 - Enable node account can be a pipeline parameter in ParallelRunStep in azureml.pipeline.steps
- **azureml-pipeline-steps**
 - This change allows user to provide an optional runconfig to the moduleVersion when calling module.Publish_python_script.
- **azureml-train-automl-client**
 - Add support for multiple languages for deep learning transformer models such as BERT in automated ML.
 - Remove deprecated lag_length parameter from documentation.
 - The forecasting parameters documentation was improved. The lag_length parameter was deprecated.
- **azureml-train-automl-runtime**
 - Enabling explanations for AutoML Prophet and ensembled models that contain Prophet model.
 - Documentation updates to azureml-train-automl-* packages.
- **azureml-train-core**
 - Supporting TensorFlow version 2.1 in the PyTorch Estimator
 - Improvements to azureml-train-core package.

2020-05-26

Azure Machine Learning SDK for Python v1.6.0

- New features

- **azureml-automl-runtime**
 - AutoML Forecasting now supports customers forecast beyond the pre-specified max-horizon without retraining the model. When the forecast destination is farther into the future than the specified maximum horizon, the forecast() function will still make point predictions out to the later date using a recursive operation mode. For the illustration of the new feature, please see the "Forecasting farther than the maximum horizon" section of "forecasting-forecast-function" notebook in [folder](#)."
- **azureml-pipeline-steps**
 - ParallelRunStep is now released and is part of **azureml-pipeline-steps** package. Existing ParallelRunStep in **azureml-contrib-pipeline-steps** package is deprecated. Changes from public preview version:
 - Added `run_max_try` optional configurable parameter to control max call to run method for any given batch, default value is 3.
 - No PipelineParameters are autogenerated anymore. Following configurable values can be set as PipelineParameter explicitly.
 - `mini_batch_size`

- node_count
- process_count_per_node
- logging_level
- run_invocation_timeout
- run_max_try
- Default value for process_count_per_node is changed to 1. User should tune this value for better performance. Best practice is to set as the number of GPU or CPU node has.
- ParallelRunStep does not inject any packages, user needs to include `azureml-core` and `azureml-dataprep[pandas, fuse]` packages in environment definition. If custom docker image is used with `user_managed_dependencies` then user need to install conda on the image.

- **Breaking changes**

- **azureml-pipeline-steps**
 - Deprecated the use of `azureml.dprep.Dataflow` as a valid type of input for `AutoMLConfig`
- **azureml-train-automl-client**
 - Deprecated the use of `azureml.dprep.Dataflow` as a valid type of input for `AutoMLConfig`

- **Bug fixes and improvements**

- **azureml-automl-core**
 - Fixed the bug where a warning may be printed during `get_output` that asked user to downgrade client.
 - Updated Mac to rely on `cudatoolkit=9.0` as it is not available at version 10 yet.
 - Removing restrictions on prophet and xgboost models when trained on remote compute.
 - Improved logging in AutoML
 - The error handling for custom featurization in forecasting tasks was improved.
 - Added functionality to allow users to include lagged features to generate forecasts.
 - Updates to error message to correctly display user error.
 - Support for `cv_split_column_names` to be used with `training_data`
 - Update logging the exception message and traceback.
- **azureml-automl-runtime**
 - Enable guardrails for forecasting missing value imputations.
 - Improved logging in AutoML
 - Added fine grained error handling for data prep exceptions
 - Removing restrictions on prophet and xgboost models when trained on remote compute.
 - `azureml-train-automl-runtime` and `azureml-automl-runtime` have updated dependencies for `pytorch`, `scipy`, and `cudatoolkit`. We now support `pytorch==1.4.0`, `scipy>=1.0.0,<=1.3.1`, and `cudatoolkit==10.1.243`.
 - The error handling for custom featurization in forecasting tasks was improved.
 - The forecasting data set frequency detection mechanism was improved.
 - Fixed issue with Prophet model training on some data sets.
 - The auto detection of max horizon during the forecasting was improved.
 - Added functionality to allow users to include lagged features to generate forecasts.
 - Adds functionality in the `forecast` function to enable providing forecasts beyond the trained horizon without retraining the forecasting model.
 - Support for `cv_split_column_names` to be used with `training_data`
- **azureml-contrib-automl-dnn-forecasting**
 - Improved logging in AutoML

- **azureml-contrib-mir**
 - Added support for Windows services in ManagedInferencing
 - Remove old MIR workflows such as attach MIR compute, SingleModelMirWebservice class -
Clean out model profiling placed in contrib-mir package
- **azureml-contrib-pipeline-steps**
 - Minor fix for YAML support
 - ParallelRunStep is released to General Availability - azureml.contrib.pipeline.steps has a deprecation notice and is move to azureml.pipeline.steps
- **azureml-contrib-reinforcementlearning**
 - RL Load testing tool
 - RL estimator has smart defaults
- **azureml-core**
 - Remove old MIR workflows such as attach MIR compute, SingleModelMirWebservice class -
Clean out model profiling placed in contrib-mir package
 - Fixed the information provided to the user in case of profiling failure: included request ID and reworded the message to be more meaningful. Added new profiling workflow to profiling runners
 - Improved error text in case of Dataset execution failures.
 - Workspace private link CLI support added.
 - Added an optional parameter `invalid_lines` to `Dataset.Tabular.from_json_lines_files` that allows for specifying how to handle lines that contain invalid JSON.
 - We will be deprecating the run-based creation of compute in the next release. We recommend creating an actual Amlcompute cluster as a persistent compute target, and using the cluster name as the compute target in your run configuration. See example notebook here:
[aka.ms/amlcomputenb](#)
 - Improved error messages in case of Dataset execution failures.
- **azureml-datatransform**
 - Made warning to upgrade pyarrow version more explicit.
 - Improved error handling and message returned in case of failure to execute dataflow.
- **azureml-interpret**
 - Documentation updates to azureml-interpret package.
 - Fixed interpretability packages and notebooks to be compatible with latest sklearn update
- **azureml-opendatasets**
 - return None when there is no data returned.
 - Improve the performance of `to_pandas_dataframe`.
- **azureml-pipeline-core**
 - Quick fix for ParallelRunStep where loading from YAML was broken
 - ParallelRunStep is released to General Availability - azureml.contrib.pipeline.steps has a deprecation notice and is move to azureml.pipeline.steps - new features include: 1. Datasets as PipelineParameter 2. New parameter `run_max_retry` 3. Configurable `append_row` output file name
- **azureml-pipeline-steps**
 - Deprecated `azureml.dprep.Dataflow` as a valid type for input data.
 - Quick fix for ParallelRunStep where loading from YAML was broken
 - ParallelRunStep is released to General Availability - azureml.contrib.pipeline.steps has a deprecation notice and is move to azureml.pipeline.steps - new features include:
 - Datasets as PipelineParameter
 - New parameter `run_max_retry`

- Configurable append_row output file name
- **azureml-telemetry**
 - Update logging the exception message and traceback.
- **azureml-train-automl-client**
 - Improved logging in AutoML
 - Updates to error message to correctly display user error.
 - Support for cv_split_column_names to be used with training_data
 - Deprecated azureml.dprep.Dataflow as a valid type for input data.
 - Updated Mac to rely on cudatoolkit=9.0 as it is not available at version 10 yet.
 - Removing restrictions on prophet and xgboost models when trained on remote compute.
 - `azureml-train-automl-runtime` and `azureml-automl-runtime` have updated dependencies for `pytorch`, `scipy`, and `cudatoolkit`. We now support `pytorch==1.4.0`, `scipy>=1.0.0,<=1.3.1`, and `cudatoolkit==10.1.243`.
 - Added functionality to allow users to include lagged features to generate forecasts.
- **azureml-train-automl-runtime**
 - Improved logging in AutoML
 - Added fine grained error handling for data prep exceptions
 - Removing restrictions on prophet and xgboost models when trained on remote compute.
 - `azureml-train-automl-runtime` and `azureml-automl-runtime` have updated dependencies for `pytorch`, `scipy`, and `cudatoolkit`. We now support `pytorch==1.4.0`, `scipy>=1.0.0,<=1.3.1`, and `cudatoolkit==10.1.243`.
 - Updates to error message to correctly display user error.
 - Support for cv_split_column_names to be used with training_data
- **azureml-train-core**
 - Added a new set of HyperDrive specific exceptions. `azureml.train.hyperdrive` will now throw detailed exceptions.
- **azureml-widgets**
 - AzureML Widgets is not displaying in JupyterLab

2020-05-11

Azure Machine Learning SDK for Python v1.5.0

- **New features**
 - Preview features
 - **azureml-contrib-reinforcementlearning**
 - Azure Machine Learning is releasing preview support for reinforcement learning using the [Ray](#) framework. The `ReinforcementLearningEstimator` enables training of reinforcement learning agents across GPU and CPU compute targets in Azure Machine Learning.
- **Bug fixes and improvements**
 - **azure-cli-ml**
 - Fixes an accidentally left behind warning log in my previous PR. The log was used for debugging and accidentally was left behind.
 - Bug fix: inform clients about partial failure during profiling
 - **azureml-automl-core**
 - Speed up Prophet/AutoArima model in AutoML forecasting by enabling parallel fitting for the time series when data sets have multiple time series. In order to benefit from this new feature,

you are recommended to set "max_cores_per_iteration = -1" (that is, using all the available cpu cores) in AutoMLConfig.

- Fix KeyError on printing guardrails in console interface
- Fixed error message for experimentation_timeout_hours
- Deprecated TensorFlow models for AutoML.
- **azureml-automl-runtime**
 - Fixed error message for experimentation_timeout_hours
 - Fixed unclassified exception when trying to deserialize from cache store
 - Speed up Prophet/AutoArima model in AutoML forecasting by enabling parallel fitting for the time series when data sets have multiple time series.
 - Fixed the forecasting with enabled rolling window on the data sets where test/prediction set does not contain one of grains from the training set.
 - Improved handling of missing data
 - Fixed issue with prediction intervals during forecasting on data sets, containing time series, which are not aligned in time.
 - Added better validation of data shape for the forecasting tasks.
 - Improved the frequency detection.
 - Created better error message if the cross validation folds for forecasting tasks cannot be generated.
 - Fix console interface to print missing value guardrail correctly.
 - Enforcing datatype checks on cv_split_indices input in AutoMLConfig.
- **azureml-cli-common**
 - Bug fix: inform clients about partial failure during profiling
- **azureml-contrib-mir**
 - Adds a class azureml.contrib.mir.RevisionStatus which relays information about the currently deployed MIR revision and the most recent version specified by the user. This class is included in the MirWebservice object under 'deployment_status' attribute.
 - Enables update on Webservices of type MirWebservice and its child class SingleModelMirWebservice.
- **azureml-contrib-reinforcementlearning**
 - Added support for Ray 0.8.3
 - AmlWindowsCompute only supports Azure Files as mounted storage
 - Renamed health_check_timeout to health_check_timeout_seconds
 - Fixed some class/method descriptions.
- **azureml-core**
 - Enabled WASB -> Blob conversions in Azure Government and China clouds.
 - Fixes bug to allow Reader roles to use az ml run CLI commands to get run information
 - Removed unnecessary logging during Azure ML Remote Runs with input Datasets.
 - RCranPackage now supports "version" parameter for the CRAN package version.
 - Bug fix: inform clients about partial failure during profiling
 - Added European-style float handling for azureml-core.
 - Enabled workspace private link features in Azure ml sdk.
 - When creating a TabularDataset using `from_delimited_files`, you can specify whether empty values should be loaded as None or as empty string by setting the boolean argument `empty_as_string`.
 - Added European-style float handling for datasets.
 - Improved error messages on dataset mount failures.

- **azureml-datadrift**
 - Data Drift results query from the SDK had a bug that didn't differentiate the minimum, maximum, and mean feature metrics, resulting in duplicate values. We have fixed this bug by prefixing target or baseline to the metric names. Before: duplicate min, max, mean. After: target_min, target_max, target_mean, baseline_min, baseline_max, baseline_mean.
- **azureml-datatprep**
 - Improve handling of write restricted Python environments when ensuring .NET Dependencies required for data delivery.
 - Fixed Dataflow creation on file with leading empty records.
 - Added error handling options for `to_partition_iterator` similar to `to_pandas_dataframe`.
- **azureml-interpret**
 - Reduced explanation path length limits to reduce likelihood of going over Windows limit
 - Bugfix for sparse explanations created with the mimic explainer using a linear surrogate model.
- **azureml-opendatasets**
 - Fix issue of MNIST's columns are parsed as string, which should be int.
- **azureml-pipeline-core**
 - Allowing the option to regenerate_outputs when using a module that is embedded in a ModuleStep.
- **azureml-train-automl-client**
 - Deprecated TensorFlow models for AutoML.
 - Fix users allow listing unsupported algorithms in local mode
 - Doc fixes to AutoMLConfig.
 - Enforcing datatype checks on cv_split_indices input in AutoMLConfig.
 - Fixed issue with AutoML runs failing in show_output
- **azureml-train-automl-runtime**
 - Fixing a bug in Ensemble iterations that was preventing model download timeout from kicking in successfully.
- **azureml-train-core**
 - Fix typo in `azureml.train.dnn.Nccl` class.
 - Supporting PyTorch version 1.5 in the PyTorch Estimator
 - Fix the issue that framework image can't be fetched in Azure Government region when using training framework estimators

2020-05-04

New Notebook Experience

You can now create, edit, and share machine learning notebooks and files directly inside the studio web experience of Azure Machine Learning. You can use all the classes and methods available in [Azure Machine Learning Python SDK](#) from inside these notebooks. To get started, visit the [Run Jupyter Notebooks in your workspace](#) article.

New Features Introduced:

- Improved editor (Monaco editor) used by VS Code
- UI/UX improvements
- Cell Toolbar
- New Notebook Toolbar and Compute Controls
- Notebook Status Bar
- Inline Kernel Switching

- R Support
- Accessibility and Localization improvements
- Command Palette
- Additional Keyboard Shortcuts
- Auto save
- Improved performance and reliability

Access the following web-based authoring tools from the studio:

WEB-BASED TOOL	DESCRIPTION
Azure ML Studio Notebooks	First in-class authoring for notebook files and support all operation available in the Azure ML Python SDK.

2020-04-27

Azure Machine Learning SDK for Python v1.4.0

- **New features**
 - AmlCompute clusters now support setting up a managed identity on the cluster at the time of provisioning. Just specify whether you would like to use a system-assigned identity or a user-assigned identity, and pass an identityId for the latter. You can then set up permissions to access various resources like Storage or ACR in a way that the identity of the compute gets used to securely access the data, instead of a token-based approach that AmlCompute employs today. Check out our SDK reference for more information on the parameters.
- **Breaking changes**
 - AmlCompute clusters supported a Preview feature around run-based creation, that we are planning on deprecating in two weeks. You can continue to create persistent compute targets as always by using the Amlcompute class, but the specific approach of specifying the identifier "amlcompute" as the compute target in run config will not be supported in the near future.
- **Bug fixes and improvements**
 - **azureml-automl-runtime**
 - Enable support for unhashable type when calculating number of unique values in a column.
 - **azureml-core**
 - Improved stability when reading from Azure Blob Storage using a TabularDataset.
 - Improved documentation for the `grant_workspace_msi` parameter for `Datastore.register_azure_blob_store`.
 - Fixed bug with `datastore.upload` to support the `src_dir` argument ending with a `/` or `\`.
 - Added actionable error message when trying to upload to an Azure Blob Storage datastore that does not have an access key or SAS token.
 - **azureml-interpret**
 - Added upper bound to file size for the visualization data on uploaded explanations.
 - **azureml-train-automl-client**
 - Explicitly checking for `label_column_name` & `weight_column_name` parameters for `AutoMLConfig` to be of type string.
 - **azureml-contrib-pipeline-steps**
 - ParallelRunStep now supports dataset as pipeline parameter. User can construct pipeline with sample dataset and can change input dataset of the same type (file or tabular) for new pipeline run.

2020-04-13

Azure Machine Learning SDK for Python v1.3.0

- Bug fixes and improvements

- **azureml-automl-core**

- Added additional telemetry around post-training operations.
 - Speeds up automatic ARIMA training by using conditional sum of squares (CSS) training for series of length longer than 100. The length used is stored as the constant ARIMA_TRIGGER_CSS_TRAINING_LENGTH w/in the TimeSeriesInternal class at /src/azureml-automl-core/azureml/automl/core/shared/constants.py
 - The user logging of forecasting runs was improved, now more information on what phase is currently running will be shown in the log
 - Disallowed target_rolling_window_size to be set to values less than 2

- **azureml-automl-runtime**

- Improved the error message shown when duplicated timestamps are found.
 - Disallowed target_rolling_window_size to be set to values less than 2.
 - Fixed the lag imputation failure. The issue was caused by the insufficient number of observations needed to seasonally decompose a series. The "de-seasonalized" data is used to compute a partial autocorrelation function (PACF) to determine the lag length.
 - Enabled column purpose featurization customization for forecasting tasks by featurization config. Numerical and Categorical as column purpose for forecasting tasks is now supported.
 - Enabled drop column featurization customization for forecasting tasks by featurization config.
 - Enabled imputation customization for forecasting tasks by featurization config. Constant value imputation for target column and mean, median, most_frequent, and constant value imputation for training data are now supported.

- **azureml-contrib-pipeline-steps**

- Accept string compute names to be passed to ParallelRunConfig

- **azureml-core**

- Added Environment.clone(new_name) API to create a copy of Environment object
 - Environment.docker.base_dockerfile accepts filepath. If able to resolve a file, the content will be read into base_dockerfile environment property
 - Automatically reset mutually exclusive values for base_image and base_dockerfile when user manually sets a value in Environment.docker
 - Added user_managed flag in RSection that indicates whether the environment is managed by user or by AzureML.
 - Dataset: Fixed dataset download failure if data path containing unicode characters.
 - Dataset: Improved dataset mount caching mechanism to respect the minimum disk space requirement in Azure Machine Learning Compute, which avoids making the node unusable and causing the job to be canceled.
 - Dataset: We add an index for the time series column when you access a time series dataset as a pandas dataframe, which is used to speed up access to time series-based data access.

Previously, the index was given the same name as the timestamp column, confusing users about which is the actual timestamp column and which is the index. We now don't give any specific name to the index since it should not be used as a column.

- Dataset: Fixed dataset authentication issue in sovereign cloud.
 - Dataset: Fixed `Dataset.to_spark_dataframe` failure for datasets created from Azure PostgreSQL datastores.

- **azureml-interpret**

- Added global scores to visualization if local importance values are sparse

- Updated azureml-interpret to use interpret-community 0.9.*
- Fixed issue with downloading explanation that had sparse evaluation data
- Added support of sparse format of the explanation object in AutoML
- **azureml-pipeline-core**
 - Support ComputeInstance as compute target in pipelines
- **azureml-train-automl-client**
 - Added additional telemetry around post-training operations.
 - Fixed the regression in early stopping
 - Deprecated azureml.dprep.Dataflow as a valid type for input data.
 - Changing default AutoML experiment time out to six days.
- **azureml-train-automl-runtime**
 - Added additional telemetry around post-training operations.
 - added sparse AutoML end to end support
- **azureml-opendatasets**
 - Added additional telemetry for service monitor.
 - Enable front door for blob to increase stability

2020-03-23

Azure Machine Learning SDK for Python v1.2.0

- **Breaking changes**
 - Drop support for Python 2.7
- **Bug fixes and improvements**
 - **azure-cli-ml**
 - Adds "--subscription-id" to `az ml model/computetarget/service` commands in the CLI
 - Adding support for passing customer-managed key(CMK) vault_url, key_name and key_version for ACI deployment
 - **azureml-automl-core**
 - Enabled customized imputation with constant value for both X and y data forecasting tasks.
 - Fixed the issue in with showing error messages to user.
 - **azureml-automl-runtime**
 - Fixed the issue in with forecasting on the data sets, containing grains with only one row
 - Decreased the amount of memory required by the forecasting tasks.
 - Added better error messages if time column has incorrect format.
 - Enabled customized imputation with constant value for both X and y data forecasting tasks.
 - **azureml-core**
 - Added support for loading ServicePrincipal from environment variables:
AZUREML_SERVICE_PRINCIPAL_ID, AZUREML_SERVICE_PRINCIPAL_TENANT_ID, and
AZUREML_SERVICE_PRINCIPAL_PASSWORD
 - Introduced a new parameter `support_multi_line` to `Dataset.Tabular.from_delimited_files` : By default (`support_multi_line=False`), all line breaks, including those in quoted field values, will be interpreted as a record break. Reading data this way is faster and more optimized for parallel execution on multiple CPU cores. However, it may result in silently producing more records with misaligned field values. This should be set to `True` when the delimited files are known to contain quoted line breaks.
 - Added the ability to register ADLS Gen2 in the Azure Machine Learning CLI
 - Renamed parameter 'fine_grain_timestamp' to 'timestamp' and parameter

'coarse_grain_timestamp' to 'partition_timestamp' for the with_timestamp_columns() method in TabularDataset to better reflect the usage of the parameters.

- Increased max experiment name length to 255.
- **azureml-interpret**
 - Updated azureml-interpret to interpret-community 0.7.*
- **azureml-sdk**
 - Changing to dependencies with compatible version Tilde for the support of patching in pre-release and stable releases.

2020-03-11

Azure Machine Learning SDK for Python v1.1.5

- Feature deprecation

- Python 2.7
 - Last version to support Python 2.7

- Breaking changes

- Semantic Versioning 2.0.0
 - Starting with version 1.1 Azure ML Python SDK adopts [Semantic Versioning 2.0.0](#). All subsequent versions will follow new numbering scheme and semantic versioning contract.

- Bug fixes and improvements

- **azure-cli-ml**
 - Change the endpoint CLI command name from 'az ml endpoint aks' to 'az ml endpoint real time' for consistency.
 - update CLI installation instructions for stable and experimental branch CLI
 - Single instance profiling was fixed to produce a recommendation and was made available in core sdk.
 - **azureml-automl-core**
 - Enabled the Batch mode inference (taking multiple rows once) for AutoML ONNX models
 - Improved the detection of frequency on the data sets, lacking data or containing irregular data points
 - Added the ability to remove data points not complying with the dominant frequency.
 - Changed the input of the constructor to take a list of options to apply the imputation options for corresponding columns.
 - The error logging has been improved.
 - **azureml-automl-runtime**
 - Fixed the issue with the error thrown if the grain was not present in the training set appeared in the test set
 - Removed the y_query requirement during scoring on forecasting service
 - Fixed the issue with forecasting when the data set contains short grains with long time gaps.
 - Fixed the issue when the auto max horizon is turned on and the date column contains dates in form of strings. Proper conversion and error messages were added for when conversion to date is not possible
 - Using native NumPy and SciPy for serializing and deserializing intermediate data for FileCacheStore (used for local AutoML runs)
 - Fixed a bug where failed child runs could get stuck in Running state.
 - Increased speed of featurization.
 - Fixed the frequency check during scoring, now the forecasting tasks do not require strict

frequency equivalence between train and test set.

- Changed the input of the constructor to take a list of options to apply the imputation options for corresponding columns.
- Fixed errors related to lag type selection.
- Fixed the unclassified error raised on the data sets, having grains with the single row
- Fixed the issue with frequency detection slowness.
- Fixes a bug in AutoML exception handling that caused the real reason for training failure to be replaced by an AttributeError.
- **azureml-cli-common**
 - Single instance profiling was fixed to produce a recommendation and was made available in core sdk.
- **azureml-contrib-mir**
 - Adds functionality in the MirWebservice class to retrieve the Access Token
 - Use token auth for MirWebservice by default during MirWebservice.run() call - Only refresh if call fails
 - Mir webservice deployment now requires proper Skus [Standard_DS2_v2, Standard_F16, Standard_A2_v2] instead of [Ds2v2, A2v2, and F16] respectively.
- **azureml-contrib-pipeline-steps**
 - Optional parameter side_inputs added to ParallelRunStep. This parameter can be used to mount folder on the container. Currently supported types are DataReference and PipelineData.
 - Parameters passed in ParallelRunConfig can be overwritten by passing pipeline parameters now. New pipeline parameters supported aml_mini_batch_size, aml_error_threshold, aml_logging_level, aml_run_invocation_timeout (aml_node_count and aml_process_count_per_node are already part of earlier release).
- **azureml-core**
 - Deployed AzureML Webservices will now default to `INFO` logging. This can be controlled by setting the `AZUREML_LOG_LEVEL` environment variable in the deployed service.
 - Python sdk uses discovery service to use 'api' endpoint instead of 'pipelines'.
 - Swap to the new routes in all SDK calls.
 - Changed routing of calls to the ModelManagementService to a new unified structure.
 - Made workspace update method publicly available.
 - Added image_build_compute parameter in workspace update method to allow user updating the compute for image build.
 - Added deprecation messages to the old profiling workflow. Fixed profiling cpu and memory limits.
 - Added RSection as part of Environment to run R jobs.
 - Added validation to `Dataset.mount` to raise error when source of the dataset is not accessible or does not contain any data.
 - Added `--grant-workspace-msi-access` as an additional parameter for the Datastore CLI for registering Azure Blob Container that will allow you to register Blob Container that is behind a VNet.
 - Single instance profiling was fixed to produce a recommendation and was made available in core sdk.
 - Fixed the issue in `aks.py _deploy`.
 - Validates the integrity of models being uploaded to avoid silent storage failures.
 - User may now specify a value for the auth key when regenerating keys for webservices.
 - Fixed bug where uppercase letters cannot be used as dataset's input name.
- **azureml-defaults**

- `azureml-dataprep` will now be installed as part of `azureml-defaults`. It is no longer required to install data prep[fuse] manually on compute targets to mount datasets.
- **azureml-interpret**
 - Updated `azureml-interpret` to `interpret-community 0.6.*`
 - Updated `azureml-interpret` to depend on `interpret-community 0.5.0`
 - Added `azureml`-style exceptions to `azureml-interpret`
 - Fixed DeepScoringExplainer serialization for keras models
- **azureml-mlflow**
 - Add support for sovereign clouds to `azureml.mlflow`
- **azureml-pipeline-core**
 - Pipeline batch scoring notebook now uses `ParallelRunStep`
 - Fixed a bug where `PythonScriptStep` results could be incorrectly reused despite changing the arguments list
 - Added the ability to set columns' type when calling the `parse_*` methods on `PipelineOutputFileDataset`
- **azureml-pipeline-steps**
 - Moved the `AutoMLStep` to the `azureml-pipeline-steps` package. Deprecated the `AutoMLStep` within `azureml-train-automl-runtime`.
 - Added documentation example for dataset as `PythonScriptStep` input
- **azureml-tensorboard**
 - Updated `azureml-tensorboard` to support TensorFlow 2.0
 - Show correct port number when using a custom TensorBoard port on a Compute Instance
- **azureml-train-automl-client**
 - Fixed an issue where certain packages may be installed at incorrect versions on remote runs.
 - fixed `FeaturizationConfig` overriding issue that filters custom featurization config.
- **azureml-train-automl-runtime**
 - Fixed the issue with frequency detection in the remote runs
 - Moved the `AutoMLStep` in the `azureml-pipeline-steps` package. Deprecated the `AutoMLStep` within `azureml-train-automl-runtime`.
- **azureml-train-core**
 - Supporting PyTorch version 1.4 in the PyTorch Estimator

2020-03-02

Azure Machine Learning SDK for Python v1.1.2rc0 (Pre-release)

- Bug fixes and improvements
 - **azureml-automl-core**
 - Enabled the Batch mode inference (taking multiple rows once) for AutoML ONNX models
 - Improved the detection of frequency on the data sets, lacking data or containing irregular data points
 - Added the ability to remove data points not complying with the dominant frequency.
 - **azureml-automl-runtime**
 - Fixed the issue with the error thrown if the grain was not present in the training set appeared in the test set
 - Removed the `y_query` requirement during scoring on forecasting service
 - **azureml-contrib-mir**
 - Adds functionality in the `MirWebservice` class to retrieve the Access Token

- **azureml-core**
 - Deployed AzureML Webservices will now default to `INFO` logging. This can be controlled by setting the `AZUREML_LOG_LEVEL` environment variable in the deployed service.
 - Fix iterating on `Dataset.get_all` to return all datasets registered with the workspace.
 - Improve error message when invalid type is passed to `path` argument of dataset creation APIs.
 - Python sdk uses discovery service to use 'api' endpoint instead of 'pipelines'.
 - Swap to the new routes in all SDK calls
 - Changes routing of calls to the ModelManagementService to a new unified structure
 - Made workspace update method publicly available.
 - Added `image_build_compute` parameter in workspace update method to allow user updating the compute for image build
 - Added deprecation messages to the old profiling workflow. Fixed profiling cpu and memory limits
- **azureml-interpret**
 - update `azureml-interpret` to `interpret-community 0.6.*`
- **azureml-mlflow**
 - Add support for sovereign clouds to `azureml.mlflow`
- **azureml-pipeline-steps**
 - Moved the `AutoMLStep` to the `azureml-pipeline-steps package`. Deprecated the `AutoMLStep` within `azureml-train-automl-runtime`.
- **azureml-train-automl-client**
 - Fixed an issue where certain packages may be installed at incorrect versions on remote runs.
- **azureml-train-automl-runtime**
 - Fixed the issue with frequency detection in the remote runs
 - Moved the `AutoMLStep` to the `azureml-pipeline-steps package`. Deprecated the `AutoMLStep` within `azureml-train-automl-runtime`.
- **azureml-train-core**
 - Moved the `AutoMLStep` to the `azureml-pipeline-steps package`. Deprecated the `AutoMLStep` within `azureml-train-automl-runtime`.

2020-02-18

Azure Machine Learning SDK for Python v1.1.1rc0 (Pre-release)

- Bug fixes and improvements
 - **azure-cli-ml**
 - Single instance profiling was fixed to produce a recommendation and was made available in core sdk.
 - **azureml-automl-core**
 - The error logging has been improved.
 - **azureml-automl-runtime**
 - Fixed the issue with forecasting when the data set contains short grains with long time gaps.
 - Fixed the issue when the auto max horizon is turned on and the date column contains dates in form of strings. We added proper conversion and sensible error if conversion to date is not possible
 - Using native NumPy and SciPy for serializing and deserializing intermediate data for `FileCacheStore` (used for local AutoML runs)
 - Fixed a bug where failed child runs could get stuck in Running state.
 - **azureml-cli-common**

- Single instance profiling was fixed to produce a recommendation and was made available in core sdk.
- **azureml-core**
 - Added `--grant-workspace-msi-access` as an additional parameter for the Datastore CLI for registering Azure Blob Container that will allow you to register Blob Container that is behind a VNet
 - Single instance profiling was fixed to produce a recommendation and was made available in core sdk.
 - Fixed the issue in `aks.py _deploy`
 - Validates the integrity of models being uploaded to avoid silent storage failures.
- **azureml-interpret**
 - added azureml-style exceptions to `azureml-interpret`
 - fixed DeepScoringExplainer serialization for keras models
- **azureml-pipeline-core**
 - Pipeline batch scoring notebook now uses `ParallelRunStep`
- **azureml-pipeline-steps**
 - Moved the `AutoMLStep` in the `azureml-pipeline-steps` package. Deprecated the `AutoMLStep` within `azureml-train-automl-runtime`.
- **azureml-contrib-pipeline-steps**
 - Optional parameter `side_inputs` added to `ParallelRunStep`. This parameter can be used to mount folder on the container. Currently supported types are `DataReference` and `PipelineData`.
- **azureml-tensorboard**
 - Updated `azureml-tensorboard` to support TensorFlow 2.0
- **azureml-train-automl-client**
 - Fixed `FeaturizationConfig` overriding issue that filters custom featurization config.
- **azureml-train-automl-runtime**
 - Moved the `AutoMLStep` in the `azureml-pipeline-steps` package. Deprecated the `AutoMLStep` within `azureml-train-automl-runtime`.
- **azureml-train-core**
 - Supporting PyTorch version 1.4 in the PyTorch Estimator

2020-02-04

Azure Machine Learning SDK for Python v1.1.0rc0 (Pre-release)

- **Breaking changes**
 - **Semantic Versioning 2.0.0**
 - Starting with version 1.1 Azure ML Python SDK adopts [Semantic Versioning 2.0.0](#). All subsequent versions will follow new numbering scheme and semantic versioning contract.
- **Bug fixes and improvements**
 - **azureml-automl-runtime**
 - Increased speed of featurization.
 - Fixed the frequency check during scoring, now in the forecasting tasks we do not require strict frequency equivalence between train and test set.
 - **azureml-core**
 - User may now specify a value for the auth key when regenerating keys for webservices.
 - **azureml-interpret**
 - Updated `azureml-interpret` to depend on `interpret-community` 0.5.0

- **azureml-pipeline-core**
 - Fixed a bug where PythonScriptStep results could be incorrectly reused despite changing the arguments list
- **azureml-pipeline-steps**
 - Added documentation example for dataset as PythonScriptStep input
- **azureml-contrib-pipeline-steps**
 - Parameters passed in ParallelRunConfig can be overwritten by passing pipeline parameters now. New pipeline parameters supported aml_mini_batch_size, aml_error_threshold, aml_logging_level, aml_run_invocation_timeout (aml_node_count and aml_process_count_per_node are already part of earlier release).

2020-01-21

Azure Machine Learning SDK for Python v1.0.85

- **New features**
 - **azureml-core**
 - Get the current core usage and quota limitation for AmlCompute resources in a given workspace and subscription
 - **azureml-contrib-pipeline-steps**
 - Enable user to pass tabular dataset as intermediate result from previous step to parallelrunstep
- **Bug fixes and improvements**
 - **azureml-automl-runtime**
 - Removed the requirement of y_query column in the request to the deployed forecasting service.
 - The 'y_query' was removed from the Dominick's Orange Juice notebook service request section.
 - Fixed the bug preventing forecasting on the deployed models, operating on data sets with date time columns.
 - Added Matthews Correlation Coefficient as a classification metric, for both binary and multiclass classification.
 - **azureml-contrib-interpret**
 - Removed text explainers from azureml-contrib-interpret as text explanation has been moved to the interpret-text repo that will be released soon.
 - **azureml-core**
 - Dataset: usages for file dataset no longer depend on numpy and pandas to be installed in the Python env.
 - Changed LocalWebservice.wait_for_deployment() to check the status of the local Docker container before trying to ping its health endpoint, greatly reducing the amount of time it takes to report a failed deployment.
 - Fixed the initialization of an internal property used in LocalWebservice.reload() when the service object is created from an existing deployment using the LocalWebservice() constructor.
 - Edited error message for clarification.
 - Added a new method called get_access_token() to AksWebservice that will return AksServiceAccessToken object, which contains access token, refresh after timestamp, expiry on timestamp and token type.
 - Deprecated existing get_token() method in AksWebservice as the new method returns all of the information this method returns.

- Modified output of az ml service get-access-token command. Renamed token to accessToken and refreshBy to refreshAfter. Added expiryOn and tokenType properties.
- Fixed get_active_runs
- **azureml-explain-model**
 - updated shap to 0.33.0 and interpret-community to 0.4.*
- **azureml-interpret**
 - updated shap to 0.33.0 and interpret-community to 0.4.*
- **azureml-train-automl-runtime**
 - Added Matthews Correlation Coefficient as a classification metric, for both binary and multiclass classification.
 - Deprecate preprocess flag from code and replaced with featurization -featurization is on by default

2020-01-06

Azure Machine Learning SDK for Python v1.0.83

- New features

- Dataset: Add two options `on_error` and `out_of_range_datetime` for `to_pandas_dataframe` to fail when data has error values instead of filling them with `None`.
- Workspace: Added the `hbi_workspace` flag for workspaces with sensitive data that enables further encryption and disables advanced diagnostics on workspaces. We also added support for bringing your own keys for the associated Cosmos DB instance, by specifying the `cmk_keyvault` and `resource_cmk_uri` parameters when creating a workspace, which creates a Cosmos DB instance in your subscription while provisioning your workspace. To learn more, see the [Azure Cosmos DB section of data encryption article](#).

- Bug fixes and improvements

- **azureml-automl-runtime**
 - Fixed a regression that caused a TypeError to be raised when running AutoML on Python versions below 3.5.4.
- **azureml-core**
 - Fixed bug in `datastore.upload_files` where relative path that didn't start with `./` was not able to be used.
 - Added deprecation messages for all Image class code paths
 - Fixed Model Management URL construction for Azure China 21Vianet region.
 - Fixed issue where models using source_dir couldn't be packaged for Azure Functions.
 - Added an option to `Environment.build_local()` to push an image into AzureML workspace container registry
 - Updated the SDK to use new token library on Azure synapse in a back compatible manner.
- **azureml-interpret**
 - Fixed bug where None was returned when no explanations were available for download. Now raises an exception, matching behavior elsewhere.
- **azureml-pipeline-steps**
 - Disallowed passing `DatasetConsumptionConfig`s to `Estimator`'s `inputs` parameter when the `Estimator` will be used in an `EstimatorStep`.
- **azureml-sdk**
 - Added AutoML client to azureml-sdk package, enabling remote AutoML runs to be submitted without installing the full AutoML package.

- **azureml-train-automl-client**
 - Corrected alignment on console output for AutoML runs
 - Fixed a bug where incorrect version of pandas may be installed on remote amlcompute.

2019-12-23

Azure Machine Learning SDK for Python v1.0.81

- Bug fixes and improvements
 - **azureml-contrib-interpret**
 - defer shap dependency to interpret-community from azureml-interpret
 - **azureml-core**
 - Compute target can now be specified as a parameter to the corresponding deployment config objects. This is specifically the name of the compute target to deploy to, not the SDK object.
 - Added CreatedBy information to Model and Service objects. May be accessed through.created_by
 - Fixed ContainerImage.run(), which was not correctly setting up the Docker container's HTTP port.
 - Make `azureml-dataprep` optional for `az ml dataset register` CLI command
 - Fixed a bug where `TabularDataset.to_pandas_dataframe` would incorrectly fall back to an alternate reader and print out a warning.
 - **azureml-explain-model**
 - defer shap dependency to interpret-community from azureml-interpret
 - **azureml-pipeline-core**
 - Added new pipeline step `NotebookRunnerStep`, to run a local notebook as a step in pipeline.
 - Removed deprecated get_all functions for PublishedPipelines, Schedules, and PipelineEndpoints
 - **azureml-train-automl-client**
 - Started deprecation of data_script as an input to AutoML.

2019-12-09

Azure Machine Learning SDK for Python v1.0.79

- Bug fixes and improvements
 - **azureml-automl-core**
 - Removed featurizationConfig to be logged
 - Updated logging to log "auto"/"off"/"customized" only.
 - **azureml-automl-runtime**
 - Added support for pandas.Series and pandas.Categorical for detecting column data type. Previously only supported numpy.ndarray
 - Added related code changes to handle categorical dtype correctly.
 - The forecast function interface was improved: the y_pred parameter was made optional. -The docstrings were improved.
 - **azureml-contrib-dataset**
 - Fixed a bug where labeled datasets could not be mounted.
 - **azureml-core**
 - Bug fix for `Environment.from_existing_conda_environment(name, conda_environment_name)`. User can create an instance of Environment that is exact replica of the local environment
 - Changed time series-related Datasets methods to `include_boundary=True` by default.
 - **azureml-train-automl-client**

- Fixed issue where validation results are not printed when show output is set to false.

2019-11-25

Azure Machine Learning SDK for Python v1.0.76

● Breaking changes

- Azureml-Train-AutoML upgrade issues
 - Upgrading to `azureml-train-automl` >= 1.0.76 from `azureml-train-automl` < 1.0.76 can cause partial installations, causing some AutoML imports to fail. To resolve this, you can run the setup script found at https://github.com/Azure/MachineLearningNotebooks/blob/master/how-to-use-azureml/automated-machine-learning/automl_setup.cmd. Or if you are using pip directly you can:
 - "pip install --upgrade azureml-train-automl"
 - "pip install --ignore-installed azureml-train-automl-client"
 - or you can uninstall the old version before upgrading
 - "pip uninstall azureml-train-automl"
 - "pip install azureml-train-automl"

● Bug fixes and improvements

- **azureml-automl-runtime**
 - AutoML will now take into account both true and false classes when calculating averaged scalar metrics for binary classification tasks.
 - Moved Machine learning and training code in AzureML-AutoML-Core to a new package AzureML-AutoML-Runtime.
- **azureml-contrib-dataset**
 - When calling `to_pandas_dataframe` on a labeled dataset with the download option, you can now specify whether to overwrite existing files or not.
 - When calling `keep_columns` or `drop_columns` that results in a time series, label, or image column being dropped, the corresponding capabilities will be dropped for the dataset as well.
 - Fixed an issue with pytorch loader for the object detection task.
- **azureml-contrib-interpret**
 - Removed explanation dashboard widget from `azureml-contrib-interpret`, changed package to reference the new one in `interpret_community`
 - Updated version of `interpret-community` to 0.2.0
- **azureml-core**
 - Improve performance of `workspace.datasets`.
 - Added the ability to register Azure SQL Database Datastore using username and password authentication
 - Fix for loading RunConfigurations from relative paths.
 - When calling `keep_columns` or `drop_columns` that results in a time series column being dropped, the corresponding capabilities will be dropped for the dataset as well.
- **azureml-interpret**
 - updated version of `interpret-community` to 0.2.0
- **azureml-pipeline-steps**
 - Documented supported values for `runconfig_pipeline_params` for Azure machine learning pipeline steps.
- **azureml-pipeline-core**
 - Added CLI option to download output in json format for Pipeline commands.

- **azureml-train-automl**
 - Split AzureML-Train-AutoML into two packages, a client package AzureML-Train-AutoML-Client and an ML training package AzureML-Train-AutoML-Runtime
- **azureml-train-automl-client**
 - Added a thin client for submitting AutoML experiments without needing to install any machine learning dependencies locally.
 - Fixed logging of automatically detected lags, rolling window sizes and maximal horizons in the remote runs.
- **azureml-train-automl-runtime**
 - Added a new AutoML package to isolate machine learning and runtime components from the client.
- **azureml-contrib-train-rl**
 - Added reinforcement learning support in SDK.
 - Added AmlWindowsCompute support in RL SDK.

2019-11-11

Azure Machine Learning SDK for Python v1.0.74

- Preview features

- **azureml-contrib-dataset**
 - After importing azureml-contrib-dataset, you can call `Dataset.Labeled.from_json_lines` instead of `._Labeled` to create a labeled dataset.
 - When calling `to_pandas_dataframe` on a labeled dataset with the download option, you can now specify whether to overwrite existing files or not.
 - When calling `keep_columns` or `drop_columns` that results in a time series, label, or image column being dropped, the corresponding capabilities will be dropped for the dataset as well.
 - Fixed issues with PyTorch loader when calling `dataset.to_torchvision()`.

- Bug fixes and improvements

- **azure-cli-ml**
 - Added Model Profiling to the preview CLI.
 - Fixes breaking change in Azure Storage causing AzureML CLI to fail.
 - Added Load Balancer Type to MLC for AKS types
- **azureml-automl-core**
 - Fixed the issue with detection of maximal horizon on time series, having missing values and multiple grains.
 - Fixed the issue with failures during generation of cross validation splits.
 - Replace this section with a message in markdown format to appear in the release notes: - Improved handling of short grains in the forecasting data sets.
 - Fixed the issue with masking of some user information during logging. -Improved logging of the errors during forecasting runs.
 - Adding psutil as a conda dependency to the autogenerated yml deployment file.
- **azureml-contrib-mir**
 - Fixes breaking change in Azure Storage causing AzureML CLI to fail.
- **azureml-core**
 - Fixes a bug that caused models deployed on Azure Functions to produce 500s.
 - Fixed an issue where the amlignore file was not applied on snapshots.
 - Added a new API `amlcompute.get_active_runs` that returns a generator for running and queued

- runs on a given amlcompute.
- Added Load Balancer Type to MLC for AKS types.
 - Added append_prefix bool parameter to download_files in run.py and download_artifacts_from_prefix in artifacts_client. This flag is used to selectively flatten the origin filepath so only the file or folder name is added to the output_directory
 - Fix deserialization issue for `run_config.yml` with dataset usage.
 - When calling `keep_columns` or `drop_columns` that results in a time series column being dropped, the corresponding capabilities will be dropped for the dataset as well.
 - **azureml-interpret**
 - Updated interpret-community version to 0.1.0.3
 - **azureml-train-automl**
 - Fixed an issue where automl_step might not print validation issues.
 - Fixed register_model to succeed even if the model's environment is missing dependencies locally.
 - Fixed an issue where some remote runs were not docker enabled.
 - Add logging of the exception that is causing a local run to fail prematurely.
 - **azureml-train-core**
 - Consider resume_from runs in the calculation of automated hyperparameter tuning best child runs.
 - **azureml-pipeline-core**
 - Fixed parameter handling in pipeline argument construction.
 - Added pipeline description and step type yaml parameter.
 - New yaml format for Pipeline step and added deprecation warning for old format.

2019-11-04

Web experience

The collaborative workspace landing page at <https://ml.azure.com> has been enhanced and rebranded as the Azure Machine Learning studio.

From the studio, you can train, test, deploy, and manage Azure Machine Learning assets such as datasets, pipelines, models, endpoints, and more.

Access the following web-based authoring tools from the studio:

WEB-BASED TOOL	DESCRIPTION
Notebook VM(preview)	Fully managed cloud-based workstation
Automated machine learning (preview)	No code experience for automating machine learning model development
Designer	Drag-and-drop machine learning modeling tool formerly known as the visual interface

Azure Machine Learning designer enhancements

- Formerly known as the visual interface
- 11 new [modules](#) including recommenders, classifiers, and training utilities including feature engineering, cross validation, and data transformation.

R SDK

Data scientists and AI developers use the [Azure Machine Learning SDK for R](#) to build and run machine learning workflows with Azure Machine Learning.

The Azure Machine Learning SDK for R uses the `reticulate` package to bind to the Python SDK. By binding directly to Python, the SDK for R allows you access to core objects and methods implemented in the Python SDK from any R environment you choose.

Main capabilities of the SDK include:

- Manage cloud resources for monitoring, logging, and organizing your machine learning experiments.
- Train models using cloud resources, including GPU-accelerated model training.
- Deploy your models as webservices on Azure Container Instances (ACI) and Azure Kubernetes Service (AKS).

See the [package website](#) for complete documentation.

Azure Machine Learning integration with Event Grid

Azure Machine Learning is now a resource provider for Event Grid, you can configure machine learning events through the Azure portal or Azure CLI. Users can create events for run completion, model registration, model deployment, and data drift detected. These events can be routed to event handlers supported by Event Grid for consumption. See machine learning event [schema](#) and [tutorial](#) articles for more details.

2019-10-31

Azure Machine Learning SDK for Python v1.0.72

• New features

- Added dataset monitors through the [azureml-datadrift](#) package, allowing for monitoring time series datasets for data drift or other statistical changes over time. Alerts and events can be triggered if drift is detected or other conditions on the data are met. See [our documentation](#) for details.
- Announcing two new editions (also referred to as a SKU interchangeably) in Azure Machine Learning. With this release, you can now create either a Basic or Enterprise Azure Machine Learning workspace. All existing workspaces will be defaulted to the Basic edition, and you can go to the Azure portal or to the studio to upgrade the workspace anytime. You can create either a Basic or Enterprise workspace from the Azure portal. Read [our documentation](#) to learn more. From the SDK, the edition of your workspace can be determined using the "sku" property of your workspace object.
- We have also made enhancements to Azure Machine Learning Compute - you can now view metrics for your clusters (like total nodes, running nodes, total core quota) in Azure Monitor, besides viewing Diagnostic logs for debugging. In addition, you can also view currently running or queued runs on your cluster and details such as the IPs of the various nodes on your cluster. You can view these either in the portal or by using corresponding functions in the SDK or CLI.
- Preview features
 - We are releasing preview support for disk encryption of your local SSD in Azure Machine Learning Compute. Raise a technical support ticket to get your subscription allow listed to use this feature.
 - Public Preview of Azure Machine Learning Batch Inference. Azure Machine Learning Batch Inference targets large inference jobs that are not time-sensitive. Batch Inference provides cost-effective inference compute scaling, with unparalleled throughput for asynchronous applications. It is optimized for high-throughput, fire-and-forget inference over large collections of data.

- o **azureml-contrib-dataset**

- o Enabled functionalities for labeled dataset

```
import azureml.core
from azureml.core import Workspace, Datastore, Dataset
import azureml.contrib.dataset
from azureml.contrib.dataset import FileHandlingOption, LabeledDatasetTask

# create a labeled dataset by passing in your JSON lines file
dataset = Dataset._Labeled.from_json_lines(datastore.path('path/to/file.jsonl'),
LabeledDatasetTask.IMAGE_CLASSIFICATION)

# download or mount the files in the `image_url` column
dataset.download()
dataset.mount()

# get a pandas dataframe
from azureml.data.dataset_type_definitions import FileHandlingOption
dataset.to_pandas_dataframe(FileHandlingOption.DOWNLOAD)
dataset.to_pandas_dataframe(FileHandlingOption.MOUNT)

# get a Torchvision dataset
dataset.to_torchvision()
```

- Bug fixes and improvements

- o **azure-cli-ml**

- o CLI now supports model packaging.
- o Added dataset CLI. For more information: `az ml dataset --help`
- o Added support for deploying and packaging supported models (ONNX, scikit-learn, and TensorFlow) without an InferenceConfig instance.
- o Added overwrite flag for service deployment (ACI and AKS) in SDK and CLI. If provided, will overwrite the existing service if service with name already exists. If service doesn't exist, will create new service.
- o Models can be registered with two new frameworks, Onnx and TensorFlow. - Model registration accepts sample input data, sample output data and resource configuration for the model.

- o **azureml-automl-core**

- o Training an iteration would run in a child process only when runtime constraints are being set.
- o Added a guardrail for forecasting tasks, to check whether a specified max_horizon will cause a memory issue on the given machine or not. If it will, a guardrail message will be displayed.
- o Added support for complex frequencies like two years and one month. -Added comprehensible error message if frequency cannot be determined.
- o Add `azureml-defaults` to auto generated conda env to solve the model deployment failure
- o Allow intermediate data in Azure Machine Learning Pipeline to be converted to tabular dataset and used in `AutoMLStep`.
- o Implemented column purpose update for streaming.
- o Implemented transformer parameter update for Imputer and HashOneHotEncoder for streaming.
- o Added the current data size and the minimum required data size to the validation error messages.
- o Updated the minimum required data size for Cross-validation to guarantee a minimum of two samples in each validation fold.

- o **azureml-cli-common**

- o CLI now supports model packaging.

- Models can be registered with two new frameworks, Onnx and TensorFlow.
- Model registration accepts sample input data, sample output data and resource configuration for the model.
- **azureml-contrib-gbdt**
 - fixed the release channel for the notebook
 - Added a warning for non-AmlCompute compute target that we don't support
 - Added LightGMB Estimator to azureml-contrib-gbdt package
- **azureml-core**
 - CLI now supports model packaging.
 - Add deprecation warning for deprecated Dataset APIs. See Dataset API change notice at <https://aka.ms/tabular-dataset>.
 - Change `Dataset.get_by_id` to return registration name and version if the dataset is registered.
 - Fix a bug that ScriptRunConfig with dataset as argument cannot be used repeatedly to submit experiment run.
 - Datasets retrieved during a run will be tracked and can be seen in the run details page or by calling `run.get_details()` after the run is complete.
 - Allow intermediate data in Azure Machine Learning Pipeline to be converted to tabular dataset and used in `AutoMLStep`.
 - Added support for deploying and packaging supported models (ONNX, scikit-learn, and TensorFlow) without an InferenceConfig instance.
 - Added overwrite flag for service deployment (ACI and AKS) in SDK and CLI. If provided, will overwrite the existing service if service with name already exists. If service doesn't exist, will create new service.
 - Models can be registered with two new frameworks, Onnx and TensorFlow. Model registration accepts sample input data, sample output data and resource configuration for the model.
 - Added new datastore for Azure Database for MySQL. Added example for using Azure Database for MySQL in DataTransferStep in Azure Machine Learning Pipelines.
 - Added functionality to add and remove tags from experiments. Added functionality to remove tags from runs
 - Added overwrite flag for service deployment (ACI and AKS) in SDK and CLI. If provided, will overwrite the existing service if service with name already exists. If service doesn't exist, will create new service.
- **azureml-datadrift**
 - Moved from `azureml-contrib-datadrift` into `azureml-datadrift`
 - Added support for monitoring time series datasets for drift and other statistical measures
 - New methods `create_from_model()` and `create_from_dataset()` to the `DataDriftDetector` class. The `create()` method will be deprecated.
 - Adjustments to the visualizations in Python and UI in the Azure Machine Learning studio.
 - Support weekly and monthly monitor scheduling, in addition to daily for dataset monitors.
 - Support backfill of data monitor metrics to analyze historical data for dataset monitors.
 - Various bug fixes
- **azureml-pipeline-core**
 - `azureml-dataprep` is no longer needed to submit an Azure Machine Learning Pipeline run from the pipeline `yaml` file.
- **azureml-train-automl**
 - Add `azureml-defaults` to auto generated conda env to solve the model deployment failure
 - AutoML remote training now includes `azureml-defaults` to allow reuse of training env for inference.

- **azureml-train-core**
 - Added PyTorch 1.3 support in `PyTorch` estimator

2019-10-21

Visual interface (preview)

- The Azure Machine Learning visual interface (preview) has been overhauled to run on [Azure Machine Learning pipelines](#). Pipelines (previously known as experiments) authored in the visual interface are now fully integrated with the core Azure Machine Learning experience.
 - Unified management experience with SDK assets
 - Versioning and tracking for visual interface models, pipelines, and endpoints
 - Redesigned UI
 - Added batch inference deployment
 - Added Azure Kubernetes Service (AKS) support for inference compute targets
 - New Python-step pipeline authoring workflow
 - New [landing page](#) for visual authoring tools
- **New modules**
 - Apply math operation
 - Apply SQL transformation
 - Clip values
 - Summarize data
 - Import from SQL Database

2019-10-14

Azure Machine Learning SDK for Python v1.0.69

- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Limiting model explanations to best run rather than computing explanations for every run. Making this behavior change for local, remote and ADB.
 - Added support for on-demand model explanations for UI
 - Added psutil as a dependency of `automl` and included psutil as a conda dependency in `amlcompute`.
 - Fixed the issue with heuristic lags and rolling window sizes on the forecasting data sets some series of which can cause linear algebra errors
 - Added print out for the heuristically determined parameters in the forecasting runs.
 - **azureml-contrib-datadrift**
 - Added protection while creating output metrics if dataset level drift is not in the first section.
 - **azureml-contrib-interpret**
 - `azureml-contrib-explain-model` package has been renamed to `azureml-contrib-interpret`
 - **azureml-core**
 - Added API to unregister datasets. `dataset.unregister_all_versions()`
 - `azureml-contrib-explain-model` package has been renamed to `azureml-contrib-interpret`.
 - **azureml-core**
 - Added API to unregister datasets. `dataset.unregister_all_versions()`.
 - Added Dataset API to check data changed time. `dataset.data_changed_time`.
 - Being able to consume `FileDataset` and `TabularDataset` as inputs to `PythonScriptStep`,

`EstimatorStep`, and `HyperDriveStep` in Azure Machine Learning Pipeline

- Performance of `FileDataset.mount` has been improved for folders with a large number of files
- Being able to consume `FileDataset` and `TabularDataset` as inputs to `PythonScriptStep`, `EstimatorStep`, and `HyperDriveStep` in the Azure Machine Learning Pipeline.
- Performance of `FileDataset.mount()` has been improved for folders with a large number of files
- Added URL to known error recommendations in run details.
- Fixed a bug in `run.get_metrics` where requests would fail if a run had too many children
- Fixed a bug in `run.get_metrics` where requests would fail if a run had too many children
- Added support for authentication on Arcadia cluster.
- Creating an `Experiment` object gets or creates the experiment in the Azure Machine Learning workspace for run history tracking. The experiment ID and archived time are populated in the `Experiment` object on creation. Example: `experiment = Experiment(workspace, "New Experiment")` `experiment_id = experiment.id` `archive()` and `reactivate()` are functions that can be called on an experiment to hide and restore the experiment from being shown in the UX or returned by default in a call to list experiments. If a new experiment is created with the same name as an archived experiment, you can rename the archived experiment when reactivating by passing a new name. There can only be one active experiment with a given name. Example: `experiment1 = Experiment(workspace, "Active Experiment")` `experiment1.archive() # Create new active experiment with the same name as the archived. experiment2. = Experiment(workspace, "Active Experiment")` `experiment1/reactivate(new_name="Previous Active Experiment")` The static method `list()` on `Experiment` can take a name filter and `ViewType` filter. `ViewType` values are `"ACTIVE_ONLY"`, `"ARCHIVED_ONLY"` and `"ALL"`. Example: `archived_experiments = Experiment.list(workspace, view_type="ARCHIVED_ONLY")` `all_first_experiments = Experiment.list(workspace, name="First Experiment", view_type="ALL")`
- Support using environment for model deployment, and service update
- **azureml-datadrift**
 - The `show` attribute of `DataDriftDetector` class won't support optional argument '`with_details`' anymore. The `show` attribute will only present data drift coefficient and data drift contribution of feature columns.
 - `DataDriftDetector` attribute '`get_output`' behavior changes:
 - Input parameter `start_time`, `end_time` are optional instead of mandatory;
 - Input specific `start_time` and/or `end_time` with a specific `run_id` in the same invoking will result in value error exception because they are mutually exclusive
 - By input specific `start_time` and/or `end_time`, only results of scheduled runs will be returned;
 - Parameter '`daily_latest_only`' is deprecated.
 - Support retrieving Dataset-based Data Drift outputs.
- **azureml-explain-model**
 - Renames AzureML-explain-model package to AzureML-interpret, keeping the old package for backwards compatibility for now
 - fixed `automl` bug with raw explanations set to classification task instead of regression by default on download from `ExplanationClient`
 - Add support for `ScoringExplainer` to be created directly using `MimicWrapper`
- **azureml-pipeline-core**
 - Improved performance for large Pipeline creation
- **azureml-train-core**
 - Added TensorFlow 2.0 support in TensorFlow Estimator
- **azureml-train-automl**
 - Creating an `Experiment` object gets or creates the experiment in the Azure Machine

Learning workspace for run history tracking. The experiment ID and archived time are populated in the Experiment object on creation. Example:

```
experiment = Experiment(workspace, "New Experiment")
experiment_id = experiment.id
```

[archive\(\)](#) and [reactivate\(\)](#) are functions that can be called on an experiment to hide and restore the experiment from being shown in the UX or returned by default in a call to list experiments. If a new experiment is created with the same name as an archived experiment, you can rename the archived experiment when reactivating by passing a new name. There can only be one active experiment with a given name. Example:

```
experiment1 = Experiment(workspace, "Active Experiment")
experiment1.archive()
# Create new active experiment with the same name as the archived.
experiment2 = Experiment(workspace, "Active Experiment")
experiment1.reactivate(new_name="Previous Active Experiment")
```

The static method [list\(\)](#) on Experiment can take a name filter and ViewType filter. ViewType values are "ACTIVE_ONLY", "ARCHIVED_ONLY" and "ALL". Example:

```
archived_experiments = Experiment.list(workspace, view_type="ARCHIVED_ONLY")
all_first_experiments = Experiment.list(workspace, name="First Experiment",
view_type="ALL")
```

- Support using environment for model deployment, and service update.
- **azureml-datadrift**
 - The show attribute of [DataDriftDetector](#) class won't support optional argument 'with_details' anymore. The show attribute will only present data drift coefficient and data drift contribution of feature columns.
 - DataDriftDetector function [get_output]python/api/azureml-datadrift/azureml.datadrift.datadriftdetector.datadriftdetector#get-output-start-time-none--end-time-none--run-id-none- behavior changes:
 - Input parameter start_time, end_time are optional instead of mandatory;
 - Input specific start_time and/or end_time with a specific run_id in the same invoking will result in value error exception because they are mutually exclusive;
 - By input specific start_time and/or end_time, only results of scheduled runs will be returned;
 - Parameter 'daily_latest_only' is deprecated.
 - Support retrieving Dataset-based Data Drift outputs.
- **azureml-explain-model**
 - Add support for [ScoringExplainer](#) to be created directly using MimicWrapper
- **azureml-pipeline-core**
 - Improved performance for large Pipeline creation.
- **azureml-train-core**
 - Added TensorFlow 2.0 support in [TensorFlow](#) Estimator.
- **azureml-train-automl**
 - The parent run will no longer be failed when setup iteration failed, as the orchestration already takes care of it.
 - Added local-docker and local-conda support for AutoML experiments

- Added local-docker and local-conda support for AutoML experiments.

2019-10-08

New web experience (preview) for Azure Machine Learning workspaces

The Experiment tab in the [new workspace portal](#) has been updated so data scientists can monitor experiments in a more performant way. You can explore the following features:

- Experiment metadata to easily filter and sort your list of experiments
- Simplified and performant experiment details pages that allow you to visualize and compare your runs
- New design to run details pages to understand and monitor your training runs

2019-09-30

Azure Machine Learning SDK for Python v1.0.65

- **New features**

- Added curated environments. These environments have been pre-configured with libraries for common machine learning tasks, and have been pre-build and cached as Docker images for faster execution. They appear by default in Workspace's list of environment, with prefix "AzureML".
- Added curated environments. These environments have been pre-configured with libraries for common machine learning tasks, and have been pre-build and cached as Docker images for faster execution. They appear by default in [Workspace](#)'s list of environment, with prefix "AzureML".

- **azureml-train-automl**

- [azureml-train-automl](#)

- Added the ONNX conversion support for the ADB and HDI

- **Preview features**

- **azureml-train-automl**

- [azureml-train-automl](#)

- Supported BERT and BiLSTM as text featurizer (preview only)
- Supported featurization customization for column purpose and transformer parameters (preview only)
- Supported raw explanations when user enables model explanation during training (preview only)
- Added Prophet for `timeseries` forecasting as a trainable pipeline (preview only)

- **azureml-contrib-datadrift**

- Packages relocated from `azureml-contrib-datadrift` to `azureml-datadrift`; the `contrib` package will be removed in a future release

- **Bug fixes and improvements**

- **azureml-automl-core**

- Introduced `FeaturizationConfig` to `AutoMLConfig` and `AutoMLBaseSettings`
- Introduced `FeaturizationConfig` to [AutoMLConfig](#) and `AutoMLBaseSettings`
 - Override Column Purpose for Featurization with given column and feature type
 - Override transformer parameters
- Added deprecation message for `explain_model()` and `retrieve_model_explanations()`
- Added Prophet as a trainable pipeline (preview only)

- Added deprecation message for `explain_model()` and `retrieve_model_explanations()`.
- Added Prophet as a trainable pipeline (preview only).
- Added support for automatic detection of target lags, rolling window size, and maximal horizon. If one of `target_lags`, `target_rolling_window_size` or `max_horizon` is set to 'auto', the heuristics will be applied to estimate the value of corresponding parameter based on training data.
- Fixed forecasting in the case when data set contains one grain column, this grain is of a numeric type and there is a gap between train and test set
- Fixed the error message about the duplicated index in the remote run in forecasting tasks
- Fixed forecasting in the case when data set contains one grain column, this grain is of a numeric type and there is a gap between train and test set.
- Fixed the error message about the duplicated index in the remote run in forecasting tasks.
- Added a guardrail to check whether a dataset is imbalanced or not. If it is, a guardrail message would be written to the console.
- **azureml-core**
 - Added ability to retrieve SAS URL to model in storage through the `model` object. Ex: `model.get_sas_url()`
 - Introduce `run.get_details()['datasets']` to get datasets associated with the submitted run
 - Add API `Dataset.Tabular.from_json_lines_files` to create a TabularDataset from JSON Lines files. To learn about this tabular data in JSON Lines files on TabularDataset, visit [this article](#) for documentation.
 - Added additional VM size fields (OS Disk, number of GPUs) to the `supported_vmsizes()` function
 - Added additional fields to the `list_nodes()` function to show the run, the private and the public IP, the port etc.
 - Ability to specify a new field during cluster provisioning `--remotelogin_port_public_access` which can be set to enabled or disabled depending on whether you would like to leave the SSH port open or closed at the time of creating the cluster. If you do not specify it, the service will smartly open or close the port depending on whether you are deploying the cluster inside a VNet.
- **azureml-explain-model**
- **azureml-core**
 - Added ability to retrieve SAS URL to model in storage through the `model` object. Ex: `model.get_sas_url()`
 - Introduce `run.get_details()['datasets']` to get datasets associated with the submitted run
 - Add API `Dataset.Tabular.from_json_lines_files()` to create a TabularDataset from JSON Lines files. To learn about this tabular data in JSON Lines files on TabularDataset, visit <https://aka.ms/azureml-data> for documentation.
 - Added additional VM size fields (OS Disk, number of GPUs) to the `supported_vmsizes()` function
 - Added additional fields to the `list_nodes()` function to show the run, the private, and the public IP, the port etc.
 - Ability to specify a new field during cluster `provisioning` that can be set to enabled or disabled depending on whether you would like to leave the SSH port open or closed at the time of creating the cluster. If you do not specify it, the service will smartly open or close the port depending on whether you are deploying the cluster inside a VNet.
- **azureml-explain-model**
 - Improved documentation for Explanation outputs in the classification scenario.
 - Added the ability to upload the predicted y values on the explanation for the evaluation

- examples. Unlocks more useful visualizations.
- Added explainer property to MimicWrapper to enable getting the underlying MimicExplainer.
 - **azureml-pipeline-core**
 - Added notebook to describe Module, ModuleVersion, and ModuleStep
 - **azureml-pipeline-steps**
 - Added RScriptStep to support R script run via AML pipeline.
 - Fixed metadata parameters parsing in AzureBatchStep that was causing the error message "assignment for parameter SubscriptionId is not specified."
 - **azureml-train-automl**
 - Supported training_data, validation_data, label_column_name, weight_column_name as data input format
 - Added deprecation message for explain_model() and retrieve_model_explanations()
 - **azureml-pipeline-core**
 - Added a [notebook](#) to describe [Module](#), [ModuleVersion, and [ModuleStep](#).
 - **azureml-pipeline-steps**
 - Added [RScriptStep](#) to support R script run via AML pipeline.
 - Fixed metadata parameters parsing in [AzureBatchStep that was causing the error message "assignment for parameter SubscriptionId is not specified".
 - **azureml-train-automl**
 - Supported training_data, validation_data, label_column_name, weight_column_name as data input format.
 - Added deprecation message for [explain_model\(\)](#) and [retrieve_model_explanations\(\)](#).

2019-09-16

Azure Machine Learning SDK for Python v1.0.62

- New features

- Introduced the `timeseries` trait on TabularDataset. This trait enables easy timestamp filtering on data a TabularDataset, such as taking all data between a range of time or the most recent data. <https://github.com/Azure/MachineLearningNotebooks/blob/master/how-to-use-azureml/work-with-data/datasets-tutorial/timeseries-datasets/tabular-timeseries-dataset-filtering.ipynb> for an example notebook.
- Enabled training with TabularDataset and FileDataset.
- **azureml-train-core**
 - Added `Nccl` and `Gloo` support in PyTorch estimator

- Bug fixes and improvements

- **azureml-automl-core**
 - Deprecated the AutoML setting 'lag_length' and the LaggingTransformer.
 - Fixed correct validation of input data if they are specified in a Dataflow format
 - Modified the fit_pipeline.py to generate the graph json and upload to artifacts.
 - Rendered the graph under `userrun` using `Cytoscape`.
- **azureml-core**
 - Revisited the exception handling in ADB code and make changes to as per new error handling
 - Added automatic MSI authentication for Notebook VMs.
 - Fixes bug where corrupt or empty models could be uploaded because of failed retries.
 - Fixed the bug where `DataReference` name changes when the `DataReference` mode changes

(for example, when calling `as_upload`, `as_download`, or `as_mount`).

- Make `mount_point` and `target_path` optional for `FileDataset.mount` and `FileDataset.download`.
 - Exception that timestamp column cannot be found will be throw out if the time serials-related API is called without fine timestamp column assigned or the assigned timestamp columns are dropped.
 - Time serials columns should be assigned with column whose type is Date, otherwise exception is expected
 - Time serials columns assigning API 'with_timestamp_columns' can take None value fine/coarse timestamp column name, which will clear previously assigned timestamp columns.
 - Exception will be thrown out when either coarse grain or fine grained timestamp column is dropped with indication for user that dropping can be done after either excluding timestamp column in dropping list or call with_time_stamp with None value to release timestamp columns
 - Exception will be thrown out when either coarse grain or fine grained timestamp column is not included in keep columns list with indication for user that keeping can be done after either including timestamp column in keep column list or call with_time_stamp with None value to release timestamp columns.
 - Added logging for the size of a registered model.
- **azureml-explain-model**
 - Fixed warning printed to console when "packaging" Python package is not installed: "Using older than supported version of lightgbm, please upgrade to version greater than 2.2.1"
 - Fixed download model explanation with sharding for global explanations with many features
 - Fixed mimic explainer missing initialization examples on output explanation
 - Fixed immutable error on set properties when uploading with explanation client using two different types of models
 - Added a `get_raw` param to `scoring_explainer.explain()` so one scoring explainer can return both engineered and raw values.
 - **azureml-train-automl**
 - Introduced public APIs from AutoML for supporting explanations from `automl` explain SDK - Newer way of supporting AutoML explanations by decoupling AutoML featurization and explain SDK - Integrated raw explanation support from `azureml` explain SDK for AutoML models.
 - Removing `azureml-defaults` from remote training environments.
 - Changed default cache store location from `FileCacheStore` based one to `AzureFileCacheStore` one for AutoML on Azure Databricks code path.
 - Fixed correct validation of input data if they are specified in a Dataflow format
 - **azureml-train-core**
 - Reverted `source_directory_data_store` deprecation.
 - Added ability to override `azureml` installed package versions.
 - Added dockerfile support in `environment_definition` parameter in estimators.
 - Simplified distributed training parameters in estimators.

```
from azureml.train.dnn import TensorFlow, Mpi, ParameterServer
```

2019-09-09

New web experience (preview) for Azure Machine Learning workspaces

The new web experience enables data scientists and data engineers to complete their end-to-end machine learning lifecycle from prepping and visualizing data to training and deploying models in a single location.

The screenshot shows the Microsoft Azure Machine Learning interface in preview. On the left, there's a navigation sidebar with sections like Home, Authoring, Assets, and Manage. The main area has a 'Welcome!' section with four cards: 'Create New' (with a plus icon), 'Automated ML' (with a gear icon), 'Visual Interface' (with a cube icon), and 'Notebooks' (with a document icon). Below this is a 'My Recent Resources' section divided into 'Runs' and 'Compute' tables.

Runs				Compute		
Run Number	Experiment	Creation Time	Status	Name	Type	Provis...
98	spawnruns	9/9/2019, 9:30:14 ...	NotSt...	greg10	Virtual Machine	Succe...
583	Schedule_Run	9/9/2019, 9:07:48 ...	Compl...	greg11	Virtual Machine	Succe...
581	Schedule_Run	9/9/2019, 8:07:48 ...	Compl...	wb	Virtual Machine	Succe...
579	Schedule_Run	9/9/2019, 7:07:49 ...	Compl...	sindhup-weNBVM	Virtual Machine	Succe...
577	Schedule_Run	9/9/2019, 6:07:48 ...	Compl...	sindhup-VM	Virtual Machine	Succe...

Key features:

Using this new Azure Machine Learning interface, you can now:

- Manage your notebooks or link out to Jupyter
- Run automated ML experiments
- Create datasets from local files, datastores, & web files
- Explore & prepare datasets for model creation
- Monitor data drift for your models
- View recent resources from a dashboard

At the time of this release, the following browsers are supported: Chrome, Firefox, Safari, and Microsoft Edge Preview.

Known issues:

1. Refresh your browser if you see "Something went wrong! Error loading chunk files" when deployment is in progress.
2. Can't delete or rename file in Notebooks and Files. During Public Preview, you can use Jupyter UI or Terminal in Notebook VM to perform update file operations. Because it is a mounted network file system all changes you make on Notebook VM are immediately reflected in the Notebook Workspace.
3. To SSH into the Notebook VM:
 - a. Find the SSH keys that were created during VM setup. Or, find the keys in the Azure Machine Learning workspace > open Compute tab > locate Notebook VM in the list > open its properties: copy the keys from the dialog.
 - b. Import those public and private SSH keys to your local machine.
 - c. Use them to SSH into the Notebook VM.

2019-09-03

Azure Machine Learning SDK for Python v1.0.60

• New features

- Introduced FileDataset, which references single or multiple files in your datastores or public urls. The files can be of any format. FileDataset provides you with the ability to download or mount the files to your compute.
- Added Pipeline Yaml Support for PythonScript Step, Adla Step, Databricks Step, DataTransferStep, and AzureBatch Step

• Bug fixes and improvements

◦ azureml-automl-core

- AutoArima is now a suggestable pipeline for preview only.
- Improved error reporting for forecasting.
- Improved the logging by using custom exceptions instead of generic in the forecasting tasks.
- Removed the check on max_concurrent_iterations to be less than total number of iterations.
- AutoML models now return AutoMLExceptions
- This release improves the execution performance of automated machine learning local runs.

◦ azureml-core

- Introduce Dataset.get_all(workspace), which returns a dictionary of `TabularDataset` and `FileDataset` objects keyed by their registration name.

```
workspace = Workspace.from_config()
all_datasets = Dataset.get_all(workspace)
mydata = all_datasets['my-data']
```

- Introduce `partition_format` as argument to `Dataset.Tabular.from_delimited_files` and `Dataset.Tabular.from_parquet_files`. The partition information of each data path will be extracted into columns based on the specified format. '{column_name}' creates string column, and '{column_name:yyyy/MM/dd/HH/mm/ss}' creates datetime column, where 'yyyy', 'MM', 'dd', 'HH', 'mm' and 'ss' are used to extract year, month, day, hour, minute, and second for the datetime type. The partition_format should start from the position of first partition key until the end of file path. For example, given the path './USA/2019/01/01/data.csv' where the partition is by country and time, `partition_format='/{Country}/{PartitionDate:yyyy/MM/dd}/data.csv'` creates string column 'Country' with value 'USA' and datetime column 'PartitionDate' with value '2019-01-01'.

```
workspace = Workspace.from_config()
all_datasets = Dataset.get_all(workspace)
mydata = all_datasets['my-data']
```

- Introduce `partition_format` as argument to `Dataset.Tabular.from_delimited_files` and `Dataset.Tabular.from_parquet_files`. The partition information of each data path will be extracted into columns based on the specified format. '{column_name}' creates string column, and '{column_name:yyyy/MM/dd/HH/mm:ss}' creates datetime column, where 'yyyy', 'MM', 'dd', 'HH', 'mm' and 'ss' are used to extract year, month, day, hour, minute, and second for the datetime type. The partition_format should start from the position of first partition key until the end of file path. For example, given the path './USA/2019/01/01/data.csv' where the partition is by country and time, `partition_format='/{Country}/{PartitionDate:yyyy/MM/dd}/data.csv'` creates string column 'Country' with value 'USA' and datetime column 'PartitionDate' with value '2019-01-01'.

- `to_csv_files` and `to_parquet_files` methods have been added to `TabularDataset`. These methods enable conversion between a `TabularDataset` and a `FileDataset` by converting the data to files of the specified format.
- Automatically log into the base image registry when saving a Dockerfile generated by `Model.package()`.
- 'gpu_support' is no longer necessary; AML now automatically detects and uses the nvidia docker extension when it is available. It will be removed in a future release.
- Added support to create, update, and use PipelineDrafts.
- This release improves the execution performance of automated machine learning local runs.
- Users can query metrics from run history by name.
- Improved the logging by using custom exceptions instead of generic in the forecasting tasks.
- **azureml-explain-model**
 - Added feature_maps parameter to the new MimicWrapper, allowing users to get raw feature explanations.
 - Dataset uploads are now off by default for explanation upload, and can be re-enabled with `upload_datasets=True`
 - Added "is_law" filtering parameters to explanation list and download functions.
 - Adds method `get_raw_explanation(feature_maps)` to both global and local explanation objects.
 - Added version check to lightgbm with printed warning if below supported version
 - Optimized memory usage when batching explanations
 - AutoML models now return AutoMLExceptions
- **azureml-pipeline-core**
 - Added support to create, update, and use PipelineDrafts - can be used to maintain mutable pipeline definitions and use them interactively to run
- **azureml-train-automl**
 - Created feature to install specific versions of gpu-capable pytorch v1.1.0, cuda toolkit 9.0, pytorch-transformers, which is required to enable BERT/ XLNet in the remote Python runtime environment.
- **azureml-train-core**
 - Early failure of some hyperparameter space definition errors directly in the sdk instead of server side.

Azure Machine Learning Data Prep SDK v1.1.4

- **Bug fixes and improvements**
 - Enabled writing to ADLS/ADLSSGen2 using raw path and credentials.
 - Fixed a bug that caused `include_path=True` to not work for `read_parquet`.
 - Fixed `to_pandas_dataframe()` failure caused by exception "Invalid property value: hostSecret".
 - Fixed a bug where files could not be read on DBFS in Spark mode.

2019-08-19

Azure Machine Learning SDK for Python v1.0.57

- **New features**

- Enabled `TabularDataset` to be consumed by AutomatedML. To learn more about `TabularDataset`, visit <https://aka.ms/azureml/howto/createdatasets>.
- Bug fixes and improvements
 - **azure-cli-ml**
 - You can now update the TLS/SSL certificate for the scoring endpoint deployed on AKS cluster both for Microsoft generated and customer certificate.
 - **azureml-automl-core**
 - Fixed an issue in AutoML where rows with missing labels were not removed properly.
 - Improved error logging in AutoML; full error messages will now always be written to the log file.
 - AutoML has updated its package pinning to include `azureml-defaults`, `azureml-explain-model`, and `azureml-datatransform`. AutoML will no longer warn on package mismatches (except for `azureml-train-automl` package).
 - Fixed an issue in `timeseries` where cv splits are of unequal size causing bin calculation to fail.
 - When running ensemble iteration for the Cross-Validation training type, if we ended up having trouble downloading the models trained on the entire dataset, we were having an inconsistency between the model weights and the models that were being fed into the voting ensemble.
 - Fixed the error, raised when training and/or validation labels (`y` and `y_valid`) are provided in the form of pandas dataframe but not as numpy array.
 - Fixed the issue with the forecasting tasks when None was encountered in the Boolean columns of input tables.
 - Allow AutoML users to drop training series that are not long enough when forecasting. - Allow AutoML users to drop grains from the test set that does not exist in the training set when forecasting.
 - **azureml-core**
 - Fixed issue with `blob_cache_timeout` parameter ordering.
 - Added external fit and transform exception types to system errors.
 - Added support for Key Vault secrets for remote runs. Add a `azureml.core.keyvault.KeyVault` class to add, get, and list secrets from the keyvault associated with your workspace. Supported operations are:
 - `azureml.core.workspace.Workspace.get_default_keyvault()`
 - `azureml.core.keyvault.KeyVault.set_secret(name, value)`
 - `azureml.core.keyvault.KeyVault.set_secrets(secrets_dict)`
 - `azureml.core.keyvault.KeyVault.get_secret(name)`
 - `azureml.core.keyvault.KeyVault.get_secrets(secrets_list)`
 - `azureml.core.keyvault.KeyVault.list_secrets()`
 - Additional methods to obtain default keyvault and get secrets during remote run:
 - `azureml.core.workspace.Workspace.get_default_keyvault()`
 - `azureml.core.run.Run.get_secret(name)`
 - `azureml.core.run.Run.get_secrets(secrets_list)`
 - Added additional override parameters to submit-hyperdrive CLI command.
 - Improve reliability of API calls by expanding retries to common requests library exceptions.
 - Add support for submitting runs from a submitted run.
 - Fixed expiring SAS token issue in FileWatcher, which caused files to stop being uploaded after their initial token had expired.
 - Supported importing HTTP csv/tsv files in dataset Python SDK.
 - Deprecated the `Workspace.setup()` method. Warning message shown to users suggests using

- create() or get()/from_config() instead.
- Added Environment.add_private_pip_wheel(), which enables uploading private custom Python packages `whl` to the workspace and securely using them to build/materialize the environment.
- You can now update the TLS/SSL certificate for the scoring endpoint deployed on AKS cluster both for Microsoft generated and customer certificate.
- azureml-explain-model**
 - Added parameter to add a model ID to explanations on upload.
 - Added `is_raw` tagging to explanations in memory and upload.
 - Added pytorch support and tests for azureml-explain-model package.
- azureml-opendatasets**
 - Support detecting and logging auto test environment.
 - Added classes to get US population by county and zip.
- azureml-pipeline-core**
 - Added label property to input and output port definitions.
- azureml-telemetry**
 - Fixed an incorrect telemetry configuration.
- azureml-train-automl**
 - Fixed the bug where on setup failure, error was not getting logged in "errors" field for the setup run and hence was not stored in parent run "errors".
 - Fixed an issue in AutoML where rows with missing labels were not removed properly.
 - Allow AutoML users to drop training series that are not long enough when forecasting.
 - Allow AutoML users to drop grains from the test set that does not exist in the training set when forecasting.
 - Now AutoMLStep passes through `automl` config to backend to avoid any issues on changes or additions of new config parameters.
 - AutoML Data Guardrail is now in public preview. User will see a Data Guardrail report (for classification/regression tasks) after training and also be able to access it through SDK API.
- azureml-train-core**
 - Added torch 1.2 support in PyTorch Estimator.
- azureml-widgets**
 - Improved confusion matrix charts for classification training.

Azure Machine Learning Data Prep SDK v1.1.12

- New features**
 - Lists of strings can now be passed in as input to `read_*` methods.
- Bug fixes and improvements**
 - The performance of `read_parquet` has been improved when running in Spark.
 - Fixed an issue where `column_type_builder` failed in case of a single column with ambiguous date formats.

Azure portal

- Preview Feature**
 - Log and output file streaming is now available for run details pages. The files will stream updates in real time when the preview toggle is turned on.
 - Ability to set quota at a workspace level is released in preview. AmlCompute quotas are allocated at the subscription level, but we now allow you to distribute that quota between workspaces and allocate it for fair sharing and governance. Just click on the **Usages + Quotas** blade in the left navigation bar of your workspace and select the **Configure Quotas** tab. You must be a subscription admin to be

able to set quotas at the workspace level since this is a cross-workspace operation.

2019-08-05

Azure Machine Learning SDK for Python v1.0.55

- **New features**
 - Token-based authentication is now supported for the calls made to the scoring endpoint deployed on AKS. We will continue to support the current key based authentication and users can use one of these authentication mechanisms at a time.
 - Ability to register a blob storage that is behind the virtual network (VNet) as a datastore.
- **Bug fixes and improvements**
 - **azureml-automl-core**
 - Fixes a bug where validation size for CV splits is small and results in bad predicted vs. true charts for regression and forecasting.
 - The logging of forecasting tasks on the remote runs improved, now user is provided with comprehensive error message if the run was failed.
 - Fixed failures of `Timeseries` if preprocess flag is True.
 - Made some forecasting data validation error messages more actionable.
 - Reduced memory consumption of AutoML runs by dropping and/or lazy loading of datasets, especially in between process spawns
 - **azureml-contrib-explain-model**
 - Added model_task flag to explainers to allow user to override default automatic inference logic for model type
 - Widget changes: Automatically installs with `contrib`, no more `nbextension` install/enable - support explanation with global feature importance (for example, Permutative)
 - Dashboard changes: - Box plots and violin plots in addition to `beeswarm` plot on summary page
 - Much faster rerendering of `beeswarm` plot on 'Top -k' slider change - helpful message explaining how top-k is computed - Useful customizable messages in place of charts when data not provided
 - **azureml-core**
 - Added Model.package() method to create Docker images and Dockerfiles that encapsulate models and their dependencies.
 - Updated local webservices to accept InferenceConfigs containing Environment objects.
 - Fixed Model.register() producing invalid models when '.' (for the current directory) is passed as the model_path parameter.
 - Add Run.submit_child, the functionality mirrors Experiment.submit while specifying the run as the parent of the submitted child run.
 - Support configuration options from Model.register in Run.register_model.
 - Ability to run JAR jobs on existing cluster.
 - Now supporting instance_pool_id and cluster_log_dbfs_path parameters.
 - Added support for using an Environment object when deploying a Model to a Webservice. The Environment object can now be provided as a part of the InferenceConfig object.
 - Add appinsifht mapping for new regions - centralus - westus - northcentralus
 - Added documentation for all the attributes in all the Datastore classes.
 - Added blob_cache_timeout parameter to `Datastore.register_azure_blob_container`.
 - Added save_to_directory and load_from_directory methods to `azureml.core.environment.Environment`.
 - Added the "az ml environment download" and "az ml environment register" commands to the

- CLI.
 - Added Environment.add_private_pip_wheel method.
- **azureml-explain-model**
 - Added dataset tracking to Explanations using the Dataset service (preview).
 - Decreased default batch size when streaming global explanations from 10k to 100.
 - Added model_task flag to explainers to allow user to override default automatic inference logic for model type.
- **azureml-mlflow**
 - Fixed bug in mlflow.azureml.build_image where nested directories are ignored.
- **azureml-pipeline-steps**
 - Added ability to run JAR jobs on existing Azure Databricks cluster.
 - Added support instance_pool_id and cluster_log_dbfs_path parameters for DatabricksStep step.
 - Added support for pipeline parameters in DatabricksStep step.
- **azureml-train-automl**
 - Added `docstrings` for the Ensemble related files.
 - Updated docs to more appropriate language for `max_cores_per_iteration` and `max_concurrent_iterations`
 - The logging of forecasting tasks on the remote runs improved, now user is provided with comprehensive error message if the run was failed.
 - Removed get_data from pipeline `automlstep` notebook.
 - Started support `dataprep` in `automlstep`.

Azure Machine Learning Data Prep SDK v1.1.10

- New features
 - You can now request to execute specific inspectors (for example, histogram, scatter plot, etc.) on specific columns.
 - Added a parallelize argument to `append_columns`. If True, data will be loaded into memory but execution will run in parallel; if False, execution will be streaming but single-threaded.

2019-07-23

Azure Machine Learning SDK for Python v1.0.53

- New features
 - Automated Machine Learning now supports training ONNX models on the remote compute target
 - Azure Machine Learning now provides ability to resume training from a previous run, checkpoint, or model files.
 - Learn how to [use estimators to resume training from a previous run](#)
- Bug fixes and improvements
 - **azure-cli-ml**
 - CLI commands "model deploy" and "service update" now accept parameters, config files, or a combination of the two. Parameters have precedence over attributes in files.
 - Model description can now be updated after registration
 - **azureml-automl-core**
 - Update NimbusML dependency to 1.2.0 version (current latest).
 - Adding support for NimbusML estimators & pipelines to be used within AutoML estimators.
 - Fixing a bug in the Ensemble selection procedure that was unnecessarily growing the resulting ensemble even if the scores remained constant.

- Enable reuse of some featurizations across CV Splits for forecasting tasks. This speeds up the run-time of the setup run by roughly a factor of n_cross_validations for expensive featurizations like lags and rolling windows.
- Addressing an issue if time is out of pandas supported time range. We now raise a DataException if time is less than pd.Timestamp.min or greater than pd.Timestamp.max
- Forecasting now allows different frequencies in train and test sets if they can be aligned. For example, "quarterly starting in January" and at "quarterly starting in October" can be aligned.
- The property "parameters" was added to the TimeSeriesTransformer.
- Remove old exception classes.
- In forecasting tasks, the `target_lags` parameter now accepts a single integer value or a list of integers. If the integer was provided, only one lag will be created. If a list is provided, the unique values of lags will be taken. `target_lags=[1, 2, 2, 4]` will create lags of one, two and four periods.
- Fix the bug about losing columns types after the transformation (bug linked);
- In `model.forecast(x, y_query)`, allow `y_query` to be an object type containing None(s) at the begin (#459519).
- Add expected values to `automl` output
- **azureml-contrib-datadrift**
 - Improvements to example notebook including switch to azureml-opendatasets instead of azureml-contrib-opendatasets and performance improvements when enriching data
- **azureml-contrib-explain-model**
 - Fixed transformations argument for LIME explainer for raw feature importance in azureml-contrib-explain-model package
 - Added segmentations to image explanations in image explainer for the AzureML-contrib-explain-model package
 - Add scipy sparse support for LimeExplainer
 - Added `batch_size` to mimic explainer when `include_local=False`, for streaming global explanations in batches to improve execution time of DecisionTreeExplainableModel
- **azureml-contrib-featureengineering**
 - Fix for calling `set_featurizer_timeseries_params()`: dict value type change and null check - Add notebook for `timeseries` featurizer
 - Update NimbusML dependency to 1.2.0 version (current latest).
- **azureml-core**
 - Added the ability to attach DBFS datastores in the AzureML CLI
 - Fixed the bug with datastore upload where an empty folder is created if `target_path` started with `/`
 - Fixed `deepcopy` issue in ServicePrincipalAuthentication.
 - Added the "az ml environment show" and "az ml environment list" commands to the CLI.
 - Environments now support specifying a `base_dockerfile` as an alternative to an already-built `base_image`.
 - The unused RunConfiguration setting `auto_prepare_environment` has been marked as deprecated.
 - Model description can now be updated after registration
 - Bugfix: Model and Image delete now provides more information about retrieving upstream objects that depend on them if delete fails due to an upstream dependency.
 - Fixed bug that printed blank duration for deployments that occur when creating a workspace for some environments.
 - Improved failure exceptions for workspace creation. Such that users don't see "Unable to create workspace. Unable to find..." as the message and instead see the actual creation failure.

- Add support for token authentication in AKS webservices.
- Add `get_token()` method to `Weservice` objects.
- Added CLI support to manage machine learning datasets.
- `Datastore.register_azure_blob_container` now optionally takes a `blob_cache_timeout` value (in seconds) which configures blobfuse's mount parameters to enable cache expiration for this datastore. The default is no timeout, such as when a blob is read, it will stay in the local cache until the job is finished. Most jobs will prefer this setting, but some jobs need to read more data from a large dataset than will fit on their nodes. For these jobs, tuning this parameter will help them succeed. Take care when tuning this parameter: setting the value too low can result in poor performance, as the data used in an epoch may expire before being used again. All reads will be done from blob storage/network rather than the local cache, which negatively impacts training times.
- Model description can now properly be updated after registration
- Model and Image deletion now provides more information about upstream objects that depend on them, which causes the delete to fail
- Improve resource utilization of remote runs using `azureml.mlflow`.
- **`azureml-explain-model`**
 - Fixed transformations argument for LIME explainer for raw feature importance in `azureml-contrib-explain-model` package
 - add `scipy sparse` support for `LimeExplainer`
 - added shape linear explainer wrapper, as well as another level to tabular explainer for explaining linear models
 - for mimic explainer in explain model library, fixed error when `include_local=False` for sparse data input
 - add expected values to `automl` output
 - fixed permutation feature importance when transformations argument supplied to get raw feature importance
 - added `batch_size` to mimic explainer when `include_local=False`, for streaming global explanations in batches to improve execution time of `DecisionTreeExplainableModel`
 - for model explainability library, fixed blackbox explainers where pandas dataframe input is required for prediction
 - Fixed a bug where `explanation.expected_values` would sometimes return a float rather than a list with a float in it.
- **`azureml-mlflow`**
 - Improve performance of `mlflow.set_experiment(experiment_name)`
 - Fix bug in use of `InteractiveLoginAuthentication` for `mlflow tracking_uri`
 - Improve resource utilization of remote runs using `azureml.mlflow`.
 - Improve the documentation of the `azureml-mlflow` package
 - Patch bug where `mlflow.log_artifacts("my_dir")` would save artifacts under `my_dir/<artifact-paths>` instead of `<artifact-paths>`
- **`azureml-opendatasets`**
 - Pin `pyarrow` of `opendatasets` to old versions (<0.14.0) because of memory issue newly introduced there.
 - Move `azureml-contrib-opendatasets` to `azureml-opendatasets`.
 - Allow open dataset classes to be registered to Azure Machine Learning workspace and leverage AML Dataset capabilities seamlessly.
 - Improve NoaalsdWeather enrich performance in non-SPARK version significantly.
- **`azureml-pipeline-steps`**

- DBFS Datastore is now supported for Inputs and Outputs in DatabricksStep.
- Updated documentation for Azure Batch Step with regard to inputs/outputs.
- In AzureBatchStep, changed `delete_batch_job_after_finish` default value to `true`.
- **azureml-telemetry**
 - Move `azureml-contrib-opendatasets` to `azureml-opendatasets`.
 - Allow open dataset classes to be registered to Azure Machine Learning workspace and leverage AML Dataset capabilities seamlessly.
 - Improve NoaalsdWeather enrich performance in non-SPARK version significantly.
- **azureml-train-automl**
 - Updated documentation on `get_output` to reflect the actual return type and provide additional notes on retrieving key properties.
 - Update NimbusML dependency to 1.2.0 version (current latest).
 - add expected values to `automl` output
- **azureml-train-core**
 - Strings are now accepted as compute target for Automated Hyperparameter Tuning
 - The unused RunConfiguration setting `auto_prepare_environment` has been marked as deprecated.

Azure Machine Learning Data Prep SDK v1.1.9

- **New features**
 - Added support for reading a file directly from an http or https url.
- **Bug fixes and improvements**
 - Improved error message when attempting to read a Parquet Dataset from a remote source (which is not currently supported).
 - Fixed a bug when writing to Parquet file format in ADLS Gen 2, and updating the ADLS Gen 2 container name in the path.

2019-07-09

Visual Interface

- **Preview features**
 - Added "Execute R script" module in visual interface.

Azure Machine Learning SDK for Python v1.0.48

- **New features**
 - **azureml-opendatasets**
 - `azureml-contrib-opendatasets` is now available as `azureml-opendatasets`. The old package can still work, but we recommend you using `azureml-opendatasets` moving forward for richer capabilities and improvements.
 - This new package allows you to register open datasets as Dataset in Azure Machine Learning workspace, and leverage whatever functionalities that Dataset offers.
 - It also includes existing capabilities such as consuming open datasets as Pandas/SPARK dataframes, and location joins for some dataset like weather.
- **Preview features**
 - HyperDriveConfig can now accept pipeline object as a parameter to support hyperparameter tuning using a pipeline.
- **Bug fixes and improvements**

- **azureml-train-automl**
 - Fixed the bug about losing columns types after the transformation.
 - Fixed the bug to allow y_query to be an object type containing None(s) at the beginning.
 - Fixed the issue in the Ensemble selection procedure that was unnecessarily growing the resulting ensemble even if the scores remained constant.
 - Fixed the issue with allow_list_models and block_list_models settings in AutoMLStep.
 - Fixed the issue that prevented the usage of preprocessing when AutoML would have been used in the context of Azure ML Pipelines.
- **azureml-opendatasets**
 - Moved azureml-contrib-opendatasets to azureml-opendatasets.
 - Allowed open dataset classes to be registered to Azure Machine Learning workspace and leverage AML Dataset capabilities seamlessly.
 - Improved NoaalsdWeather enrich performance in non-SPARK version significantly.
- **azureml-explain-model**
 - Updated online documentation for interpretability objects.
 - Added `batch_size` to mimic explainer when `include_local=False`, for streaming global explanations in batches to improve execution time of DecisionTreeExplainableModel for model explainability library.
 - Fixed the issue where `explanation.expected_values` would sometimes return a float rather than a list with a float in it.
 - Added expected values to `automl` output for mimic explainer in explain model library.
 - Fixed permutation feature importance when transformations argument supplied to get raw feature importance.
- **azureml-core**
 - Added the ability to attach DBFS datastores in the AzureML CLI.
 - Fixed the issue with datastore upload where an empty folder is created if `target_path` started with `/`.
 - Enabled comparison of two datasets.
 - Model and Image delete now provides more information about retrieving upstream objects that depend on them if delete fails due to an upstream dependency.
 - Deprecated the unused RunConfiguration setting in `auto_prepare_environment`.
- **azureml-mlflow**
 - Improved resource utilization of remote runs that use `azureml.mlflow`.
 - Improved the documentation of the `azureml-mlflow` package.
 - Fixed the issue where `mlflow.log_artifacts("my_dir")` would save artifacts under "my_dir/artifact-paths" instead of "artifact-paths".
- **azureml-pipeline-core**
 - Parameter `hash_paths` for all pipeline steps is deprecated and will be removed in future. By default contents of the `source_directory` is hashed (except files listed in `.amlignore` or `.gitignore`)
 - Continued improving Module and ModuleStep to support compute type-specific modules, to prepare for RunConfiguration integration and other changes to unlock compute type-specific module usage in pipelines.
- **azureml-pipeline-steps**
 - `AzureBatchStep`: Improved documentation with regard to inputs/outputs.
 - `AzureBatchStep`: Changed `delete_batch_job_after_finish` default value to true.
- **azureml-train-core**
 - Strings are now accepted as compute target for Automated Hyperparameter Tuning.

- Deprecated the unused RunConfiguration setting in auto_prepare_environment.
- Deprecated parameters `conda_dependencies_file_path` and `pip_requirements_file_path` in favor of `conda_dependencies_file` and `pip_requirements_file` respectively.
- **azureml-opendatasets**
 - Improve NoaalsdWeather enrich performance in non-SPARK version significantly.

2019-04-26

Azure Machine Learning SDK for Python v1.0.33 released.

- Azure ML Hardware Accelerated Models on [FPGAs](#) is generally available.
 - You can now [use the azureml-accel-models package](#) to:
 - Train the weights of a supported deep neural network (ResNet 50, ResNet 152, DenseNet-121, VGG-16, and SSD-VGG)
 - Use transfer learning with the supported DNN
 - Register the model with Model Management Service and containerize the model
 - Deploy the model to an Azure VM with an FPGA in an Azure Kubernetes Service (AKS) cluster
 - Deploy the container to an [Azure Data Box Edge](#) server device
 - Score your data with the gRPC endpoint with this [sample](#)

Automated Machine Learning

- Feature sweeping to enable dynamically adding featurizers for performance optimization. New featurizers: work embeddings, weight of evidence, target encodings, text target encoding, cluster distance
- Smart CV to handle train/valid splits inside automated ML
- Few memory optimization changes and runtime performance improvement
- Performance improvement in model explanation
- ONNX model conversion for local run
- Added Subsampling support
- Intelligent Stopping when no exit criteria defined
- Stacked ensembles
- Time Series Forecasting
 - New predict forecast function
 - You can now use rolling-origin cross validation on time series data
 - New functionality added to configure time series lags
 - New functionality added to support rolling window aggregate features
 - New Holiday detection and featurizer when country code is defined in experiment settings
- Azure Databricks
 - Enabled time series forecasting and model explainability/interpretability capability
 - You can now cancel and resume (continue) automated ML experiments
 - Added support for multicore processing

MLOps

- **Local deployment & debugging for scoring containers**

You can now deploy an ML model locally and iterate quickly on your scoring file and dependencies to ensure they behave as expected.

- **Introduced InferenceConfig & Model.deploy()**
Model deployment now supports specifying a source folder with an entry script, the same as a RunConfig. Additionally, model deployment has been simplified to a single command.
- **Git reference tracking**
Customers have been requesting basic Git integration capabilities for some time as it helps maintain a complete audit trail. We have implemented tracking across major entities in Azure ML for Git-related metadata (repo, commit, clean state). This information will be collected automatically by the SDK and CLI.
- **Model profiling & validation service**
Customers frequently complain of the difficulty to properly size the compute associated with their inference service. With our model profiling service, the customer can provide sample inputs and we will profile across 16 different CPU / memory configurations to determine optimal sizing for deployment.
- **Bring your own base image for inference**
Another common complaint was the difficulty in moving from experimentation to inference RE sharing dependencies. With our new base image sharing capability, you can now reuse your experimentation base images, dependencies and all, for inference. This should speed up deployments and reduce the gap from the inner to the outer loop.
- **Improved Swagger schema generation experience**
Our previous swagger generation method was error prone and impossible to automate. We have a new in-line way of generating swagger schemas from any Python function via decorators. We have open-sourced this code and our schema generation protocol is not coupled to the Azure ML platform.
- **Azure ML CLI is generally available (GA)**
Models can now be deployed with a single CLI command. We got common customer feedback that no one deploys an ML model from a Jupyter notebook. The [CLI reference documentation](#) has been updated.

2019-04-22

Azure Machine Learning SDK for Python v1.0.30 released.

The [PipelineEndpoint](#) was introduced to add a new version of a published pipeline while maintaining same endpoint.

2019-04-15

Azure portal

- You can now resubmit an existing Script run on an existing remote compute cluster.
- You can now run a published pipeline with new parameters on the Pipelines tab.
- Run details now supports a new Snapshot file viewer. You can view a snapshot of the directory when you submitted a specific run. You can also download the notebook that was submitted to start the run.
- You can now cancel parent runs from the Azure portal.

2019-04-08

Azure Machine Learning SDK for Python v1.0.23

- **New features**
 - The Azure Machine Learning SDK now supports Python 3.7.
 - Azure Machine Learning DNN Estimators now provide built-in multi-version support. For example, [TensorFlow](#) estimator now accepts a [framework_version](#) parameter, and users can specify version '1.10' or '1.12'. For a list of the versions supported by your current SDK release, call

`get_supported_versions()` on the desired framework class (for example, `TensorFlow.get_supported_versions()`). For a list of the versions supported by the latest SDK release, see the [DNN Estimator documentation](#).

2019-03-25

Azure Machine Learning SDK for Python v1.0.21

- **New features**
 - The `azureml.core.Run.create_children` method allows low-latency creation of multiple child-runs with a single call.

2019-03-11

Azure Machine Learning SDK for Python v1.0.18

- **Changes**
 - The `azureml-tensorboard` package replaces `azureml-contrib-tensorboard`.
 - With this release, you can set up a user account on your managed compute cluster (`amlcompute`), while creating it. This can be done by passing these properties in the provisioning configuration. You can find more details in the [SDK reference documentation](#).

Azure Machine Learning Data Prep SDK v1.0.17

- **New features**
 - Now supports adding two numeric columns to generate a resultant column using the expression language.
- **Bug fixes and improvements**
 - Improved the documentation and parameter checking for `random_split`.

2019-02-27

Azure Machine Learning Data Prep SDK v1.0.16

- **Bug fix**
 - Fixed a Service Principal authentication issue that was caused by an API change.

2019-02-25

Azure Machine Learning SDK for Python v1.0.17

- **New features**
 - Azure Machine Learning now provides first class support for popular DNN framework Chainer. Using `Chainer` class users can easily train and deploy Chainer models.
 - Learn how to [run distributed training with ChainerMN](#)
 - Learn how to [run hyperparameter tuning with Chainer using HyperDrive](#)
 - Azure Machine Learning Pipelines added ability to trigger a Pipeline run based on datastore modifications. The pipeline `schedule notebook` is updated to showcase this feature.
- **Bug fixes and improvements**
 - We have added support in Azure Machine Learning pipelines for setting the `source_directory_data_store` property to a desired datastore (such as a blob storage) on `RunConfigurations` that are supplied to the `PythonScriptStep`. By default Steps use Azure File store as the backing datastore, which may run into throttling issues when a large number of steps are executed

concurrently.

Azure portal

- **New features**

- New drag and drop table editor experience for reports. Users can drag a column from the well to the table area where a preview of the table will be displayed. The columns can be rearranged.
- New Logs file viewer
- Links to experiment runs, compute, models, images, and deployments from the activities tab

Next steps

Read the overview for [Azure Machine Learning](#).

Azure Machine Learning CLI (v2) release notes

9/22/2022 • 7 minutes to read • [Edit Online](#)

APPLIES TO:  [Azure CLI ml extension v2 \(current\)](#)

In this article, learn about Azure Machine Learning CLI (v2) releases.

RSS feed: Get notified when this page is updated by copying and pasting the following URL into your feed reader:

`https://learn.microsoft.com/api/search/rss?search=%22Azure+machine+learning+release+notes-v2%22&locale=en-us`

2022-05-24

Azure Machine Learning CLI (v2) v2.4.0

- The Azure Machine Learning CLI (v2) is now GA.
- `az ml job`
 - The command group is marked as GA.
 - Added AutoML job type in public preview.
 - Added `schedules` property to pipeline job in public preview.
 - Added an option to list only archived jobs.
 - Improved reliability of `az ml job download` command.
- `az ml data`
 - The command group is marked as GA.
 - Added MLTable data type in public preview.
 - Added an option to list only archived data assets.
- `az ml environment`
 - Added an option to list only archived environments.
- `az ml model`
 - The command group is marked as GA.
 - Allow models to be created from job outputs.
 - Added an option to list only archived models.
- `az ml online-deployment`
 - The command group is marked as GA.
 - Removed timeout waiting for deployment creation.
 - Improved online deployment list view.
- `az ml online-endpoint`
 - The command group is marked as GA.
 - Added `mirror_traffic` property to online endpoints in public preview.
 - Improved online endpoint list view.
- `az ml batch-deployment`
 - The command group is marked as GA.
 - Added support for `uri_file` and `uri_folder` as invocation input.
 - Fixed a bug in batch deployment update.
 - Fixed a bug in batch deployment list-jobs output.
- `az ml batch-endpoint`

- The command group is marked as GA.
- Added support for `uri_file` and `uri_folder` as invocation input.
- Fixed a bug in batch endpoint update.
- Fixed a bug in batch endpoint list-jobs output.
- `az ml component`
 - The command group is marked as GA.
 - Added an option to list only archived components.
- `az ml code`
 - This command group is removed.

2022-03-14

Azure Machine Learning CLI (v2) v2.2.1

- `az ml job`
 - For all job types, flattened the `code` section of the YAML schema. Instead of `code.local_path` to specify the path to the source code directory, it is now just `code`
 - For all job types, changed the schema for defining data inputs to the job in the job YAML. Instead of specifying the data path using either the `file` or `folder` fields, use the `path` field to specify either a local path, a URI to a cloud path containing the data, or a reference to an existing registered Azure ML data asset via `path: azureml:<data_name>:<data_version>`. Also specify the `type` field to clarify whether the data source is a single file (`uri_file`) or a folder (`uri_folder`). If `type` field is omitted, it defaults to `type: uri_folder`. For more information, see the section of any of the [job YAML references](#) that discuss the schema for specifying input data.
 - In the [sweep job YAML schema](#), changed the `sampling_algorithm` field from a string to an object in order to support additional configurations for the random sampling algorithm type
 - Removed the component job YAML schema. With this release, if you want to run a command job inside a pipeline that uses a component, just specify the component to the `component` field of the command job YAML definition.
 - For all job types, added support for referencing the latest version of a nested asset in the job YAML configuration. When referencing a registered environment or data asset to use as input in a job, you can alias by latest version rather than having to explicitly specify the version. For example:
 - `environment: azureml:AzureML-Minimal@latest`
 - For pipeline jobs, introduced the `${\{ parent \}}` context for binding inputs and outputs between steps in a pipeline. For more information, see [Expression syntax for binding inputs and outputs between steps in a pipeline job](#).
 - Added support for downloading named outputs of job via the `--output-name` argument for the `az ml job download` command
- `az ml data`
 - Deprecated the `az ml dataset` subgroup, now using `az ml data` instead
 - There are two types of data that can now be created, either from a single file source (`type: uri_file`) or a folder (`type: uri_folder`). When creating the data asset, you can either specify the data source from a local file / folder or from a URI to a cloud path location. See the [data YAML schema](#) for the full schema
- `az ml environment`
 - In the [environment YAML schema](#), renamed the `build.local_path` field to `build.path`
 - Removed the `build.context_uri` field, the URI of the uploaded build context location will be accessible via `build.path` when the environment is returned
- `az ml model`

- In the [model YAML schema](#), `model_uri` and `local_path` fields removed and consolidated to one `path` field that can take either a local path or a cloud path URI. `model_format` field renamed to `type`; the default type is `custom_model`, but you can specify one of the other types (`mlflow_model`, `triton_model`) to use the model in no-code deployment scenarios
- For `az ml model create`, `--model-uri` and `--local-path` arguments removed and consolidated to one `--path` argument that can take either a local path or a cloud path URI
- Added the `az ml model download` command to download a model's artifact files
- `az ml online-deployment`
 - In the [online deployment YAML schema](#), flattened the `code` section of the `code_configuration` field. Instead of `code_configuration.code.local_path` to specify the path to the source code directory containing the scoring files, it is now just `code_configuration.code`
 - Added an `environment_variables` field to the online deployment YAML schema to support configuring environment variables for an online deployment
- `az ml batch-deployment`
 - In the [batch deployment YAML schema](#), flattened the `code` section of the `code_configuration` field. Instead of `code_configuration.code.local_path` to specify the path to the source code directory containing the scoring files, it is now just `code_configuration.code`
- `az ml component`
 - Flattened the `code` section of the [command component YAML schema](#). Instead of `code.local_path` to specify the path to the source code directory, it is now just `code`
 - Added support for referencing the latest version of a registered environment to use in the component YAML configuration. When referencing a registered environment, you can alias by latest version rather than having to explicitly specify the version. For example:
 - `environment: azureml:AzureML-Minimal@latest`
 - Renamed the component input and output type value from `path` to `uri_folder` for the `type` field when defining a component input or output
- Removed the `delete` commands for assets (model, component, data, environment). The existing delete functionality is only a soft delete, so the `delete` commands will be reintroduced in a later release once hard delete is supported
- Added support for archiving and restoring assets (model, component, data, environment) and jobs, e.g. `az ml model archive` and `az ml model restore`. You can now archive assets and jobs, which will hide the archived entity from list queries (e.g. `az ml model list`).

2021-10-04

Azure Machine Learning CLI (v2) v2.0.2

- `az ml workspace`
 - Updated [workspace YAML schema](#)
- `az ml compute`
 - Updated YAML schemas for [AmlCompute](#) and [Compute Instance](#)
 - Removed support for legacy AKS attach via `az ml compute attach`. Azure Arc-enabled Kubernetes attach will be supported in the next release
- `az ml datastore`
 - Updated YAML schemas for [Azure blob](#), [Azure file](#), [Azure Data Lake Gen1](#), and [Azure Data Lake Gen2](#) datastores
 - Added support for creating Azure Data Lake Storage Gen1 and Gen2 datastores
- `az ml job`
 - Updated YAML schemas for [command job](#) and [sweep job](#)

- Added support for running pipeline jobs ([pipeline job YAML schema](#))
- Added support for job input literals and input data URIs for all job types
- Added support for job outputs for all job types
- Changed the expression syntax from `{ <expression> }` to `${{ <expression> }}`. For more information, see [Expression syntax for configuring Azure ML jobs](#)
- `az ml environment`
 - Updated [environment YAML schema](#)
 - Added support for creating environments from Docker build context
- `az ml model`
 - Updated [model YAML schema](#)
 - Added new `model_format` property to Model for no-code deployment scenarios
- `az ml dataset`
 - Renamed `az ml data` subgroup to `az ml dataset`
 - Updated dataset YAML schema
- `az ml component`
 - Added the `az ml component` commands for managing Azure ML components
 - Added support for command components ([command component YAML schema](#))
- `az ml online-endpoint`
 - `az ml endpoint` subgroup split into two separate groups: `az ml online-endpoint` and `az ml batch-endpoint`
 - Updated [online endpoint YAML schema](#)
 - Added support for local endpoints for dev/test scenarios
 - Added interactive VSCode debugging support for local endpoints (added the `--vscode-debug` flag to `az ml batch-endpoint create/update`)
- `az ml online-deployment`
 - `az ml deployment` subgroup split into two separate groups: `az ml online-deployment` and `az ml batch-deployment`
 - Updated [managed online deployment YAML schema](#)
 - Added autoscaling support via integration with Azure Monitor Autoscale
 - Added support for updating multiple online deployment properties in the same update operation
 - Added support for performing concurrent operations on deployments under the same endpoint
- `az ml batch-endpoint`
 - `az ml endpoint` subgroup split into two separate groups: `az ml online-endpoint` and `az ml batch-endpoint`
 - Updated [batch endpoint YAML schema](#)
 - Removed `traffic` property; replaced with a configurable default deployment property
 - Added support for input data URIs for `az ml batch-endpoint invoke`
 - Added support for VNet ingress (private link)
- `az ml batch-deployment`
 - `az ml deployment` subgroup split into two separate groups: `az ml online-deployment` and `az ml batch-deployment`
 - Updated [batch deployment YAML schema](#)

2021-05-25

Announcing the CLI (v2) (preview) for Azure Machine Learning

The `ml` extension to the Azure CLI is the next-generation interface for Azure Machine Learning. It enables you to

train and deploy models from the command line, with features that accelerate scaling data science up and out while tracking the model lifecycle. [Install and get started.](#)

Enable preview features for Azure Machine Learning

9/22/2022 • 2 minutes to read • [Edit Online](#)

In Azure Machine Learning, new features and improvements are often first released as preview features before they're made generally available (GA). As new features are introduced, you can turn them on or off in the Azure Machine Learning studio at your convenience. That way, you get a chance to use the latest features, evaluate how they fit your work needs and provide feedback to shape the product. Your feedback is very valuable and it helps us constantly improve the product.

Some preview features provide access to entire new functionality while others may reflect a change to the user interface, but little or no change in functionality.

NOTE

The amount of time a feature remains in preview can vary based on user feedback, quality checks, and long-term road maps.

IMPORTANT

The preview features are provided without a service-level agreement, and are not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Quickstart: Create workspace resources](#).

How do I enable preview features?

You can enable or disable preview features anytime in [Azure Machine Learning studio](#). Use the following steps to discover preview features:

1. From the [Azure Machine Learning studio](#), select the megaphone icon from the top-right corner of the page. The **Preview Features** panel will appear.



2. Find the feature you would like to try out and select the toggle next to it to enable or disable the feature.

TIP

When you disable a feature, a text box will appear that can be used to provide feedback on the feature. To learn how to provide feedback without disabling a feature, see [How do I provide feedback?](#).



Manage preview features

X

Check out the current preview features before they are generally available.

Enable a toggle to immediately turn on a feature, or return here any time to turn it off. You can also provide your feedback about each feature to help inform future development and improvements.

Customizable job metrics experience

Disabled

This new experience makes enhancements to the metrics tab of a job to help you better understand your experimentation results. New capabilities include the ability to create custom charts, rearrange the layout, save personalized views, and share views with your teammates. [Learn more](#)



Tell us more about your experience

How do I provide feedback?

Use the following steps to provide feedback on a feature.

1. From the [Azure Machine Learning studio](#), select the **megaphone** icon from the top-right corner of the page. The **Preview Features** panel will appear.
2. Find the feature you would like to provide feedback on and select the **smile** or **frown**. A text box will appear where you can provide more details.



Manage preview features

X

Check out the current preview features before they are generally available.

Enable a toggle to immediately turn on a feature, or return here any time to turn it off. You can also provide your feedback about each feature to help inform future development and improvements.

Customizable job metrics experience

Disabled

This new experience makes enhancements to the metrics tab of a job to help you better understand your experimentation results. New capabilities include the ability to create custom charts, rearrange the layout, save personalized views, and share views with your teammates. [Learn more](#)



Tell us more about your experience

The text box will also appear when you disable a feature.

Next steps

- [Feature availability across cloud regions](#)

Azure Machine Learning feature availability across clouds regions

9/22/2022 • 8 minutes to read • [Edit Online](#)

Learn what Azure Machine Learning features are available in the Azure Government, Azure Germany, and Azure China 21Vianet regions.

In the list of global Azure regions, there are several regions that serve specific markets in addition to the public cloud regions. For example, the Azure Government and the Azure China 21Vianet regions. Azure Machine Learning is deployed into the following regions, in addition to public cloud regions:

- Azure Government regions **US-Arizona** and **US-Virginia**.
- Azure China 21Vianet region **China-East-2**.

Azure Machine Learning is still in development in air-gap Regions.

The information in the rest of this document provides information on what features of Azure Machine Learning are available in these regions, along with region-specific information on using these features.

Azure Government

FEATURE	PUBLIC CLOUD STATUS	US-VIRGINIA	US-ARIZONA
Automated machine learning			
Create and run experiments in notebooks	GA	YES	YES
Create and run experiments in studio web experience	Public Preview	YES	YES
Industry-leading forecasting capabilities	GA	YES	YES
Support for deep learning and other advanced learners	GA	YES	YES
Large data support (up to 100 GB)	Public Preview	YES	YES
Azure Databricks integration	GA	NO	NO
SQL, Azure Cosmos DB, and HDInsight integrations	GA	YES	YES
Machine Learning pipelines			

FEATURE	PUBLIC CLOUD STATUS	US-VIRGINIA	US-ARIZONA
Create, run, and publish pipelines using the Azure ML SDK	GA	YES	YES
Create pipeline endpoints using the Azure ML SDK	GA	YES	YES
Create, edit, and delete scheduled runs of pipelines using the Azure ML SDK	GA	YES*	YES*
View pipeline run details in studio	GA	YES	YES
Create, run, visualize, and publish pipelines in Azure ML designer	GA	YES	YES
Azure Databricks Integration with ML Pipeline	GA	NO	NO
Create pipeline endpoints in Azure ML designer	GA	YES	YES
Integrated notebooks			
Workspace notebook and file sharing	GA	YES	YES
R and Python support	GA	YES	YES
Virtual Network support	GA	YES	YES
Compute instance			
Managed compute Instances for integrated Notebooks	GA	YES	YES
Jupyter, JupyterLab Integration	GA	YES	YES
Virtual Network (VNet) support	GA	YES	YES
SDK support			
Python SDK support	GA	YES	YES
Security			

Feature	Public Cloud Status	US-Virginia	US-Arizona
Virtual Network (VNet) support for training	GA	YES	YES
Virtual Network (VNet) support for inference	GA	YES	YES
Scoring endpoint authentication	Public Preview	YES	YES
Workplace private endpoint	GA	GA	GA
ACI behind VNet	Public Preview	NO	NO
ACR behind VNet	GA	YES	YES
Private IP of AKS cluster	Public Preview	NO	NO
Network isolation for managed online endpoints	Public Preview	NO	NO
Compute			
quota management across workspaces	GA	YES	YES
Kubernetes compute	GA	NO	NO
Data for machine learning			
Create, view, or edit datasets and datastores from the SDK	GA	YES	YES
Create, view, or edit datasets and datastores from the UI	GA	YES	YES
View, edit, or delete dataset drift monitors from the SDK	Public Preview	YES	YES
View, edit, or delete dataset drift monitors from the UI	Public Preview	YES	YES
Machine learning lifecycle			
Model profiling	GA	YES	PARTIAL
The Azure ML CLI 1.0	GA	YES	YES
FPGA-based Hardware Accelerated Models	GA	NO	NO

FEATURE	PUBLIC CLOUD STATUS	US-VIRGINIA	US-ARIZONA
Visual Studio Code integration	Public Preview	NO	NO
Event Grid integration	Public Preview	NO	NO
Integrate Azure Stream Analytics with Azure Machine Learning	Public Preview	NO	NO
Labeling images and text			
Labeling Project Management Portal	GA	YES	YES
Labeler Portal	GA	YES	YES
Labeling using private workforce	GA	YES	YES
ML assisted labeling (Image classification and object detection)	Public Preview	YES	YES
Responsible ML			
Explainability in UI	Public Preview	NO	NO
Differential privacy SmartNoise toolkit	OSS	NO	NO
Custom tags in Azure Machine Learning to implement datasheets	GA	NO	NO
Fairness AzureML Integration	Public Preview	NO	NO
Interpretability SDK	GA	YES	YES
Training			
Experimentation log streaming	GA	YES	YES
Reinforcement Learning	Public Preview	NO	NO
Experimentation UI	Public Preview	YES	YES
.NET integration ML.NET 1.0	GA	YES	YES
Inference			

FEATURE	PUBLIC CLOUD STATUS	US-VIRGINIA	US-ARIZONA
Managed online endpoints	GA	YES	YES
Batch inferencing	GA	YES	YES
Azure Stack Edge with FPGA	Public Preview	NO	NO
Other			
Open Datasets	Public Preview	YES	YES
Custom Cognitive Search	Public Preview	YES	YES

Azure Government scenarios

SCENARIO	US-VIRGINIA	US-ARIZONA	LIMITATIONS
General security setup			
Disable/control internet access (inbound and outbound) and specific VNet	PARTIAL	PARTIAL	
Placement for all associated resources/services	YES	YES	
Encryption at-rest and in-transit.	YES	YES	
Root and SSH access to compute resources.	YES	YES	
Maintain the security of deployed systems (instances, endpoints, etc.), including endpoint protection, patching, and logging	PARTIAL	PARTIAL	ACI behind VNet currently not available
Control (disable/limit/restrict) the use of ACI/AKS integration	PARTIAL	PARTIAL	ACI behind VNet currently not available
Azure role-based access control (Azure RBAC) - Custom Role Creations	YES	YES	
Control access to ACR images used by ML Service (Azure provided/maintained versus custom)	PARTIAL	PARTIAL	
General Machine Learning Service Usage			

SCENARIO	US-VIRGINIA	US-ARIZONA	LIMITATIONS
Ability to have a development environment to build a model, train that model, host it as an endpoint, and consume it via a webapp	YES	YES	
Ability to pull data from ADLS (Data Lake Storage)	YES	YES	
Ability to pull data from Azure Blob Storage	YES	YES	

Other Azure Government limitations

- For Azure Machine Learning compute instances, the ability to refresh a token lasting more than 24 hours is not available in Azure Government.
- Model Profiling does not support 4 CPUs in the US-Arizona region.
- Sample notebooks may not work in Azure Government if it needs access to public data.
- IP addresses: The CLI command used in the [required public internet access](#) instructions does not return IP ranges. Use the [Azure IP ranges and service tags for Azure Government](#) instead.
- For scheduled pipelines, we also provide a blob-based trigger mechanism. This mechanism is not supported for CMK workspaces. For enabling a blob-based trigger for CMK workspaces, you have to do extra setup. For more information, see [Trigger a run of a machine learning pipeline from a Logic App](#).
- Firewalls: When using an Azure Government region, add the following hosts to your firewall setting:
 - For Arizona use: `usgovarizona.api.ml.azure.us`
 - For Virginia use: `usgovvirginia.api.ml.azure.us`
 - For both: `graph.windows.net`

Azure China 21Vianet

FEATURE	PUBLIC CLOUD STATUS	CH-EAST-2	CH-NORTH-3
Automated machine learning			
Create and run experiments in notebooks	GA	YES	N/A
Create and run experiments in studio web experience	Preview	YES	N/A
Industry-leading forecasting capabilities	GA	YES	N/A
Support for deep learning and other advanced learners	GA	YES	N/A

Feature	Public Cloud Status	CH-East-2	CH-North-3
Large data support (up to 100 GB)	Preview	YES	N/A
Azure Databricks Integration	GA	YES	N/A
SQL, Azure Cosmos DB, and HDInsight integrations	GA	YES	N/A
Machine Learning pipelines			
Create, run, and publish pipelines using the Azure ML SDK	GA	YES	N/A
Create pipeline endpoints using the Azure ML SDK	GA	YES	N/A
Create, edit, and delete scheduled runs of pipelines using the Azure ML SDK	GA	YES	N/A
View pipeline run details in studio	GA	YES	N/A
Create, run, visualize, and publish pipelines in Azure ML designer	GA	YES	N/A
Azure Databricks Integration with ML Pipeline	GA	YES	N/A
Create pipeline endpoints in Azure ML designer	GA	YES	N/A
Integrated notebooks			
Workspace notebook and file sharing	GA	YES	N/A
R and Python support	GA	YES	N/A
Virtual Network support	GA	YES	N/A
Compute instance			
Managed compute Instances for integrated Notebooks	GA	YES	N/A
Jupyter, JupyterLab Integration	GA	YES	N/A

Feature	Public Cloud Status	CH-East-2	CH-North-3
Virtual Network (VNet) support	GA	YES	N/A
SDK support			
Python SDK support	GA	YES	N/A
Security			
Virtual Network (VNet) support for training	GA	YES	N/A
Virtual Network (VNet) support for inference	GA	YES	N/A
Scoring endpoint authentication	Preview	YES	N/A
Workplace Private Endpoint	GA	YES	N/A
ACI behind VNet	Preview	NO	N/A
ACR behind VNet	GA	YES	N/A
Private IP of AKS cluster	Preview	NO	N/A
Network isolation for managed online endpoints	Preview	NO	N/A
Compute			
quota management across workspaces	GA	YES	N/A
Kubernetes compute	GA	NO	NO
Data for machine learning			
Create, view, or edit datasets and datastores from the SDK	GA	YES	N/A
Create, view, or edit datasets and datastores from the UI	GA	YES	N/A
View, edit, or delete dataset drift monitors from the SDK	Preview	YES	N/A
View, edit, or delete dataset drift monitors from the UI	Preview	YES	N/A

FEATURE	PUBLIC CLOUD STATUS	CH-EAST-2	CH-NORTH-3
Machine learning lifecycle			
Model profiling	GA	YES	N/A
The Azure DevOps extension for Machine Learning & the Azure ML CLI	GA	YES	N/A
FPGA-based Hardware Accelerated Models	Deprecating	Deprecating	N/A
Visual Studio Code integration	Preview	NO	N/A
Event Grid integration	Preview	YES	N/A
Integrate Azure Stream Analytics with Azure Machine Learning	Preview	NO	N/A
Labeling			
Labeling Project Management Portal	GA	YES	N/A
Labeler Portal	GA	YES	N/A
Labeling using private workforce	GA	YES	N/A
ML assisted labeling (Image classification and object detection)	Preview	YES	N/A
Responsible AI			
Explainability in UI	Preview	NO	N/A
Differential privacy SmartNoise toolkit	OSS	NO	N/A
custom tags in Azure Machine Learning to implement datasheets	GA	YES	N/A
Fairness AzureML Integration	Preview	NO	N/A
Interpretability SDK	GA	YES	N/A
Training			

FEATURE	PUBLIC CLOUD STATUS	CH-EAST-2	CH-NORTH-3
Experimentation log streaming	GA	YES	N/A
Reinforcement Learning	Deprecating	Deprecating	N/A
Experimentation UI	GA	YES	N/A
.NET integration ML.NET 1.0	GA	YES	N/A
Inference			
Managed online endpoints	GA	YES	N/A
Batch inferencing	GA	YES	N/A
Azure Stack Edge with FPGA	Deprecating	Deprecating	N/A
Other			
Open Datasets	Preview	YES	N/A
Custom Cognitive Search	Preview	YES	N/A

Other Azure China limitations

- Azure China has limited VM SKU, especially for GPU SKU. It only has NCv3 family (V100).
- REST API Endpoints are different from global Azure. Use the following table to find the REST API endpoint for Azure China regions:

REST ENDPOINT	GLOBAL AZURE	CHINA-GOVERNMENT
Management plane	https://management.azure.com/	https://management.chinacloudapi.cn/
Data plane	https://{{location}}.experiments.azureml.net	https://{{location}}.experiments.ml.azure.cn
Azure Active Directory	https://login.microsoftonline.com	https://login.chinacloudapi.cn

- Sample notebook may not work, if it needs access to public data.
- IP address ranges: The CLI command used in the [required public internet access](#) instructions does not return IP ranges. Use the [Azure IP ranges and service tags](#) for Azure China instead.
- Azure Machine Learning compute instances preview is not supported in a workspace where Private Endpoint is enabled for now, but CI will be supported in the next deployment for the service expansion to all AzureML regions.
- Searching for assets in the web UI with Chinese characters will not work correctly.

Next steps

To learn more about the regions that Azure Machine learning is available in, see [Products by region](#).

What happened to Azure Machine Learning Workbench?

9/22/2022 • 4 minutes to read • [Edit Online](#)

The Azure Machine Learning Workbench application and some other early features were deprecated and replaced in the [September 2018](#) release to make way for an improved [architecture](#).

To improve your experience, the release contains many significant updates prompted by customer feedback. The core functionality from experiment runs to model deployment hasn't changed. But now, you can use the robust [Python SDK](#), and the [Azure CLI](#) to accomplish your machine learning tasks and pipelines.

Most of the artifacts that were created in the earlier version of Azure Machine Learning are stored in your own local or cloud storage. These artifacts won't ever disappear.

In this article, you learn about what changed and how it affects your pre-existing work with the Azure Machine Learning Workbench and its APIs.

WARNING

This article is not for Azure Machine Learning Studio users. It is for Azure Machine Learning customers who have installed the Workbench (preview) application and/or have experimentation and model management preview accounts.

What changed?

The latest release of Azure Machine Learning includes the following features:

- A [simplified Azure resources model](#).
- A [new portal UI](#) to manage your experiments and compute targets.
- A new, more comprehensive [Python SDK](#).
- The new expanded [Azure CLI extension](#) for machine learning.

The [architecture](#) was redesigned for ease of use. Instead of multiple Azure resources and accounts, you only need an [Azure Machine Learning Workspace](#). You can create workspaces quickly in the [Azure portal](#). By using a workspace, multiple users can store training and deployment compute targets, model experiments, Docker images, deployed models, and so on.

Although there are new improved CLI and SDK clients in the current release, the desktop workbench application itself has been retired. Experiments can be managed in the [workspace dashboard in Azure Machine Learning studio](#). Use the dashboard to get your experiment history, manage the compute targets attached to your workspace, manage your models and Docker images, and even deploy web services.

Support timeline

On January 9th, 2019 support for Machine Learning Workbench, Azure Machine Learning Experimentation and Model Management accounts, and their associated SDK and CLI ended.

All the latest capabilities are available by using this [SDK](#), the [CLI](#), and the [Azure portal](#).

What about run histories?

Older run histories are no longer accessible, how you can still see your runs in the latest version.

Run histories are now called **experiments**. You can collect your model's experiments and explore them by using the SDK, the CLI, or the Azure Machine Learning studio.

The Azure Machine Learning studio is supported on Microsoft Edge, Chrome, and Firefox browsers only:

Experiment	Latest job	Last submitted	Created
Tutorial-Batch-Scoring	busy_lettuce_0r104hnk	Jun 21, 2022 10:49 AM	Jun 21, 2022 10:49 AM
Datadriftpipeline	yellow_fox_3snl67qt	Jun 21, 2022 10:08 AM	Jun 21, 2022 10:08 AM
pytorch-with-milflow	ivory_cloud_lt2t15fz	Jun 17, 2022 11:00 AM	Jun 17, 2022 10:29 AM
diabetes-experiment	crimson_napkin_9w421snt	Jun 17, 2022 10:16 AM	Jun 17, 2022 10:15 AM

Start training your models and tracking the run histories using the new CLI and SDK. You can learn how with the [Tutorial: train models with Azure Machine Learning](#).

Will projects persist?

You won't lose any code or work. In the older version, projects are cloud entities with a local directory. In the latest version, you attach local directories to the Azure Machine Learning workspace by using a local config file. See a [diagram of the latest architecture](#).

Much of the project content was already on your local machine. So you just need to create a config file in that directory and reference it in your code to connect to your workspace. To continue using the local directory containing your files and scripts, specify the directory's name in the '[experiment.submit](#)' Python command or using the `az ml project attach` CLI command. For example:

APPLIES TO: [Python SDK azureml v1](#)

```
run = exp.submit(source_directory=script_folder,
                  script='train.py', run_config=run_config_system_managed)
```

[Create a workspace](#) to get started.

What about my registered models and images?

The models that you registered in your old model registry must be migrated to your new workspace if you want to continue to use them. To migrate your models, download the models and re-register them in your new workspace.

The images that you created in your old image registry cannot be directly migrated to the new workspace. In most cases, the model can be deployed without having to create an image. If needed, you can create an image for the model in the new workspace. For more information, see [Manage, register, deploy, and monitor machine learning models](#).

What about deployed web services?

Now that support for the old CLI has ended, you can no longer redeploy models or manage the web services

you originally deployed with your Model Management account. However, those web services will continue to work for as long as Azure Container Service (ACS) is still supported.

In the latest version, models are deployed as web services to Azure Container Instances (ACI) or Azure Kubernetes Service (AKS) clusters. You can also deploy to FPGAs.

Learn more in these articles:

- [Where and how to deploy models](#)
- [Tutorial: Train and deploy a model](#)

Next steps

Learn about the [latest architecture for Azure Machine Learning](#).

For an overview of the service, read [What is Azure Machine Learning?](#).

Start with [Quickstart: Get started with Azure Machine Learning](#). Then use these resources to create your first experiment with your preferred method:

- [Run a "Hello world!" Python script \(part 1 of 3\)](#)
- [Use a Jupyter notebook to train image classification models](#)
- [Use automated machine learning](#)
- [Use the designer's drag & drop capabilities](#)
- [Use the ML extension to the CLI](#)

Use a keyboard to use Azure Machine Learning designer

9/22/2022 • 2 minutes to read • [Edit Online](#)

Learn how to use a keyboard and screen reader to use Azure Machine Learning designer. For a list of keyboard shortcuts that work everywhere in the Azure portal, see [Keyboard shortcuts in the Azure portal](#)

This workflow has been tested with [Narrator](#) and [JAWS](#), but it should work with other standard screen readers.

Navigate the pipeline graph

The pipeline graph is organized as a nested list. The outer list is a component list, which describes all the components in the pipeline graph. The inner list is a connection list, which describes input/output ports and details for a specific component connection.

The following keyboard actions help you navigate a pipeline graph:

- Tab: Move to first node > each port of the node > next node.
- Up/down arrow keys: Move to next or previous node by its position in the graph.
- Ctrl+G when focus is on a port: Go to the connected port. When there's more than one connection from one port, open a list view to select the target. Use the Esc key to go to the selected target.

Edit the pipeline graph

Add a component to the graph

1. Use Ctrl+F6 to switch focus from the canvas to the component tree.
2. Find the desired component in the component tree using standard treeview control.

Connect a component to another component

1. Use the Tab key to move focus to a port.
The screen reader reads the port information, which includes whether this port is a valid source port to set a connection to other components.
2. If the current port is a valid source port, press access key + C to start connecting. This command sets this port as the connection source.
3. Using the Tab key, move focus through every available destination port.
4. To use the current port as the destination port and set up the connection, press Enter.
5. To cancel the connection, press Esc.

Edit setting of a component

- Use access key + A to open the component setting panel. Then, use the Tab key to move focus to the setting panel, where you can edit the settings.

Navigation shortcuts

KEYSTROKE	DESCRIPTION
Ctrl + F6	Toggle focus between canvas and component tree
Ctrl + F1	Open the information card when focusing on a node in component tree
Ctrl + Shift + H	Open the connection helper when focus is on a node
Ctrl + Shift + E	Open component properties when focus is on a node
Ctrl + G	Move focus to first failed node if the pipeline run failed

Action shortcuts

Use the following shortcuts with the access key. For more information on access keys, see https://en.wikipedia.org/wiki/Access_key.

KEYSTROKE	ACTION
Access key + R	Run
Access key + P	Publish
Access key + C	Clone
Access key + D	Deploy
Access key + I	Create/update inference pipeline
Access key + B	Create/update batch inference pipeline
Access key + K	Open "Create inference pipeline" dropdown
Access key + U	Open "Update inference pipeline" dropdown
Access key + M	Open more(...) dropdown
Access key + A	Open component settings

Next steps

- [Turn on high contrast or change theme](#)
- [Accessibility related tools at Microsoft](#)