

Contents

Azure Machine Learning Documentation

Overview

[What is Azure Machine Learning?](#)

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

[Architecture & terms](#)

Tutorials

[Python get started \(Day 1\)](#)

1. Set up local computer
2. "Hello world"
3. Train your model
4. Bring your own data

[Jupyter Notebooks](#)

[Image classification \(MNIST data\)](#)

1. Set up compute instance
2. Train a model
3. Deploy a model

[Regression with Automated ML \(NYC Taxi data\)](#)

[Auto-train an ML model](#)

[Batch score models with pipelines](#)

[Prep your code for production](#)

Studio

[Automated ML \(UI\)](#)

[Create automated ML experiments](#)

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

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

[1. Train a regression model](#)

[2. Deploy that model](#)

[Label image data](#)

[R SDK](#)

[Create first ML experiment \(R\)](#)

[CLI](#)

[Train & deploy with CLI](#)

[Visual Studio Code](#)

[Set up Azure Machine Learning extension](#)

[Train and deploy a TensorFlow image classification model](#)

[Microsoft Power BI integration](#)

[Part 1: Train and deploy models](#)

[Use Notebooks \(code\)](#)

[Use designer](#)

[Use automated ML](#)

[Part 2: Consume in Power BI](#)

[Samples](#)

[Jupyter Notebooks](#)

[Example repository](#)

[Designer examples & datasets](#)

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

[Open Datasets \(public\)](#)

[Concepts](#)

[Plan and manage costs](#)

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

[Designer overview](#)

[Algorithm cheat sheet](#)

[How to select algorithms](#)

[Automated ML](#)

[Automated ML overview](#)

[Overfitting & imbalanced data](#)

[Workspace](#)

[Environments](#)

[Compute instance](#)

[Compute target](#)

[Data](#)

- [Data access](#)
- [Data ingestion](#)
- [Data processing](#)
- [Model training](#)
- [Distributed training](#)
- [Deep learning](#)
- [Model management \(MLOps\)](#)
- [Model portability \(ONNX\)](#)
- [Open source integrations](#)
- [ML pipelines](#)
- [Security](#)
 - [Enterprise security overview](#)
 - [Security baseline](#)
- [Responsible ML](#)
 - [Responsible ML overview](#)
 - [Model interpretability](#)
 - [Fairness in Machine Learning](#)
 - [Differential Privacy](#)
- [How-to guides](#)
 - [Security](#)
 - [Identity & access management](#)
 - [Set up authentication](#)
 - [Manage users and roles](#)
 - [Use managed identities for access control](#)
 - [Use Azure AD identity in AKS deployments](#)
 - [Network security](#)
 - [Virtual network overview](#)
 - [Secure workspace resources](#)
 - [Secure training environment](#)
 - [Secure inferencing environment](#)
 - [Use studio in a virtual network](#)
 - [Use Private Link](#)

- [Use custom DNS](#)
- [Configure secure web services](#)
- [Configure Firewall](#)
- [Data protection](#)
 - [Increase resiliency](#)
 - [Regenerate storage access keys](#)
- [Monitor Azure Machine Learning](#)
- [Secure coding](#)
- [Audit and manage compliance](#)
- [Create & manage workspaces](#)
 - [Use Azure portal or Python SDK](#)
 - [Use Azure CLI](#)
 - [Use REST](#)
 - [Use Resource Manager template](#)
- [Create & manage compute resources](#)
 - [Compute instance](#)
 - [Compute cluster](#)
 - [Azure Kubernetes Service](#)
 - [Attach your own](#)
 - [Use studio](#)
- [Set up your environment](#)
 - [Set up dev environments](#)
 - [Run Jupyter Notebooks](#)
 - [Set up software environments](#)
 - [Use private Python packages](#)
 - [Set input & output directories](#)
 - [Visual Studio Code](#)
 - [Git integration](#)
 - [How to troubleshoot environments](#)
- [Work with data](#)
 - [Label data](#)
 - [Get data labeled](#)

- [Label images](#)
- [Create datasets with labels](#)
- [Get data](#)
 - [Data ingestion with Azure Data Factory](#)
 - [DevOps for data ingestion](#)
 - [Import data in the designer](#)
- [Access data](#)
 - [Connect to Azure storage \(Python\)](#)
 - [Get data from a datastore \(Python\)](#)
 - [Connect to data \(UI\)](#)
- [Manage & consume data](#)
 - [Train with datasets](#)
 - [Detect drift on datasets](#)
 - [Version & track datasets](#)
 - [Preserve data privacy](#)
- [Train models](#)
 - [Configure & submit training run](#)
 - [Tune hyperparameters](#)
 - [Work with MLflow](#)
 - [Track experiments with MLflow](#)
 - [Track Azure Databricks runs with MLflow](#)
 - [Scikit-learn](#)
 - [TensorFlow](#)
 - [Keras](#)
 - [PyTorch](#)
 - [Train with custom Docker image](#)
 - [Migrate from Estimators to ScriptRunConfig](#)
 - [Track & monitor training](#)
 - [Start, monitor or cancel runs](#)
 - [Enable logs](#)
 - [View logs](#)
 - [Visualize runs with TensorBoard](#)

Interpret & explain models

Interpret ML models

Explain automated ML models

Assess and mitigate model fairness

Use Key Vault when training

Reinforcement learning

Automated machine learning

Use automated ML (Python)

Use automated ML (interface)

Use automated ML with Databricks

Use remote compute targets

Auto-train a forecast model

Data splits & cross-validation (Python)

Featurization in automated ML (Python)

Use automated ML in ML pipelines (Python)

Understand charts and metrics

Use ONNX model in .NET application

Deploy & serve models

Where and how to deploy

Deployment targets

Use existing models

Azure ML compute instances

Azure Kubernetes Service

Azure Container Instances

GPU inference

Azure App Service

Azure Functions

Azure Cognitive Search

Use custom Docker image

IoT Edge

FPGA inference

Troubleshooting deployment

- [Troubleshooting deployment \(local\)](#)
- [Consume web service](#)
- [Update web service](#)
- [Advanced entry script authoring](#)
- [Monitor web services](#)
 - [Collect & evaluate model data](#)
 - [Monitor with Application Insights](#)
- [High-performance serving with Triton](#)
- [Continuously deploy models](#)
- [Deploy encrypted inferencing service](#)
- [Profile models](#)
- [Deploy designer models](#)
- [\(Preview\) No code deployment](#)
- [Package models](#)
- [Build & use ML pipelines](#)
 - [Create ML pipelines \(Python\)](#)
 - [Moving data into and between ML pipeline steps \(Python\)](#)
 - [Deploy ML pipelines \(Python\)](#)
 - [Trigger a pipeline](#)
 - [Designer \(drag-n-drop\)](#)
 - [Log metrics](#)
 - [Transform data](#)
 - [Retrain using published pipelines](#)
 - [Batch predictions](#)
 - [Execute Python code](#)
 - [Troubleshooting pipelines](#)
 - [Log pipeline data to Application Insights](#)
 - [Troubleshooting the ParallelRunStep](#)
 - [Azure Pipelines for CI/CD](#)
 - [GitHub Actions for CI/CD](#)
 - [Manage resource quotas](#)
 - [Manage resources VS Code](#)

[VS Code interactive debugging](#)

[Export and delete data](#)

[Create event driven workflows](#)

[Reference](#)

[Python SDK](#)

[R SDK](#)

[CLI](#)

[REST API](#)

[Designer module reference](#)

[Monitor data reference](#)

[Machine learning pipeline YAML reference](#)

[Sovereign cloud parity](#)

[Azure Policy built-ins](#)

[Curated environments](#)

[Resources](#)

[Release notes](#)

[Azure roadmap](#)

[Pricing](#)

[Service limits](#)

[Regional availability](#)

[User forum](#)

[Stack Overflow](#)

[Compare our ML products](#)

[What happened to Workbench](#)

[Designer accessibility features](#)

What is Azure Machine Learning?

12/23/2020 • 5 minutes to read • [Edit Online](#)

In this article, you learn about Azure Machine Learning, a cloud-based environment you can use to train, deploy, automate, manage, and track ML models.

Azure Machine Learning can be used for any kind of machine learning, from classical ml to deep learning, supervised, and unsupervised learning. Whether you prefer to write Python or R code with the SDK or work with no-code/low-code options in [the studio](#), you can build, train, and track machine learning and deep-learning models in an Azure Machine Learning Workspace.

Start training on your local machine and then scale out to the cloud.

The service also interoperates with popular deep learning and reinforcement open-source tools such as PyTorch, TensorFlow, scikit-learn, and Ray RLlib.

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](#) today. 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.

What is machine learning?

Machine learning is a data science technique that allows computers to use existing data to forecast future behaviors, outcomes, and trends. By using machine learning, computers learn without being explicitly programmed.

Forecasts or predictions from machine learning can make apps and devices smarter. For example, when you shop online, machine learning helps recommend other products you might want based on what you've bought. Or when your credit card is swiped, machine learning compares the transaction to a database of transactions and helps detect fraud. And when your robot vacuum cleaner vacuums a room, machine learning helps it decide whether the job is done.

Machine learning tools to fit each task

Azure Machine Learning provides all the tools developers and data scientists need for their machine learning workflows, including:

- The [Azure Machine Learning designer](#): drag-n-drop modules to build your experiments and then deploy pipelines.
- Jupyter notebooks: use our [example notebooks](#) or create your own notebooks to leverage our [SDK for Python](#) samples for your machine learning.
- R scripts or notebooks in which you use the [SDK for R](#) to write your own code, or use the R modules in the designer.
- - The [Many Models Solution Accelerator](#) (preview) builds on Azure Machine Learning and enables you to train, operate, and manage hundreds or even thousands of machine learning models.

- Machine learning extension for Visual Studio Code users
- Machine learning CLI
- Open-source frameworks such as PyTorch, TensorFlow, and scikit-learn and many more
- Reinforcement learning with Ray RLlib

You can even use [MLflow](#) to track metrics and deploy models or Kubeflow to build end-to-end workflow pipelines.

Build ML models in Python or R

Start training on your local machine using the Azure Machine Learning [Python SDK](#) or [R SDK](#). Then, you can scale out to the cloud.

With many available [compute targets](#), like Azure Machine Learning Compute and [Azure Databricks](#), and with [advanced hyperparameter tuning services](#), you can build better models faster by using the power of the cloud.

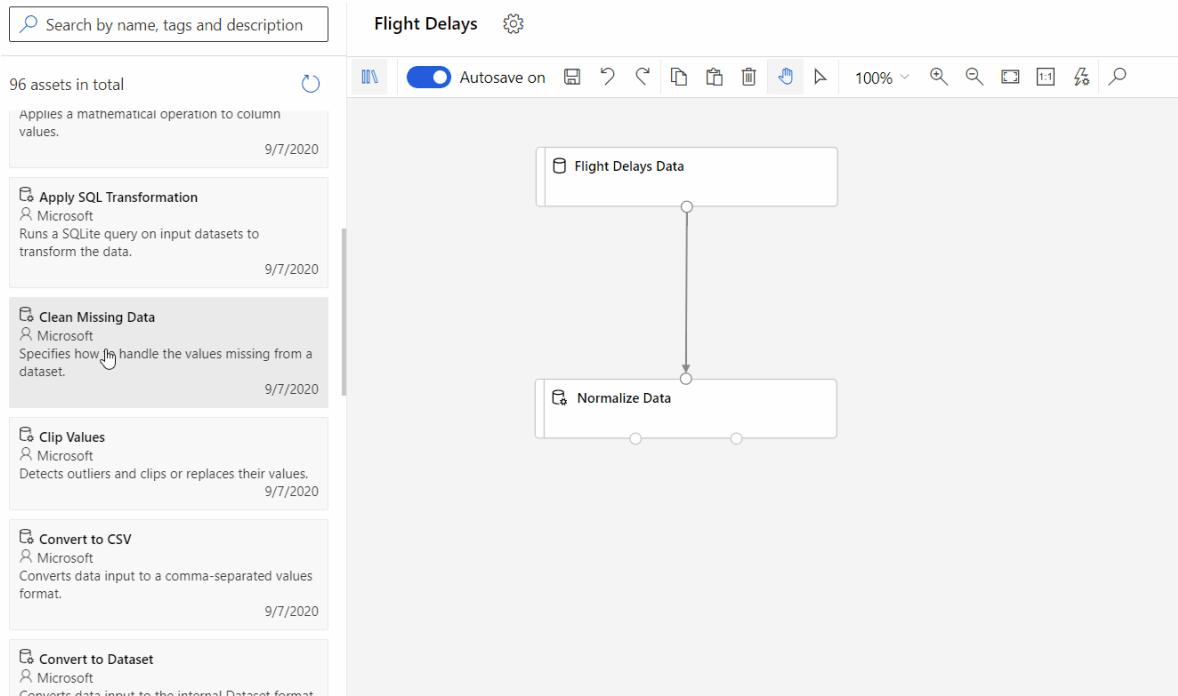
You can also [automate model training and tuning](#) using the SDK.

Build ML models in the studio

[Azure Machine Learning studio](#) is a web portal in Azure Machine Learning for low-code and no-code options for model training, deployment, and asset management. The studio integrates with the Azure Machine Learning SDK for a seamless experience. For more information, see [What is Azure Machine Learning studio](#).

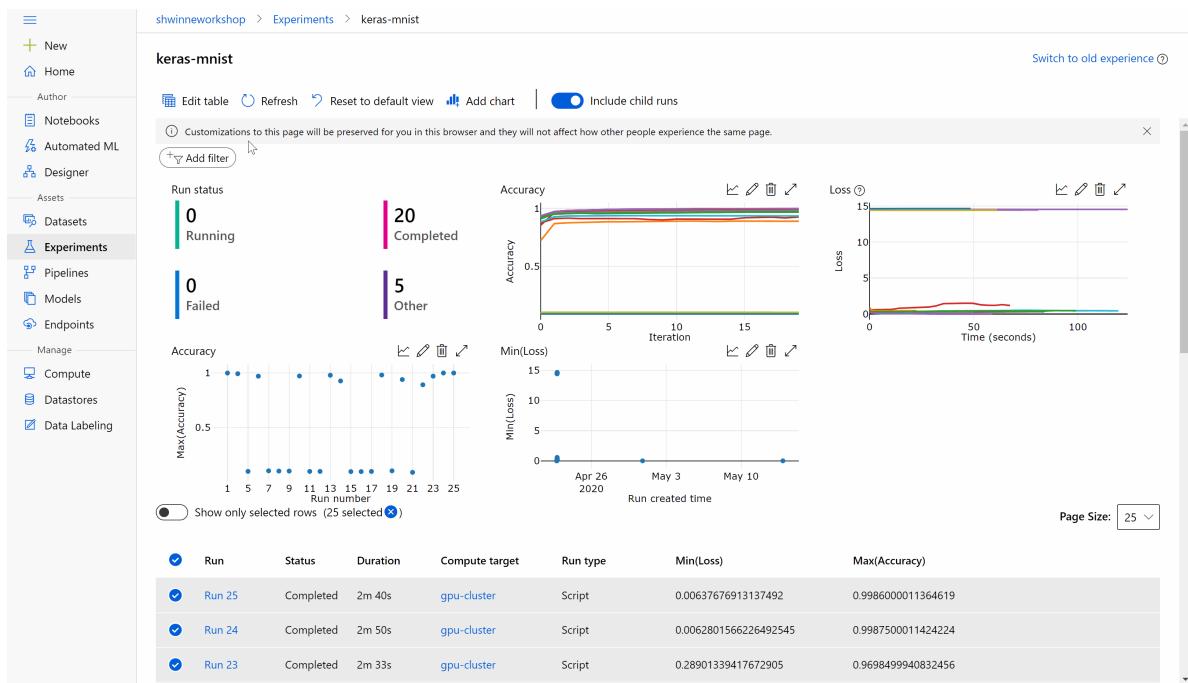
- [Azure Machine Learning designer](#)

Use [the designer](#) to train and deploy machine learning models without writing any code. Try the [designer tutorial](#) to get started.



- [Track experiments](#)

Learn how to [track and visualize data science experiments](#) in the studio.



- And much more...

Visit Azure Machine Learning studio at ml.azure.com.

MLOps: Deploy & lifecycle management

When you have the right model, you can easily use it in a web service, on an IoT device, or from Power BI. For more information, see the article on [how to deploy and where](#).

Then you can manage your deployed models by using the [Azure Machine Learning SDK for Python](#), [Azure Machine Learning studio](#), or the [machine learning CLI](#).

These models can be consumed and return predictions in [real time](#) or [asynchronously](#) on large quantities of data.

And with advanced [machine learning pipelines](#), you can collaborate on each step from data preparation, model training and evaluation, through deployment. Pipelines allow you to:

- Automate the end-to-end machine learning process in the cloud
- Reuse components and only rerun steps when needed
- Use different compute resources in each step
- Run batch scoring tasks

If you want to use scripts to automate your machine learning workflow, the [machine learning CLI](#) provides command-line tools that perform common tasks, such as submitting a training run or deploying a model.

To get started using Azure Machine Learning, see [Next steps](#).

Integration with other services

Azure Machine Learning works with other services on the Azure platform, and also integrates with open source tools such as Git and MLFlow.

- Compute targets such as [Azure Kubernetes Service](#), [Azure Container Instances](#), [Azure Databricks](#), [Azure Data Lake Analytics](#), and [Azure HDInsight](#). For more information on compute targets, see [What are compute targets?](#)
- [Azure Event Grid](#). For more information, see [Consume Azure Machine Learning events](#).
- [Azure Monitor](#). For more information, see [Monitoring Azure Machine Learning](#).

- Data stores such as [Azure Storage accounts](#), [Azure Data Lake Storage](#), [Azure SQL Database](#), [Azure Database for PostgreSQL](#), and [Azure Open Datasets](#). For more information, see [Access data in Azure storage services](#) and [Create datasets with Azure Open Datasets](#).
- [Azure Virtual Networks](#). For more information, see [Virtual network isolation and privacy overview](#).
- [Azure Pipelines](#). For more information, see [Train and deploy machine learning models](#).
- [Git repository logs](#). For more information, see [Git integration](#).
- [MLFlow](#). For more information, see [MLflow to track metrics and deploy models](#)
- [Kubeflow](#). For more information, see [build end-to-end workflow pipelines](#).

Secure communications

Your Azure Storage account, compute targets, and other resources can be used securely inside a virtual network to train models and perform inference. For more information, see [Virtual network isolation and privacy overview](#).

Next steps

- Create your first experiment with your preferred method:
 - [Get started in your own development environment](#)
 - Use Jupyter notebooks on a compute instance to train & deploy ML models
 - Use R Markdown to train & deploy ML models
 - Use automated machine learning to train & deploy ML models
 - Use the designer's drag & drop capabilities to train & deploy
 - Use the machine learning CLI to train and deploy a model
- Learn about [machine learning pipelines](#) to build, optimize, and manage your machine learning scenarios.
- Read the in-depth [Azure Machine Learning architecture and concepts](#) article.

What is Azure Machine Learning studio?

12/23/2020 • 3 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:

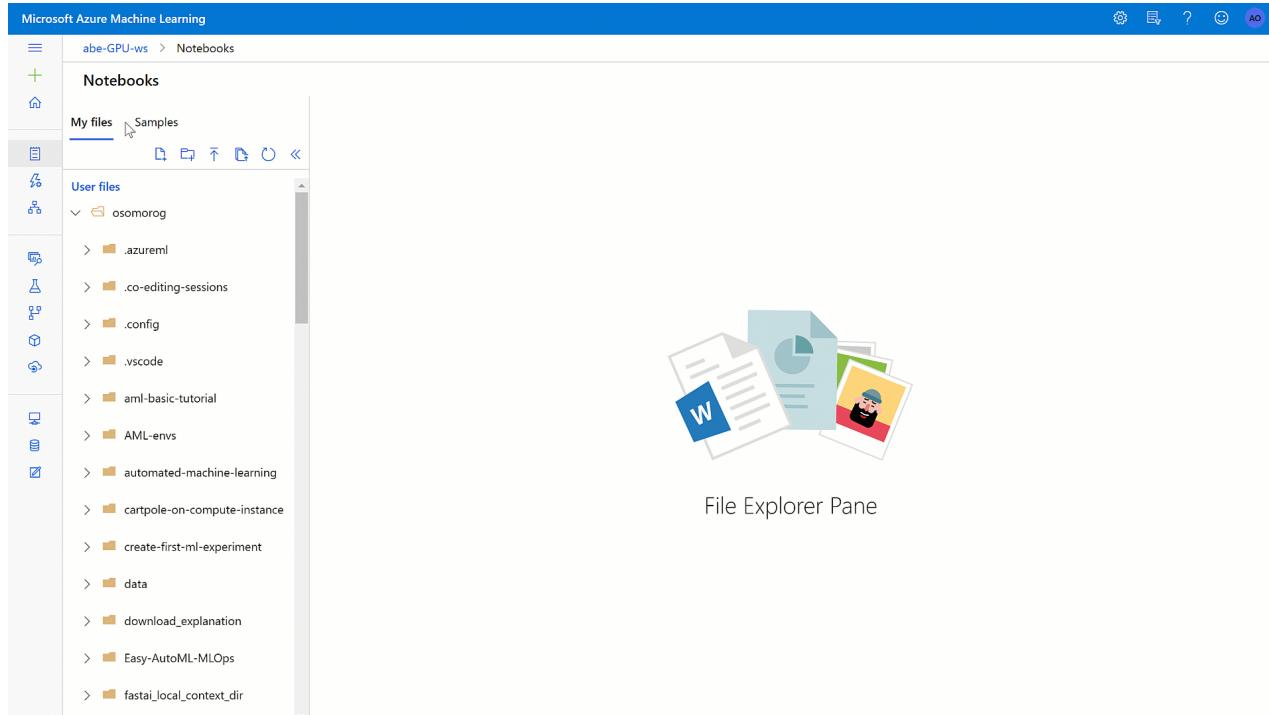
- Microsoft Edge (The new Microsoft Edge, latest version. Not Microsoft Edge legacy)
- Safari (latest version, Mac only)
- Chrome (latest version)
- Firefox (latest version)

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 modules to create ML pipelines. Try out the [designer tutorial](#).

Flight Delays

96 assets in total

Flight Delays Data

Normalize Data

Apply SQL Transformation

Clean Missing Data

Clip Values

Convert to CSV

Convert to Dataset

- **Automated machine learning UI**

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

Preview Microsoft Azure | Machine learning

adb_automation_eastus2_ws > Welcome

Automated machine learning

No recent automated ML runs to display.

Click "New automated ML run" to create your first run

Documentation

View all documentation

Concept: What is automated machine learning?

Tutorial: Create your first classification model with automated machine learning

Blog: Build more accurate forecasts with new capabilities in automated machine learning

- **Data labeling**

Use [Azure Machine Learning data labeling](#) to efficiently coordinate data 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

Released in 2015, **ML Studio (classic)** was our first drag-and-drop machine learning builder. It 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.

We recommend that new users choose **Azure Machine Learning**, instead of ML Studio (classic), for the latest range of data science tools.

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	Unsupported	Fully integrated with Azure Machine Learning Python and R SDKs
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 CI/CD tooling, CPU and GPU deployments and more

FEATURE	ML STUDIO (CLASSIC)	AZURE MACHINE LEARNING
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

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:

- ○ [Get started in your own development environment](#)
- [Use Jupyter notebooks on a compute instance to train & deploy 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: Architecture and concepts

12/23/2020 • 14 minutes to read • [Edit Online](#)

Learn about the architecture and concepts for [Azure Machine Learning](#). This article gives you a high-level understanding of the components and how they work together to assist in the process of building, deploying, and maintaining machine learning models.

Workspace

A [machine learning workspace](#) is the top-level resource for Azure Machine Learning.

The workspace is the centralized place to:

- Manage resources you use for training and deployment of models, such as [computes](#)
- Store assets you create when you use Azure Machine Learning, including:
 - [Environments](#)
 - [Experiments](#)
 - [Pipelines](#)
 - [Datasets](#)
 - [Models](#)
 - [Endpoints](#)

A workspace includes other Azure resources that are used by the workspace:

- [Azure Container Registry \(ACR\)](#): Registers docker containers that you use during training and when you deploy a model. To minimize costs, ACR is only created when deployment images are created.
- [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.
- [Azure Application Insights](#): Stores monitoring information about your models.
- [Azure Key Vault](#): Stores secrets that are used by compute targets and other sensitive information that's needed by the workspace.

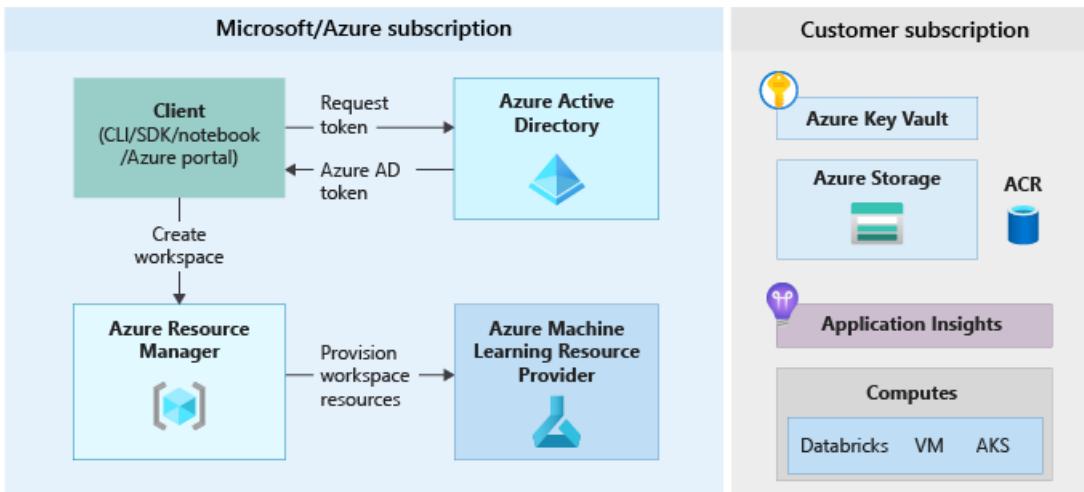
You can share a workspace with others.

Create workspace

The following diagram shows the create workspace workflow.

- You sign in to Azure AD from one of the supported Azure Machine Learning clients (Azure CLI, Python SDK, Azure portal) and request the appropriate Azure Resource Manager token.
- You call Azure Resource Manager to create the workspace.
- Azure Resource Manager contacts the Azure Machine Learning resource provider to provision the workspace.
- If you don't specify existing resources, additional required resources are created in your subscription..

You can also provision other compute targets that are attached to a workspace (like Azure Kubernetes Service or VMs) as needed.



Computes

A [compute target](#) is any machine or set of machines you use to run your training script or host your service deployment. You can use your local machine or a remote compute resource as a compute target. With compute targets, you can start training on your local machine and then scale out to the cloud without changing your training script.

Azure Machine Learning introduces two fully managed cloud-based virtual machines (VM) that are configured for machine learning tasks:

- **Compute instance:** A compute instance is a VM that includes multiple tools and environments installed for machine learning. The primary use of a compute instance is for your development workstation. You can start running sample notebooks with no setup required. A compute instance can also be used as a compute target for training and inferencing jobs.
- **Compute clusters:** Compute clusters are a cluster of VMs with multi-node scaling capabilities. Compute clusters are better suited for compute targets for large jobs and production. The cluster scales up automatically when a job is submitted. Use as a training compute target or for dev/test deployment.

For more information about training compute targets, see [Training compute targets](#). For more information about deployment compute targets, see [Deployment targets](#).

Datasets and datastores

[Azure Machine Learning Datasets](#) make it easier to access and work with your data. By creating a dataset, 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.

For more information, see [Create and register Azure Machine Learning Datasets](#). For more examples using Datasets, see the [sample notebooks](#).

Datasets use [datastores](#) to securely connect to your Azure storage services. Datastores store connection information without putting your authentication credentials and the integrity of your original data source at risk. They store connection information, like your subscription ID and token authorization in your Key Vault associated with the workspace, so you can securely access your storage without having to hard code them in your script.

Environments

[Workspace > Environments](#)

An [environment](#) is the encapsulation of the environment where training or scoring of your machine learning model happens. The environment specifies the Python packages, environment variables, and software settings

around your training and scoring scripts.

For code samples, see the "Manage environments" section of [How to use environments](#).

Experiments

[Workspace](#) > [Experiments](#)

An experiment is a grouping of many runs from a specified script. It always belongs to a workspace. When you submit a run, you provide an experiment name. Information for the run is stored under that experiment. If the name doesn't exist when you submit an experiment, a new experiment is automatically created.

For an example of using an experiment, see [Tutorial: Train your first model](#).

Runs

[Workspace](#) > [Experiments](#) > [Run](#)

A run is a single execution of a training script. An experiment will typically contain multiple runs.

Azure Machine Learning records all runs and stores the following information in the experiment:

- Metadata about the run (timestamp, duration, and so on)
- Metrics that are logged by your script
- Output files that are autocollected by the experiment or explicitly uploaded by you
- A snapshot of the directory that contains your scripts, prior to the run

You produce a run when you submit a script to train a model. A run can have zero or more child runs. For example, the top-level run might have two child runs, each of which might have its own child run.

Run configurations

[Workspace](#) > [Experiments](#) > [Run](#) > [Run configuration](#)

A run configuration defines how a script should be run in a specified compute target. You use the configuration to specify the script, the compute target and Azure ML environment to run on, any distributed job-specific configurations, and some additional properties. For more information on the full set of configurable options for runs, see [ScriptRunConfig](#).

A run configuration can be persisted into a file inside the directory that contains your training script. Or it can be constructed as an in-memory object and used to submit a run.

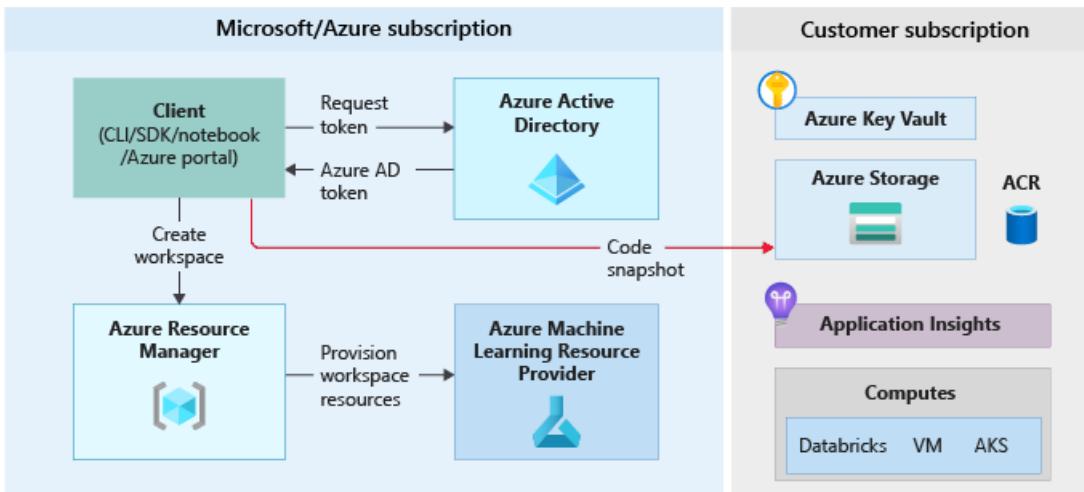
For example run configurations, see [Configure a training run](#).

Snapshots

[Workspace](#) > [Experiments](#) > [Run](#) > [Snapshot](#)

When you submit a run, Azure Machine Learning compresses the directory that contains the script as a zip file and sends it to the compute target. The zip file is then extracted, and the script is run there. Azure Machine Learning also stores the zip file as a snapshot as part of the run record. Anyone with access to the workspace can browse a run record and download the snapshot.

The following diagram shows the code snapshot workflow.



Logging

Azure Machine Learning automatically logs standard run metrics for you. However, you can also [use the Python SDK to log arbitrary metrics](#).

There are multiple ways to view your logs: monitoring run status in real time, or viewing results after completion. For more information, see [Monitor and view ML run logs](#).

NOTE

To prevent unnecessary files from being included in the snapshot, make an ignore file (`.gitignore` or `.amlignore`) 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 `.gitignore`. The `.amlignore` file uses the same syntax. *If both files exist, the `.amlignore` file takes precedence.*

Git tracking and integration

When you start a training run where the source directory is a local Git repository, information about the repository is stored in the run history. This works with runs submitted using a script run configuration or ML pipeline. It also works for runs submitted from the SDK or Machine Learning CLI.

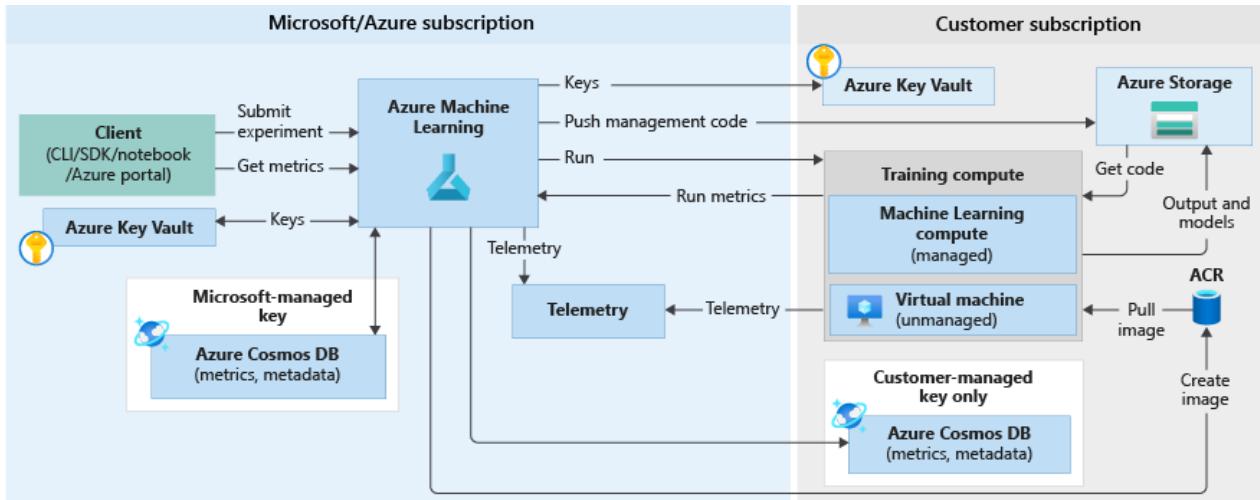
For more information, see [Git integration for Azure Machine Learning](#).

Training workflow

When you run an experiment to train a model, the following steps happen. These are illustrated in the training workflow diagram below:

- Azure Machine Learning is called with the snapshot ID for the code snapshot saved in the previous section.
- Azure Machine Learning creates a run ID (optional) and a Machine Learning service token, which is later used by compute targets like Machine Learning Compute/VMs to communicate with the Machine Learning service.
- You can choose either a managed compute target (like Machine Learning Compute) or an unmanaged compute target (like VMs) to run training jobs. Here are the data flows for both scenarios:
 - VMs/HDIInsight, accessed by SSH credentials in a key vault in the Microsoft subscription. Azure Machine Learning runs management code on the compute target that:
 1. Prepares the environment. (Docker is an option for VMs and local computers. See the following steps for Machine Learning Compute to understand how running experiments on Docker containers works.)
 2. Downloads the code.
 3. Sets up environment variables and configurations.
 4. Runs user scripts (the code snapshot mentioned in the previous section).

- Machine Learning Compute, accessed through a workspace-managed identity. Because Machine Learning Compute is a managed compute target (that is, it's managed by Microsoft) it runs under your Microsoft subscription.
1. Remote Docker construction is kicked off, if needed.
 2. Management code is written to the user's Azure Files share.
 3. The container is started with an initial command. That is, management code as described in the previous step.
- After the run completes, you can query runs and metrics. In the flow diagram below, this step occurs when the training compute target writes the run metrics back to Azure Machine Learning from storage in the Cosmos DB database. Clients can call Azure Machine Learning. Machine Learning will in turn pull metrics from the Cosmos DB database and return them back to the client.



Models

At its simplest, a model is a piece of code that takes an input and produces output. Creating a machine learning model involves selecting an algorithm, providing it with data, and [tuning hyperparameters](#). Training is an iterative process that produces a trained model, which encapsulates what the model learned during the training process.

You can bring a model that was trained outside of Azure Machine Learning. Or you can train a model by submitting a [run](#) of an [experiment](#) to a [compute target](#) in Azure Machine Learning. Once you have a model, you [register the model](#) in the workspace.

Azure Machine Learning is framework agnostic. When you create a model, you can use any popular machine learning framework, such as Scikit-learn, XGBoost, PyTorch, TensorFlow, and Chainer.

For an example of training a model using Scikit-learn, see [Tutorial: Train an image classification model with Azure Machine Learning](#).

Model registry

[Workspace > Models](#)

The **model registry** lets you keep track of all the models in your Azure Machine Learning workspace.

Models are identified by name and version. Each time you register a model with the same name as an existing one, the registry assumes that it's a new version. The version is incremented, and the new model is registered under the same name.

When you register the model, you can provide additional metadata tags and then use the tags when you search for 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 is stored in multiple files, you can register them as a single model in your Azure Machine Learning workspace. After registration, you can then download or deploy the registered model and receive all the files that were registered.

You can't delete a registered model that is being used by an active deployment.

For an example of registering a model, see [Train an image classification model with Azure Machine Learning](#).

Deployment

You deploy a [registered model](#) as a service endpoint. You need the following components:

- **Environment.** This environment encapsulates the dependencies required to run your model for inference.
- **Scoring code.** This script accepts requests, scores the requests by using the model, and returns the results.
- **Inference configuration.** The inference configuration specifies the environment, entry script, and other components needed to run the model as a service.

For more information about these components, see [Deploy models with Azure Machine Learning](#).

Endpoints

[Workspace](#) > [Endpoints](#)

An endpoint is an instantiation of your model into either a web service that can be hosted in the cloud or an IoT module for integrated device deployments.

Web service endpoint

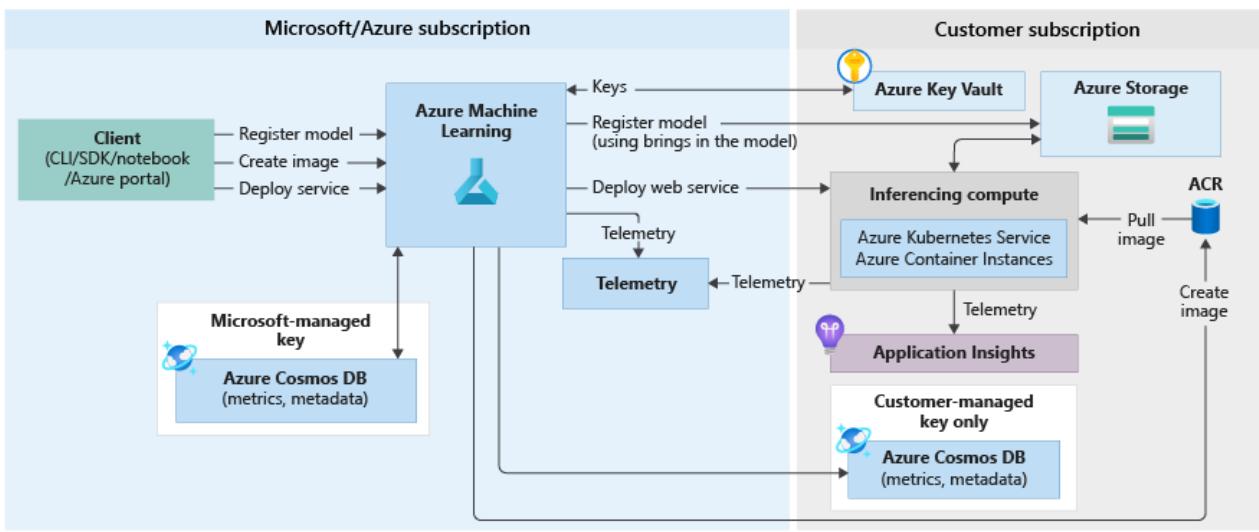
When deploying a model as a web service, the endpoint can be deployed on Azure Container Instances, Azure Kubernetes Service, or FPGAs. You create the service from your model, script, and associated files. These are placed into a base container image, which contains the execution environment for the model. The image has a load-balanced, HTTP endpoint that receives scoring requests that are sent to the web service.

You can enable Application Insights telemetry or model telemetry to monitor your web service. The telemetry data is accessible only to you. It's stored in your Application Insights and storage account instances. If you've enabled automatic scaling, Azure automatically scales your deployment.

The following diagram shows the inference workflow for a model deployed as a web service endpoint:

Here are the details:

- The user registers a model by using a client like the Azure Machine Learning SDK.
- The user creates an image by using a model, a score file, and other model dependencies.
- The Docker image is created and stored in Azure Container Registry.
- The web service is deployed to the compute target (Container Instances/AKS) using the image created in the previous step.
- Scoring request details are stored in Application Insights, which is in the user's subscription.
- Telemetry is also pushed to the Microsoft/Azure subscription.



For an example of deploying a model as a web service, see [Deploy an image classification model in Azure Container Instances](#).

Real-time endpoints

When you deploy a trained model in the designer, you can [deploy the model as a real-time endpoint](#). A real-time endpoint commonly receives a single request via the REST endpoint and returns a prediction in real-time. This is in contrast to batch processing, which processes multiple values at once and saves the results after completion to a datastore.

Pipeline endpoints

Pipeline endpoints let you call your [ML Pipelines](#) programmatically via a REST endpoint. Pipeline endpoints let you automate your pipeline workflows.

A pipeline endpoint is a collection of published pipelines. This logical organization lets you manage and call multiple pipelines using the same endpoint. Each published pipeline in a pipeline endpoint is versioned. You can select a default pipeline for the endpoint, or specify a version in the REST call.

IoT module endpoints

A deployed IoT module endpoint is a Docker container that includes your model and associated script or application and any additional dependencies. You deploy these modules by using Azure IoT Edge on edge devices.

If you've enabled monitoring, Azure collects telemetry data from the model inside the Azure IoT Edge module. The telemetry data is accessible only to you, and it's stored in your storage account instance.

Azure IoT Edge ensures that your module is running, and it monitors the device that's hosting it.

Automation

Azure Machine Learning CLI

The [Azure Machine Learning CLI](#) is an extension to the Azure CLI, a cross-platform command-line interface for the Azure platform. This extension provides commands to automate your machine learning activities.

ML Pipelines

You use [machine learning pipelines](#) to create and manage workflows that stitch together machine learning phases. For example, a pipeline might include data preparation, model training, model deployment, and inference/scoring phases. Each phase can encompass multiple steps, each of which can run unattended in various compute targets.

Pipeline steps are reusable, and can be run without rerunning the previous steps if the output of those steps hasn't changed. For example, you can retrain a model without rerunning costly data preparation steps if the data hasn't changed. Pipelines also allow data scientists to collaborate while working on separate areas of a machine learning workflow.

Monitoring and logging

Azure Machine Learning provides the following monitoring and logging capabilities:

- For **Data Scientists**, you can monitor your experiments and log information from your training runs. For more information, see the following articles:
 - [Start, monitor, and cancel training runs](#)
 - [Log metrics for training runs](#)
 - [Track experiments with MLflow](#)
 - [Visualize runs with TensorBoard](#)
- For **Administrators**, you can monitor information about the workspace, related Azure resources, and events such as resource creation and deletion by using Azure Monitor. For more information, see [How to monitor Azure Machine Learning](#).
- For **DevOps** or **MLOps**, you can monitor information generated by models deployed as web services or IoT Edge modules to identify problems with the deployments and gather data submitted to the service. For more information, see [Collect model data](#) and [Monitor with Application Insights](#).

Interacting with your workspace

Studio

[Azure Machine Learning studio](#) provides a web view of all the artifacts in your workspace. You can view results and details of your datasets, experiments, pipelines, models, and endpoints. You can also manage compute resources and datastores in the studio.

The studio is also where you access the interactive tools that are part of Azure Machine Learning:

- [Azure Machine Learning designer](#) to perform workflow steps without writing code
- Web experience for [automated machine learning](#)
- [Azure Machine Learning notebooks](#) to write and run your own code in integrated Jupyter notebook servers.
- [Data labeling projects](#) to create, manage, and monitor projects to label your data

Programming tools

IMPORTANT

Tools 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](#).

- Interact with the service in any Python environment with the [Azure Machine Learning SDK for Python](#).
- Interact with the service in any R environment with the [Azure Machine Learning SDK for R](#) (preview).
- Use [Azure Machine Learning designer](#) to perform the workflow steps without writing code.
- Use [Azure Machine Learning CLI](#) for automation.
- The [Many Models Solution Accelerator](#) (preview) builds on Azure Machine Learning and enables you to train, operate, and manage hundreds or even thousands of machine learning models.

Next steps

To get started with Azure Machine Learning, see:

- [What is Azure Machine Learning?](#)
- [Create an Azure Machine Learning workspace](#)
- [Tutorial \(part 1\): Train a model](#)

Tutorial: Get started with Azure Machine Learning in your development environment (part 1 of 4)

12/23/2020 • 4 minutes to read • [Edit Online](#)

In this *four-part tutorial series*, you'll learn the fundamentals of Azure Machine Learning and complete jobs-based Python machine learning tasks on the Azure cloud platform.

In part 1 of this tutorial series, you will:

- Install the Azure Machine Learning SDK.
- Set up the directory structure for code.
- Create an Azure Machine Learning workspace.
- Configure your local development environment.
- Set up a compute cluster.

NOTE

This tutorial series focuses the Azure Machine Learning concepts suited to Python *jobs-based* machine learning tasks that are compute-intensive and/or require reproducibility. If you are more interested in an exploratory workflow, you could instead use [Jupyter](#) or [RStudio](#) on an [Azure Machine Learning compute instance](#).

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try [Azure Machine Learning](#).
- Familiarity with Python and [Machine Learning concepts](#). Examples include environments, training, and scoring.
- Local development environment, such as Visual Studio Code, Jupyter, or PyCharm.
- Python (version 3.5 to 3.7).

Install the Azure Machine Learning SDK

Throughout this tutorial, we make use of the Azure Machine Learning SDK for Python.

You can use the tools most familiar to you (for example, Conda and pip) to set up a Python environment to use throughout this tutorial. Install into your Python environment the Azure Machine Learning SDK for Python via pip:

```
pip install azureml-sdk
```

[I ran into an issue](#)

Create a directory structure for code

We recommend that you set up the following simple directory structure for this tutorial:

```
tutorial  
└─.azureml
```

- `tutorial` : Top-level directory of the project.
- `.azureml` : Hidden subdirectory for storing Azure Machine Learning configuration files.

TIP

If you're on a Mac, in a Finder window use **Command + Shift + .** to toggle the ability to see and create directories that begin with a dot. Or use the command terminal to create the directory.

Learn into an issue

Create an Azure Machine Learning workspace

A workspace is a top-level resource for Azure Machine Learning and is a centralized place to:

- Manage resources such as compute.
- Store assets like notebooks, environments, datasets, pipelines, models, and endpoints.
- Collaborate with other team members.

In the top-level directory, `tutorial`, add a new Python file called `01-create-workspace.py` by using the following code. Adapt the parameters (name, subscription ID, resource group, and [location](#)) with your preferences.

You can run the code in an interactive session or as a Python file.

NOTE

When you're using a local development environment (for example, your computer), you'll be asked to authenticate to your workspace by using a *device code* the first time you run the following code. Follow the on-screen instructions.

```
# tutorial/01-create-workspace.py
from azureml.core import Workspace

ws = Workspace.create(name='<my_workspace_name>', # provide a name for your workspace
                      subscription_id='<azure-subscription-id>', # provide your subscription ID
                      resource_group='<myresourcegroup>', # provide a resource group name
                      create_resource_group=True,
                      location='<NAME_OF_REGION>') # For example: 'westeurope' or 'eastus2' or 'westus2' or
                      'southeastasia'.

# write out the workspace details to a configuration file: .azureml/config.json
ws.write_config(path='.azureml')
```

Run this code from the `tutorial` directory:

```
cd <path/to/tutorial>
python ./01-create-workspace.py
```

TIP

If running this code gives you an error that you do not have access to the subscription, see [Create a workspace](#) for information on authentication options.

After you've successfully run `01-create-workspace.py`, your folder structure will look like:

```
tutorial
└─.azureml
   └─config.json
   └─01-create-workspace.py
```

The file `.azureml/config.json` contains the metadata necessary to connect to your Azure Machine Learning workspace. Namely, it contains your subscription ID, resource group, and workspace name.

NOTE

The contents of `config.json` are not secrets. It's fine to share these details.

Authentication is still required to interact with your Azure Machine Learning workspace.

[I ran into an issue](#)

Create an Azure Machine Learning compute cluster

Create a Python script in the `tutorial` top-level directory called `02-create-compute.py`. Populate it with the following code to create an Azure Machine Learning compute cluster that will autoscale between zero and four nodes:

```
# tutorial/02-create-compute.py
from azureml.core import Workspace
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

ws = Workspace.from_config() # This automatically looks for a directory .azureml

# Choose a name for your CPU cluster
cpu_cluster_name = "cpu-cluster"

# Verify that the cluster does not exist already
try:
    cpu_cluster = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print('Found existing cluster, use it.')
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                           idle_seconds_before_scaledown=2400,
                                                           min_nodes=0,
                                                           max_nodes=4)
    cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

cpu_cluster.wait_for_completion(show_output=True)
```

Run the Python file:

```
python ./02-create-compute.py
```

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).

Your folder structure will now look as follows:

```
tutorial
└─.azureml
  └─config.json
  └─01-create-workspace.py
  └─02-create-compute.py
```

[I ran into an issue](#)

Next steps

In this setup tutorial, you have:

- Created an Azure Machine Learning workspace.
- Set up your local development environment.
- Created an Azure Machine Learning compute cluster.

In the other parts of this tutorial you will learn:

- Part 2. Run code in the cloud by using the Azure Machine Learning SDK for Python.
- Part 3. Manage the Python environment that you use for model training.
- Part 4. Upload data to Azure and consume that data in training.

Continue to the next tutorial, to walk through submitting a script to the Azure Machine Learning compute cluster.

[Tutorial: Run a "Hello world!" Python script on Azure](#)

Tutorial: Run a "Hello world!" Python script (part 2 of 4)

12/23/2020 • 5 minutes to read • [Edit Online](#)

In this tutorial, you learn how to use the Azure Machine Learning SDK for Python to submit and run a Python "Hello world!" script.

This tutorial is *part 2 of a four-part tutorial series* in which you learn the fundamentals of Azure Machine Learning and complete jobs-based machine learning tasks in Azure. This tutorial builds on the work you completed in [Part 1: Set up your local machine for Azure Machine Learning](#).

In this tutorial, you will:

- Create and run a "Hello world!" Python script locally.
- 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

- Completion of [part 1](#) if you don't already have an Azure Machine Learning workspace.
- Introductory knowledge of the Python language and machine learning workflows.
- Local development environment, such as Visual Studio Code, Jupyter, or PyCharm.
- Python (version 3.5 to 3.7).

Create and run a Python script locally

Create a new subdirectory called `src` under the `tutorial` directory to store code that you want to run on an Azure Machine Learning compute cluster. In the `src` subdirectory, create the `hello.py` Python script:

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

Your project directory structure will now look like:

```
tutorial
└── .azureml
    └── config.json
└── src
    └── hello.py
    └── 01-create-workspace.py
    └── 02-create-compute.py
```

Test your script locally

You can run your code locally, by using your favorite IDE or a terminal. Running code locally has the benefit of interactive debugging of code.

```
cd <path/to/tutorial>
python ./src/hello.py
```

[Report an issue](#)

Create a control script

A *control script* allows you to run your `hello.py` script in the cloud. You use the control script to control how and where your machine learning code is run.

In your tutorial directory, create a new Python file called `03-run-hello.py` and copy/paste the following code into that file:

```
# tutorial/03-run-hello.py
from azureml.core import Workspace, Experiment, Environment, ScriptRunConfig

ws = Workspace.from_config()
experiment = Experiment(workspace=ws, name='day1-experiment-hello')

config = ScriptRunConfig(source_directory='./src', script='hello.py', compute_target='cpu-cluster')

run = experiment.submit(config)
aml_url = run.get_portal_url()
print(aml_url)
```

Understand the code

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

```
ws = Workspace.from_config()
```

`Workspace` connects to your Azure Machine Learning workspace, so that you can communicate with your Azure Machine Learning resources.

```
experiment = Experiment(...)
```

`Experiment` provides a simple way to organize multiple runs under a single name. Later you can see how experiments make it easy to compare metrics between dozens of runs.

```
config = ScriptRunConfig(...)
```

`ScriptRunConfig` wraps your `hello.py` code and passes it to your workspace. As the name suggests, you can use this class to *configure* how you want your *script* to *run* in Azure Machine Learning. It also specifies what compute target the script will run on. In this code, the target is the compute cluster that you created in the [setup tutorial](#).

```
run = experiment.submit(config)
```

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

```
aml_url = run.get_portal_url()
```

The `run` 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.

[Report an issue](#)

Submit and run your code in the cloud

Run your control script, which in turn runs `hello.py` on the compute cluster that you created in the [setup tutorial](#).

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 runs 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 run has completed.

```
python 03-run-hello.py
```

TIP

If running this code gives you an error that you do not have access to the subscription, see [Connect to a workspace](#) for information on authentication options.

[I ran into an issue](#)

Monitor your code in the cloud by using the studio

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

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

Follow the link and go to the **Outputs + logs** tab. There you can see a `70_driver_log.txt` file that looks like this:

```
1: [2020-08-04T22:15:44.407305] Entering context manager injector.
2: [context_manager_injector.py] Command line Options: Namespace(inject=['ProjectPythonPath:context_managers.ProjectPythonPath', 'RunHistory:context_managers.RunHistory', 'TrackUserError:context_managers.TrackUserError', 'UserExceptions:context_managers.UserExceptions'], invocation=['hello.py'])
3: Starting the daemon thread to refresh tokens in background for process with pid = 31263
4: Entering Run History Context Manager.
5: Preparing to call script [ hello.py ] with arguments: []
6: After variable expansion, calling script [ hello.py ] with arguments: []
7:
8: Hello world!
9: Starting the daemon thread to refresh tokens in background for process with pid = 31263
10:
11:
12: The experiment completed successfully. Finalizing run...
13: Logging experiment finalizing status in history service.
14: [2020-08-04T22:15:46.541334] TimeoutHandler __init__
15: [2020-08-04T22:15:46.541396] TimeoutHandler __enter__
16: Cleaning up all outstanding Run operations, waiting 300.0 seconds
17: 1 items cleaning up...
18: Cleanup took 0.1812913417816162 seconds
19: [2020-08-04T22:15:47.040203] TimeoutHandler __exit__
```

On line 8, you see the "Hello world!" output.

The `70_driver_log.txt` file contains the standard output from a run. This file can be useful when you're debugging remote runs in the cloud.

[I ran into an issue](#)

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 3 of 4)

12/23/2020 • 8 minutes to read • [Edit Online](#)

This tutorial shows you how to train a machine learning model in Azure Machine Learning.

This tutorial is *part 3 of a four-part tutorial series* in which you learn the fundamentals of Azure Machine Learning and complete jobs-based machine learning tasks in Azure. This tutorial builds on the work that you completed in [Part 1: Set up](#) and [Part 2: Run "Hello world!"](#) of the series.

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 2](#) of the series.
- Introductory knowledge of the Python language and machine learning workflows.
- Local development environment, such as Visual Studio Code, Jupyter, or PyCharm.
- Python (version 3.5 to 3.7).

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 following 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.

```
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
```

Next you define the training script. 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` subdirectory:

```

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")

```

You now have the following directory structure:

```
tutorial
└─.azureml
|   └─config.json
└─src
|   └─hello.py
|   └─model.py
|   └─train.py
└─01-create-workspace.py
└─02-create-compute.py
└─03-run-hello.py
```

[I ran into an issue](#)

Create a Python environment

For demonstration purposes, we're going to use a Conda environment. (The steps for a pip virtual environment are almost identical.)

Create a file called `pytorch-env.yml` in the `.azureml` hidden directory:

```
name: pytorch-env
channels:
- defaults
- pytorch
dependencies:
- python=3.6.2
- pytorch
- torchvision
```

This environment has all the dependencies that your model and training script require. Notice there's no dependency on the Azure Machine Learning SDK for Python.

[I ran into an issue](#)

Test locally

Use the following code to test your script runs locally in this environment:

```
conda env create -f .azureml/pytorch-env.yml      # create conda environment
conda activate pytorch-env                          # activate conda environment
python src/train.py                                # train model
```

After you run this script, you'll see the data downloaded into a directory called `tutorial/data`.

[I ran into an issue](#)

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 `tutorial` directory called `04-run-pytorch.py`:

```

# 04-run-pytorch.py
from azureml.core import Workspace
from azureml.core import Experiment
from azureml.core import Environment
from azureml.core import ScriptRunConfig

if __name__ == "__main__":
    ws = Workspace.from_config()
    experiment = Experiment(workspace=ws, name='day1-experiment-train')
    config = ScriptRunConfig(source_directory='./src',
                            script='train.py',
                            compute_target='cpu-cluster')

    # set up pytorch environment
    env = Environment.from_conda_specification(
        name='pytorch-env',
        file_path='./azureml/pytorch-env.yml'
    )
    config.run_config.environment = env

    run = experiment.submit(config)

    aml_url = run.get_portal_url()
    print(aml_url)

```

Understand the code changes

`env = ...`

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.

`config.run_config.environment = env`

Adds the environment to [ScriptRunConfig](#).

[I ran into an issue](#)

Submit the run to Azure Machine Learning

If you switched local environments, be sure to switch back to an environment that has the Azure Machine Learning SDK for Python installed.

Then run:

`python 04-run-pytorch.py`

NOTE

The first time you run this script, Azure Machine Learning will build a new Docker image from your PyTorch environment. The whole run might take 5 to 10 minutes to complete.

You can see the Docker build logs in the Azure Machine Learning studio. Follow the link to the studio, select the **Outputs + logs** tab, and then select `20_image_build_log.txt`.

This image will be reused in future runs to make them run much quicker.

After your image is built, select `70_driver_log.txt` to see the output of your training script.

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
...
Files already downloaded and verified
epoch=1, batch= 2000: loss 2.19
epoch=1, batch= 4000: loss 1.82
epoch=1, batch= 6000: loss 1.66
epoch=1, batch= 8000: loss 1.58
epoch=1, batch=10000: loss 1.52
epoch=1, batch=12000: loss 1.47
epoch=2, batch= 2000: loss 1.39
epoch=2, batch= 4000: loss 1.38
epoch=2, batch= 6000: loss 1.37
epoch=2, batch= 8000: loss 1.33
epoch=2, batch=10000: loss 1.31
epoch=2, batch=12000: loss 1.27
Finished Training
```

WARNING

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

Move `data` outside `src`.

Environments can be registered to a workspace with `env.register(ws)`. They can then be easily shared, reused, and versioned. Environments make it easy to reproduce previous results and to collaborate with your team.

Azure Machine Learning also maintains a collection of curated environments. These environments cover common machine learning scenarios and are backed by cached Docker images. Cached Docker images make the first remote run faster.

In short, using registered environments can save you time! Read [How to use environments](#) for more information.

[I ran into an issue](#)

Log training metrics

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

The current training script prints metrics to the terminal. Azure Machine Learning provides a mechanism for logging metrics with more functionality. By adding a few lines of code, you gain the ability to visualize metrics in the studio and to compare metrics between multiple runs.

Modify `train.py` to include logging

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
from azureml.core import Run

# ADDITIONAL CODE: get AML run from the current context
run = Run.get_context()

# 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
                run.log('loss', loss)
                print(f'epoch={epoch + 1}, batch={i + 1:5}: loss {loss:.2f}')
                running_loss = 0.0

    print('Finished Training')

```

Understand the additional two lines of code

In `train.py`, you access the `run` object from *within* the training script itself by using the `Run.get_context()` method and use it to log metrics:

```
# in train.py
run = Run.get_context()

...
run.log('loss', loss)
```

Metrics in Azure Machine Learning are:

- Organized by experiment and run, 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.

[I ran into an issue](#)

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-aml-env
channels:
  - defaults
  - pytorch
dependencies:
  - python=3.6.2
  - pytorch
  - torchvision
  - pip
  - pip:
    - azureml-sdk
```

[I ran into an issue](#)

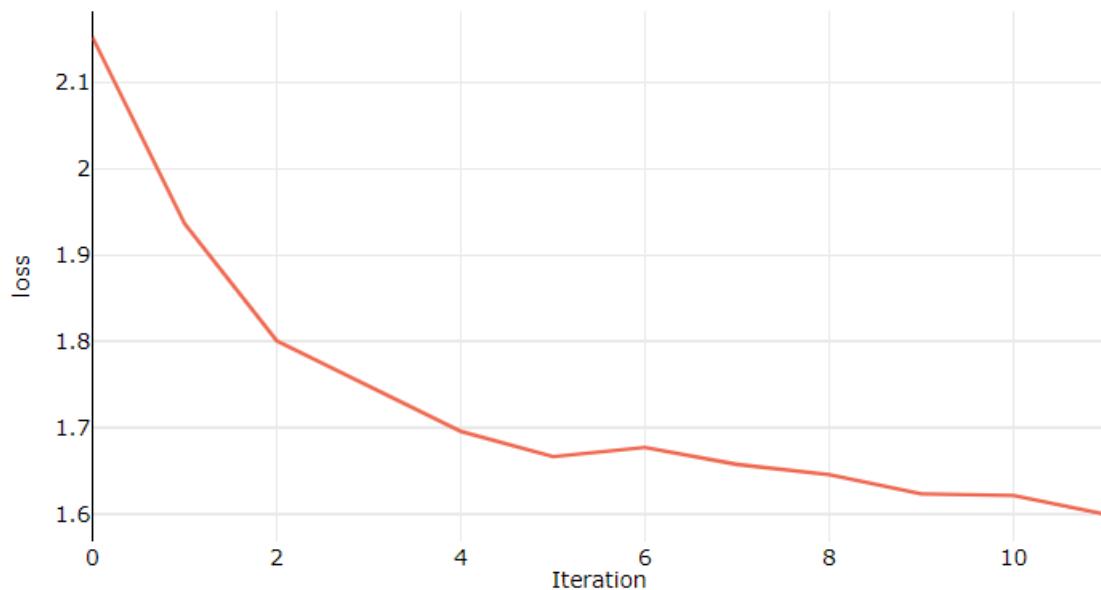
Submit the run to Azure Machine Learning

Submit this script once more:

```
python 04-run-pytorch.py
```

This time when you visit the studio, go to the **Metrics** tab where you can now see live updates on the model training loss!

loss



I ran into an issue

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 take a local Conda environment to the cloud with 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: Use your own data (part 4 of 4)

12/23/2020 • 7 minutes to read • [Edit Online](#)

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 4 of a four-part tutorial series* in which you learn the fundamentals of Azure Machine Learning and complete jobs-based machine learning tasks in Azure. This tutorial builds on the work you completed in [Part 1: Set up](#), [Part 2: Run "Hello World!"](#), and [Part 3: Train a model](#).

In [Part 3: Train a model](#), data was downloaded through the inbuilt `torchvision.datasets.CIFAR10` method in the PyTorch API. However, in many cases you'll want to use your own data in a remote training run. This article shows the workflow that you can use to work with your own data in Azure Machine Learning.

In this tutorial, you:

- Configure a training script to use data in a local directory.
- Test the training script locally.
- Upload data to Azure.
- Create a control script.
- Understand the new Azure Machine Learning concepts (passing parameters, datasets, datastores).
- Submit and run your training script.
- View your code output in the cloud.

Prerequisites

- Completion of [part 3](#) of the series.
- Introductory knowledge of the Python language and machine learning workflows.
- Local development environment, such as Visual Studio Code, Jupyter, or PyCharm.
- Python (version 3.5 to 3.7).

Adjust the training script

By now you have your training script (`tutorial/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 now 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.

```
import os
import argparse
import torch
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

```
from model import Net
from azureml.core import Run

run = Run.get_context()

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
            run.log('loss', loss) # log loss metric to AML
            print(f'epoch={epoch + 1}, batch={i + 1:5}: loss {loss:.2f}')
            running_loss = 0.0

    print('Finished Training')

```

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
)

```

[I ran into an issue](#)

Test the script locally

Your script now accepts `data path` as an argument. To start with, test it locally. Add to your tutorial directory structure a folder called `data`. Your directory structure should look like:

```

tutorial
└── azureml
    ├── config.json
    └── pytorch-env.yml
└── data
└── src
    ├── hello.py
    ├── model.py
    └── train.py
└── 01-create-workspace.py
└── 02-create-compute.py
└── 03-run-hello.py
└── 04-run-pytorch.py

```

If you didn't run `train.py` locally in the previous tutorial, you won't have the `data/` directory. In this case, run the `torchvision.datasets.CIFAR10` method locally with `download=True` in your `train.py` script.

To run the modified training script locally, call:

```
python src/train.py --data_path ./data --learning_rate 0.003 --momentum 0.92
```

You avoid having to download the CIFAR10 dataset by passing in a local path to the data. You can also experiment with different values for *learning rate* and *momentum* hyperparameters without having to hard-code them in the training script.

[I ran into an issue](#)

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 datastores that store your data. For more details, see the [datastores documentation](#).

Create a new Python control script called `05-upload-data.py` in the `tutorial` directory:

```
# 05-upload-data.py
from azureml.core import Workspace
ws = Workspace.from_config()
datastore = ws.get_default_datastore()
datastore.upload(src_dir='./data',
                 target_path='datasets/cifar10',
                 overwrite=True)
```

The `target_path` value specifies the path on the datastore where the CIFAR10 data will be uploaded.

TIP

While you're using Azure Machine Learning to upload the data, you can use [Azure Storage Explorer](#) to upload ad hoc files. If you need an ETL tool, you can use [Azure Data Factory](#) to ingest your data into Azure.

Run the Python file to upload the data. (The upload should be quick, less than 60 seconds.)

```
python 05-upload-data.py
```

You should see the following standard output:

```
Uploading ./data\cifar-10-batches-py\data_batch_2
Uploaded ./data\cifar-10-batches-py\data_batch_2, 4 files out of an estimated total of 9
.
.
Uploading ./data\cifar-10-batches-py\data_batch_5
Uploaded ./data\cifar-10-batches-py\data_batch_5, 9 files out of an estimated total of 9
Uploaded 9 files
```

[I ran into an issue](#)

Create a control script

As you've done previously, create a new Python control script called `06-run-pytorch-data.py`:

```
# 06-run-pytorch-data.py
from azureml.core import Workspace
from azureml.core import Experiment
from azureml.core import Environment
from azureml.core import ScriptRunConfig
from azureml.core import Dataset

if __name__ == "__main__":
    ws = Workspace.from_config()
    datastore = ws.get_default_datastore()
    dataset = Dataset.File.from_files(path=(datastore, 'datasets/cifar10'))

    experiment = Experiment(workspace=ws, name='day1-experiment-data')

    config = ScriptRunConfig(
        source_directory='./src',
        script='train.py',
        compute_target='cpu-cluster',
        arguments=[
            '--data_path', dataset.as_named_input('input').as_mount(),
            '--learning_rate', 0.003,
            '--momentum', 0.92],
    )
    # set up pytorch environment
    env = Environment.from_conda_specification(
        name='pytorch-env',
        file_path='./azureml/pytorch-env.yml'
    )
    config.run_config.environment = env

    run = experiment.submit(config)
    aml_url = run.get_portal_url()
    print("Submitted to compute cluster. Click link below")
    print("")
    print(aml_url)
```

Understand the code changes

The control script is similar to the one from [part 3 of this series](#), with the following new lines:

```
dataset = Dataset.File.from_files( ... )
```

A `dataset` is used to reference the data you uploaded to Azure Blob Storage. Datasets are an abstraction layer on top of your data that are designed to improve reliability and trustworthiness.

```
config = ScriptRunConfig(...)
```

`ScriptRunConfig` is modified to include a list of arguments that will be passed into `train.py`. The `dataset.as_named_input('input').as_mount()` argument means the specified directory will be *mounted* to the

compute target.

[I ran into an issue](#)

Submit the run to Azure Machine Learning

Now resubmit the run to use the new configuration:

```
python 06-run-pytorch-data.py
```

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.

[I ran into an issue](#)

Inspect the log file

In the studio, go to the experiment run (by selecting the previous URL output) followed by **Outputs + logs**. Select the `70_driver_log.txt` file. You should see the following output:

```
Processing 'input'.
Processing dataset FileDataset
{
    "source": [
        "('workspaceblobstore', 'datasets/cifar10')"
    ],
    "definition": [
        "GetDatastoreFiles"
    ],
    "registration": {
        "id": "XXXXX",
        "name": null,
        "version": null,
        "workspace": "Workspace.create(name='XXXX', subscription_id='XXXX', resource_group='X')"
    }
}
Mounting input to /tmp/tmp9kituvp3.
Mounted input to /tmp/tmp9kituvp3 as folder.
Exit __enter__ of DatasetContextManager
Entering Run History Context Manager.
Current directory: /mnt/batch/tasks/shared/LS_root/jobs/dsvm-aml/azureml/tutorial-session-3_1600171983_763c5381/mounts/workspaceblobstore/azureml/tutorial-session-3_1600171983_763c5381
Preparing to call script [ train.py ] with arguments: ['--data_path', '$input', '--learning_rate', '0.003', '--momentum', '0.92']
After variable expansion, calling script [ train.py ] with arguments: ['--data_path', '/tmp/tmp9kituvp3', '--learning_rate', '0.003', '--momentum', '0.92']

Script type = None
===== DATA =====
DATA PATH: /tmp/tmp9kituvp3
LIST FILES IN DATA PATH...
['cifar-10-batches-py', 'cifar-10-python.tar.gz']
```

Notice:

- Azure Machine Learning has mounted Blob Storage to the compute cluster automatically for you.
- The `dataset.as_named_input('input').as_mount()` used in the control script resolves to the mount point.

[I ran into an issue](#)

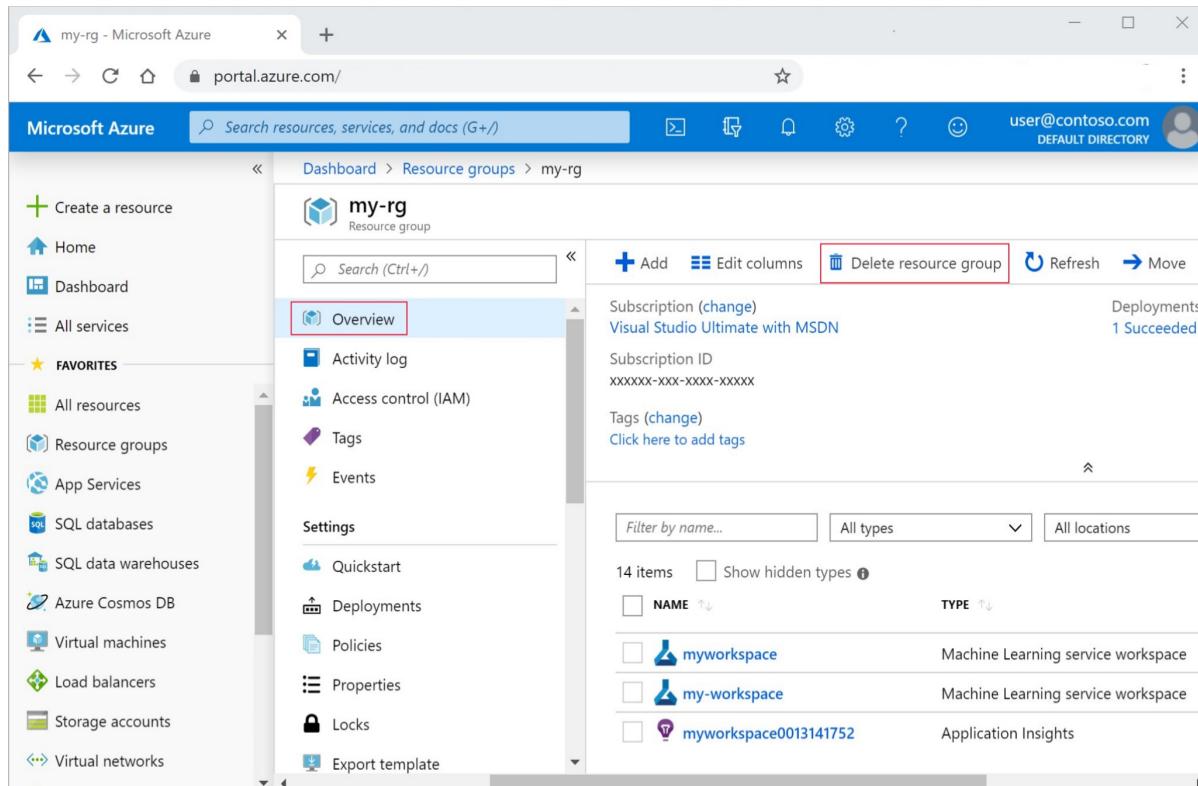
Clean up resources

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 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. The left sidebar has 'Resource groups' selected under 'All services'. The main content area shows the 'my-rg' resource group details. At the top right of the content area, there is a red box around the 'Delete resource group' button. Below it, the 'Overview' tab is selected. The main content area displays subscription information, deployment status, and a list of resources. The resource list table has columns for NAME and TYPE, showing three items: 'myworkspace' (Machine Learning service workspace), 'my-workspace' (Machine Learning service workspace), and 'myworkspace0013141752' (Application Insights).

NAME	TYPE
myworkspace	Machine Learning service workspace
my-workspace	Machine Learning service workspace
myworkspace0013141752	Application Insights

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 run.

Now that you have a model, learn:

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

Tutorial: Get started with Azure Machine Learning in Jupyter Notebooks

12/23/2020 • 5 minutes to read • [Edit Online](#)

In this tutorial, you complete the steps to get started with Azure Machine Learning by using Jupyter Notebooks on a [managed cloud-based workstation \(compute instance\)](#). This tutorial is a precursor to all other Jupyter Notebook tutorials.

In this tutorial, you:

- Create an [Azure Machine Learning workspace](#) to use in other Jupyter Notebook tutorials.
- Clone the tutorials notebook to your folder in the workspace.
- Create a cloud-based compute instance with the Azure Machine Learning Python SDK installed and preconfigured.

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.

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.

Skip to [Clone a notebook folder](#) if you already have an Azure Machine Learning workspace.

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 **+ Create a resource**.

The screenshot shows the Microsoft Azure portal interface. At the top, there are three tabs: 'Home - Microsoft Azure', 'Azure ML Workspace (Preview)', and 'Sign out'. Below the tabs, the URL is 'portal.azure.com/#home'. The main header includes a search bar with 'Search resources, services, and docs (G+)' and various navigation icons. The user's email, 'sheril.gilley@gmail.com', is displayed with 'DEFAULT DIRECTORY'.

Azure services

- Create a resource
- Virtual machines
- App Services
- Storage accounts
- SQL databases
- Azure Database for PostgreSQL
- Azure Cosmos DB
- Kubernetes services
- More services

Recent resources

NAME	TYPE	LAST VIEWED
Visual Studio Ultimate with MSDN	Subscription	30 min ago

Navigate

- Subscriptions
- Resource groups
- All resources
- Dashboard

Tools

- Microsoft Learn
- Azure Monitor
- Security Center
- Cost Management

3. Use the search bar to find **Machine Learning**.
4. Select **Machine Learning**.
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 docs-ws . Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others.
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 .
Location	Select the location closest to your users and the data resources to create your workspace.
Workspace edition	Select Basic as the workspace type for this tutorial. The workspace type determines the features to which you'll have access and pricing. Everything in this tutorial can be performed with either a Basic or Enterprise workspace.

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

WARNING

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

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

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

IMPORTANT

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

Run a notebook in your workspace

Azure Machine Learning includes a cloud notebook server in your workspace for an install-free and preconfigured experience. Use [your own environment](#) if you prefer to have control over your environment, packages, and dependencies.

Follow along with this video or use the detailed steps to clone and run the tutorial notebook from your workspace.

Clone a notebook folder

You 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.

1. Sign in to [Azure Machine Learning studio](#).
2. Select your subscription and the workspace you created.
3. On the left, select **Notebooks**.
4. At the top, select the **Samples** tab.
5. Open the **Python** folder.
6. Open the folder with a version number on it. This number represents the current release for the Python SDK.
7. Select the ... button at the right of the **tutorials** folder, and then select **Clone**.

The screenshot shows the Microsoft Azure Machine Learning interface. The top navigation bar includes 'Preview' and 'Microsoft Azure Machine Learning'. Below the navigation bar is a sidebar with various options: 'New', 'Home', 'Author', 'Notebooks' (which is highlighted with a red box), 'Automated ML', 'Designer', 'Assets', 'Datasets', 'Experiments', 'Pipelines', 'Models', 'Endpoints', 'Manage', 'Compute', and 'Datastores'. The main content area shows a 'Notebooks' list under 'my-ws > Notebooks'. The 'Sample notebooks' tab is selected (highlighted with a red box). A search bar says 'Search to filter notebooks'. Below the search bar is a 'Samples' folder, which contains a subfolder '[SDK version number]' that further contains 'how-to-use-azureml' and 'tutorials'. To the right of the 'tutorials' folder is a context menu with options: '...', 'Clone' (highlighted with a red box), and 'Copy file path'.

8. A list of folders shows each user who accesses the workspace. Select your folder to clone the **tutorials** folder there.

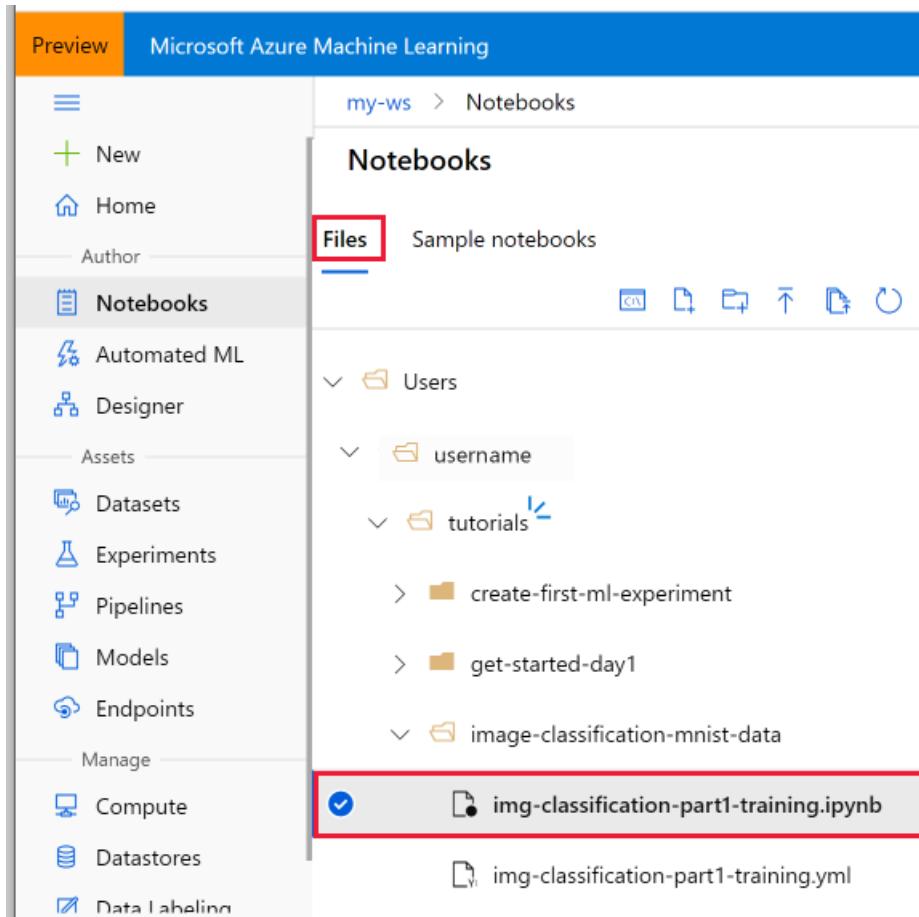
Open the cloned notebook

1. Open the **tutorials** folder that was closed into your **User files** section.

IMPORTANT

You can view notebooks in the **samples** folder but you can't run a notebook from there. To run a notebook, make sure you open the cloned version of the notebook in the **User Files** section.

2. Select the **img-classification-part1-training.ipynb** file in your **tutorials/image-classification-mnist-data** folder.



3. On the top bar, select a compute instance to use to run the notebook. These virtual machines (VMs) are preconfigured with [everything you need to run Azure Machine Learning](#).
4. If no VMs are found, select **+ Add** to create the compute instance VM.
 - a. When you create a VM, follow these rules:
 - A name is required, and the field can't be empty.
 - The name must be unique (in a case-insensitive fashion) across all existing compute instances in the Azure region of the workspace or compute instance. You'll get an alert if the name you choose isn't unique.
 - Valid characters are uppercase and lowercase letters, numbers 0 to 9, and the dash character (-).
 - The name must be between 3 and 24 characters long.
 - The name should start with a letter, not a number or a dash character.
 - If a dash character is used, it must be followed by at least one letter after the dash. For example, Test-, test-0, test-01 are invalid, while test-a0, test-0a are valid instances.
 - b. Select the VM size from the available choices. For the tutorials, the default VM is a good choice.
 - c. Then select **Create**. It can take approximately five minutes to set up your VM.
5. When the VM is available, it appears in the top toolbar. You can now run the notebook by using either **Run all** in the toolbar or **Shift+Enter** in the code cells of the notebook.

If you have custom widgets or prefer to use Jupyter or JupyterLab, select the **Jupyter** drop-down list on the far right. Then select **Jupyter** or **JupyterLab**. The new browser window opens.

Next steps

Now that you have a development environment set up, continue on to train a model in a Jupyter Notebook.

[Tutorial: Train image classification models with MNIST data and scikit-learn](#)

If you don't plan on following any other tutorials now, stop the cloud notebook server VM when you aren't using it to reduce cost.

If you used a compute instance or Notebook VM, stop the VM when you aren't using it to reduce cost.

1. In your workspace, select **Compute**.
2. From the list, select the VM.
3. Select **Stop**.
4. When you're ready to use the server again, select **Start**.

Tutorial: Train image classification models with MNIST data and scikit-learn

12/23/2020 • 13 minutes to read • [Edit Online](#)

In this tutorial, you train a machine learning model on remote compute resources. You'll use the training and deployment workflow for Azure Machine Learning in a Python Jupyter Notebook. You can then use the notebook as a template to train your own machine learning model with your own data. This tutorial is **part one of a two-part tutorial series**.

This tutorial trains a simple logistic regression by using the [MNIST](#) dataset and [scikit-learn](#) with Azure Machine Learning. MNIST is a popular dataset consisting of 70,000 grayscale images. Each image is a handwritten digit of 28 x 28 pixels, representing a number from zero to nine. The goal is to create a multi-class classifier to identify the digit a given image represents.

Learn how to take the following actions:

- Set up your development environment.
- Access and examine the data.
- Train a simple logistic regression model on a remote cluster.
- Review training results and register the best model.

You learn how to select a model and deploy it in [part two of this tutorial](#).

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.

NOTE

Code in this article was tested with [Azure Machine Learning SDK](#) version 1.13.0.

Prerequisites

- Complete the [Tutorial: Get started creating your first Azure ML experiment](#) to:
 - Create a workspace
 - Clone the tutorials notebook to your folder in the workspace.
 - Create a cloud-based compute instance.
- In your cloned `tutorials/image-classification-mnist-data` folder, open the `img-classification-part1-training.ipynb` notebook.

The tutorial and accompanying `utils.py` file is also available on [GitHub](#) if you wish to use it on your own [local environment](#). Run `pip install azureml-sdk[notebooks] azureml-opendatasets matplotlib` to install dependencies for this tutorial.

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 read along as you run the code. 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.

Set up your development environment

All the setup for your development work can be accomplished in a Python notebook. Setup includes the following actions:

- Import Python packages.
- Connect to a workspace, so that your local computer can communicate with remote resources.
- Create an experiment to track all your runs.
- Create a remote compute target to use for training.

Import packages

Import Python packages you need in this session. Also display the Azure Machine Learning SDK version:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

import azureml.core
from azureml.core import Workspace

# check core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)
```

Connect to a workspace

Create a workspace object from the existing workspace. `Workspace.from_config()` reads the file `config.json` and loads the details into an object named `ws`:

```
# load workspace configuration from the config.json file in the current folder.
ws = Workspace.from_config()
print(ws.name, ws.location, ws.resource_group, sep='\t')
```

Create an experiment

Create an experiment to track the runs in your workspace. A workspace can have multiple experiments:

```
from azureml.core import Experiment
experiment_name = 'sklearn-mnist'

exp = Experiment(workspace=ws, name=experiment_name)
```

Create or attach an existing compute target

By using Azure Machine Learning Compute, a managed service, data scientists can train machine learning models on clusters of Azure virtual machines. Examples include VMs with GPU support. In this tutorial, you create Azure Machine Learning Compute as your training environment. You will submit Python code to run on this VM later in the tutorial.

The code below creates the compute clusters for you if they don't already exist in your workspace. It sets up a

cluster that will scale down to 0 when not in use, and can scale up to a maximum of 4 nodes.

Creation of the compute target takes about five minutes. If the compute resource is already in the workspace, the code uses it and skips the creation process.

```
from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
import os

# choose a name for your cluster
compute_name = os.environ.get("AML_COMPUTE_CLUSTER_NAME", "cpu-cluster")
compute_min_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MAX_NODES", 4)

# This example uses CPU VM. For using GPU VM, set SKU to STANDARD_NC6
vm_size = os.environ.get("AML_COMPUTE_CLUSTER_SKU", "STANDARD_D2_V2")

if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('found compute target. just use it. ' + compute_name)
    else:
        print('creating a new compute target...')
        provisioning_config = AmlCompute.provisioning_configuration(vm_size=vm_size,
                                                                    min_nodes=compute_min_nodes,
                                                                    max_nodes=compute_max_nodes)

    # create the cluster
    compute_target = ComputeTarget.create(
        ws, compute_name, provisioning_config)

    # can poll for a minimum number of nodes and for a specific timeout.
    # if no min node count is provided it will use the scale settings for the cluster
    compute_target.wait_for_completion(
        show_output=True, min_node_count=None, timeout_in_minutes=20)

    # For a more detailed view of current AmlCompute status, use get_status()
    print(compute_target.get_status().serialize())
```

You now have the necessary packages and compute resources to train a model in the cloud.

Explore data

Before you train a model, you need to understand the data that you use to train it. In this section you learn how to:

- Download the MNIST dataset.
- Display some sample images.

Download the MNIST dataset

Use Azure Open Datasets to get the raw MNIST data files. [Azure Open Datasets](#) are curated public datasets that you can use to add scenario-specific features to machine learning solutions for more accurate models. Each dataset has a corresponding class, `MNIST` in this case, to retrieve the data in different ways.

This code retrieves the data as a `FileDataset` object, which is a subclass of `Dataset`. A `FileDataset` references single or multiple files of any format in your datastores or public urls. The class provides you with the ability to download or mount the files to your compute by creating a reference to the data source location. Additionally, you register the Dataset to your workspace for easy retrieval during training.

Follow the [how-to](#) to learn more about Datasets and their usage in the SDK.

```

from azureml.core import Dataset
from azureml.opendatasets import MNIST

data_folder = os.path.join(os.getcwd(), 'data')
os.makedirs(data_folder, exist_ok=True)

mnist_file_dataset = MNIST.get_file_dataset()
mnist_file_dataset.download(data_folder, overwrite=True)

mnist_file_dataset = mnist_file_dataset.register(workspace=ws,
                                                 name='mnist_opendataset',
                                                 description='training and test dataset',
                                                 create_new_version=True)

```

Display some sample images

Load the compressed files into `numpy` arrays. Then use `matplotlib` to plot 30 random images from the dataset with their labels above them. This step requires a `load_data` function that's included in an `utils.py` file. This file is included in the sample folder. Make sure it's placed in the same folder as this notebook. The `load_data` function simply parses the compressed files into numpy arrays.

```

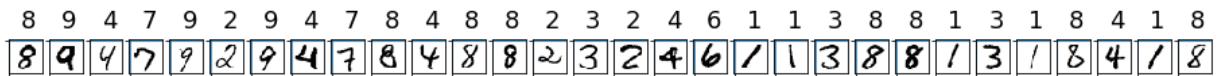
# make sure utils.py is in the same directory as this code
from utils import load_data
import glob

# note we also shrink the intensity values (X) from 0-255 to 0-1. This helps the model converge faster.
X_train = load_data(glob.glob(os.path.join(data_folder,"**/train-images-idx3-ubyte.gz")), recursive=True)[0], False) / 255.0
X_test = load_data(glob.glob(os.path.join(data_folder,"**/t10k-images-idx3-ubyte.gz")), recursive=True)[0], False) / 255.0
y_train = load_data(glob.glob(os.path.join(data_folder,"**/train-labels-idx1-ubyte.gz")), recursive=True)[0], True).reshape(-1)
y_test = load_data(glob.glob(os.path.join(data_folder,"**/t10k-labels-idx1-ubyte.gz")), recursive=True)[0], True).reshape(-1)

# now let's show some randomly chosen images from the training set.
count = 0
sample_size = 30
plt.figure(figsize=(16, 6))
for i in np.random.permutation(X_train.shape[0])[:sample_size]:
    count = count + 1
    plt.subplot(1, sample_size, count)
    plt.axhline('')
    plt.axvline('')
    plt.text(x=10, y=-10, s=y_train[i], fontsize=18)
    plt.imshow(X_train[i].reshape(28, 28), cmap=plt.cm.Greys)
plt.show()

```

A random sample of images displays:



Now you have an idea of what these images look like and the expected prediction outcome.

Train on a remote cluster

For this task, you submit the job to run on the remote training cluster you set up earlier. To submit a job you:

- Create a directory

- Create a training script
- Create a script run configuration
- Submit the job

Create a directory

Create a directory to deliver the necessary code from your computer to the remote resource.

```
import os
script_folder = os.path.join(os.getcwd(), "sklearn-mnist")
os.makedirs(script_folder, exist_ok=True)
```

Create a training script

To submit the job to the cluster, first create a training script. Run the following code to create the training script called `train.py` in the directory you just created.

```

%%writefile $script_folder/train.py

import argparse
import os
import numpy as np
import glob

from sklearn.linear_model import LogisticRegression
import joblib

from azureml.core import Run
from utils import load_data

# let user feed in 2 parameters, the dataset to mount or download, and the regularization rate of the
# logistic regression model
parser = argparse.ArgumentParser()
parser.add_argument('--data-folder', type=str, dest='data_folder', help='data folder mounting point')
parser.add_argument('--regularization', type=float, dest='reg', default=0.01, help='regularization rate')
args = parser.parse_args()

data_folder = args.data_folder
print('Data folder:', data_folder)

# load train and test set into numpy arrays
# note we scale the pixel intensity values to 0-1 (by dividing it with 255.0) so the model can converge
# faster.
X_train = load_data(glob.glob(os.path.join(data_folder, '**/train-images-idx3-ubyte.gz')), recursive=True)[0] / 255.0
X_test = load_data(glob.glob(os.path.join(data_folder, '**/t10k-images-idx3-ubyte.gz')), recursive=True)[0] / 255.0
y_train = load_data(glob.glob(os.path.join(data_folder, '**/train-labels-idx1-ubyte.gz')), recursive=True)[0], True).reshape(-1)
y_test = load_data(glob.glob(os.path.join(data_folder, '**/t10k-labels-idx1-ubyte.gz')), recursive=True)[0], True).reshape(-1)

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape, sep = '\n')

# get hold of the current run
run = Run.get_context()

print('Train a logistic regression model with regularization rate of', args.reg)
clf = LogisticRegression(C=1.0/args.reg, solver="liblinear", multi_class="auto", random_state=42)
clf.fit(X_train, y_train)

print('Predict the test set')
y_hat = clf.predict(X_test)

# calculate accuracy on the prediction
acc = np.average(y_hat == y_test)
print('Accuracy is', acc)

run.log('regularization rate', np.float(args.reg))
run.log('accuracy', np.float(acc))

os.makedirs('outputs', exist_ok=True)
# note file saved in the outputs folder is automatically uploaded into experiment record
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')

```

Notice how the script gets data and saves models:

- The training script reads an argument to find the directory that contains the data. When you submit the job later, you point to the datastore for this argument:


```
parser.add_argument('--data-folder', type=str, dest='data_folder', help='data directory mounting point')
```
- The training script saves your model into a directory named **outputs**. Anything written in this directory

is automatically uploaded into your workspace. You access your model from this directory later in the tutorial.

```
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')
```

- The training script requires the file `utils.py` to load the dataset correctly. The following code copies `utils.py` into `script_folder` so that the file can be accessed along with the training script on the remote resource.

```
import shutil  
shutil.copy('utils.py', script_folder)
```

Configure the training job

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. Configure the ScriptRunConfig by specifying:

- The directory that contains your scripts. All the files in this directory are uploaded into the cluster nodes for execution.
- The compute target. In this case, you use the Azure Machine Learning compute cluster you created.
- The training script name, `train.py`.
- An environment that contains the libraries needed to run the script.
- Arguments required from the training script.

In this tutorial, this target is `AmlCompute`. All files in the script folder are uploaded into the cluster nodes for run. The `--data_folder` is set to use the dataset.

First, create the environment that contains: the scikit-learn library, `azureml-dataset-runtime` required for accessing the dataset, and `azureml-defaults` which contains the dependencies for logging metrics. The `azureml-defaults` also contains the dependencies required for deploying the model as a web service later in the part 2 of the tutorial.

Once the environment is defined, register it with the Workspace to re-use it in part 2 of the tutorial.

```
from azureml.core.environment import Environment  
from azureml.core.conda_dependencies import CondaDependencies  
  
# to install required packages  
env = Environment('tutorial-env')  
cd = CondaDependencies.create(pip_packages=['azureml-dataset-runtime[pandas,fuse]', 'azureml-defaults'],  
    conda_packages=['scikit-learn==0.22.1'])  
  
env.python.conda_dependencies = cd  
  
# Register environment to re-use later  
env.register(workspace=ws)
```

Then, create the `ScriptRunConfig` by specifying the training script, compute target and environment.

```
from azureml.core import ScriptRunConfig  
  
args = ['--data-folder', mnist_file_dataset.as_mount(), '--regularization', 0.5]  
  
src = ScriptRunConfig(source_directory=script_folder,  
                      script='train.py',  
                      arguments=args,  
                      compute_target=compute_target,  
                      environment=env)
```

Submit the job to the cluster

Run the experiment by submitting the `ScriptRunConfig` object:

```
run = exp.submit(config=src)
run
```

Because the call is asynchronous, it returns a **Preparing** or **Running** state as soon as the job is started.

Monitor a remote run

In total, the first run takes **about 10 minutes**. But for subsequent runs, as long as the script dependencies don't change, the same image is reused. So the container startup time is much faster.

What happens while you wait:

- **Image creation:** A Docker image is created that matches the Python environment specified by the Azure ML environment. The image is uploaded to the workspace. Image creation and uploading takes **about five minutes**.
This stage happens once for each Python environment because the container is cached for subsequent runs. During image creation, logs are streamed to the run history. You can monitor the image creation progress by using these logs.
- **Scaling:** If the remote cluster requires more nodes to do the run than currently available, additional nodes are added automatically. Scaling typically takes **about five minutes**.
- **Running:** In this stage, the necessary scripts and files are sent to the compute target. Then datastores are mounted or copied. And then the `entry_script` is run. While the job is running, `stdout` and the `./logs` directory are streamed to the run history. You can monitor the run's progress by using these logs.
- **Post-processing:** The `./outputs` directory of the run is copied over to the run history in your workspace, so you can access these results.

You can check the progress of a running job in several ways. This tutorial uses a Jupyter widget and a `wait_for_completion` method.

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 to 15 seconds until the job finishes:

```
from azureml.widgets import RunDetails
RunDetails(run).show()
```

The widget will look like the following at the end of training:

Run Properties		Output Logs
Status	Completed	Uploading experiment status to history service. Adding run profile attachment azureml-logs/80_driver_log.txt
Start Time	8/10/2018 12:11:42 PM	Data folder: /mnt/batch/tasks/shared/LS_root/jobs/gpucluster225c81517743bf5/azureml/sklearn-mnist_1533921100384/mounts/workspacefilestore/mnist (60000, 784) (60000,) (10000, 784) (10000,)
Duration	0:07:20	Train a logistic regression model with regularization rate of 0.01 Predict the test set Accuracy is 0.9185
Run Id	sklearn-mnist_1533921100384	The experiment completed successfully. Starting post-processing steps.
Arguments	N/A	
regularization rate	0.01	
accuracy	0.9185	

Click here to see the run in Azure portal

If you need to cancel a run, you can follow [these instructions](#).

Get log results upon completion

Model training and monitoring happen in the background. Wait until the model has finished training before you run more code. Use `wait_for_completion` to show when the model training is finished:

```
run.wait_for_completion(show_output=False) # specify True for a verbose log
```

Display run results

You now have a model trained on a remote cluster. Retrieve the accuracy of the model:

```
print(run.get_metrics())
```

The output shows the remote model has accuracy of 0.9204:

```
{'regularization_rate': 0.8, 'accuracy': 0.9204}
```

In the next tutorial, you explore this model in more detail.

Register model

The last step in the training script wrote the file `outputs/sklearn_mnist_model.pkl` in a directory named `outputs` in the VM of the cluster where the job is run. `outputs` is a special directory in that all content in this directory is automatically uploaded to your workspace. This content appears in the run record in the experiment under your workspace. So the model file is now also available in your workspace.

You can see files associated with that run:

```
print(run.get_file_names())
```

Register the model in the workspace, so that you or other collaborators can later query, examine, and deploy this model:

```
# register model
model = run.register_model(model_name='sklearn_mnist',
                           model_path='outputs/sklearn_mnist_model.pkl')
print(model.name, model.id, model.version, sep='\t')
```

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 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 an '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 of the main area 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) Click here to add tags'. A table lists 14 items under 'Filter by name...', 'All types', and 'All locations'. The table includes columns for 'NAME' (with a sorting arrow) and 'TYPE'. Items 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**.

You can also delete just the Azure Machine Learning Compute cluster. However, autoscale is turned on, and the cluster minimum is zero. So this particular resource won't incur additional compute charges when not in use:

```
# Optionally, delete the Azure Machine Learning Compute cluster
compute_target.delete()
```

Next steps

In this Azure Machine Learning tutorial, you used Python for the following tasks:

- Set up your development environment.
- Access and examine the data.
- Train multiple models on a remote cluster using the popular scikit-learn machine learning library
- Review training details and register the best model.

You're ready to deploy this registered model by using the instructions in the next part of the tutorial series:

[Tutorial 2 - Deploy models](#)

Tutorial: Deploy an image classification model in Azure Container Instances

12/23/2020 • 7 minutes to read • [Edit Online](#)

This tutorial is [part two of a two-part tutorial series](#). In the [previous tutorial](#), you trained machine learning models and then registered a model in your workspace on the cloud. Now you're ready to deploy the model as a web service. A web service is an image, in this case a Docker image. It encapsulates the scoring logic and the model itself.

In this part of the tutorial, you use Azure Machine Learning for the following tasks:

- Set up your testing environment.
- Retrieve the model from your workspace.
- Deploy the model to Container Instances.
- Test the deployed model.

Container Instances is a great solution for testing and understanding the workflow. For scalable production deployments, consider using Azure Kubernetes Service. For more information, see [how to deploy and where](#).

NOTE

Code in this article was tested with Azure Machine Learning SDK version 1.0.83.

Prerequisites

To run the notebook, first complete the model training in [Tutorial \(part 1\): Train an image classification model](#). Then open the *img-classification-part2-deploy.ipynb* notebook in your cloned *tutorials/image-classification-mnist-data* folder.

This tutorial is also available on [GitHub](#) if you wish to use it on your own [local environment](#). Make sure you have installed `matplotlib` and `scikit-learn` in your environment.

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 read along as you run the code. 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.

Set up the environment

Start by setting up a testing environment.

Import packages

Import the Python packages needed for this tutorial.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

import azureml.core

# Display the core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)
```

Deploy as web service

Deploy the model as a web service hosted in ACI.

To build the correct environment for ACI, provide the following:

- A scoring script to show how to use the model
- A configuration file to build the ACI
- The model you trained before

Create scoring script

Create the scoring script, called score.py, used by the web service call to show how to use the model.

You must include two required functions into the scoring script:

- The `init()` function, which typically loads the model into a global object. This function is run only once when the Docker container is started.
- The `run(input_data)` function uses the model to predict a value based on the input data. Inputs and outputs to the run typically use JSON for serialization and de-serialization, but other formats are supported.

```
%%writefile score.py
import json
import numpy as np
import os
import pickle
import joblib

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)
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'sklearn_mnist_model.pkl')
    model = joblib.load(model_path)

def run(raw_data):
    data = np.array(json.loads(raw_data)['data'])
    # make prediction
    y_hat = model.predict(data)
    # you can return any data type as long as it is JSON-serializable
    return y_hat.tolist()
```

Create configuration file

Create a deployment configuration file and specify the number of CPUs and gigabyte of RAM needed for your ACI container. While it depends on your model, the default of 1 core and 1 gigabyte of RAM is usually sufficient for many models. If you feel you need more later, you would have to recreate the image and redeploy the service.

```
from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores=1,
                                              memory_gb=1,
                                              tags={"data": "MNIST", "method": "sklearn"},
                                              description='Predict MNIST with sklearn')
```

Deploy in ACI

Estimated time to complete: **about 2-5 minutes**

Configure the image and deploy. The following code goes through these steps:

1. Create environment object containing dependencies needed by the model using the environment (`tutorial-env`) saved during training.
2. Create inference configuration necessary to deploy the model as a web service using:
 - The scoring file (`score.py`)
 - environment object created in previous step
3. Deploy the model to the ACI container.
4. Get the web service HTTP endpoint.

```
%%time
from azureml.core.webservice import Webservice
from azureml.core.model import InferenceConfig
from azureml.core.environment import Environment
from azureml.core import Workspace
from azureml.core.model import Model

ws = Workspace.from_config()
model = Model(ws, 'sklearn_mnist')

myenv = Environment.get(workspace=ws, name="tutorial-env", version="1")
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)

service = Model.deploy(workspace=ws,
                      name='sklearn-mnist-svc3',
                      models=[model],
                      inference_config=inference_config,
                      deployment_config=aciconfig)

service.wait_for_deployment(show_output=True)
```

Get the scoring web service's HTTP endpoint, which accepts REST client calls. This endpoint can be shared with anyone who wants to test the web service or integrate it into an application.

```
print(service.scoring_uri)
```

Test the model

Download test data

Download the test data to the `./data/` directory

```

import os
from azureml.core import Dataset
from azureml.opendatasets import MNIST

data_folder = os.path.join(os.getcwd(), 'data')
os.makedirs(data_folder, exist_ok=True)

mnist_file_dataset = MNIST.get_file_dataset()
mnist_file_dataset.download(data_folder, overwrite=True)

```

Load test data

Load the test data from the `./data/` directory created during the training tutorial.

```

from utils import load_data
import os
import glob

data_folder = os.path.join(os.getcwd(), 'data')
# note we also shrink the intensity values (X) from 0-255 to 0-1. This helps the neural network converge
# faster
X_test = load_data(glob.glob(os.path.join(data_folder,"**/t10k-images-idx3-ubyte.gz"), recursive=True)[0],
False) / 255.0
y_test = load_data(glob.glob(os.path.join(data_folder,"**/t10k-labels-idx1-ubyte.gz"), recursive=True)[0],
True).reshape(-1)

```

Predict test data

Feed the test dataset to the model to get predictions.

The following code goes through these steps:

1. Send the data as a JSON array to the web service hosted in ACI.
2. Use the SDK's `run` API to invoke the service. You can also make raw calls using any HTTP tool such as curl.

```

import json
test = json.dumps({"data": X_test.tolist()})
test = bytes(test, encoding='utf8')
y_hat = service.run(input_data=test)

```

Examine the confusion matrix

Generate a confusion matrix to see how many samples from the test set are classified correctly. Notice the misclassified value for the incorrect predictions.

```

from sklearn.metrics import confusion_matrix

conf_mx = confusion_matrix(y_test, y_hat)
print(conf_mx)
print('Overall accuracy:', np.average(y_hat == y_test))

```

The output shows the confusion matrix:

```

[[ 960    0    1    2    1    5    6    3    1    1]
 [  0 1112    3    1    0    1    5    1   12    0]
 [  9    8 920    20   10    4   10   11   37    3]
 [  4    0   17 921    2   21    4   12   20    9]
 [  1    2    5    3 915    0   10    2    6   38]
 [ 10    2    0   41   10 770   17    7   28    7]
 [  9    3    7    2    6   20 907    1    3    0]
 [  2    7   22    5    8    1    1 950    5   27]
 [ 10   15    5   21   15   27    7   11 851   12]
 [  7    8    2   13   32   13    0   24   12 898]]

```

Overall accuracy: 0.9204

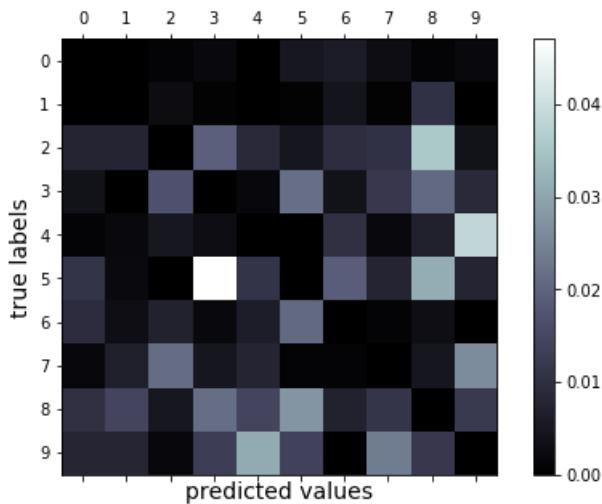
Use `matplotlib` to display the confusion matrix as a graph. In this graph, the X axis represents the actual values, and the Y axis represents the predicted values. The color in each grid represents the error rate. The lighter the color, the higher the error rate is. For example, many 5's are mis-classified as 3's. So you see a bright grid at (5,3).

```

# normalize the diagonal cells so that they don't overpower the rest of the cells when visualized
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
np.fill_diagonal(norm_conf_mx, 0)

fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(111)
cax = ax.matshow(norm_conf_mx, cmap=plt.cm.bone)
ticks = np.arange(0, 10, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(ticks)
ax.set_yticklabels(ticks)
fig.colorbar(cax)
plt.ylabel('true labels', fontsize=14)
plt.xlabel('predicted values', fontsize=14)
plt.savefig('conf.png')
plt.show()

```



Show predictions

Test the deployed model with a random sample of 30 images from the test data.

1. Print the returned predictions and plot them along with the input images. Red font and inverse image (white on black) is used to highlight the misclassified samples.

Since the model accuracy is high, you might have to run the following code a few times before you can see a misclassified sample.

```

import json

# find 30 random samples from test set
n = 30
sample_indices = np.random.permutation(X_test.shape[0])[0:n]

test_samples = json.dumps({"data": X_test[sample_indices].tolist()})
test_samples = bytes(test_samples, encoding='utf8')

# predict using the deployed model
result = service.run(input_data=test_samples)

# compare actual value vs. the predicted values:
i = 0
plt.figure(figsize = (20, 1))

for s in sample_indices:
    plt.subplot(1, n, i + 1)
    plt.axhline('')
    plt.axvline('')

    # use different color for misclassified sample
    font_color = 'red' if y_test[s] != result[i] else 'black'
    clr_map = plt.cm.gray if y_test[s] != result[i] else plt.cm.Greys

    plt.text(x=10, y=-10, s=result[i], fontsize=18, color=font_color)
    plt.imshow(X_test[s].reshape(28, 28), cmap=clr_map)

    i = i + 1
plt.show()

```

You can also send raw HTTP request to test the web service.

```

import requests

# send a random row from the test set to score
random_index = np.random.randint(0, len(X_test)-1)
input_data = "{\"data\": [" + str(list(X_test[random_index])) + "]}"

headers = {'Content-Type': 'application/json'}

# for AKS deployment you'd need to the service key in the header as well
# api_key = service.get_key()
# headers = {'Content-Type':'application/json', 'Authorization':('Bearer ' + api_key)}

resp = requests.post(service.scoring_uri, input_data, headers=headers)

print("POST to url", service.scoring_uri)
#print("input data:", input_data)
print("label:", y_test[random_index])
print("prediction:", resp.text)

```

Clean up resources

To keep the resource group and workspace for other tutorials and exploration, you can delete only the Container Instances deployment by using this API call:

```
service.delete()
```

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 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 visible with various service icons. The main content area is titled 'my-rg - Microsoft Azure' and shows the 'Resource groups > my-rg' path. On the right, the 'Overview' tab is selected. At the top of the overview page, there is a toolbar with 'Add', 'Edit columns', and a 'Delete resource group' button, which is highlighted with a red box. Below the toolbar, there are sections for 'Subscription (change)', 'Visual Studio Ultimate with MSDN', 'Subscription ID', and 'Tags (change)'. A table below lists 14 items, including 'myworkspace' and 'my-workspace' under 'Machine Learning service workspace', and 'myworkspace0013141752' under 'Application Insights'. There are filters for 'NAME' and 'TYPE' at the bottom of the list.

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

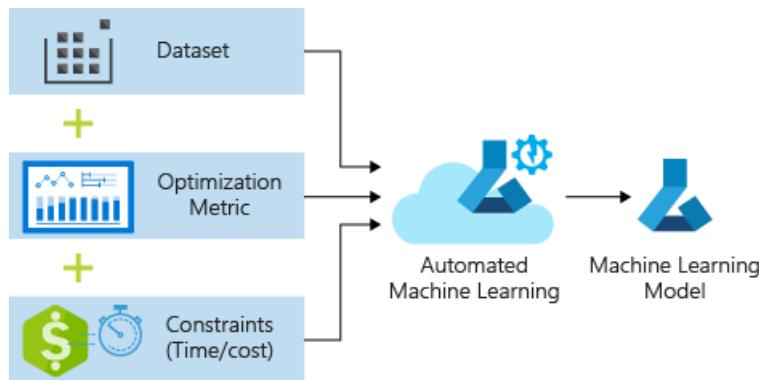
Next steps

- Learn about all of the [deployment options for Azure Machine Learning](#).
- Learn how to [create clients for the web service](#).
- [Make predictions on large quantities of data asynchronously](#).
- Monitor your Azure Machine Learning models with [Application Insights](#).
- Try out the [automatic algorithm selection](#) tutorial.

Tutorial: Use automated machine learning to predict taxi fares

12/23/2020 • 14 minutes to read • [Edit Online](#)

In this tutorial, you use automated machine learning in Azure Machine Learning to create a regression model to predict NYC taxi fare prices. This process 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.



In this tutorial you learn the following tasks:

- Download, transform, and clean data using Azure Open Datasets
- Train an automated machine learning regression model
- Calculate model accuracy

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 [setup tutorial](#) if you don't already have an Azure Machine Learning workspace or notebook virtual machine.
- After you complete the setup tutorial, open the `tutorials/regression-automl-nyc-taxi-data/regression-automated-ml.ipynb` notebook using the same notebook server.

This tutorial is also available on [GitHub](#) if you wish to run it in your own [local environment](#). To get the required packages,

- [Install the full automl client](#).
- Run `pip install azureml-opendatasets azureml-widgets` to get the required packages.

Download and prepare data

Import the necessary packages. The Open Datasets package contains a class representing each data source (`NycTlcGreen` for example) to easily filter date parameters before downloading.

```
from azureml.opendatasets import NycTlcGreen
import pandas as pd
from datetime import datetime
from dateutil.relativedelta import relativedelta
```

Begin by creating a dataframe to hold the taxi data. When working in a non-Spark environment, Open Datasets only allows downloading one month of data at a time with certain classes to avoid `MemoryError` with large datasets.

To download taxi data, iteratively fetch one month at a time, and before appending it to `green_taxi_df` randomly sample 2,000 records from each month to avoid bloating the dataframe. Then preview the data.

```
green_taxi_df = pd.DataFrame([])
start = datetime.strptime("1/1/2015", "%m/%d/%Y")
end = datetime.strptime("1/31/2015", "%m/%d/%Y")

for sample_month in range(12):
    temp_df_green = NycTlcGreen(start + relativedelta(months=sample_month), end +
relativedelta(months=sample_month)) \
        .to_pandas_dataframe()
    green_taxi_df = green_taxi_df.append(temp_df_green.sample(2000))

green_taxi_df.head(10)
```


VENDOR ID	TIME MEASURE	LPEP DRICKUPP	PASSENGER COUNT	TRIP DISTANCE	PULLOCATIONID	DOLONGITUDE	PICKUPLONGITUDE	DROPFLONGITUDE	PAYMENTTYPE	FAREAMOUNT	EXTRA	MATA	IMPROVEMENTTSURCHARGE	TIPAMOUNT	TOLLSAMOUNT	EHAILFEE	TOTALAMOUNT	TRIP TYPE
113951	10923:25:51	20150:-0923:25:51	213.055000	1.190000	None	-7.000000	40.000000	-7.000000	...	2	12.500000	0.500000	0.500000	0.000000	0.000000	0.000000	nanan	13.80
150436	20150:-1111:17:14	20150:-1111:17:14	211.190000	1.190000	None	-7.000000	40.000000	-7.000000	...	100	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	nanan	9.55

		L	P	E	P	D	R	O	P	I	C	P	O	P	A	F	M	E	T	S	T	L	S	A	H	A	M	O	T	R								
V	E	N	D	O	R	I	D	D	G	D	C	C	O	L	O	M	E	N	T	M	A	T	A	M	O	R	C	H	A	R	M	O	T	R				
E	N	D	O	R	I	M	M	E	M	E	I	A	N	O	I	T	Y	P	T	M	A	T	A	M	O	A	L	F	E	O	T	P						
E	N	D	O	R	I	M	M	E	M	E	I	A	N	O	I	T	Y	P	T	M	A	T	A	M	O	R	C	H	A	R	M	O	T	R				
E	N	D	O	R	I	M	M	E	M	E	I	A	N	O	I	T	Y	P	T	M	A	T	A	M	O	R	C	H	A	R	M	O	T	R				
4	2	2	N	N	-	4	-	...	2	5	0	0	0	0	0	0	n	6	1																			
3	0	o	o	o	7	0	7			a	.	.																			
2	1	n	n	n	3	.	3			0	5	5	3	0	0	0	n	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
1	5	e	e	e	.	7	.			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
3	-	9	9	9	1	9	4																															
6	0	1	-	2	2	2	2	3	:	1	6	:	3	3	2	2	0	1	5	-	0	1	-	2	2	2	3	:	2	0	1	6	5					

Now that the initial data is loaded, define a function to create various time-based features from the pickup datetime field. This will create new fields for the month number, day of month, day of week, and hour of day, and will allow the model to factor in time-based seasonality. Use the `apply()` function on the dataframe to iteratively

apply the `build_time_features()` function to each row in the taxi data.

```
def build_time_features(vector):
    pickup_datetime = vector[0]
    month_num = pickup_datetime.month
    day_of_month = pickup_datetime.day
    day_of_week = pickup_datetime.weekday()
    hour_of_day = pickup_datetime.hour

    return pd.Series((month_num, day_of_month, day_of_week, hour_of_day))

green_taxi_df[["month_num", "day_of_month", "day_of_week", "hour_of_day"]]
green_taxi_df[["lpepPickupDatetime"]].apply(build_time_features, axis=1)
green_taxi_df.head(10)
```

V E N D O R I D	L P E P P R I C K U P D A T E T I M E	P A S S E N T R I P D A S T I O N I D E	T R I P L O C A T T I O N I D E	D O L O P L O N G T U D E	P I C K U P K F L O N G T U D E	D R O P O F F L O N G T U D E	P A Y M E N T T Y P E	F A R E A M O U N T	M T T R A	E X T R A	M T A X	T O L L S U R C H A R G E	T I P A M O U N T	T O T A L A M O U N T	T R I P T Y P E	M O N T H - N U M	D A Y - O F - M O N T H	D A Y - O F - W E E K	H O U R - O F - D A Y		
1	2	2	2	1	0	None	None	-7	4	-	.	2	4	1	0	0	0	nan	6	1	1
1	1	0	0	.	None	None	3	0	7	20	1
2	1	1	6	6	None	None	.	8	3	.			5	0	0	5	0	30	0	0	6
9	5	5	9	9	e	e	.	1	9	.			0	0	0	0	0	0	0	0	0
8	-	-	9	9			6	6	6												
1	0	0	0	0																	
7	1	1	1	1																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.											
2	0	0	0	0																	
0	1	1	1	1																	
1	1	1	1	1																	
0	0	0	0	0																	
5	5	5	5	5																	
6	6	6	6	6																	
:	:	:	:	:																	
2	2	2	2	2																	
9	9	9	9	9																	
6	-	-	9	9			6	6	9	.</td											

V E N D O R I D	L P E P P R O C P O F D A T E T I M E	P A S S E N D I S T T C O U N T E	T R U L O C A T I O N I D E	D O L O C A T I O N I D E	P I C K U P L O C A T I O N I D E	D R O P O F L O C A T I O N I D E	P A Y M E N T T Y P E	F A R E A M O U N T	M T T R A	E X T R A	T O L L S U R C H A R G E	T I P A M O U N T	E H A M O U N T	T O T A L A M O U N T	T R I P T Y P E	M O N T H - N U M	D A Y - O F - M O N T H	D A Y - O F - W E E K	H O U R - O F - D A Y
8	1	2	2	1	None	None	-7	40	-7.	2	6.	0.	00	00	nan	7.	1.	14	619
1	0	0	.	1	none	none	3.	7.	3.	50	50	50	30	00	00	80	00		
1	1	1	1	0	ne	ne	9.	22	95										
7	5	5	-	-			6												
5	0	0	1	-	0	0													
5	0	0	0	0	0	0													
1	1	-	0	4	4	1													
9	1	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
0	0	0	9	0	ne	ne	8.	66	87										
0	0	0	8	8	8	8													
1	1	-	0	3	3	1													
1	1	-	0	3	3	1													
2	2	2	1	0	None	None	-7	40	-7.	2	6.	0.	00	00	nan	6.	1.	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
7	1	1	9	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512
2	5	5	0	0	ne	ne	7.	7.	.	00	00	00	00	00	00	80	00		
8	-	-																	
1	0	0	1	-	0	0													
1	1	-	0	3	3	1													
2	2	2	1	9	none	none	3.	7.	3.	00	00	53	00	00	00	80	00	13	512

Remove some of the columns that you won't need for training or additional feature building.

```

columns_to_remove = ["lpepPickupDatetime", "lpepDropoffDatetime", "puLocationId", "doLocationId", "extra",
"mtaTax",
"improvementSurcharge", "tollsAmount", "ehailFee", "tripType", "rateCodeID",
"storeAndFwdFlag", "paymentType", "fareAmount", "tipAmount"
]
for col in columns_to_remove:
    green_taxi_df.pop(col)

green_taxi_df.head(5)

```

Cleanse data

Run the `describe()` function on the new dataframe to see summary statistics for each field.

```
green_taxi_df.describe()
```

VEND ORID	PASS ENGE RCO UNT	TRIP DIST ANC E	PICK UPLO NGIT UDE	PICK UPLA TITU DE	DRO POFF LON GITU DE	DRO POFF LATI TUDE	TOTA LAM OUN T	MON TH_N UM DAY_ OF_ MON TH	DAY_ OF_ WEEK	HOU R_OF _DAY		
count	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00	4800 0.00
mean	1.78	1.37	2.87	- 73.8 3	40.6 9	- 73.8 4	40.7 0	14.7 5	6.50	15.1 3	3.27	13.5 2
std	0.41	1.04	2.93	2.76	1.52	2.61	1.44	12.0 8	3.45	8.45	1.95	6.83
min	1.00	0.00	0.00	- 74.6 6	0.00	- 74.6 6	0.00	- 300. 00	1.00	1.00	0.00	0.00
25%	2.00	1.00	1.06	- 73.9 6	40.7 0	- 73.9 7	40.7 0	7.80	3.75	8.00	2.00	9.00
50%	2.00	1.00	1.90	- 73.9 4	40.7 5	- 73.9 4	40.7 5	11.3 0	6.50	15.0 0	3.00	15.0 0
75%	2.00	1.00	3.60	- 73.9 2	40.8 0	- 73.9 1	40.7 9	17.8 0	9.25	22.0 0	5.00	19.0 0
max	2.00	9.00	97.5 7	0.00	41.9 3	0.00	41.9 4	450. 00	12.0 0	30.0 0	6.00	23.0 0

From the summary statistics, you see that there are several fields that have outliers or values that will reduce model accuracy. First filter the lat/long fields to be within the bounds of the Manhattan area. This will filter out longer taxi trips or trips that are outliers in respect to their relationship with other features.

Additionally filter the `tripDistance` field to be greater than zero but less than 31 miles (the haversine distance between the two lat/long pairs). This eliminates long outlier trips that have inconsistent trip cost.

Lastly, the `totalAmount` field has negative values for the taxi fares, which don't make sense in the context of our model, and the `passengerCount` field has bad data with the minimum values being zero.

Filter out these anomalies using query functions, and then remove the last few columns unnecessary for training.

```
final_df = green_taxi_df.query("pickupLatitude>=40.53 and pickupLatitude<=40.88")
final_df = final_df.query("pickupLongitude>=-74.09 and pickupLongitude<=-73.72")
final_df = final_df.query("tripDistance>=0.25 and tripDistance<31")
final_df = final_df.query("passengerCount>0 and totalAmount>0")

columns_to_remove_for_training = ["pickupLongitude", "pickupLatitude", "dropoffLongitude",
"dropoffLatitude"]
for col in columns_to_remove_for_training:
    final_df.pop(col)
```

Call `describe()` again on the data to ensure cleansing worked as expected. You now have a prepared and cleansed set of taxi, holiday, and weather data to use for machine learning model training.

```
final_df.describe()
```

Configure workspace

Create a workspace object from the existing workspace. A [Workspace](#) is a class that accepts your Azure subscription and resource information. It also creates a cloud resource to monitor and track your model runs.

`Workspace.from_config()` reads the file `config.json` and loads the authentication details into an object named `ws`. `ws` is used throughout the rest of the code in this tutorial.

```
from azureml.core.workspace import Workspace
ws = Workspace.from_config()
```

Split the data into train and test sets

Split the data into training and test sets by using the `train_test_split` function in the `scikit-learn` library. This function segregates the data into the `x (features)` data set for model training and the `y (values to predict)` data set for testing.

The `test_size` parameter determines the percentage of data to allocate to testing. The `random_state` parameter sets a seed to the random generator, so that your train-test splits are deterministic.

```
from sklearn.model_selection import train_test_split
x_train, x_test = train_test_split(final_df, test_size=0.2, random_state=223)
```

The purpose of this step is to have data points to test the finished model that haven't been used to train the model, in order to measure true accuracy.

In other words, a well-trained model should be able to accurately make predictions from data it hasn't already seen. You now have data prepared for auto-training a machine learning model.

Automatically train a model

To automatically train a model, take the following steps:

1. Define settings for the experiment run. Attach your training data to the configuration, and modify settings that control the training process.
2. Submit the experiment for model tuning. After submitting the experiment, the process iterates through different machine learning algorithms and hyperparameter settings, adhering to your defined constraints. It chooses the best-fit model by optimizing an accuracy metric.

Define training settings

Define the experiment parameter and model settings for training. View the full list of [settings](#). Submitting the experiment with these default settings will take approximately 5-20 min, but if you want a shorter run time, reduce the `experiment_timeout_hours` parameter.

PROPERTY	VALUE IN THIS TUTORIAL	DESCRIPTION
<code>iteration_timeout_minutes</code>	10	Time limit in minutes for each iteration. Increase this value for larger datasets that need more time for each iteration.
<code>experiment_timeout_hours</code>	0.3	Maximum amount of time in hours that all iterations combined can take before the experiment terminates.
<code>enable_early_stopping</code>	True	Flag to enable early termination if the score is not improving in the short term.
<code>primary_metric</code>	<code>spearman_correlation</code>	Metric that you want to optimize. The best-fit model will be chosen based on this metric.
<code>featurization</code>	<code>auto</code>	By using <code>auto</code> , the experiment can preprocess the input data (handling missing data, converting text to numeric, etc.)
<code>verbosity</code>	<code>logging.INFO</code>	Controls the level of logging.
<code>n_cross_validations</code>	5	Number of cross-validation splits to perform when validation data is not specified.

```
import logging

automl_settings = {
    "iteration_timeout_minutes": 10,
    "experiment_timeout_hours": 0.3,
    "enable_early_stopping": True,
    "primary_metric": 'spearman_correlation',
    "featurization": 'auto',
    "verbosity": logging.INFO,
    "n_cross_validations": 5
}
```

Use your defined training settings as a `**kwargs` parameter to an `AutoMLConfig` object. Additionally, specify your training data and the type of model, which is `regression` in this case.

```
from azureml.train.automl import AutoMLConfig

automl_config = AutoMLConfig(task='regression',
                             debug_log='automated_ml_errors.log',
                             training_data=x_train,
                             label_column_name="totalAmount",
                             **automl_settings)
```

NOTE

Automated machine learning pre-processing 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 pre-processing steps applied during training are applied to your input data automatically.

Train the automatic regression model

Create an experiment object in your workspace. An experiment acts as a container for your individual runs. Pass the defined `automl_config` object to the experiment, and set the output to `True` to view progress during the run.

After starting the experiment, the output shown updates live as the experiment runs. For each iteration, you see the model type, the run duration, and the training accuracy. The field `BEST` tracks the best running training score based on your metric type.

```
from azureml.core.experiment import Experiment
experiment = Experiment(ws, "taxi-experiment")
local_run = experiment.submit(automl_config, show_output=True)
```

```
Running on local machine
Parent Run ID: AutoML_1766cdf7-56cf-4b28-a340-c4aeee15b12b
Current status: DatasetFeaturization. Beginning to featurize the dataset.
Current status: DatasetEvaluation. Gathering dataset statistics.
Current status: FeaturesGeneration. Generating features for the dataset.
Current status: DatasetFeaturizationCompleted. Completed featurizing the dataset.
Current status: DatasetCrossValidationSplit. Generating individually featurized CV splits.
Current status: ModelSelection. Beginning model selection.
```

```
*****
```

```
ITERATION: The iteration being evaluated.
```

```
PIPELINE: A summary description of the pipeline being evaluated.
```

```
DURATION: Time taken for the current iteration.
```

```
METRIC: The result of computing score on the fitted pipeline.
```

```
BEST: The best observed score thus far.
```

```
*****
```

ITERATION	PIPELINE	DURATION	METRIC	BEST
0	StandardScalerWrapper RandomForest	0:00:16	0.8746	0.8746
1	MinMaxScaler RandomForest	0:00:15	0.9468	0.9468
2	StandardScalerWrapper ExtremeRandomTrees	0:00:09	0.9303	0.9468
3	StandardScalerWrapper LightGBM	0:00:10	0.9424	0.9468
4	RobustScaler DecisionTree	0:00:09	0.9449	0.9468
5	StandardScalerWrapper LassoLars	0:00:09	0.9440	0.9468
6	StandardScalerWrapper LightGBM	0:00:10	0.9282	0.9468
7	StandardScalerWrapper RandomForest	0:00:12	0.8946	0.9468
8	StandardScalerWrapper LassoLars	0:00:16	0.9439	0.9468
9	MinMaxScaler ExtremeRandomTrees	0:00:35	0.9199	0.9468
10	RobustScaler ExtremeRandomTrees	0:00:19	0.9411	0.9468
11	StandardScalerWrapper ExtremeRandomTrees	0:00:13	0.9077	0.9468
12	StandardScalerWrapper LassoLars	0:00:15	0.9433	0.9468
13	MinMaxScaler ExtremeRandomTrees	0:00:14	0.9186	0.9468
14	RobustScaler RandomForest	0:00:10	0.8810	0.9468
15	StandardScalerWrapper LassoLars	0:00:55	0.9433	0.9468
16	StandardScalerWrapper ExtremeRandomTrees	0:00:13	0.9026	0.9468
17	StandardScalerWrapper RandomForest	0:00:13	0.9140	0.9468
18	VotingEnsemble	0:00:23	0.9471	0.9471
19	StackEnsemble	0:00:27	0.9463	0.9471

Explore the results

Explore the results of automatic training with a [Jupyter widget](#). The widget allows you to see a graph and table of all individual run iterations, along with training accuracy metrics and metadata. Additionally, you can filter on different accuracy metrics than your primary metric with the dropdown selector.

```
from azureml.widgets import RunDetails
RunDetails(local_run).show()
```

AutoML_797ff8a7-a369-4986-8180-e9bbcd938259:
Status: Completed



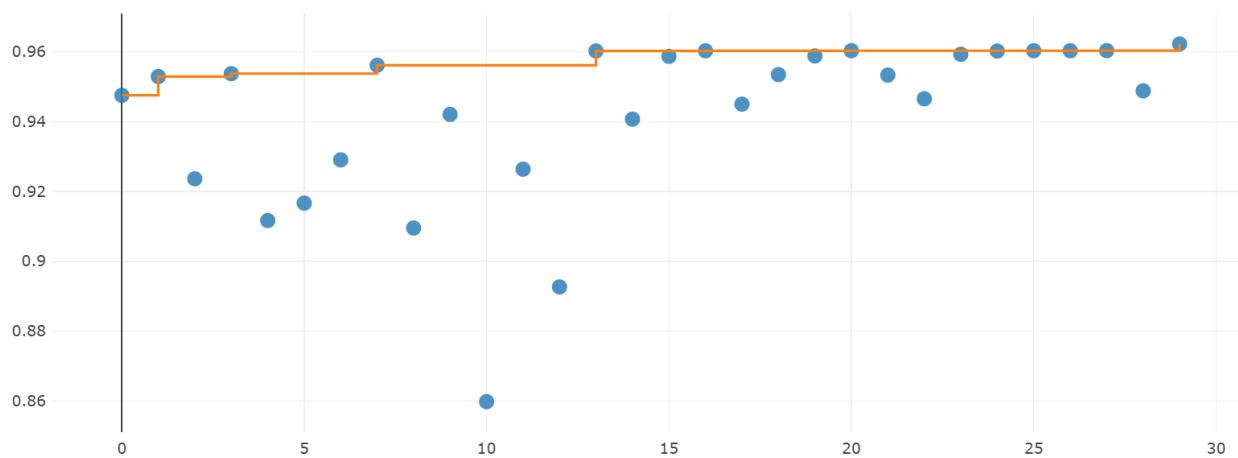
Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started	Ran
29	Ensemble	0.96225654	0.96225654	Completed	0:01:10	Dec 6, 2018 6:12 PM	29
27	MaxAbsScaler, SGD	0.96033526	0.96033526	Completed	0:00:15	Dec 6, 2018 6:11 PM	27
25	StandardScalerWrapper, ElasticNet	0.96031892	0.96031892	Completed	0:00:11	Dec 6, 2018 6:11 PM	25
20	MaxAbsScaler, SGD	0.96031661	0.96031661	Completed	0:00:20	Dec 6, 2018 6:09 PM	20
26	StandardScalerWrapper, ElasticNet	0.96031391	0.96031892	Completed	0:00:12	Dec 6, 2018 6:11 PM	26

Pages: 1 2 3 4 5 6 Next Last 5 per page

spearman_correlation



AutoML Run with metric : spearman_correlation



[Click here to see the run in Azure portal](#)

Retrieve the best model

Select the best model from your iterations. The `get_output` function returns the best run and the fitted model for the last fit invocation. By using the overloads on `get_output`, you can retrieve the best run and fitted model for any logged metric or a particular iteration.

```
best_run, fitted_model = local_run.get_output()
print(best_run)
print(fitted_model)
```

Test the best model accuracy

Use the best model to run predictions on the test data set to predict taxi fares. The function `predict` uses the best model and predicts the values of `y, trip cost`, from the `x_test` data set. Print the first 10 predicted cost values from `y_predict`.

```
y_test = x_test.pop("totalAmount")

y_predict = fitted_model.predict(x_test)
print(y_predict[:10])
```

Calculate the `root mean squared error` of the results. Convert the `y_test` data frame to a list to compare to the predicted values. The function `mean_squared_error` takes two arrays of values and calculates the average squared error between them. Taking the square root of the result gives an error in the same units as the `y` variable, `cost`. It

indicates roughly how far the taxi fare predictions are from the actual fares.

```
from sklearn.metrics import mean_squared_error
from math import sqrt

y_actual = y_test.values.flatten().tolist()
rmse = sqrt(mean_squared_error(y_actual, y_predict))
rmse
```

Run the following code to calculate mean absolute percent error (MAPE) by using the full `y_actual` and `y_predict` data sets. This metric calculates an absolute difference between each predicted and actual value and sums all the differences. Then it expresses that sum as a percent of the total of the actual values.

```
sum_actuals = sum_errors = 0

for actual_val, predict_val in zip(y_actual, y_predict):
    abs_error = actual_val - predict_val
    if abs_error < 0:
        abs_error = abs_error * -1

    sum_errors = sum_errors + abs_error
    sum_actuals = sum_actuals + actual_val

mean_abs_percent_error = sum_errors / sum_actuals
print("Model MAPE:")
print(mean_abs_percent_error)
print()
print("Model Accuracy:")
print(1 - mean_abs_percent_error)
```

```
Model MAPE:
0.14353867606052823
```

```
Model Accuracy:
0.8564613239394718
```

From the two prediction accuracy metrics, you see that the model is fairly good at predicting taxi fares from the data set's features, typically within +- \$4.00, and approximately 15% error.

The traditional machine learning model development process is highly resource-intensive, and requires significant domain knowledge and time investment to run and compare the results of dozens of models. Using automated machine learning is a great way to rapidly test many different models for your scenario.

Clean up resources

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

Stop the compute instance

If you used a compute instance or Notebook VM, stop the VM when you aren't using it to reduce cost.

1. In your workspace, select **Compute**.
2. From the list, select the VM.
3. Select **Stop**.
4. When you're ready to use the server again, select **Start**.

Delete everything

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 by using an automated regression model locally with custom parameters.
- Explored and reviewed training results.

[Deploy your model](#) with Azure Machine Learning.

Tutorial: Build an Azure Machine Learning pipeline for batch scoring

12/23/2020 • 11 minutes to read • [Edit Online](#)

In this advanced tutorial, you learn how to build an [Azure Machine Learning pipeline](#) to run a batch scoring job. Machine learning pipelines optimize your workflow with speed, portability, and reuse, so you can focus on machine learning instead of infrastructure and automation. After you build and publish a pipeline, you configure a REST endpoint that you can use to trigger the pipeline from any HTTP library on any platform.

The example uses a pretrained [Inception-V3](#) convolutional neural network model implemented in Tensorflow to classify unlabeled images.

In this tutorial, you complete the following tasks:

- Configure workspace
- Download and store sample data
- Create dataset objects to fetch and output data
- Download, prepare, and register the model in your workspace
- Provision compute targets and create a scoring script
- Use the `ParallelRunStep` class for async batch scoring
- Build, run, and publish a pipeline
- Enable a REST endpoint for the pipeline

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

- If you don't already have an Azure Machine Learning workspace or notebook virtual machine, complete [Part 1 of the setup tutorial](#).
- When you finish the setup tutorial, use the same notebook server to open the `tutorials/machine-learning-pipelines-advanced/tutorial-pipeline-batch-scoring-classification.ipynb` notebook.

If you want to run the setup tutorial in your own [local environment](#), you can access the tutorial on [GitHub](#). Run `pip install azureml-sdk[notebooks] azureml-pipeline-core azureml-pipeline-steps pandas requests` to get the required packages.

Configure workspace and create a datastore

Create a workspace object from the existing Azure Machine Learning workspace.

```
from azureml.core import Workspace  
ws = Workspace.from_config()
```

IMPORTANT

This code snippet expects the workspace configuration to be saved in the current directory or its parent. For more information on creating a workspace, see [Create and manage Azure Machine Learning workspaces](#). For more information on saving the configuration to file, see [Create a workspace configuration file](#).

Create a datastore for sample images

On the `pipelinedata` account, get the ImageNet evaluation public data sample from the `sampleddata` public blob container. Call `register_azure_blob_container()` to make the data available to the workspace under the name `images_datastore`. Then, set the workspace default datastore as the output datastore. Use the output datastore to score output in the pipeline.

For more information on accessing data, see [How to access data](#).

```
from azureml.core.datastore import Datastore

batchscore_blob = Datastore.register_azure_blob_container(ws,
    datastore_name="images_datastore",
    container_name="sampledata",
    account_name="pipelinedata",
    overwrite=True)

def_data_store = ws.get_default_datastore()
```

Create dataset objects

When building pipelines, `Dataset` objects are used for reading data from workspace datastores, and `PipelineData` objects are used for transferring intermediate data between pipeline steps.

IMPORTANT

The batch scoring example in this tutorial uses only one pipeline step. In use cases that have multiple steps, the typical flow will include these steps:

1. Use `Dataset` objects as *inputs* to fetch raw data, perform some transformation, and then *output a PipelineData object*.
2. Use the `PipelineData` *output object* in the preceding step as an *input object*. Repeat it for subsequent steps.

In this scenario, you create `Dataset` objects that correspond to the datastore directories for both the input images and the classification labels (y-test values). You also create a `PipelineData` object for the batch scoring output data.

```
from azureml.core.dataset import Dataset
from azureml.pipeline.core import PipelineData

input_images = Dataset.File.from_files((batchscore_blob, "batchscoring/images/"))
label_ds = Dataset.File.from_files((batchscore_blob, "batchscoring/labels/"))
output_dir = PipelineData(name="scores",
    datastore=def_data_store,
    output_path_on_compute="batchscoring/results")
```

Register the datasets to the workspace if you want to reuse it later. This step is optional.

```
input_images = input_images.register(workspace = ws, name = "input_images")
label_ds = label_ds.register(workspace = ws, name = "label_ds")
```

Download and register the model

Download the pretrained Tensorflow model to use it for batch scoring in a pipeline. First, create a local directory where you store the model. Then, download and extract the model.

```
import os
import tarfile
import urllib.request

if not os.path.isdir("models"):
    os.mkdir("models")

response =
urllib.request.urlretrieve("http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz",
"model.tar.gz")
tar = tarfile.open("model.tar.gz", "r:gz")
tar.extractall("models")
```

Next, register the model to your workspace, so you can easily retrieve the model in the pipeline process. In the `register()` static function, the `model_name` parameter is the key you use to locate your model throughout the SDK.

```
from azureml.core.model import Model

model = Model.register(model_path="models/inception_v3.ckpt",
                      model_name="inception",
                      tags={"pretrained": "inception"},
                      description="Imagenet trained tensorflow inception",
                      workspace=ws)
```

Create and attach the remote compute target

Machine learning pipelines can't be run locally, so you run them on cloud resources or *remote compute targets*. A remote compute target is a reusable virtual compute environment where you run experiments and machine learning workflows.

Run the following code to create a GPU-enabled `AmlCompute` target, and then attach it to your workspace. For more information about compute targets, see the [conceptual article](#).

```

from azureml.core.compute import AmlCompute, ComputeTarget
from azureml.exceptions import ComputeTargetException
compute_name = "gpu-cluster"

# checks to see if compute target already exists in workspace, else create it
try:
    compute_target = ComputeTarget(workspace=ws, name=compute_name)
except ComputeTargetException:
    config = AmlCompute.provisioning_configuration(vm_size="STANDARD_NC6",
                                                    vm_priority="lowpriority",
                                                    min_nodes=0,
                                                    max_nodes=1)

    compute_target = ComputeTarget.create(workspace=ws, name=compute_name,
                                         provisioning_configuration=config)
    compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)

```

Write a scoring script

To do the scoring, create a batch scoring script called `batch_scoring.py`, and then write it to the current directory. The script takes input images, applies the classification model, and then outputs the predictions to a results file.

The `batch_scoring.py` script takes the following parameters, which get passed from the `ParallelRunStep` you create later:

- `--model_name`: The name of the model being used.
- `--labels_dir`: The location of the `labels.txt` file.

The pipeline infrastructure uses the `ArgumentParser` class to pass parameters into pipeline steps. For example, in the following code, the first argument `--model_name` is given the property identifier `model_name`. In the `init()` function, `Model.get_model_path(args.model_name)` is used to access this property.

```

%%writefile batch_scoring.py

import os
import argparse
import datetime
import time
import tensorflow as tf
from math import ceil
import numpy as np
import shutil
from tensorflow.contrib.slim.python.slim.nets import inception_v3

from azureml.core import Run
from azureml.core.model import Model
from azureml.core.dataset import Dataset

slim = tf.contrib.slim

image_size = 299
num_channel = 3

def get_class_label_dict(labels_dir):
    label = []
    labels_path = os.path.join(labels_dir, 'labels.txt')
    proto_as_ascii_lines = tf.gfile.GFile(labels_path).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

```

```

def init():
    global g_tf_sess, probabilities, label_dict, input_images

    parser = argparse.ArgumentParser(description="Start a tensorflow model serving")
    parser.add_argument('--model_name', dest="model_name", required=True)
    parser.add_argument('--labels_dir', dest="labels_dir", required=True)
    args, _ = parser.parse_known_args()

    label_dict = get_class_label_dict(args.labels_dir)
    classes_num = len(label_dict)

    with slim.arg_scope(inception_v3.inception_v3_arg_scope()):
        input_images = tf.placeholder(tf.float32, [1, image_size, image_size, num_channel])
        logits, _ = inception_v3.inception_v3(input_images,
                                                num_classes=classes_num,
                                                is_training=False)
    probabilities = tf.argmax(logits, 1)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    g_tf_sess = tf.Session(config=config)
    g_tf_sess.run(tf.global_variables_initializer())
    g_tf_sess.run(tf.local_variables_initializer())

    model_path = Model.get_model_path(args.model_name)
    saver = tf.train.Saver()
    saver.restore(g_tf_sess, model_path)

def file_to_tensor(file_path):
    image_string = tf.read_file(file_path)
    image = tf.image.decode_image(image_string, channels=3)

    image.set_shape([None, None, None])
    image = tf.image.resize_images(image, [image_size, image_size])
    image = tf.divide(tf.subtract(image, [0]), [255])
    image.set_shape([image_size, image_size, num_channel])
    return image

def run(mini_batch):
    result_list = []
    for file_path in mini_batch:
        test_image = file_to_tensor(file_path)
        out = g_tf_sess.run(test_image)
        result = g_tf_sess.run(probabilities, feed_dict={input_images: [out]})
        result_list.append(os.path.basename(file_path) + ": " + label_dict[result[0]])
    return result_list

```

TIP

The pipeline in this tutorial has only one step, and it writes the output to a file. For multi-step pipelines, you also use `ArgumentParser` to define a directory to write output data for input to subsequent steps. For an example of passing data between multiple pipeline steps by using the `ArgumentParser` design pattern, see the [notebook](#).

Build the pipeline

Before you run the pipeline, create an object that defines the Python environment and creates the dependencies that your `batch_scoring.py` script requires. The main dependency required is Tensorflow, but you also install `azureml-core` and `azureml-dataprep[fuse]` which are required by ParallelRunStep. Also, specify Docker and Docker-GPU support.

```
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core.runconfig import DEFAULT_GPU_IMAGE

cd = CondaDependencies.create(pip_packages=["tensorflow-gpu==1.15.2",
                                            "azureml-core", "azureml-dataprep[fuse]"])
env = Environment(name="parallelenv")
env.python.conda_dependencies = cd
env.docker.base_image = DEFAULT_GPU_IMAGE
```

Create the configuration to wrap the script

Create the pipeline step using the script, environment configuration, and parameters. Specify the compute target you already attached to your workspace.

```
from azureml.pipeline.steps import ParallelRunConfig

parallel_run_config = ParallelRunConfig(
    environment=env,
    entry_script="batch_scoring.py",
    source_directory=".",
    output_action="append_row",
    mini_batch_size="20",
    error_threshold=1,
    compute_target=compute_target,
    process_count_per_node=2,
    node_count=1
)
```

Create the pipeline step

A pipeline step is an object that encapsulates everything you need to run a pipeline, including:

- Environment and dependency settings
- The compute resource to run the pipeline on
- Input and output data, and any custom parameters
- Reference to a script or SDK logic to run during the step

Multiple classes inherit from the parent class [PipelineStep](#). You can choose classes to use specific frameworks or stacks to build a step. In this example, you use the [ParallelRunStep](#) class to define your step logic by using a custom Python script. If an argument to your script is either an input to the step or an output of the step, the argument must be defined *both* in the `arguments` array *and* in either the `input` or the `output` parameter, respectively.

In scenarios where there is more than one step, an object reference in the `outputs` array becomes available as an `input` for a subsequent pipeline step.

```

from azureml.pipeline.steps import ParallelRunStep
from datetime import datetime

parallel_step_name = "batchscoring-" + datetime.now().strftime("%Y%m%d%H%M")

label_config = label_ds.as_named_input("labels_input")

batch_score_step = ParallelRunStep(
    name=parallel_step_name,
    inputs=[input_images.as_named_input("input_images")],
    output=output_dir,
    arguments=["--model_name", "inception",
               "--labels_dir", label_config],
    side_inputs=[label_config],
    parallel_run_config=parallel_run_config,
    allow_reuse=False
)

```

For a list of all the classes you can use for different step types, see the [steps package](#).

Submit the pipeline

Now, run the pipeline. First, create a `Pipeline` object by using your workspace reference and the pipeline step you created. The `steps` parameter is an array of steps. In this case, there's only one step for batch scoring. To build pipelines that have multiple steps, place the steps in order in this array.

Next, use the `Experiment.submit()` function to submit the pipeline for execution. The `wait_for_completion` function outputs logs during the pipeline build process. You can use the logs to see current progress.

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.

```

from azureml.core import Experiment
from azureml.pipeline.core import Pipeline

pipeline = Pipeline(workspace=ws, steps=[batch_score_step])
pipeline_run = Experiment(ws, 'batch_scoring').submit(pipeline)
pipeline_run.wait_for_completion(show_output=True)

```

Download and review output

Run the following code to download the output file that's created from the `batch_scoring.py` script. Then, explore the scoring results.

```
import pandas as pd

batch_run = next(pipeline_run.get_children())
batch_output = batch_run.get_output_data("scores")
batch_output.download(local_path="inception_results")

for root, dirs, files in os.walk("inception_results"):
    for file in files:
        if file.endswith("parallel_run_step.txt"):
            result_file = os.path.join(root, file)

df = pd.read_csv(result_file, delimiter=":", header=None)
df.columns = ["Filename", "Prediction"]
print("Prediction has ", df.shape[0], " rows")
df.head(10)
```

Publish and run from a REST endpoint

Run the following code to publish the pipeline to your workspace. In your workspace in Azure Machine Learning studio, you can see metadata for the pipeline, including run history and durations. You can also run the pipeline manually from the studio.

Publishing the pipeline enables a REST endpoint that you can use to run the pipeline from any HTTP library on any platform.

```
published_pipeline = pipeline_run.publish_pipeline(
    name="Inception_v3_scoring", description="Batch scoring using Inception v3 model", version="1.0")

published_pipeline
```

To run the pipeline from the REST endpoint, you need an OAuth2 Bearer-type authentication header. The following example uses interactive authentication (for illustration purposes), but for most production scenarios that require automated or headless authentication, use service principal authentication as [described in this article](#).

Service principal authentication involves creating an *App Registration* in *Azure Active Directory*. First, you generate a client secret, and then you grant your service principal *role access* to your machine learning workspace. Use the `ServicePrincipalAuthentication` class to manage your authentication flow.

Both `InteractiveLoginAuthentication` and `ServicePrincipalAuthentication` inherit from `AbstractAuthentication`. In both cases, use the `get_authentication_header()` function in the same way to fetch the header:

```
from azureml.core.authentication import InteractiveLoginAuthentication

interactive_auth = InteractiveLoginAuthentication()
auth_header = interactive_auth.get_authentication_header()
```

Get the REST URL from the `endpoint` property of the published pipeline object. You can also find the REST URL in your workspace in Azure Machine Learning studio.

Build an HTTP POST request to the endpoint. Specify your authentication header in the request. Add a JSON payload object that has the experiment name.

Make the request to trigger the run. Include code to access the `Id` key from the response dictionary to get the value of the run ID.

```
import requests

rest_endpoint = published_pipeline.endpoint
response = requests.post(rest_endpoint,
                         headers=auth_header,
                         json={"ExperimentName": "batch_scoring",
                               "ParameterAssignments": {"process_count_per_node": 6}})
run_id = response.json()["Id"]
```

Use the run ID to monitor the status of the new run. The new run takes another 10-15 min to finish.

The new run will look similar to the pipeline you ran earlier in the tutorial. You can choose not to view the full output.

```
from azureml.pipeline.core.run import PipelineRun
from azureml.widgets import RunDetails

published_pipeline_run = PipelineRun(ws.experiments["batch_scoring"], run_id)
RunDetails(published_pipeline_run).show()
```

Clean up resources

Don't complete this section if you plan to run other Azure Machine Learning tutorials.

Stop the compute instance

If you used a compute instance or Notebook VM, stop the VM when you aren't using it to reduce cost.

1. In your workspace, select **Compute**.
2. From the list, select the VM.
3. Select **Stop**.
4. When you're ready to use the server again, select **Start**.

Delete everything

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, in the left menu, select **Resource groups**.
2. In the list of resource groups, 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 then select **Delete**.

Next steps

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

- Built a pipeline with environment dependencies to run on a remote GPU compute resource.
- Created a scoring script to run batch predictions by using a pretrained Tensorflow model.
- Published a pipeline and enabled it to be run from a REST endpoint.

For more examples of how to build pipelines by using the machine learning SDK, see the [notebook repository](#).

Tutorial: Convert ML experiments to production Python code

12/23/2020 • 11 minutes to read • [Edit Online](#)

In this tutorial, you learn how to convert Jupyter notebooks into Python scripts to make it testing and automation friendly using the MLOpsPython code template and Azure Machine Learning. Typically, this process is used to take experimentation / training code from a Jupyter notebook and convert it into Python scripts. Those scripts can then be used testing and CI/CD automation in your production environment.

A machine learning project requires experimentation where hypotheses are tested with agile tools like Jupyter Notebook using real datasets. Once the model is ready for production, the model code should be placed in a production code repository. In some cases, the model code must be converted to Python scripts to be placed in the production code repository. This tutorial covers a recommended approach on how to export experimentation code to Python scripts.

In this tutorial, you learn how to:

- Clean nonessential code
- Refactor Jupyter Notebook code into functions
- Create Python scripts for related tasks
- Create unit tests

Prerequisites

- Generate the [MLOpsPython template](#) and use the `experimentation/Diabetes Ridge Regression Training.ipynb` and `experimentation/Diabetes Ridge Regression Scoring.ipynb` notebooks. These notebooks are used as an example of converting from experimentation to production. You can find these notebooks at <https://github.com/microsoft/MLOpsPython/tree/master/experimentation>.
- Install `nbconvert`. Follow only the installation instructions under section [Installing nbconvert](#) on the [Installation](#) page.

Remove all nonessential code

Some code written during experimentation is only intended for exploratory purposes. Therefore, the first step to convert experimental code into production code is to remove this nonessential code. Removing nonessential code will also make the code more maintainable. In this section, you'll remove code from the

`experimentation/Diabetes Ridge Regression Training.ipynb` notebook. The statements printing the shape of `x` and `y` and the cell calling `features.describe` are just for data exploration and can be removed. After removing nonessential code, `experimentation/Diabetes Ridge Regression Training.ipynb` should look like the following code without markdown:

```

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
import joblib
import pandas as pd

sample_data = load_diabetes()

df = pd.DataFrame(
    data=sample_data.data,
    columns=sample_data.feature_names)
df['Y'] = sample_data.target

X = df.drop('Y', axis=1).values
y = df['Y'].values

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}}

args = {
    "alpha": 0.5
}

reg_model = Ridge(**args)
reg_model.fit(data["train"]["X"], data["train"]["y"])

preds = reg_model.predict(data["test"]["X"])
mse = mean_squared_error(preds, y_test)
metrics = {"mse": mse}
print(metrics)

model_name = "sklearn_regression_model.pkl"
joblib.dump(value=reg, filename=model_name)

```

Refactor code into functions

Second, the Jupyter code needs to be refactored into functions. Refactoring code into functions makes unit testing easier and makes the code more maintainable. In this section, you'll refactor:

- The Diabetes Ridge Regression Training notebook([experimentation/Diabetes Ridge Regression Training.ipynb](#))
- The Diabetes Ridge Regression Scoring notebook([experimentation/Diabetes Ridge Regression Scoring.ipynb](#))

Refactor Diabetes Ridge Regression Training notebook into functions

In [experimentation/Diabetes Ridge Regression Training.ipynb](#), complete the following steps:

1. Create a function called `split_data` to split the data frame into test and train data. The function should take the dataframe `df` as a parameter, and return a dictionary containing the keys `train` and `test`.

Move the code under the *Split Data into Training and Validation Sets* heading into the `split_data` function and modify it to return the `data` object.

2. Create a function called `train_model`, which takes the parameters `data` and `args` and returns a trained model.

Move the code under the heading *Training Model on Training Set* into the `train_model` function and modify it to return the `reg_model` object. Remove the `args` dictionary, the values will come from the `args` parameter.

3. Create a function called `get_model_metrics`, which takes parameters `reg_model` and `data`, and evaluates the model then returns a dictionary of metrics for the trained model.

Move the code under the *Validate Model on Validation Set* heading into the `get_model_metrics` function and modify it to return the `metrics` object.

The three functions should be as follows:

```
# Split the dataframe into test and train data
def split_data(df):
    X = df.drop('Y', axis=1).values
    y = df['Y'].values

    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}}
    return data

# Train the model, return the model
def train_model(data, args):
    reg_model = Ridge(**args)
    reg_model.fit(data["train"]["X"], data["train"]["y"])
    return reg_model

# Evaluate the metrics for the model
def get_model_metrics(reg_model, data):
    preds = reg_model.predict(data["test"]["X"])
    mse = mean_squared_error(preds, data["test"]["y"])
    metrics = {"mse": mse}
    return metrics
```

Still in `experimentation/Diabetes Ridge Regression Training.ipynb`, complete the following steps:

1. Create a new function called `main`, which takes no parameters and returns nothing.
2. Move the code under the "Load Data" heading into the `main` function.
3. Add invocations for the newly written functions into the `main` function:

```
# Split Data into Training and Validation Sets
data = split_data(df)
```

```
# Train Model on Training Set
args = {
    "alpha": 0.5
}
reg = train_model(data, args)
```

```
# Validate Model on Validation Set
metrics = get_model_metrics(reg, data)
```

4. Move the code under the "Save Model" heading into the `main` function.

The `main` function should look like the following code:

```
def main():
    # Load Data
    sample_data = load_diabetes()

    df = pd.DataFrame(
        data=sample_data.data,
        columns=sample_data.feature_names)
    df['Y'] = sample_data.target

    # Split Data into Training and Validation Sets
    data = split_data(df)

    # Train Model on Training Set
    args = {
        "alpha": 0.5
    }
    reg = train_model(data, args)

    # Validate Model on Validation Set
    metrics = get_model_metrics(reg, data)

    # Save Model
    model_name = "sklearn_regression_model.pkl"

    joblib.dump(value=reg, filename=model_name)
```

At this stage, there should be no code remaining in the notebook that isn't in a function, other than import statements in the first cell.

Add a statement that calls the `main` function.

```
main()
```

After refactoring, `experimentation/Diabetes Ridge Regression Training.ipynb` should look like the following code without the markdown:

```

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
import pandas as pd
import joblib

# Split the dataframe into test and train data
def split_data(df):
    X = df.drop('Y', axis=1).values
    y = df['Y'].values

    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}}
    return data

# Train the model, return the model
def train_model(data, args):
    reg_model = Ridge(**args)
    reg_model.fit(data["train"]["X"], data["train"]["y"])
    return reg_model

# Evaluate the metrics for the model
def get_model_metrics(reg_model, data):
    preds = reg_model.predict(data["test"]["X"])
    mse = mean_squared_error(preds, data["test"]["y"])
    metrics = {"mse": mse}
    return metrics

def main():
    # Load Data
    sample_data = load_diabetes()

    df = pd.DataFrame(
        data=sample_data.data,
        columns=sample_data.feature_names)
    df['Y'] = sample_data.target

    # Split Data into Training and Validation Sets
    data = split_data(df)

    # Train Model on Training Set
    args = {
        "alpha": 0.5
    }
    reg = train_model(data, args)

    # Validate Model on Validation Set
    metrics = get_model_metrics(reg, data)

    # Save Model
    model_name = "sklearn_regression_model.pkl"

    joblib.dump(value=reg, filename=model_name)

main()

```

Refactor Diabetes Ridge Regression Scoring notebook into functions

In `experimentation/Diabetes Ridge Regression Scoring.ipynb`, complete the following steps:

1. Create a new function called `init`, which takes no parameters and return nothing.
2. Copy the code under the "Load Model" heading into the `init` function.

The `init` function should look like the following code:

```
def init():
    model_path = Model.get_model_path(
        model_name="sklearn_regression_model.pkl")
    model = joblib.load(model_path)
```

Once the `init` function has been created, replace all the code under the heading "Load Model" with a single call to `init` as follows:

```
init()
```

In `experimentation/Diabetes Ridge Regression.ipynb`, complete the following steps:

1. Create a new function called `run`, which takes `raw_data` and `request_headers` as parameters and returns a dictionary of results as follows:

```
{"result": result.tolist()}
```

2. Copy the code under the "Prepare Data" and "Score Data" headings into the `run` function.

The `run` function should look like the following code (Remember to remove the statements that set the variables `raw_data` and `request_headers`, which will be used later when the `run` function is called):

```
def run(raw_data, request_headers):
    data = json.loads(raw_data)["data"]
    data = numpy.array(data)
    result = model.predict(data)

    return {"result": result.tolist()}
```

Once the `run` function has been created, replace all the code under the "Prepare Data" and "Score Data" headings with the following code:

```
raw_data = '{"data":[[1,2,3,4,5,6,7,8,9,10],[10,9,8,7,6,5,4,3,2,1]]}'
request_header = {}
prediction = run(raw_data, request_header)
print("Test result: ", prediction)
```

The previous code sets variables `raw_data` and `request_header`, calls the `run` function with `raw_data` and `request_header`, and prints the predictions.

After refactoring, `experimentation/Diabetes Ridge Regression.ipynb` should look like the following code without the markdown:

```

import json
import numpy
from azureml.core.model import Model
import joblib

def init():
    model_path = Model.get_model_path(
        model_name="sklearn_regression_model.pkl")
    model = joblib.load(model_path)

def run(raw_data, request_headers):
    data = json.loads(raw_data)["data"]
    data = numpy.array(data)
    result = model.predict(data)

    return {"result": result.tolist()}

init()
test_row = '{"data":[[1,2,3,4,5,6,7,8,9,10],[10,9,8,7,6,5,4,3,2,1]]}'
request_header = {}
prediction = run(test_row, {})
print("Test result: ", prediction)

```

Combine related functions in Python files

Third, related functions need to be merged into Python files to better help code reuse. In this section, you'll be creating Python files for the following notebooks:

- The Diabetes Ridge Regression Training notebook([experimentation/Diabetes Ridge Regression Training.ipynb](#))
- The Diabetes Ridge Regression Scoring notebook([experimentation/Diabetes Ridge Regression Scoring.ipynb](#))

Create Python file for the Diabetes Ridge Regression Training notebook

Convert your notebook to an executable script by running the following statement in a command prompt, which uses the `nbconvert` package and the path of `experimentation/Diabetes Ridge Regression Training.ipynb`:

```
jupyter nbconvert -- to script "Diabetes Ridge Regression Training.ipynb" -output train
```

Once the notebook has been converted to `train.py`, remove any unwanted comments. Replace the call to `main()` at the end of the file with a conditional invocation like the following code:

```

if __name__ == '__main__':
    main()

```

Your `train.py` file should look like the following code:

```

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
import pandas as pd
import joblib

# Split the dataframe into test and train data
def split_data(df):
    X = df.drop('Y', axis=1).values
    y = df['Y'].values

    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}}
    return data

# Train the model, return the model
def train_model(data, args):
    reg_model = Ridge(**args)
    reg_model.fit(data["train"]["X"], data["train"]["y"])
    return reg_model

# Evaluate the metrics for the model
def get_model_metrics(reg_model, data):
    preds = reg_model.predict(data["test"]["X"])
    mse = mean_squared_error(preds, data["test"]["y"])
    metrics = {"mse": mse}
    return metrics

def main():
    # Load Data
    sample_data = load_diabetes()

    df = pd.DataFrame(
        data=sample_data.data,
        columns=sample_data.feature_names)
    df['Y'] = sample_data.target

    # Split Data into Training and Validation Sets
    data = split_data(df)

    # Train Model on Training Set
    args = {
        "alpha": 0.5
    }
    reg = train_model(data, args)

    # Validate Model on Validation Set
    metrics = get_model_metrics(reg, data)

    # Save Model
    model_name = "sklearn_regression_model.pkl"

    joblib.dump(value=reg, filename=model_name)

if __name__ == '__main__':
    main()

```

`train.py` can now be invoked from a terminal by running `python train.py`. The functions from `train.py` can also be called from other files.

The `train_aml.py` file found in the `diabetes_regression/training` directory in the MLOpsPython repository calls the functions defined in `train.py` in the context of an Azure Machine Learning experiment run. The functions can also be called in unit tests, covered later in this guide.

Create Python file for the Diabetes Ridge Regression Scoring notebook

Convert your notebook to an executable script by running the following statement in a command prompt that which uses the `nbconvert` package and the path of `experimentation/Diabetes Ridge Regression Scoring.ipynb`:

```
jupyter nbconvert -- to script "Diabetes Ridge Regression.ipynb" -output score
```

Once the notebook has been converted to `score.py`, remove any unwanted comments. Your `score.py` file should look like the following code:

```
import json
import numpy
from azureml.core.model import Model
import joblib

def init():
    model_path = Model.get_model_path(
        model_name="sklearn_regression_model.pkl")
    model = joblib.load(model_path)

def run(raw_data, request_headers):
    data = json.loads(raw_data)["data"]
    data = numpy.array(data)
    result = model.predict(data)

    return {"result": result.tolist()}

init()
test_row = '{"data":[[1,2,3,4,5,6,7,8,9,10],[10,9,8,7,6,5,4,3,2,1]]}'
request_header = {}
prediction = run(test_row, request_header)
print("Test result: ", prediction)
```

The `model` variable needs to be global so that it's visible throughout the script. Add the following statement at the beginning of the `init` function:

```
global model
```

After adding the previous statement, the `init` function should look like the following code:

```
def init():
    global model

    # load the model from file into a global object
    model_path = Model.get_model_path(
        model_name="sklearn_regression_model.pkl")
    model = joblib.load(model_path)
```

Create unit tests for each Python file

Fourth, create unit tests for your Python functions. Unit tests protect code against functional regressions and make it easier to maintain. In this section, you'll be creating unit tests for the functions in `train.py`.

`train.py` contains multiple functions, but we'll only create a single unit test for the `train_model` function using the

Pytest framework in this tutorial. Pytest isn't the only Python unit testing framework, but it's one of the most commonly used. For more information, visit [Pytest](#).

A unit test usually contains three main actions:

- Arrange object - creating and setting up necessary objects
- Act on an object
- Assert what is expected

The unit test will call `train_model` with some hard-coded data and arguments, and validate that `train_model` acted as expected by using the resulting trained model to make a prediction and comparing that prediction to an expected value.

```
import numpy as np
from code.training.train import train_model

def test_train_model():
    # Arrange
    X_train = np.array([1, 2, 3, 4, 5, 6]).reshape(-1, 1)
    y_train = np.array([10, 9, 8, 8, 6, 5])
    data = {"train": {"X": X_train, "y": y_train}}

    # Act
    reg_model = train_model(data, {"alpha": 1.2})

    # Assert
    preds = reg_model.predict([[1], [2]])
    np.testing.assert_almost_equal(preds, [9.939393939394, 9.030303030303])
```

Next steps

Now that you understand how to convert from an experiment to production code, see the following links for more information and next steps:

- [MLOpsPython](#): Build a CI/CD pipeline to train, evaluate and deploy your own model using Azure Pipelines and Azure Machine Learning
- [Monitor Azure ML experiment runs and metrics](#)
- [Monitor and collect data from ML web service endpoints](#)

Tutorial: Create a classification model with automated ML in Azure Machine Learning

12/23/2020 • 11 minutes to read • [Edit Online](#)

In this tutorial, you learn how to create a simple classification model without writing a single line of code using automated machine learning 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 machine learning, 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.

For a time-series forecasting example, see [Tutorial: Demand forecasting & AutoML](#).

In this tutorial, you learn how to do the following tasks:

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

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 + **Create a resource**.

The screenshot shows the Microsoft Azure portal homepage. At the top, there are three tabs: 'Home - Microsoft Azure', 'Azure ML Workspace (Preview)', and 'Sign out'. Below the tabs is the Microsoft Azure logo and a search bar with the placeholder 'Search resources, services, and docs (G+/-)'. The user's email, 'sheril.gilley@gmail.com', is displayed in the top right corner.

Azure services

- Create a resource
- Virtual machines
- App Services
- Storage accounts
- SQL databases
- Azure Database for PostgreSQL
- Azure Cosmos DB
- Kubernetes services
- More services

Recent resources

NAME	TYPE	LAST VIEWED
Visual Studio Ultimate with MSDN	Subscription	30 min ago

Navigate

- Subscriptions
- Resource groups
- All resources
- Dashboard

Tools

- Microsoft Learn
- Azure Monitor
- Security Center

Cost Management

3. Use the search bar to find **Machine Learning**.
4. Select **Machine Learning**.
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 docs-ws . Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others.
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 .
Location	Select the location closest to your users and the data resources to create your workspace.
Workspace edition	Select Basic as the workspace type for this tutorial. The workspace type determines the features to which you'll have access and pricing. Everything in this tutorial can be performed with either a Basic or Enterprise workspace.

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

WARNING

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

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

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

IMPORTANT

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

Get started in Azure Machine Learning 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 top navigation bar includes a gear icon, a copy icon, a question mark icon, and a smiley face icon. The left sidebar has a 'New' button, a 'Home' link, and a 'Author' section containing 'Notebooks', 'Automated ML' (which is highlighted with a red box), 'Designer', and 'Assets'. Under 'Assets', there are 'Datasets', 'Experiments', 'Pipelines', 'Models', and 'Endpoints'. The main content area shows the 'automl_ws > Automated ML' path. It features a heading 'Automated ML' with a sub-instruction: '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 run' (also highlighted with a red box). A message states 'No recent Automated ML runs to display.' with a call to action 'Click "New Automated ML run" to create your first run' and a link 'Learn more on creating Automated ML runs'. At the bottom, there's a 'Documentation' section with three items: '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 run**.

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 **Browse**.
 - e. Choose the **bankmarketing_train.csv** file on your local computer. This is the file you downloaded as a [prerequisite](#).
 - f. Give your dataset a unique name and provide an optional description.
 - g. 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.
- h. 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

- i. The **Schema** form allows for further configuration of your data for this experiment. For this example, we don't make any selections. Select **Next**.
- j. 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.
- k. Select **Create** to complete the creation of your dataset.
- l. Select your dataset once it appears in the list.

m. Review the **Data preview** to ensure you didn't include `day_of_week` then, select **Close**.

n. Select **Next**.

Configure run

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 Run** 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 **+Create a new compute** and 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 **Virtual Machine** form to set up your compute.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
Virtual machine priority	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 [\(1\)](#)

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 * [\(1\)](#)
automl-compute1

Minimum number of nodes * [\(1\)](#)
1

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 * [\(1\)](#)
6

Idle seconds before scale down * [\(1\)](#)
120

Enable SSH access [\(1\)](#)

Advanced settings

Enable virtual network [\(1\)](#)

Assign a managed identity [\(1\)](#)

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

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

d. Select **Next**.

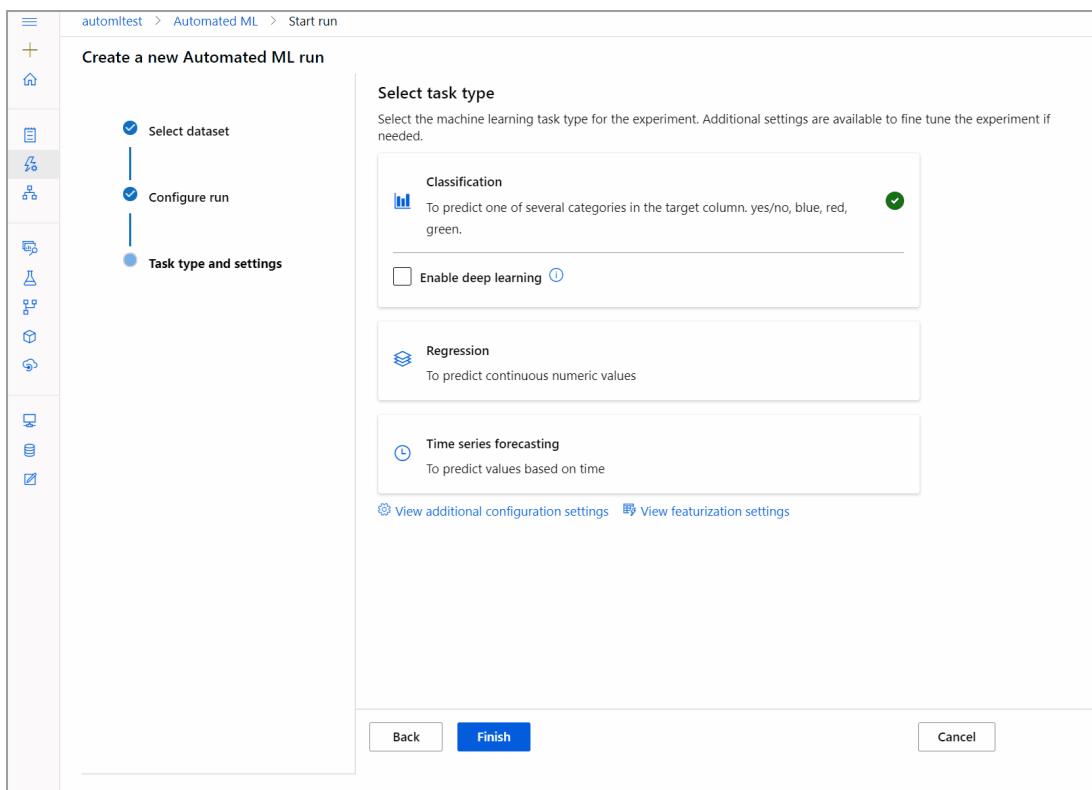
3. On the **Task type and settings** form, complete the setup for your automated ML experiment by specifying the machine learning task type and configuration settings.
- Select **Classification** as the machine learning task type.
 - Select **View additional configuration settings** and populate the 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
Exit criterion	If a criteria is met, the training job is stopped.	Training job time (hours): 1 Metric score threshold: None
Validation	Choose a cross-validation type and number of tests.	Validation type: k-fold cross-validation Number of validations: 2

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

Select **Save**.

- c. Select **View featurization settings**. For this example, select the toggle switch for the **day_of_week** feature, so as to not include it for featurization in this experiment.



Select **Save**.

4. Select **Finish** to run the experiment. The **Run Detail** screen opens with the **Run 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.

The screenshot shows the Azure Machine Learning studio interface. At the top, the navigation path is: automl_ws > Automated ML (preview) > new-experiment > Run 1. Below this, the title "Run 1" is followed by a green circle icon with a plus sign and the text "Running". There are "Refresh" and "Cancel" buttons. A horizontal menu bar includes "Details", "Data guardrails", "Models", "Outputs + Logs", "Child runs", and "Snapshot".

The main content area is divided into two sections:

- Properties** (left section):
 - Status: Running
 - Created: Jul 7, 2020 4:37 PM
 - Compute target: automl-compute
 - Run ID: AutoML_badd2cb1-c415-44c2-b0a8-1e6ee55fd8b8
 - Run number: 1
 - Script name: --
 - Created by: Nina Baccam
 - Input datasets:
 - Input name: input_data, ID: 442ec7297840
 - Output datasets: (partially visible)
- Run summary** (right section):
 - Task type: Classification [View all run settings](#)
 - Primary metric: AUC weighted
 - Run status: Running
 - Experiment name: new-experiment

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 run page by selecting **Run 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** button in the top-left.
3. Populate the **Deploy a model** pane as follows:

FIELD	VALUE
Deployment name	my-automl-deploy

FIELD	VALUE
Deployment description	My first automated machine learning experiment deployment
Compute type	Select Azure Compute Instance (ACI)
Enable authentication	Disable.
Use custom deployments	Disable. 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 **Run** 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 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 items like 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). It includes sections for Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Deployments, Policies, Properties, Locks, and Export template. The 'Overview' tab is 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 details about the subscription (Visual Studio Ultimate with MSDN) and deployment status (1 Succeeded). A table lists 14 items under 'Deployments', showing entries for 'myworkspace', 'my-workspace', and 'myworkspace0013141752' with their respective types: Machine Learning service workspace or 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 automated machine learning

12/23/2020 • 9 minutes to read • [Edit Online](#)

In this tutorial, you use automated machine learning, or automated ML, in the Azure Machine Learning studio to create a time-series forecasting model to predict rental demand for a bike sharing service.

For a classification model example, see [Tutorial: Create a classification model with automated ML in Azure Machine Learning](#).

In this tutorial, you 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.

Prerequisites

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

Get started in Azure Machine Learning 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 run**.

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 **Browse**.
- 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.	Use headers from the first file
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.
- l. Select **Next**.

Configure run

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 run** form as follows:

- a. Enter an experiment name:

b. Select **cnt** as the target column, what you want to predict. This column indicates the number of total bike share rentals.

c. Select **Create a new compute** and configure your compute target. Automated ML only supports Azure Machine Learning compute.

a. Populate the **Virtual Machine** form to set up your compute.

FIELD	DESCRIPTION	VALUE FOR TUTORIAL
Virtual machine priority	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.

d. 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 **forecast horizon** is the length of time into the future you want to predict. Deselect **Autodetect** and type **14** in the field.

4. 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
Validation	Choose a cross-validation type and number of tests.	Validation type: k-fold cross-validation Number of validations: 5
Concurrency	The maximum number of parallel iterations executed per iteration	Max concurrent iterations: 6

Select **Save**.

Run experiment

To run your experiment, select **Finish**. The **Run details** screen opens with the **Run status** at the top next to the run 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 run. 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. At the top, there is a breadcrumb navigation: automl.ws > Automated ML (preview) > new-experiment > Run 1. Below the navigation, the title "Run 1" is followed by a status indicator "Running". There are "Refresh" and "Cancel" buttons. A horizontal navigation bar includes "Details", "Data guardrails", "Models", "Outputs + Logs", "Child runs", and "Snapshot". The "Details" tab is currently selected. On the left, under the "Properties" section, there are several key details: Status (Running), Created (Jul 7, 2020 4:37 PM), Compute target (automl-compute), Run ID (AutoML_badd2cb1-c415-44c2-b0a8-1e6ee55fd8b8), Run number (1), Script name (--), and Created by (Nina Baccam). Under Input datasets, it lists an input dataset with ID 442ec7297840. On the right, the "Run summary" section shows Task type (Classification), Primary metric (AUC weighted), and Run status (Running). The Experiment name is listed as new-experiment.

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 run is complete, navigate back to parent run page by selecting **Run 1** at the top of your screen.

In the **Best model summary** section, **StackEnsemble** is considered the best model in the context of this experiment, 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 **StackEnsemble** 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 **Run** 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 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 'Create a resource' button and a list of services including Home, Dashboard, All services, Favorites (All resources, Resource groups, App Services, SQL databases, SQL data warehouses, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks), and Settings (Quickstart, Deployments, Policies, Properties, Locks, Export template). The main area shows the 'my-rg' Resource group details. The 'Overview' tab is 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 (Visual Studio Ultimate with MSDN), Deployment ID (xxxxxx-xxx-xxxx-xxxx), and Tags (Click here to add tags). A table lists 14 items under 'NAME' and 'TYPE': myworkspace (Machine Learning service workspace), my-workspace (Machine Learning service workspace), and myworkspace0013141752 (Application Insights). There are also filters for name, type, and location.

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: Predict automobile price with the designer

12/23/2020 • 12 minutes to read • [Edit Online](#)

In this two-part tutorial, you learn how to use the Azure Machine Learning designer to train and deploy a machine learning model that predicts the price of any car. The designer is a drag-and-drop tool that lets you create machine learning models without a single line of code.

In part one of the tutorial, you'll 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'll 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

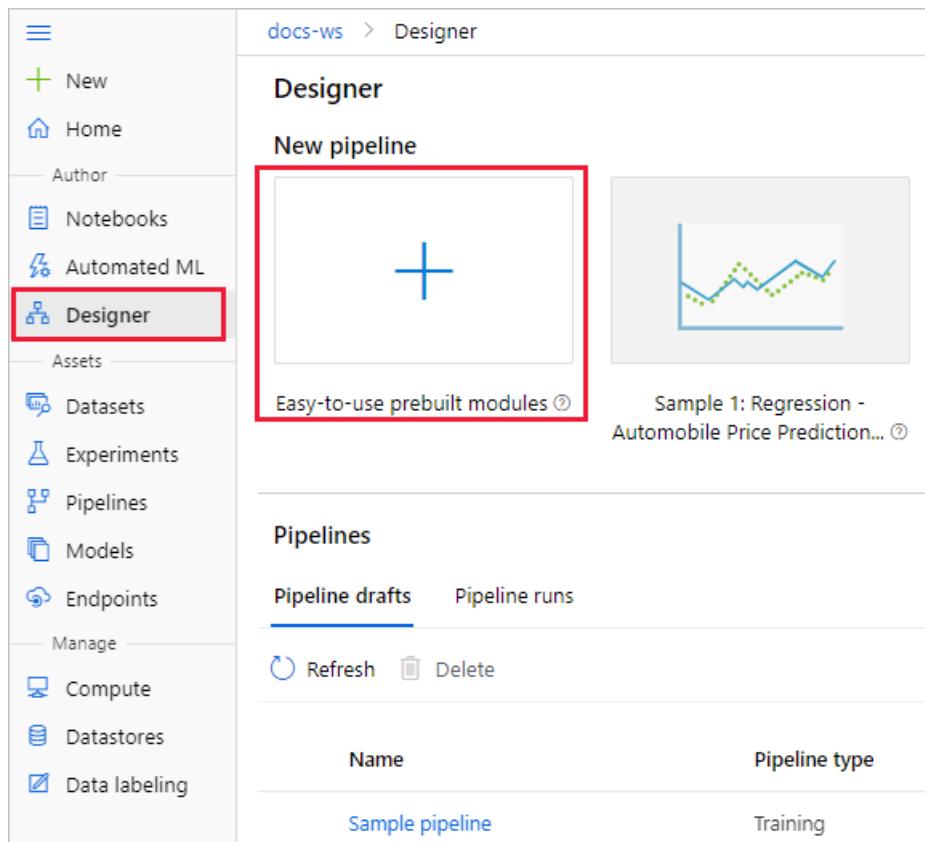
You need an Azure Machine Learning workspace to use the designer. The workspace is the top-level resource for Azure Machine Learning, it provides a centralized place to work with all the artifacts you create in Azure Machine Learning. For instruction on creating a workspace, see [Create and manage Azure Machine Learning workspaces](#).

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 modules.
4. At the top of the canvas, select the default pipeline name **Pipeline-Created-on**. Rename it to *Automobile price prediction*. The name doesn't need to be unique.

Set the default compute target

A pipeline runs 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 runs.

You can set a **Default compute target** for the entire pipeline, which will tell every module to use the same compute target by default. However, you can specify compute targets on a per-module basis.

1. Next to the pipeline name, select the **Gear icon** at the top of the canvas to open the **Settings** pane.
2. In the **Settings** pane to the right of the canvas, select **Select compute target**.

If you already have an available compute target, you can select it to run this pipeline.

NOTE

The designer can only run training experiments on Azure Machine Learning Compute but other compute targets won't be shown.

3. Enter a name for the compute resource.

4. Select **Save**.

NOTE

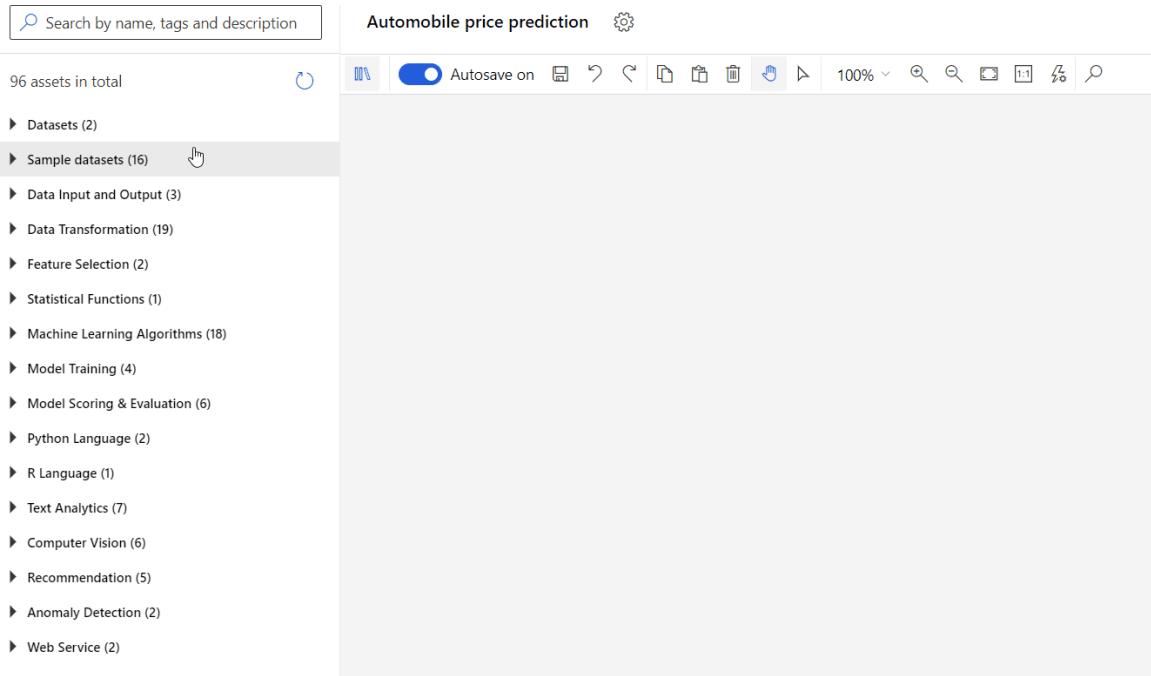
It takes approximately five minutes to create a compute resource. After the resource is created, you can reuse it and skip this wait time for future runs.

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 modules. Select **Sample datasets** to view the available sample datasets.
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 **Visualize**.
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.

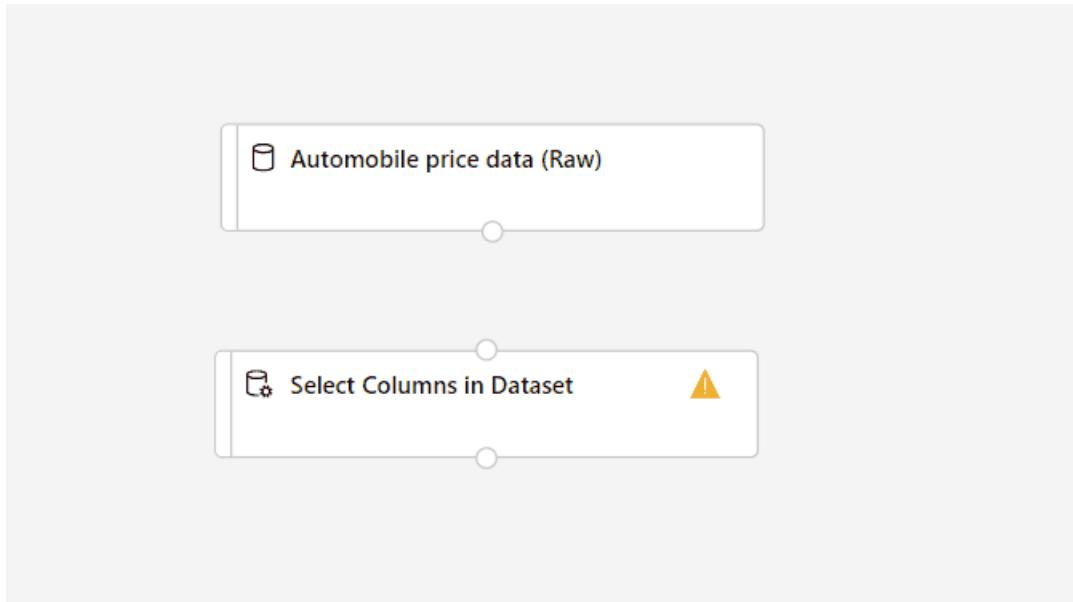
Remove a column

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 will exclude that column from the model altogether.

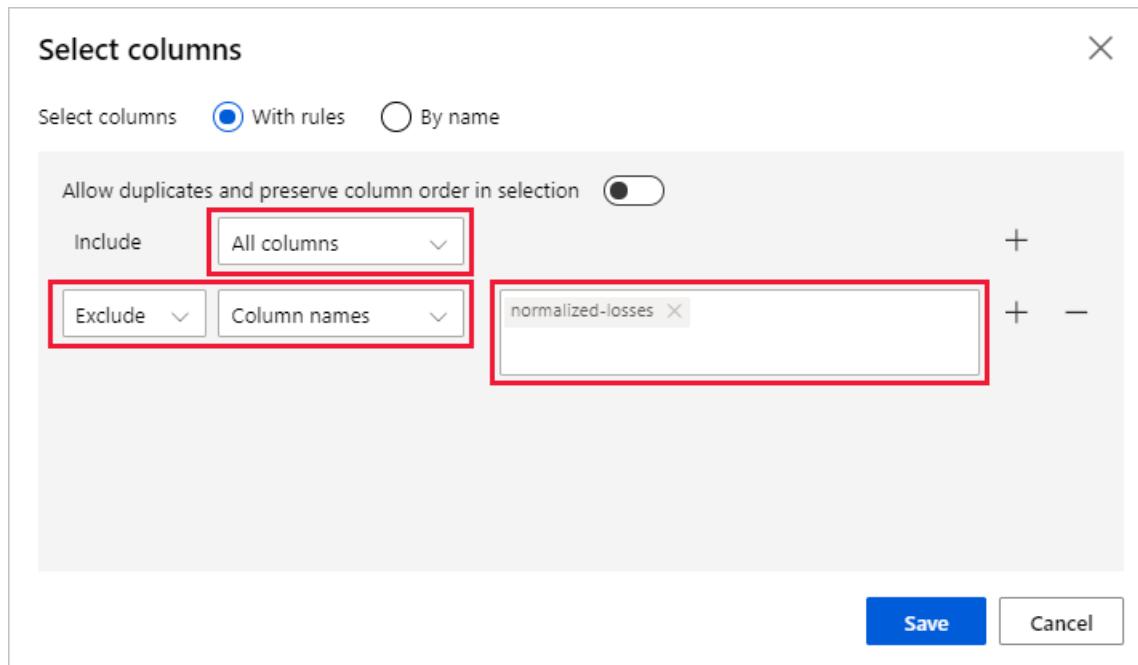
1. In the module palette to the left of the canvas, expand the **Data Transformation** section and find the **Select Columns in Dataset** module.
2. Drag the **Select Columns in Dataset** module onto the canvas. Drop the module below the dataset module.
3. Connect the **Automobile price data (Raw)** dataset to the **Select Columns in Dataset** module. 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 module.

TIP

You create a flow of data through your pipeline when you connect the output port of one module to an input port of another.



4. Select the **Select Columns in Dataset** module.
5. In the module details pane to the right of the canvas, select **Edit column**.
6. Expand the **Column names** drop down next to **Include**, and select **All columns**.
7. Select the **+** to add a new rule.
8. From the drop-down menus, select **Exclude** and **Column names**.
9. Enter *normalized-losses* in the text box.
10. In the lower right, select **Save** to close the column selector.



11. Select the **Select Columns in Dataset** module.
12. In the module details pane to the right of the canvas, 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

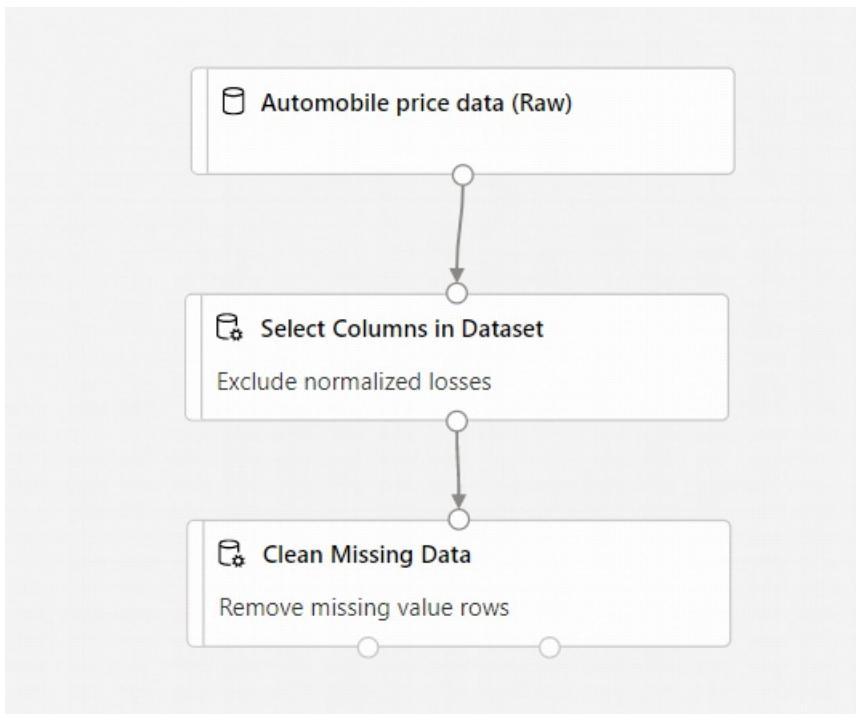
Your dataset still has missing values after you remove the **normalized-losses** column. You can remove the remaining missing data by using the **Clean Missing Data** module.

TIP

Cleaning the missing values from input data is a prerequisite for using most of the modules in the designer.

1. In the module palette to the left of the canvas, expand the section **Data Transformation**, and find the **Clean Missing Data** module.
2. Drag the **Clean Missing Data** module to the pipeline canvas. Connect it to the **Select Columns in Dataset** module.
3. Select the **Clean Missing Data** module.
4. In the module details pane to the right of the canvas, select **Edit Column**.
5. In the **Columns to be cleaned** window that appears, expand the drop-down menu next to **Include**. Select **All columns**
6. Select **Save**
7. In the module details pane to the right of the canvas, select **Remove entire row** under **Cleaning mode**.
8. In the module details pane to the right of the canvas, select the **Comment** 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 modules in place to process the data, you can set up the training modules.

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 will 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 module palette, expand the section **Data Transformation** and find the **Split Data** module.
2. Drag the **Split Data** module to the pipeline canvas.
3. Connect the left port of the **Clean Missing Data** module to the **Split Data** module.

IMPORTANT

Be sure that the left output ports 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** module.
 5. In the module details pane to the right of the canvas, 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.
6. In the module details pane to the right of the canvas, select the **Comment** box, and enter *Split the dataset into training set (0.7) and test set (0.3)*.

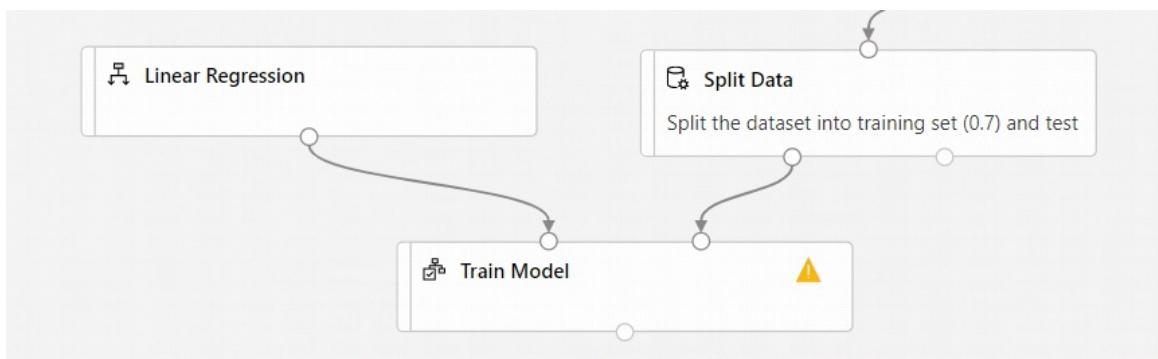
Train the model

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 module palette, expand **Machine Learning Algorithms**.
This option displays several categories of modules that you can use to initialize learning algorithms.
2. Select **Regression > Linear Regression**, and drag it to the pipeline canvas.
3. In the module palette, expand the section **Module training**, and drag the **Train Model** module to the canvas.
4. Connect the output of the **Linear Regression** module to the left input of the **Train Model** module.
5. Connect the training data output (left port) of the **Split Data** module to the right input of the **Train Model** module.

IMPORTANT

Be sure that the left output ports of **Split Data** connects to **Train Model**. The left port contains the training set. The right port contains the test set.

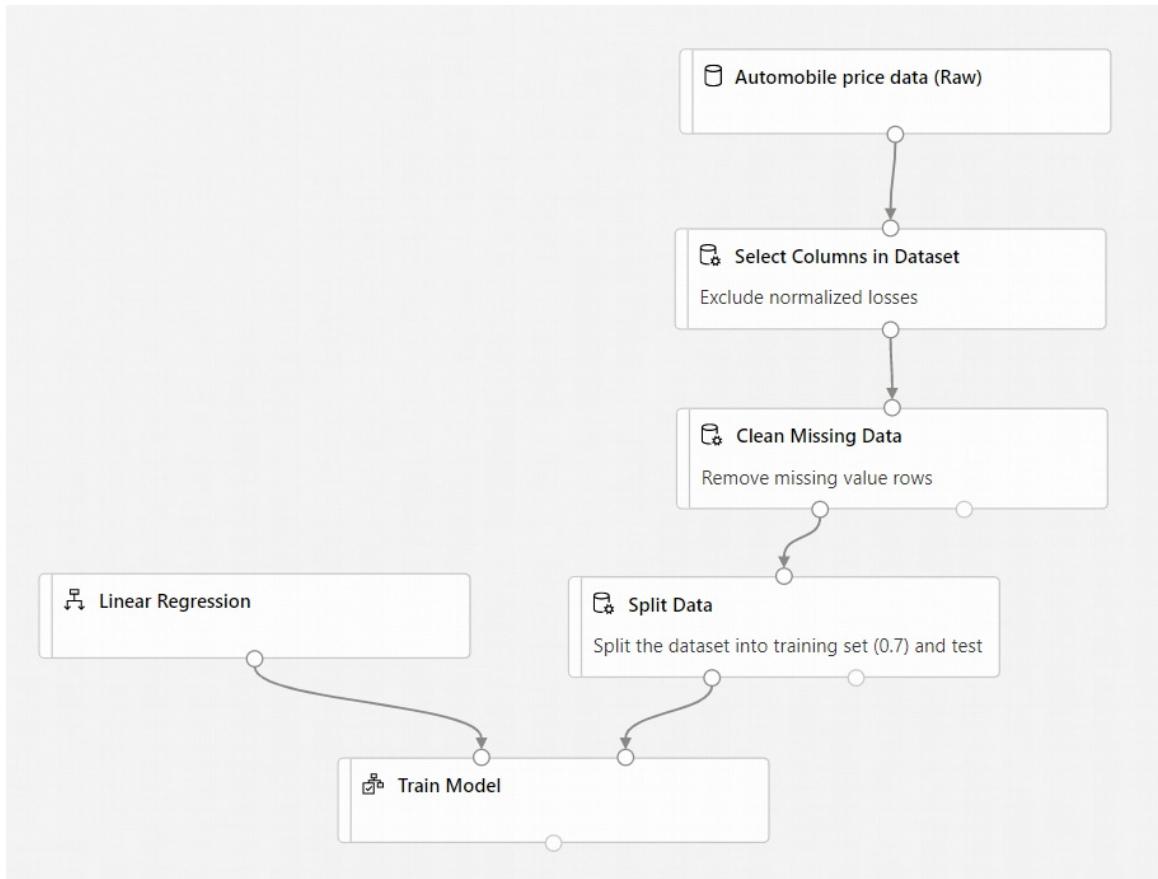


6. Select the **Train Model** module.
7. In the module details pane to the right of the canvas, select **Edit column selector**.
8. In the **Label column** dialog box, expand the drop-down menu and select **Column names**.
9. In the text box, enter *price* to specify the value that your model is going 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 module

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.

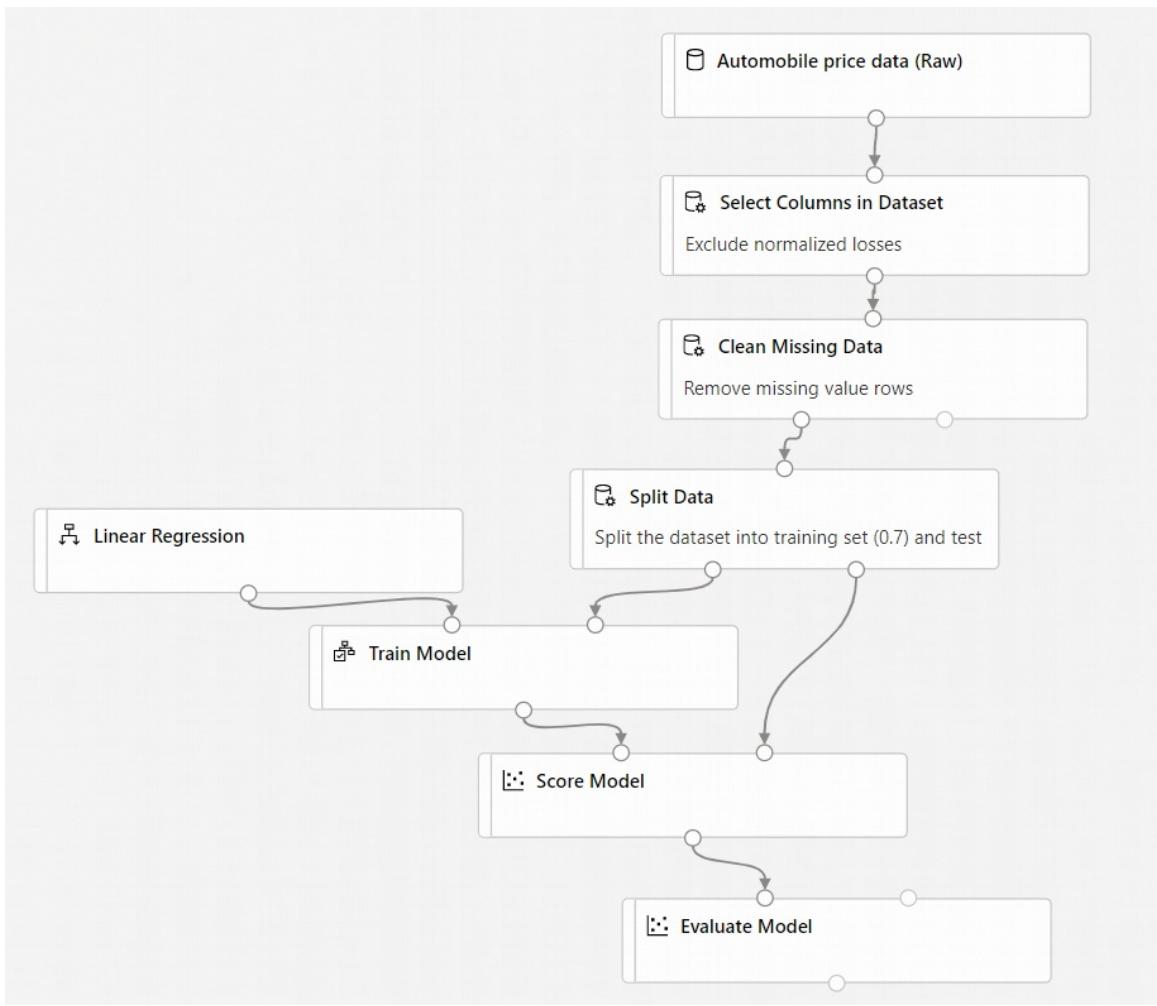
1. Enter *score model* in the search box to find the **Score Model** module. Drag the module to the pipeline canvas.
2. Connect the output of the **Train Model** module to the left input port of **Score Model**. Connect the test data output (right port) of the **Split Data** module to the right input port of **Score Model**.

Add the Evaluate Model module

Use the **Evaluate Model** module to evaluate how well your model scored the test dataset.

1. Enter *evaluate* in the search box to find the **Evaluate Model** module. Drag the module to the pipeline canvas.
2. Connect the output of the **Score Model** module 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 run to train your machine learning model. You can submit a valid pipeline run 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 run** dialog box, select **Create new**.

NOTE

Experiments group similar pipeline runs together. If you run a pipeline multiple times, you can select the same experiment for successive runs.

- a. Enter a descriptive name for **New experiment Name**.

- b. Select **Submit**.

You can view run status and details at the top right of the canvas.

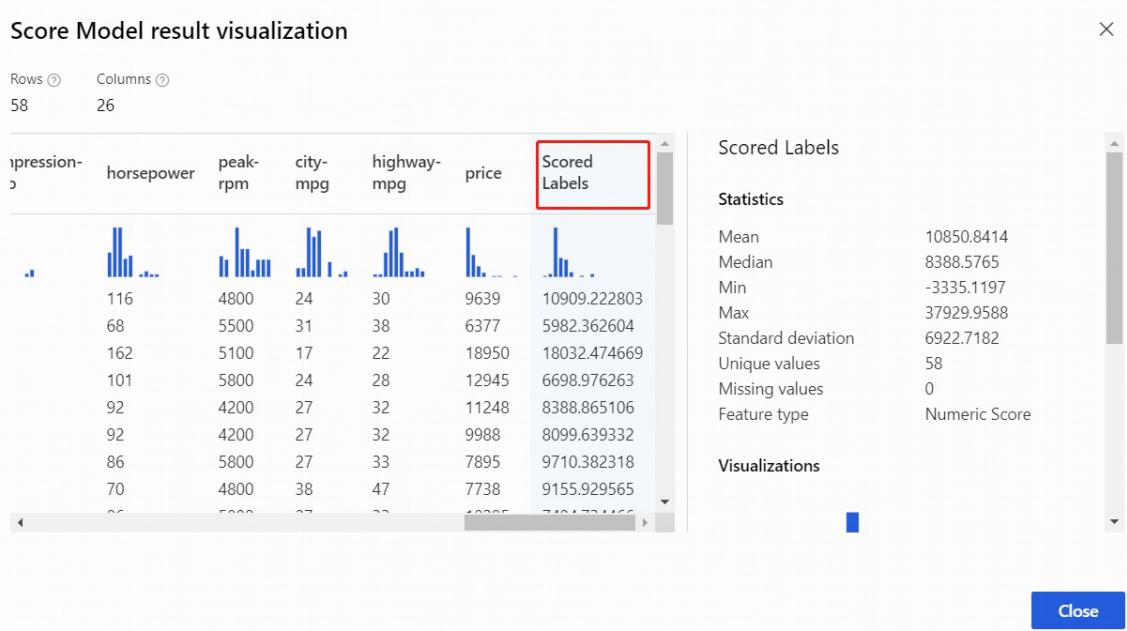
If this is the first run, it may take up to 20 minutes for your pipeline to finish running. The default compute settings have a minimum node size of 0, which means that the designer must allocate resources after being idle. Repeated pipeline runs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each module to further improve efficiency.

View scored labels

After the run completes, you can view the results of the pipeline run. First, look at the predictions generated by the regression model.

1. Right-click the **Score Model** module, and select **Visualize** 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** to see how well the trained model performed on the test dataset.

1. Right-click the **Evaluate Model** module and select **Visualize** 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](#).

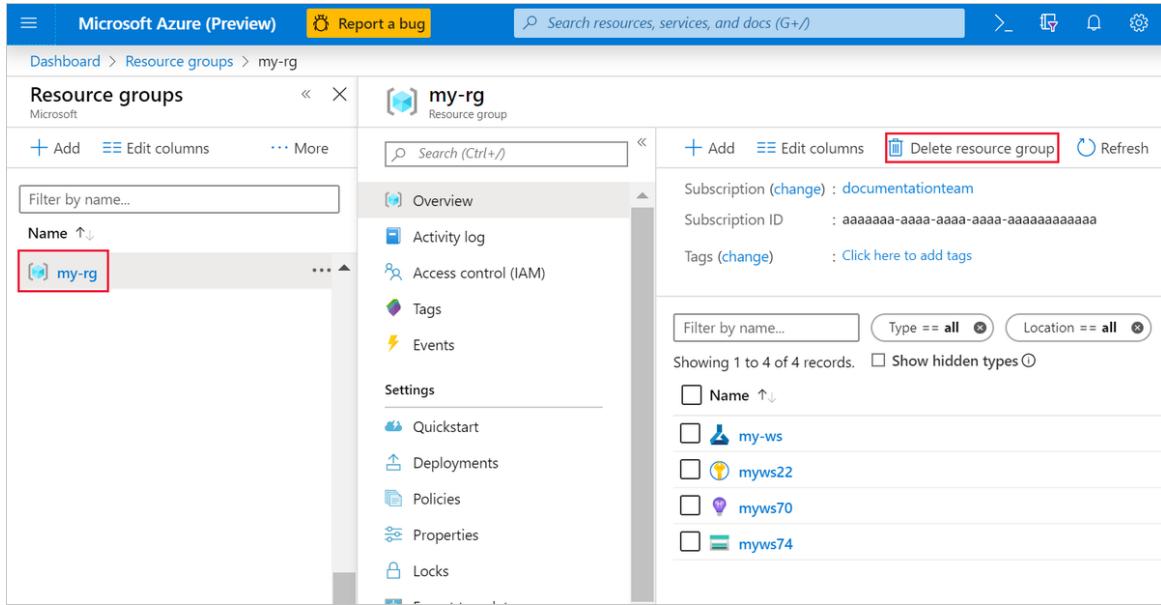
IMPORTANT

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 Azure portal interface. On the left, there's a sidebar with 'Resource groups' selected. The main content area shows a list of resource groups under 'my-rg'. One item, 'my-rg', is highlighted with a red box. On the far right, there's a detailed view of the selected resource group, including its subscription information ('Subscription (change) : documentationteam'), tags ('Tags (change) : Click here to add tags'), and a list of resources ('my-ws', 'myws22', 'myws70', 'myws74'). The 'Delete resource group' button 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 navigation sidebar with sections like Home, Author, Automated ML, Designer, Notebooks, Assets, Datasets, Experiments, Pipelines, Models, Endpoints, Manage, and Compute. The Compute section is highlighted with a red box. The main area is titled 'Compute' and has tabs for Compute Instances, Training Clusters, Inference Clusters (which is selected and underlined), and Attached Compute. Below these tabs is a search bar labeled 'Search to filter items...'. There's a row of buttons: '+ New', 'Refresh', 'Delete' (which is also highlighted with a red box), and three dots. A table below lists datasets, with one entry for 'aks-compute' (Kubernetes Serv...). At the bottom, there are navigation arrows for 'Prev' and 'Next'.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the Azure Machine Learning Datasets blade. The left sidebar has sections like 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 area shows a dataset named 'TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1'. The 'Details' tab is selected. In the top navigation bar, there are buttons for Refresh, Generate profile, Unregister (which is highlighted with a red box), and New version. Below the navigation, there are two main sections: 'Attributes' and 'Tags'. The 'Attributes' section contains fields for Properties, Description (with a note about automatic promotion), 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'. On the right, there's a 'Sample usage' section with some Python code:

```
# azureml-core of version 1.0.72 or higher is required
from azureml.core import Workspace, Dataset

subscription_id = 'ee85ed72-b2b6-48f6-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: Deploy a machine learning model with the designer

12/23/2020 • 4 minutes to read • [Edit Online](#)

You can deploy the predictive model developed in [part one of the tutorial](#) to give others a chance to use it. In part one, you trained your model. Now, it's time to generate new predictions based on user input. In this part of the tutorial, you will:

- 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

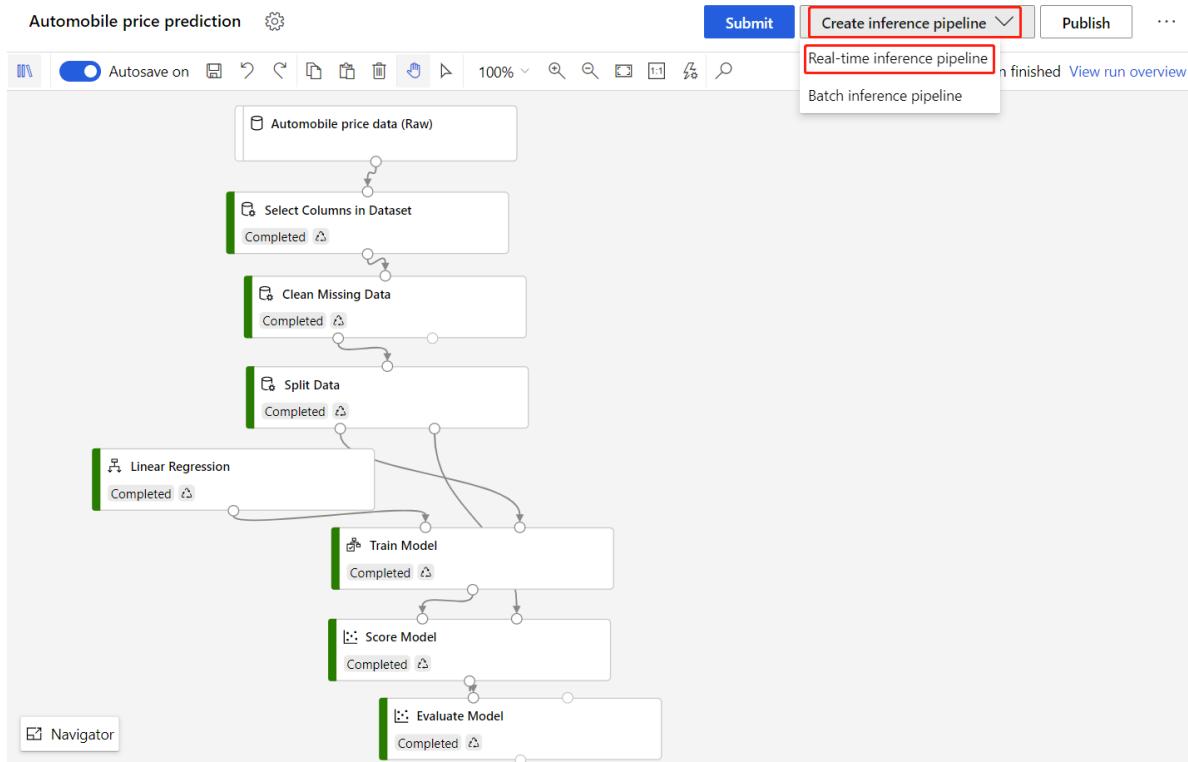
If you do not see graphical elements mentioned in this document, such as buttons in studio or designer, you may not have the right level of permissions to the workspace. Please contact your Azure subscription administrator to verify that you have been granted the correct level of access. For more information, see [Manage users and roles](#).

Create a real-time inference pipeline

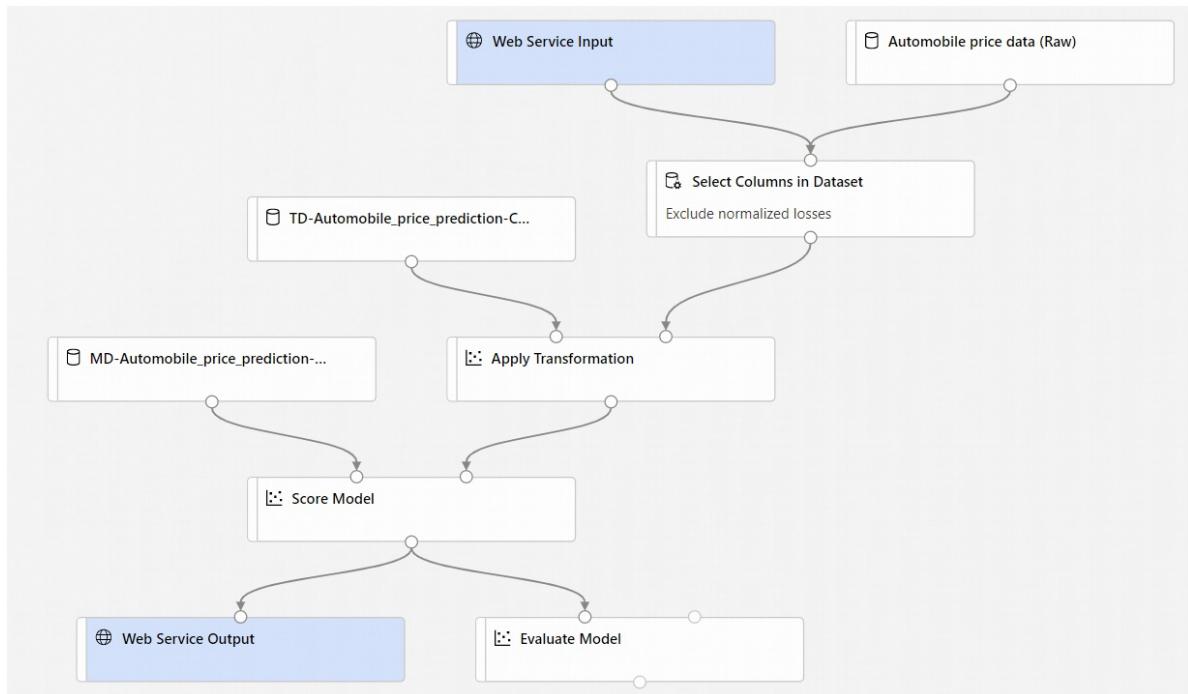
To deploy your pipeline, you must first convert the training pipeline into a real-time inference pipeline. This process removes training modules and adds web service inputs and outputs to handle requests.

Create a real-time inference pipeline

1. Above the pipeline canvas, select **Create inference pipeline > Real-time inference pipeline**.



Your pipeline should now look like this:



When you select **Create inference pipeline**, several things happen:

- The trained model is stored as a **Dataset** module in the module palette. You can find it under **My Datasets**.
- Training modules 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** modules are added. These modules 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 training data used to create the predictive pipeline. In this scenario, price is included in the schema. However, price isn't used as a factor during prediction.

2. Select **Submit**, and use the same compute target and experiment that you used in part one.

If this is the first run, 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 runs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each module to further improve efficiency.

3. Select **Deploy**.

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 'Compute' page in the Azure Machine Learning studio. The left sidebar has sections for Home, Author, Automated ML, Designer, Notebooks, Assets, Datasets, Experiments, Pipelines, Models, Endpoints, and Manage. Under Manage, the 'Compute' option is highlighted with a red box. The main area is titled 'Compute' and contains tabs for Compute Instances, Training Clusters, Inference Clusters (which is underlined and highlighted with a red box), and Attached Compute. Below the tabs are buttons for '+ New', 'Refresh', 'Delete', and 'Detach'. A table lists an inference cluster named 'aks-compute' of type 'Kubernetes Service' with a 'Succeeded' provisioning state. Navigation arrows for 'Prev' and 'Next' are at the bottom right of the table.

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.
4. Select **Deploy**.

Set up real-time endpoint

Deploy new real-time endpoint Replace an existing real-time endpoint

Name * (i) (o)

Description (i)

Compute type * (i)
 (v)

Compute name * (i)
 (v)

> Advanced (v)

Deploy Cancel

A success notification above the canvas appears after deployment finishes. It might take a few minutes.

View 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.

2. In the **Details** tab, you can see more information such as the REST URI, status, and tags.
3. In the **Consume** tab, you can find security keys and set authentication methods.
4. In the **Deployment logs** tab, you can find the detailed deployment logs of your real-time endpoint.

For more information on consuming your web service, see [Consume a model deployed as a webservice](#)

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 interface. On the left, the 'Resource groups' blade is open, displaying a list of resource groups. One group, 'my-rg', is highlighted with a red box. On the right, the details for 'my-rg' are shown, including its subscription information ('Subscription (change) : documentationteam'), subscription ID, and tags. The 'Delete resource group' button 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 navigation sidebar with sections like Home, Author, Automated ML, Designer, Notebooks, Assets, Datasets, Experiments, Pipelines, Models, Endpoints, Manage, and Compute. The Compute section 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 these tabs is a toolbar with New, Refresh, Delete (which is also highlighted with a red box), and an ellipsis. There's also a search bar labeled 'Search to filter items...'. A table lists an inference cluster named 'aks-compute' with details: Name, Type (Kubernetes Service), Provisioning Status (Success), Creation Date (10/17/...), and Status (STAN...). 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 Azure Machine Learning studio dataset details page for 'TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1'. The sidebar on the left has sections like 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 dataset details: Attributes (Properties: File, Description: This is a dataset promoted by inference graph generation automatically on 11/12/...), Tags (CreatedByAMLStudio: true), and Sample usage code. The sample usage code is as follows:

```
# 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 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](#).

Tutorial: Create a labeling project for multi-class image classification

12/23/2020 • 8 minutes to read • [Edit Online](#)

This tutorial shows you how to manage the process of labeling (also referred to as tagging) images to be used as data for building machine learning models. Data labeling in Azure Machine Learning is in public preview.

If you want to train a machine learning model to classify images, you need hundreds or even thousands of images that are correctly labeled. Azure Machine Learning helps you manage the progress of your private team of domain experts as they label your data.

In this tutorial, you'll use images of cats and dogs. Since each image is either a cat or a dog, this is a *multi-class* labeling project. You'll learn how to:

- Create an Azure storage account and upload images to the account.
- Create an Azure Machine Learning workspace.
- Create a multi-class image labeling project.
- Label your data. Either you or your labelers can perform this task.
- Complete the project by reviewing and exporting the data.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#).

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 + **Create a resource**.

The screenshot shows the Microsoft Azure portal homepage. At the top, there are three tabs: 'Home - Microsoft Azure', 'Azure ML Workspace (Preview)', and 'Sign out'. Below the tabs is a search bar with the placeholder 'Search resources, services, and docs (G+/-)'. The main content area is titled 'Azure services' and includes icons for 'Create a resource', 'Virtual machines', 'App Services', 'Storage accounts', 'SQL databases', 'Azure Database for PostgreSQL', 'Azure Cosmos DB', 'Kubernetes services', and 'More services'. A section titled 'Recent resources' lists a single item: 'Visual Studio Ultimate with MSDN' (Subscription, last viewed 30 min ago). Below this is a 'Navigate' section with links for 'Subscriptions', 'Resource groups', 'All resources', and 'Dashboard'. A 'Tools' section contains links for 'Microsoft Learn', 'Azure Monitor', and 'Security Center'. At the bottom left, there is a 'Cost Management' link.

3. Use the search bar to find **Machine Learning**.
4. Select **Machine Learning**.
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 docs-ws . Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others.
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 .
Location	Select the location closest to your users and the data resources to create your workspace.
Workspace edition	Select Basic as the workspace type for this tutorial. The workspace type determines the features to which you'll have access and pricing. Everything in this tutorial can be performed with either a Basic or Enterprise workspace.

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

WARNING

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

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

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

Start a labeling project

Next you will manage the data labeling project in Azure Machine Learning studio, a consolidated 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.

Create a datastore

Azure Machine Learning datastores are used to store connection information, like your subscription ID and token authorization. Here you use a datastore to connect to the storage account that contains the images for this tutorial.

1. On the left side of your workspace, select **Datastores**.
2. Select **+ New datastore**.
3. Fill out the form with these settings:

FIELD	DESCRIPTION
Datastore name	Give the datastore a name. Here we use labeling_tutorial .
Datastore type	Select the type of storage. Here we use Azure Blob Storage , the preferred storage for images.
Account selection method	Select Enter manually .
URL	<code>https://azureopendatastorage.blob.core.windows.net/openimagescontainer</code>
Authentication type	Select SAS token .
Account key	<code>?sv=2019-02-02&ss=bfqt&srt=sco&sp=r&se=2025-03-25T04:51:17Z&st=2020-24T20:51:17Z&spr=https&sig=7D7SdKQidGT6pURQ9R4SUzWGxZ%2BH1NPCstoSRRVg</code>

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

Create a labeling project

Now that you have access to the data you want to have labeled, create your labeling project.

1. At the top of the page, select **Projects**.
2. Select **+ Add project**.

Project details

1. Use the following input for the **Project details** form:

FIELD	DESCRIPTION
Project name	Give your project a name. Here we'll use tutorial-cats-n-dogs .
Labeling task type	Select Image Classification Multi-class .

Select **Next** to continue creating the project.

Select or create a dataset

1. On the **Select or create a dataset** form, select the second choice, **Create a dataset**, then select the link **From datastore**.
2. Use the following input for the **Create dataset from datastore** form:
 - a. On the **Basic info** form, add a name, here we'll use **images-for-tutorial**. Add a description if you wish. Then select **Next**.
 - b. On the **Datastore selection** form, use the dropdown to select your **Previously created datastore**, for example **tutorial_images (Azure Blob Storage)**
 - c. Next, still on the **Datastore selection** form, select **Browse** and then select **MultiClass - DogsCats**. Select **Save** to use **/MultiClass - DogsCats** as the path.
 - d. Select **Next** to confirm details and then **Create** to create the dataset.
 - e. Select the circle next to the dataset name in the list, for example **images-for-tutorial**.
3. Select **Next** to continue creating the project.

Incremental refresh

If you plan to add new images to your dataset, incremental refresh will find these new images and add them to your project. When you enable this feature, the project will periodically check for new images. You won't be adding new images to the datastore for this tutorial, so leave this feature unchecked.

Select **Next** to continue.

Label classes

1. On the **Label classes** form, type a label name, then select **+ Add label** to type the next label. For this project, the labels are **Cat**, **Dog**, and **Uncertain**.
2. Select **Next** when have added all the labels.

Labeling instructions

1. On the **Labeling instructions** form, you can provide a link to a website that provides detailed instructions for your

labelers. We'll leave it blank for this tutorial.

2. You can also add a short description of the task directly on the form. Type **Labeling tutorial - Cats & Dogs**.
3. Select **Next**.
4. In the **ML assisted labeling** section, leave the checkbox unchecked. ML assisted labeling requires more data than you'll be using in this tutorial.
5. Select **Create project**.

This page doesn't automatically refresh. After a pause, manually refresh the page until the project's status changes to **Created**.

Start labeling

You have now set up your Azure resources, and configured a data labeling project. It's time to add labels to your data.

Tag the images

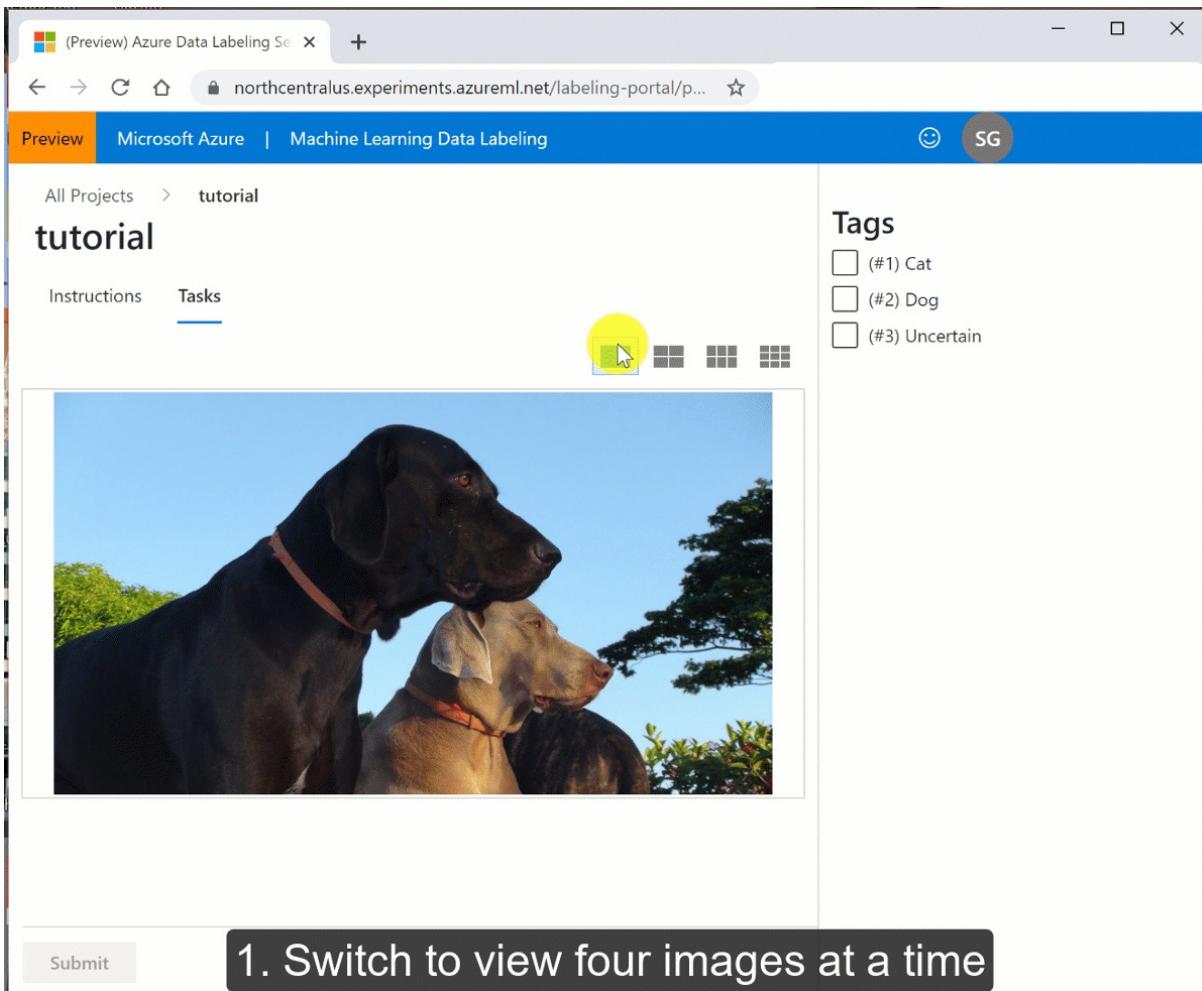
In this part of the tutorial, you'll switch roles from the *project administrator* to that of a *labeler*. Anyone who has contributor access to your workspace can become a labeler.

1. In [Machine Learning studio](#), select **Data labeling** on the left-hand side to find your project.
2. Select **Label** link for the project.
3. Read the instructions, then select **Tasks**.
4. Select a thumbnail image on the right to display the number of images you wish to label in one go. You must label all these images before you can move on. Only switch layouts when you have a fresh page of unlabeled data. Switching layouts clears the page's in-progress tagging work.
5. Select one or more images, then select a tag to apply to the selection. The tag appears below the image. Continue to select and tag all images on the page. To select all the displayed images simultaneously, select **Select all**. Select at least one image to apply a tag.

TIP

You can select the first nine tags by using the number keys on your keyboard.

6. Once all the images on the page are tagged, select **Submit** to submit these labels.



7. After you submit tags for the data at hand, Azure refreshes the page with a new set of images from the work queue.

Complete the project

Now you'll switch roles back to the *project administrator* for the labeling project.

As a manager, you may want to review the work of your labeler.

Review labeled data

1. In [Machine Learning studio](#), select **Data labeling** on the left-hand side to find your project.
2. Select the project name link.
3. The Dashboard shows you the progress of your project.
4. At the top of the page, select **Data**.
5. On the left side, select **Labeled data** to see your tagged images.
6. When you disagree with a label, select the image and then select **Reject** at the bottom of the page. The tags will be removed and the image is put back in the queue of unlabeled images.

Export labeled data

You can export the label data for Machine Learning experimentation at any time. Users often export multiple times and train different models, rather than wait for all the images to be labeled.

Image labels can be exported in [COCO format](#) or as an Azure Machine Learning dataset. The dataset format makes it easy to use for training in Azure Machine Learning.

1. In [Machine Learning studio](#), select **Data labeling** on the left-hand side to find your project.
2. Select the project name link.
3. Select **Export** and choose **Export as Azure ML Dataset**.

The status of the export appears just below the **Export** button.

- Once the labels are successfully exported, select **Datasets** on the left side to view the results.

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 the resources that you created, delete them so you don't incur any charges:

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

The screenshot shows the Microsoft Azure portal interface. The left sidebar is collapsed, showing options like 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES' (with 'All resources' selected), '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 - Microsoft Azure' and shows the 'Resource groups' blade for the 'my-rg' resource group. The 'Overview' tab is active. 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 also present. On the right, it says 'Deployments 1 Succeeded'. At the bottom, there's a table listing 14 items, with columns for 'NAME' and 'TYPE'. The items listed are: 'myworkspace' (Machine Learning service workspace), 'my-workspace' (Machine Learning service workspace), and 'myworkspace0013141752' (Application Insights).

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

Next steps

In this tutorial, you labeled images. Now use your labeled data:

[Train a machine learning image recognition model.](#)

Tutorial: Use R to create a machine learning model (preview)

12/23/2020 • 15 minutes to read • [Edit Online](#)

IMPORTANT

The Azure Machine Learning R SDK 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 tutorial you'll use the Azure Machine Learning R SDK (preview) to create a logistic regression model that predicts the likelihood of a fatality in a car accident. You'll see how the Azure Machine Learning cloud resources work with R to provide a scalable environment for training and deploying a model.

In this tutorial, you perform the following tasks:

- Create an Azure Machine Learning workspace
- Open RStudio from your workspace
- Clone <https://github.com/Azure/azureml-sdk-for-r> the files necessary to run this tutorial into your workspace
- Load data and prepare for training
- Upload data to a datastore so it is available for remote training
- Create a compute resource to train the model remotely
- Train a `caret` model to predict probability of fatality
- Deploy a prediction endpoint
- Test the model from R

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.

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 + **Create a resource**.

The screenshot shows the Microsoft Azure portal homepage. At the top, there are three tabs: 'Home - Microsoft Azure', 'Azure ML Workspace (Preview)', and 'Sign out'. Below the tabs is a search bar with the placeholder 'Search resources, services, and docs (G+)'. The main content area is titled 'Azure services' and includes a row of icons for 'Create a resource', 'Virtual machines', 'App Services', 'Storage accounts', 'SQL databases', 'Azure Database for PostgreSQL', 'Azure Cosmos DB', 'Kubernetes services', and 'More services'. Below this is a section titled 'Recent resources' with a table showing one item: 'Visual Studio Ultimate with MSDN' (Subscription type) last viewed 30 min ago. There are also 'Navigate' and 'Tools' sections with various links like 'Subscriptions', 'Resource groups', 'All resources', 'Dashboard', 'Microsoft Learn', 'Azure Monitor', 'Security Center', and 'Cost Management'.

3. Use the search bar to find **Machine Learning**.
4. Select **Machine Learning**.
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 docs-ws . Names must be unique across the resource group. Use a name that's easy to recall and to differentiate from workspaces created by others.
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 .
Location	Select the location closest to your users and the data resources to create your workspace.
Workspace edition	Select Basic as the workspace type for this tutorial. The workspace type determines the features to which you'll have access and pricing. Everything in this tutorial can be performed with either a Basic or Enterprise workspace.

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

WARNING

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

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

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

IMPORTANT

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

Open RStudio

This example uses a compute instance in your workspace for an install-free and pre-configured experience. Use [your own environment](#) if you prefer to have control over your environment, packages and dependencies on your own machine.

Use RStudio on an Azure ML compute instance to run this tutorial.

1. Select **Compute** on the left.
2. Add a compute resource if one does not already exist.
3. Once the compute is running, use the **RStudio** link to open RStudio.

Clone the sample vignettes

Clone the <https://github.com/Azure/azureml-sdk-for-r> GitHub repository for a copy of the vignette files you will run in this tutorial.

1. In RStudio, navigate to the "Terminal" tab and cd into the directory where you would like to clone the repository.
2. Run `git clone https://github.com/Azure/azureml-sdk-for-r` in the terminal to clone the repository.
3. In RStudio, navigate to the *vignettes* folder of the cloned *azureml-sdk-for-r* folder. Under *vignettes*, select the *train-and-deploy-first-model.Rmd* file to find vignette used in this tutorial. The additional files used for the vignette are located in the *train-and-deploy-first-model* subfolder. Once you've opened the vignette, set the working directory to the file's location via **Session > Set Working Directory > To Source File Location**.

IMPORTANT

The rest of this article contains the same content as you see in the *train-and-deploy-first-model.Rmd* file. If you are experienced with RMarkdown, feel free to use the code from that file. Or you can copy/paste the code snippets from there, or from this article into an R script or the command line.

Set up your development environment

The setup for your development work in this tutorial includes the following actions:

- Install required packages
- Connect to a workspace, so that your compute instance can communicate with remote resources
- Create an experiment to track your runs

- Create a remote compute target to use for training

Install required packages

The compute instance already has the latest version of the R SDK from CRAN installed. If you would like to install the development version from GitHub instead to pick up the latest bug fixes, please run the following:

```
remotes::install_github('https://github.com/Azure/azureml-sdk-for-r')
azuremlsdk::install_azureml()
```

WARNING

During the installation process, if you get the prompt "Would you like to install Miniconda? [Y/n]: ", please respond with "`n`" as the compute instance already has Anaconda installed and a Miniconda installation is not needed.

Now go ahead and import the `azuremlsdk` package.

```
library(azuremlsdk)
```

The training and scoring scripts (`accidents.R` and `accident_predict.R`) have some additional dependencies. If you plan on running those scripts locally, make sure you have those required packages as well.

Load your workspace

Instantiate a workspace object from your existing workspace. The following code will load the workspace details from the `config.json` file. You can also retrieve a workspace using [`get_workspace\(\)`](#).

```
ws <- load_workspace_from_config()
```

Create an experiment

An Azure ML experiment tracks a grouping of runs, typically from the same training script. Create an experiment to track the runs for training the caret model on the accidents data.

```
experiment_name <- "accident-logreg"
exp <- experiment(ws, experiment_name)
```

Create a compute target

By using Azure Machine Learning Compute (AmlCompute), a managed service, data scientists can train machine learning models on clusters of Azure virtual machines. Examples include VMs with GPU support. In this tutorial, you create a single-node AmlCompute cluster as your training environment. The code below creates the compute cluster for you if it doesn't already exist in your workspace.

You may need to wait a few minutes for your compute cluster to be provisioned if it doesn't already exist.

```

cluster_name <- "rcluster"
compute_target <- get_compute(ws, cluster_name = cluster_name)
if (is.null(compute_target)) {
  vm_size <- "STANDARD_D2_V2"
  compute_target <- create_aml_compute(workspace = ws,
                                         cluster_name = cluster_name,
                                         vm_size = vm_size,
                                         max_nodes = 1)
}
wait_for_provisioning_completion(compute_target)

```

Prepare data for training

This tutorial uses data from the US [National Highway Traffic Safety Administration](#) (with thanks to [Mary C. Meyer](#) and [Tremika Finney](#)). This dataset includes data from over 25,000 car crashes in the US, with variables you can use to predict the likelihood of a fatality. First, import the data into R and transform it into a new dataframe `accidents` for analysis, and export it to an `Rdata` file.

```

nassCDS <- read.csv("nassCDS.csv",
                     colClasses=c("factor","numeric","factor",
                                "factor","factor","numeric",
                                "factor","numeric","numeric",
                                "numeric","character","character",
                                "numeric","numeric","character"))

accidents <-
na.omit(nassCDS[,c("dead","dvcat","seatbelt","frontal","sex","ageOfOcc","yearVeh","airbag","occRole")])
accidents$frontal <- factor(accidents$frontal, labels=c("notfrontal","frontal"))
accidents$occRole <- factor(accidents$occRole)
accidents$dvcat <- ordered(accidents$dvcat,
                            levels=c("1-9km/h","10-24","25-39","40-54","55+"))

saveRDS(accidents, file="accidents.Rd")

```

Upload data to the datastore

Upload data to the cloud so that it can be accessed by your remote training environment. Each Azure Machine Learning workspace comes with a default datastore that stores the connection information to the Azure blob container that is provisioned in the storage account attached to the workspace. The following code will upload the accidents data you created above to that datastore.

```

ds <- get_default_datastore(ws)

target_path <- "accidentdata"
upload_files_to_datastore(ds,
                         list("./accidents.Rd"),
                         target_path = target_path,
                         overwrite = TRUE)

```

Train a model

For this tutorial, fit a logistic regression model on your uploaded data using your remote compute cluster. To submit a job, you need to:

- Prepare the training script
- Create an estimator
- Submit the job

Prepare the training script

A training script called `accidents.R` has been provided for you in the `train-and-deploy-first-model` directory.

Notice the following details **inside the training script** that have been done to leverage Azure Machine Learning for training:

- The training script takes an argument `-d` to find the directory that contains the training data. When you define and submit your job later, you point to the datastore for this argument. Azure ML will mount the storage folder to the remote cluster for the training job.
- The training script logs the final accuracy as a metric to the run record in Azure ML using `log_metric_to_run()`. The Azure ML SDK provides a set of logging APIs for logging various metrics during training runs. These metrics are recorded and persisted in the experiment run record. The metrics can then be accessed at any time or viewed in the run details page in [studio](#). See the [reference](#) for the full set of logging methods `log_*`.
- The training script saves your model into a directory named **outputs**. The `./outputs` folder receives special treatment by Azure ML. During training, files written to `./outputs` are automatically uploaded to your run record by Azure ML and persisted as artifacts. By saving the trained model to `./outputs`, you'll be able to access and retrieve your model file even after the run is over and you no longer have access to your remote training environment.

Create an estimator

An Azure ML estimator encapsulates the run configuration information needed for executing a training script on the compute target. Azure ML runs are run as containerized jobs on the specified compute target. By default, the Docker image built for your training job will include R, the Azure ML SDK, and a set of commonly used R packages. See the full list of default packages included here.

To create the estimator, define:

- The directory that contains your scripts needed for training (`source_directory`). All the files in this directory are uploaded to the cluster node(s) for execution. The directory must contain your training script and any additional scripts required.
- The training script that will be executed (`entry_script`).
- The compute target (`compute_target`), in this case the AmlCompute cluster you created earlier.
- The parameters required from the training script (`script_params`). Azure ML will run your training script as a command-line script with `Rscript`. In this tutorial you specify one argument to the script, the data directory mounting point, which you can access with `ds$path(target_path)`.
- Any environment dependencies required for training. The default Docker image built for training already contains the three packages (`caret`, `e1071`, and `optparse`) needed in the training script. So you don't need to specify additional information. If you are using R packages that are not included by default, use the estimator's `cran_packages` parameter to add additional CRAN packages. See the [estimator\(\)](#) reference for the full set of configurable options.

```
est <- estimator(source_directory = "train-and-deploy-first-model",
                  entry_script = "accidents.R",
                  script_params = list("--data_folder" = ds$path(target_path)),
                  compute_target = compute_target
                )
```

Submit the job on the remote cluster

Finally submit the job to run on your cluster. `submit_experiment()` returns a Run object that you then use to interface with the run. In total, the first run takes **about 10 minutes**. But for later runs, the same Docker image is reused as long as the script dependencies don't change. In this case, the image is cached and the container startup time is much faster.

```
run <- submit_experiment(exp, est)
```

You can view the run's details in RStudio Viewer. Clicking the "Web View" link provided will bring you to Azure Machine Learning studio, where you can monitor the run in the UI.

```
view_run_details(run)
```

Model training happens in the background. Wait until the model has finished training before you run more code.

```
wait_for_run_completion(run, show_output = TRUE)
```

You -- and colleagues with access to the workspace -- can submit multiple experiments in parallel, and Azure ML will take care of scheduling the tasks on the compute cluster. You can even configure the cluster to automatically scale up to multiple nodes, and scale back when there are no more compute tasks in the queue. This configuration is a cost-effective way for teams to share compute resources.

Retrieve training results

Once your model has finished training, you can access the artifacts of your job that were persisted to the run record, including any metrics logged and the final trained model.

Get the logged metrics

In the training script `accidents.R`, you logged a metric from your model: the accuracy of the predictions in the training data. You can see metrics in the [studio](#), or extract them to the local session as an R list as follows:

```
metrics <- get_run_metrics(run)
metrics
```

If you've run multiple experiments (say, using differing variables, algorithms, or hyperparameters), you can use the metrics from each run to compare and choose the model you'll use in production.

Get the trained model

You can retrieve the trained model and look at the results in your local R session. The following code will download the contents of the `./outputs` directory, which includes the model file.

```
download_files_from_run(run, prefix="outputs/")
accident_model <- readRDS("outputs/model.rds")
summary(accident_model)
```

You see some factors that contribute to an increase in the estimated probability of death:

- higher impact speed
- male driver
- older occupant
- passenger

You see lower probabilities of death with:

- presence of airbags
- presence seatbelts
- frontal collision

The vehicle year of manufacture does not have a significant effect.

You can use this model to make new predictions:

```
newdata <- data.frame( # valid values shown below
  dvcat="10-24",      # "1-9km/h" "10-24"    "25-39"    "40-54"    "55+"
  seatbelt="none",    # "none"     "belted"
  frontal="frontal", # "notfrontal" "frontal"
  sex="f",           # "f" "m"
  ageOfocc=16,        # age in years, 16-97
  yearVeh=2002,       # year of vehicle, 1955-2003
  airbag="none",      # "none"     "airbag"
  occRole="pass"      # "driver"   "pass"
)

## predicted probability of death for these variables, as a percentage
as.numeric(predict(accident_model,newdata, type="response"))*100
```

Deploy as a web service

With your model, you can predict the danger of death from a collision. Use Azure ML to deploy your model as a prediction service. In this tutorial, you will deploy the web service in [Azure Container Instances](#) (ACI).

Register the model

First, register the model you downloaded to your workspace with `register_model()`. A registered model can be any collection of files, but in this case the R model object is sufficient. Azure ML will use the registered model for deployment.

```
model <- register_model(ws,
  model_path = "outputs/model.rds",
  model_name = "accidents_model",
  description = "Predict probability of auto accident")
```

Define the inference dependencies

To create a web service for your model, you first need to create a scoring script (`entry_script`), an R script that will take as input variable values (in JSON format) and output a prediction from your model. For this tutorial, use the provided scoring file `accident_predict.R`. The scoring script must contain an `init()` method that loads your model and returns a function that uses the model to make a prediction based on the input data. See the [documentation](#) for more details.

Next, define an Azure ML **environment** for your script's package dependencies. With an environment, you specify R packages (from CRAN or elsewhere) that are needed for your script to run. You can also provide the values of environment variables that your script can reference to modify its behavior. By default, Azure ML will build the same default Docker image used with the estimator for training. Since the tutorial has no special requirements, create an environment with no special attributes.

```
r_env <- r_environment(name = "basic_env")
```

If you want to use your own Docker image for deployment instead, specify the `custom_docker_image` parameter. See the `r_environment()` reference for the full set of configurable options for defining an environment.

Now you have everything you need to create an **inference config** for encapsulating your scoring script and environment dependencies.

```
inference_config <- inference_config(
  entry_script = "accident_predict.R",
  source_directory = "train-and-deploy-first-model",
  environment = r_env)
```

Deploy to ACI

In this tutorial, you will deploy your service to ACI. This code provisions a single container to respond to inbound requests, which is suitable for testing and light loads. See [aci_webservice_deployment_config\(\)](#) for additional configurable options. (For production-scale deployments, you can also [deploy to Azure Kubernetes Service](#).)

```
aci_config <- aci_webservice_deployment_config(cpu_cores = 1, memory_gb = 0.5)
```

Now you deploy your model as a web service. Deployment **can take several minutes**.

```
aci_service <- deploy_model(ws,
  'accident-pred',
  list(model),
  inference_config,
  aci_config)

wait_for_deployment(aci_service, show_output = TRUE)
```

Test the deployed service

Now that your model is deployed as a service, you can test the service from R using [invoke_webservice\(\)](#). Provide a new set of data to predict from, convert it to JSON, and send it to the service.

```
library(jsonlite)

newdata <- data.frame( # valid values shown below
  dvcat="10-24",      # "1-9km/h" "10-24"    "25-39"    "40-54"    "55+"
  seatbelt="none",    # "none"     "belted"
  frontal="frontal", # "notfrontal" "frontal"
  sex="f",           # "f" "m"
  ageOfFocc=22,       # age in years, 16-97
  yearVeh=2002,       # year of vehicle, 1955-2003
  airbag="none",      # "none"     "airbag"
  occRole="pass"      # "driver"   "pass"
  )

prob <- invoke_webservice(aci_service, toJSON(newdata))
prob
```

You can also get the web service's HTTP endpoint, which accepts REST client calls. You can share this endpoint with anyone who wants to test the web service or integrate it into an application.

```
aci_service$scoring_uri
```

Clean up resources

Delete the resources once you no longer need them. Don't delete any resource you plan to still use.

Delete the web service:

```
delete_webservice(aci_service)
```

Delete the registered model:

```
delete_model(model)
```

Delete the compute cluster:

```
delete_compute(compute)
```

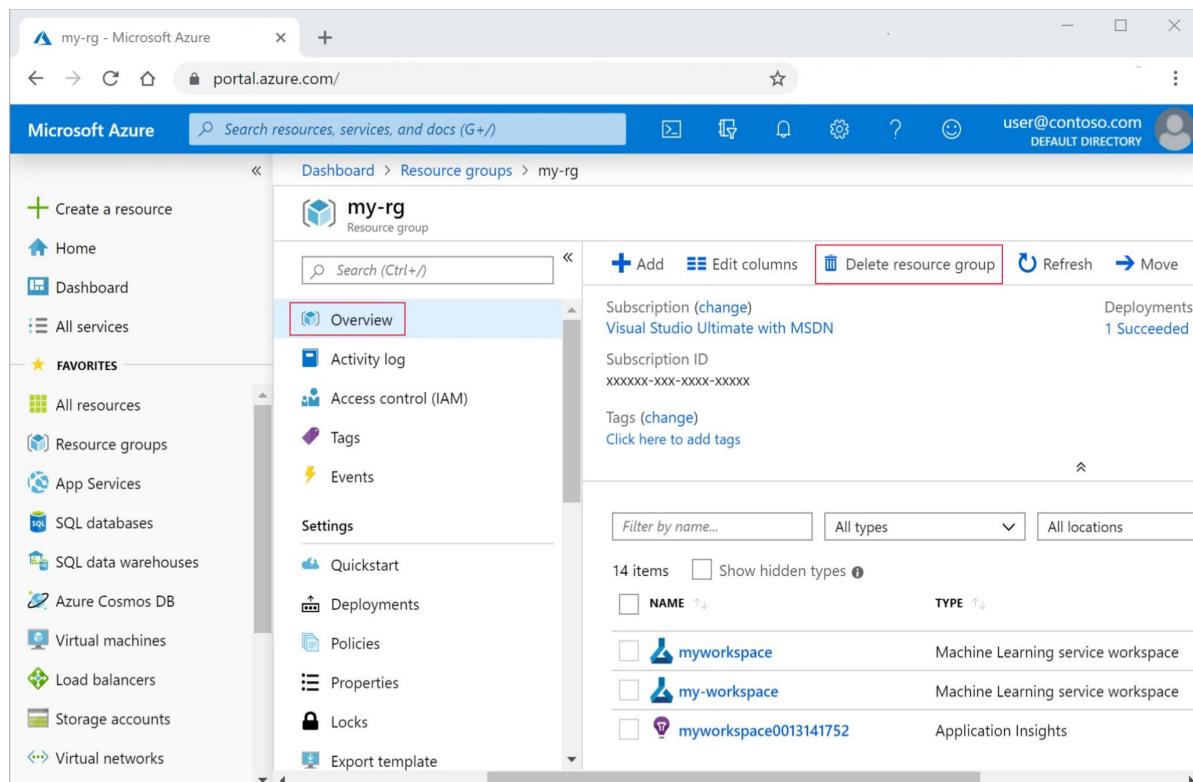
Delete everything

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 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**.



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

- Now that you've completed your first Azure Machine Learning experiment in R, learn more about the [Azure](#)

Machine Learning SDK for R.

- Learn more about Azure Machine Learning with R from the examples in the other *vignettes* folders.

Tutorial: Train and deploy a model from the CLI

12/23/2020 • 14 minutes to read • [Edit Online](#)

In this tutorial, you use the machine learning extension for the Azure CLI to train, register, and deploy a model.

The Python training scripts in this tutorial use [scikit-learn](#) to train a simple model. The focus of this tutorial is not on the scripts or the model, but the process of using the CLI to work with Azure Machine Learning.

Learn how to take the following actions:

- Install the machine learning extension
- Create an Azure Machine Learning workspace
- Create the compute resource used to train the model
- Define and register the dataset used to train the model
- Start a training run
- Register and download a model
- Deploy the model as a web service
- Score data using the web service

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.
- 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.

Download the example project

For this tutorial, download the <https://github.com/microsoft/MLOps> project. The files in the `examples/cli-train-deploy` directory are used by the steps in this tutorial.

To get a local copy of the files, either [download a .zip archive](#), or use the following Git command to clone the repository:

```
git clone https://github.com/microsoft/MLOps.git
```

Training files

The `examples/cli-train-deploy` directory from the project contains the following files, which are used when training a model:

- `.azureml\mnist.runconfig` : A **run configuration** file. This file defines the runtime environment needed to train the model. In this example, it also mounts the data used to train the model into the training environment.
- `scripts\train.py` : The training script. This file trains the model.
- `scripts\utils.py` : A helper file used by the training script.
- `.azureml\conda_dependencies.yml` : Defines the software dependencies needed to run the training script.
- `dataset.json` : The dataset definition. Used to register the MNIST dataset in the Azure Machine Learning workspace.

Deployment files

The repository contains the following files, which are used to deploy the trained model as a web service:

- `aciDeploymentConfig.yml` : A **deployment configuration** file. This file defines the hosting environment needed for the model.
- `inferenceConfig.json` : An **inference configuration** file. This file defines the software environment used by the service to score data with the model.
- `score.py` : A python script that accepts incoming data, scores it using the model, and then returns a response.
- `scoring-env.yml` : The conda dependencies needed to run the model and `score.py` script.
- `testdata.json` : A data file that can be used to test the deployed web service.

Connect to 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](#).

Install the machine learning extension

To install the machine learning extension, use the following command:

```
az extension add -n azure-cli-ml
```

If you get a message that the extension is already installed, use the following command to update to the latest version:

```
az extension update -n azure-cli-ml
```

Create a resource group

A resource group is a container of resources on the Azure platform. When working with the Azure Machine Learning, the resource group will contain your Azure Machine Learning workspace. It will also contain other Azure services used by the workspace. For example, if you train your model using a cloud-based compute resource, that

resource is created in the resource group.

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:

TIP

You should select a region where the 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:

```
{
  "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

To create a new workspace, use the following command. Replace <workspace-name> with the name you want to use for this workspace. Replace <resource-group-name> with the name of the resource group:

```
az ml workspace create -w <workspace-name> -g <resource-group-name>
```

The output of this command is similar to the following JSON:

```
{
  "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>"
}
```

Connect local project to workspace

From a terminal or command prompt, use the following commands change directories to the `cli-train-deploy` directory, then connect to your workspace:

```
cd ~/MLOps/examples/cli-train-deploy
az ml folder attach -w <workspace-name> -g <resource-group-name>
```

The output of this command is similar to the following JSON:

```
{
  "Experiment name": "model-training",
  "Project path": "/home/user/MLOps/examples/cli-train-deploy",
  "Resource group": "<resource-group-name>",
  "Subscription id": "<subscription-id>",
  "Workspace name": "<workspace-name>"
}
```

This command creates a `.azureml/config.json` file, which contains information needed to connect to your workspace. The rest of the `az ml` commands used in this tutorial will use this file, so you don't have to add the workspace and resource group to all commands.

Create the compute target for training

This example uses an Azure Machine Learning Compute cluster to train the model. To create a new compute cluster, use the following command:

```
az ml computetarget create amlcompute -n cpu-cluster --max-nodes 4 --vm-size Standard_D2_V2
```

The output of this command is similar to the following JSON:

```
{  
  "location": "<location>",  
  "name": "cpu-cluster",  
  "provisioningErrors": null,  
  "provisioningState": "Succeeded"  
}
```

This command creates a new compute target named `cpu-cluster`, with a maximum of four nodes. The VM size selected provides a VM with a GPU resource. For information on the VM size, see [VM types and sizes].

IMPORTANT

The name of the compute target (`cpu-cluster` in this case), is important; it is referenced by the `.azureml/mnist.runconfig` file used in the next section.

Define the dataset

To train a model, you can provide the training data using a dataset. To create a dataset from the CLI, you must provide a dataset definition file. The `dataset.json` file provided in the repo creates a new dataset using the MNIST data. The dataset it creates is named `mnist-dataset`.

To register the dataset using the `dataset.json` file, use the following command:

```
az ml dataset register -f dataset.json --skip-validation
```

The output of this command is similar to the following JSON:

```
{  
  "definition": [  
    "GetFiles"  
,  
    "registration": {  
      "description": "mnist dataset",  
      "id": "a13a4034-02d1-40bd-8107-b5d591a464b7",  
      "name": "mnist-dataset",  
      "tags": {  
        "sample-tag": "mnist"  
      },  
      "version": 1,  
      "workspace": "Workspace.create(name='myworkspace', subscription_id='mysubscriptionid',  
resource_group='myresourcegroup')"  
    },  
    "source": [  
      "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"  
    ]  
}
```

IMPORTANT

Copy the value of the `id` entry, as it is used in the next section.

To see a more comprehensive template for a dataset, use the following command:

```
az ml dataset register --show-template
```

Reference the dataset

To make the dataset available in the training environment, you must reference it from the runconfig file. The `.azureml/mnist.runconfig` file contains the following YAML entries:

```
# The arguments to the script file.
arguments:
- --data-folder
- DatasetConsumptionConfig:mnist

.....


# The configuration details for data.
data:
  mnist:
    # Data Location
    dataLocation:
      # the Dataset used for this run.
      dataset:
        # Id of the dataset.
        id: a13a4034-02d1-40bd-8107-b5d591a464b7
    # the DataPath used for this run.
    datapath:
      # Whether to create new folder.
      createOutputDirectories: false
    # The mode to handle
    mechanism: mount
    # Point where the data is download or mount or upload.
    environmentVariableName: mnist
    # relative path where the data is download or mount or upload.
    pathOnCompute:
      # Whether to overwrite the data if existing.
      overwrite: false
```

Change the value of the `id` entry to match the value returned when you registered the dataset. This value is used to load the data into the compute target during training.

This YAML results in the following actions during training:

- Mounts the dataset (based on the ID of the dataset) in the training environment, and stores the path to the mount point in the `mnist` environment variable.
- Passes the location of the data (mount point) inside the training environment to the script using the `--data-folder` argument.

The runconfig file also contains information used to configure the environment used by the training run. If you inspect this file, you'll see that it references the `cpu-compute` compute target you created earlier. It also lists the number of nodes to use when training (`"nodeCount": "4"`), and contains a `"condaDependencies"` section that lists the Python packages needed to run the training script.

TIP

While it is possible to manually create a runconfig file, the one in this example was created using the `generate-runconfig.py` file included in the repository. This file gets a reference to the registered dataset, creates a run config programatically, and then persists it to file.

For more information on run configuration files, see [Use compute targets for model training](#). For a complete JSON

reference, see the [runconfigschema.json](#).

Submit the training run

To start a training run on the `cpu-cluster` compute target, use the following command:

```
az ml run submit-script -c mnist -e myexperiment --source-directory scripts -t runoutput.json
```

This command specifies a name for the experiment (`myexperiment`). The experiment stores information about this run in the workspace.

The `-c mnist` parameter specifies the `.azureml/mnist.runconfig` file.

The `-t` parameter stores a reference to this run in a JSON file, and will be used in the next steps to register and download the model.

As the training run processes, it streams information from the training session on the remote compute resource. Part of the information is similar to the following text:

```
Predict the test set
Accuracy is 0.9185
```

This text is logged from the training script and displays the accuracy of the model. Other models will have different performance metrics.

If you inspect the training script, you'll notice that it also uses the alpha value when it stores the trained model to `outputs/sklearn_mnist_model.pkl`.

The model was saved to the `./outputs` directory on the compute target where it was trained. In this case, the Azure Machine Learning Compute instance in the Azure cloud. The training process automatically uploads the contents of the `./outputs` directory from the compute target where training occurs to your Azure Machine Learning workspace. It's stored as part of the experiment (`myexperiment` in this example).

Register the model

To register the model directly from the stored version in your experiment, use the following command:

```
az ml model register -n mymodel -f runoutput.json --asset-path "outputs/sklearn_mnist_model.pkl" -t registeredmodel.json
```

This command registers the `outputs/sklearn_mnist_model.pkl` file created by the training run as a new model registration named `mymodel`. The `--assets-path` references a path in an experiment. In this case, the experiment and run information are loaded from the `runoutput.json` file created by the training command. The `-t registeredmodel.json` creates a JSON file that references the new registered model created by this command, and is used by other CLI commands that work with registered models.

The output of this command is similar to the following JSON:

```
{  
    "createdTime": "2019-09-19T15:25:32.411572+00:00",  
    "description": "",  
    "experimentName": "myexperiment",  
    "framework": "Custom",  
    "frameworkVersion": null,  
    "id": "mymodel:1",  
    "name": "mymodel",  
    "properties": "",  
    "runId": "myexperiment_1568906070_5874522d",  
    "tags": "",  
    "version": 1  
}
```

Model versioning

Note the version number returned for the model. The version is incremented each time you register a new model with this name. For example, you can download the model and register it from a local file by using the following commands:

```
az ml model download -i "mymodel:1" -t .  
az ml model register -n mymodel -p "sklearn_mnist_model.pkl"
```

The first command downloads the registered model to the current directory. The file name is `sklearn_mnist_model.pkl`, which is the file referenced when you registered the model. The second command registers the local model (`-p "sklearn_mnist_model.pkl"`) with the same name as the previous registration (`mymodel`). This time, the JSON data returned lists the version as 2.

Deploy the model

To deploy a model, use the following command:

```
az ml model deploy -n myservice -m "mymodel:1" --ic inferenceConfig.json --dc aciDeploymentConfig.yml
```

NOTE

You may receive a warning about "Failed to check LocalWebservice existence" or "Failed to create Docker client". You can safely ignore this, as you are not deploying a local web service.

This command deploys a new service named `myservice`, using version 1 of the model that you registered previously.

The `inferenceConfig.yml` file provides information on how to use the model for inference. For example, it references the entry script (`score.py`) and software dependencies.

For more information on the structure of this file, see the [Inference configuration schema](#). For more information on entry scripts, see [Deploy models with Azure Machine Learning](#).

The `aciDeploymentConfig.yml` describes the deployment environment used to host the service. The deployment configuration is specific to the compute type that you use for the deployment. In this case, an Azure Container Instance is used. For more information, see the [Deployment configuration schema](#).

It will take several minutes before the deployment process completes.

TIP

In this example, Azure Container Instances is used. Deployments to ACI automatically create the needed ACI resource. If you were to instead deploy to Azure Kubernetes Service, you must create an AKS cluster ahead of time and specify it as part of the `az ml model deploy` command. For an example of deploying to AKS, see [Deploy a model to an Azure Kubernetes Service cluster](#).

After several minutes, information similar to the following JSON is returned:

```
ACI service creation operation finished, operation "Succeeded"
{
  "computeType": "ACI",
  {...ommitted for space...}
  "runtimeType": null,
  "scoringUri": "http://6c061467-4e44-4f05-9db5-9f9a22ef7a5d.eastus2.azurecontainer.io/score",
  "state": "Healthy",
  "tags": "",
  "updatedAt": "2019-09-19T18:22:32.227401+00:00"
}
```

The scoring URI

The `scoringUri` returned from the deployment is the REST endpoint for a model deployed as a web service. You can also get this URI by using the following command:

```
az ml service show -n myservice
```

This command returns the same JSON document, including the `scoringUri`.

The REST endpoint can be used to send data to the service. For information on creating a client application that sends data to the service, see [Consume an Azure Machine Learning model deployed as a web service](#)

Send data to the service

While you can create a client application to call the endpoint, the machine learning CLI provides a utility that can act as a test client. Use the following command to send data in the `testdata.json` file to the service:

```
az ml service run -n myservice -d @testdata.json
```

TIP

If you use PowerShell, use the following command instead:

```
az ml service run -n myservice -d `@testdata.json
```

The response from the command is similar to [3].

Clean up resources

IMPORTANT

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

Delete deployed service

If you plan on continuing to use the Azure Machine Learning workspace, but want to get rid of the deployed service to reduce costs, use the following command:

```
az ml service delete -n myservice
```

This command returns a JSON document that contains the name of the deleted service. It may take several minutes before the service is deleted.

Delete the training compute

If you plan on continuing to use the Azure Machine Learning workspace, but want to get rid of the `cpu-cluster` compute target created for training, use the following command:

```
az ml computetarget delete -n cpu-cluster
```

This command returns a JSON document that contains the ID of the deleted compute target. It may take several minutes before the compute target has been deleted.

Delete everything

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

To delete the resource group, and all the Azure resources created in this document, use the following command. Replace `<resource-group-name>` with the name of the resource group you created earlier:

```
az group delete -g <resource-group-name> -y
```

Next steps

In this Azure Machine Learning tutorial, you used the machine learning CLI for the following tasks:

- Install the machine learning extension
- Create an Azure Machine Learning workspace
- Create the compute resource used to train the model
- Define and register the dataset used to train the model
- Start a training run
- Register and download a model
- Deploy the model as a web service
- Score data using the web service

For more information on using the CLI, see [Use the CLI extension for Azure Machine Learning](#).

Set up Azure Machine Learning Visual Studio Code extension (preview)

12/23/2020 • 3 minutes to read • [Edit Online](#)

Learn how to install and run scripts using the Azure Machine Learning Visual Studio Code extension.

In this tutorial, you learn the following tasks:

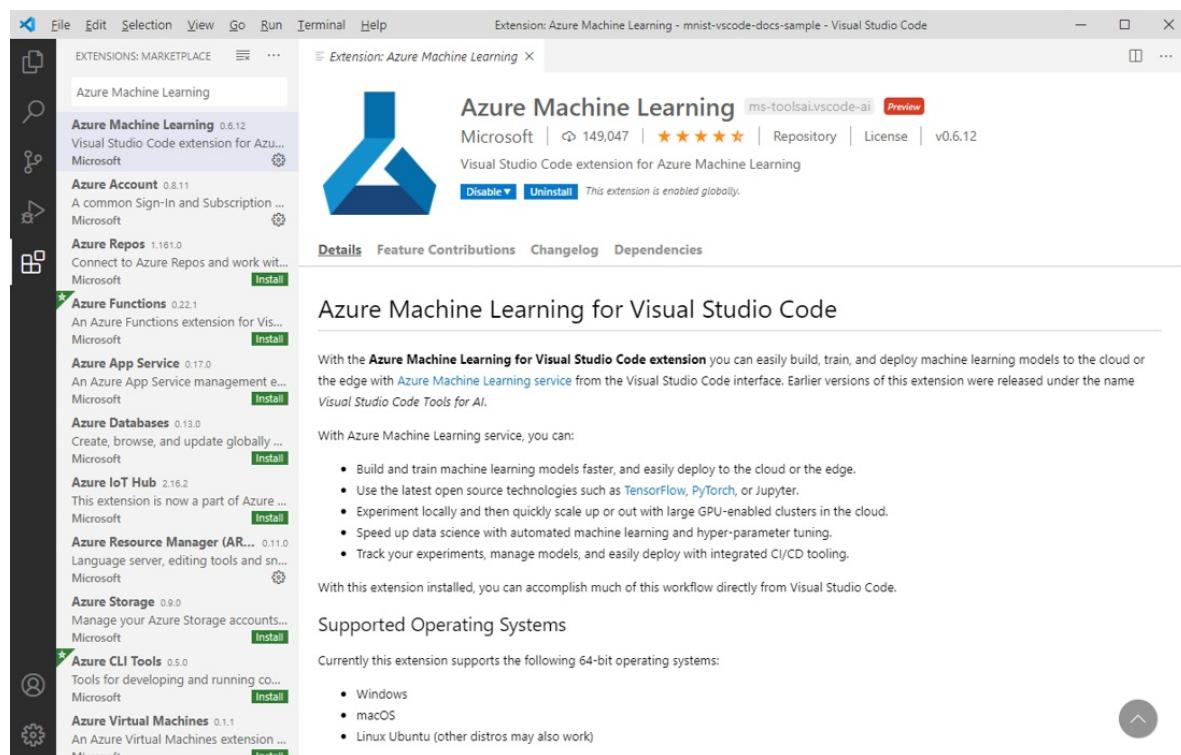
- Install the Azure Machine Learning Visual Studio Code extension
- Sign into your Azure account from Visual Studio Code
- Use the Azure Machine Learning extension to run a sample script

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 3](#)

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 for "Azure Machine Learning".
4. Select **Install**.



NOTE

Alternatively, you can install the Azure Machine Learning extension via the Visual Studio Marketplace by [downloading the installer directly](#).

The rest of the steps in this tutorial have been tested with **version 0.6.8** of the extension.

Sign in to your Azure Account

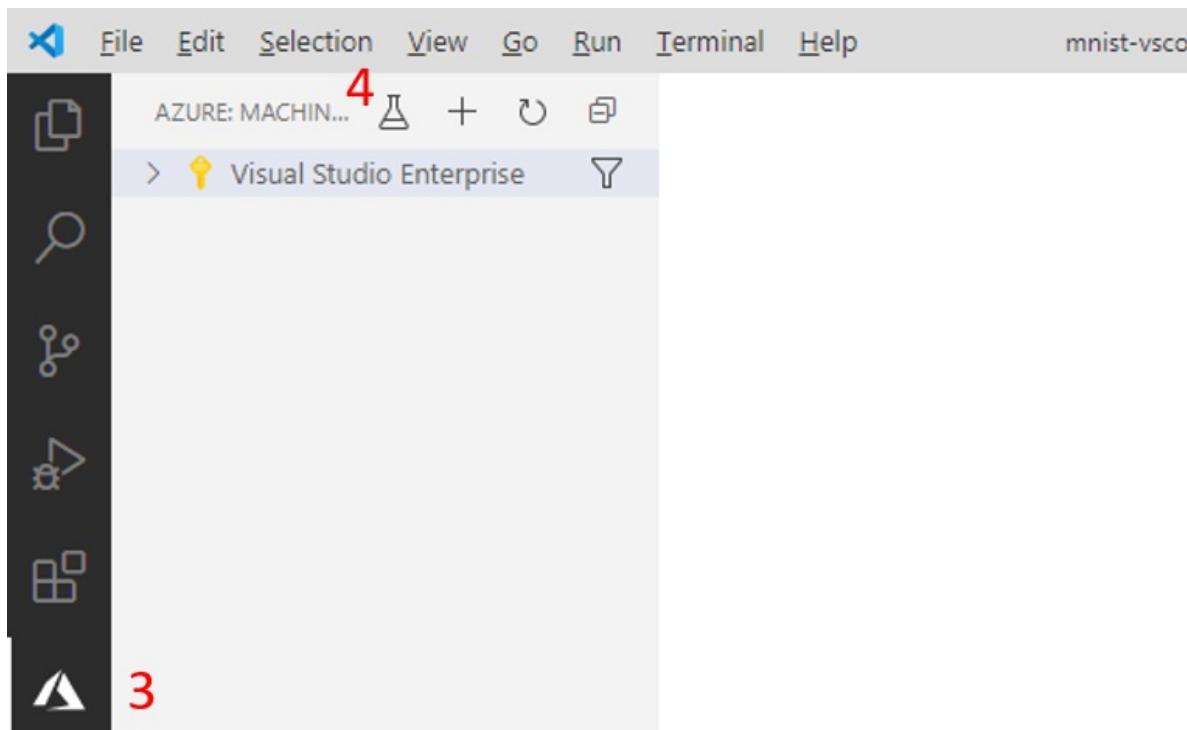
In order to provision resources and run workloads on Azure, you have to sign in with your Azure account credentials. To assist with account management, Azure Machine Learning automatically installs the Azure Account extension. Visit the following site to [learn more about the Azure Account extension](#).

1. Open the command palette by selecting **View > Command Palette** from the menu bar.
2. Enter the command "Azure: Sign In" into the command palette to start the sign in process.

Run a machine learning model training script in Azure

Now that you have signed into Azure with your account credentials, Use the steps in this section to learn how to use the extension to train a machine learning model.

1. Download and unzip the [VS Code Tools for AI repository](#) anywhere on your computer.
2. Open the `mnist-vscode-docs-sample` directory in Visual Studio Code.
3. Select the **Azure** icon in the Activity Bar.
4. Select the **Run Experiment** icon at the top of the Azure Machine Learning View.



5. When the command palette expands, follow the prompts.

NOTE

If you already have existing Azure Machine Learning resources provisioned, see [how to run experiments in VS Code guide](#).

- a. Select your Azure subscription.
 - b. From the list of environments, select **Conda dependencies file**.
 - c. Press **Enter** to browse the Conda dependencies file. This file contains the dependencies required to run your script. In this case, the dependencies file is the `env.yml` file inside the `mnist-vscode-docs-sample` directory.
 - d. Press **Enter** to browse the training script file. This is the file that contains code to a machine learning model that categorize images of handwritten digits. In this case, the script to train the model is the `train.py` file inside the `mnist-vscode-docs-sample` directory.
6. At this point, a configuration file similar to the one below appears in the text editor. The configuration contains the information required to run the training job like the file that contains the code to train the model and any Python dependencies specified in the previous step.

```
{
  "workspace": "WS06271500",
  "resourceGroup": "WS06271500-rg2",
  "location": "South Central US",
  "experiment": "WS06271500-exp1",
  "compute": {
    "name": "WS06271500-com1",
    "vmSize": "Standard_D1_v2, Cores: 1; RAM: 3.5GB;"
  },
  "runConfiguration": {
    "filename": "WS06271500-com1-rc1",
    "environment": {
      "name": "WS06271500-env1",
      "conda_dependencies": [
        "python=3.6.2",
        "tensorflow=1.15.0",
        "pip"
      ],
      "pip_dependencies": [
        "azureml-defaults"
      ],
      "environment_variables": {}
    }
  }
}
```

7. Once you're satisfied with your configuration, submit your experiment by opening the command palette and entering the following command:

```
Azure ML: Submit Experiment
```

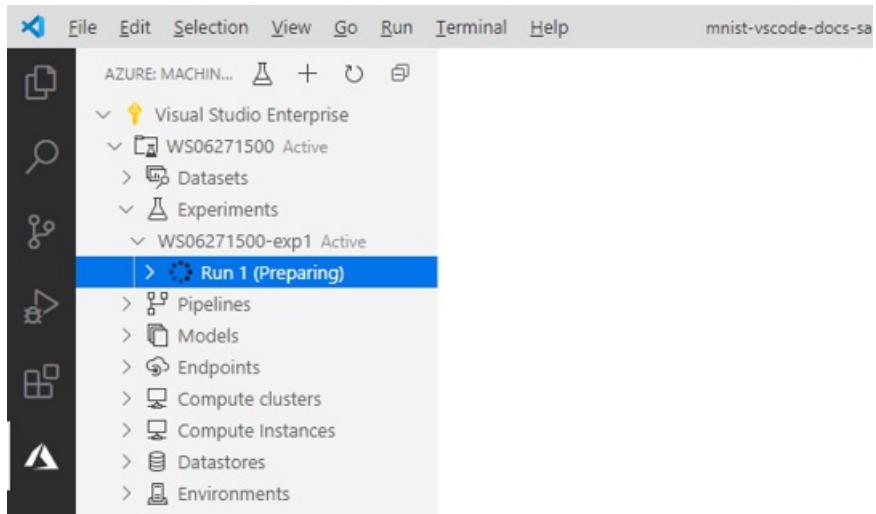
This sends the `train.py` and configuration file to your Azure Machine Learning workspace. The training job is then started on a compute resource in Azure.

Track the progress of the training script

Running your script can take several minutes. To track its progress:

1. Select the **Azure** icon from the activity bar.
2. Expand your subscription node.
3. Expand your currently running experiment's node. This is located inside the `{workspace}/Experiments/{experiment}` node where the values for your workspace and experiment are the same as the properties defined in the configuration file.
4. All of the runs for the experiment are listed, as well as their status. To get the most recent status, click the

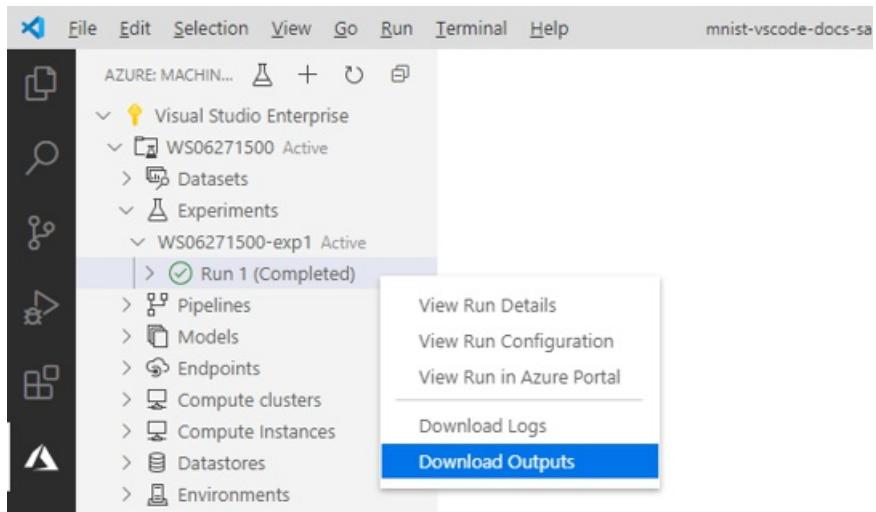
refresh icon at the top of the Azure Machine Learning View.



Download the trained model

When the experiment run is complete, the output is a trained model. To download the outputs locally:

1. Right-click the most recent run and select **Download Outputs**.



2. Select a location where to save the outputs to.
3. A folder with the name of your run is downloaded locally. Navigate to it.
4. The model files are inside the `outputs/outputs/model` directory.

Next steps

- [Tutorial: Train and deploy an image classification TensorFlow model using the Azure Machine Learning Visual Studio Code Extension.](#)

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

12/23/2020 • 9 minutes to read • [Edit Online](#)

Learn how to train and deploy 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:

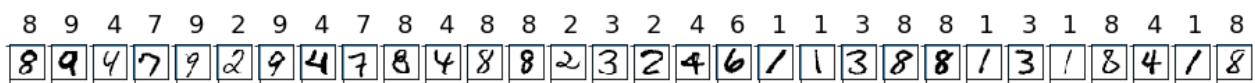
- Understand the code
- Create a workspace
- Create an experiment
- Configure Computer Targets
- Run a configuration file
- Train a model
- Register a model
- Deploy a model

Prerequisites

- Azure subscription. If you don't have one, sign up to try the [free or paid version of Azure Machine Learning](#).
- 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 tutorial](#)

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.

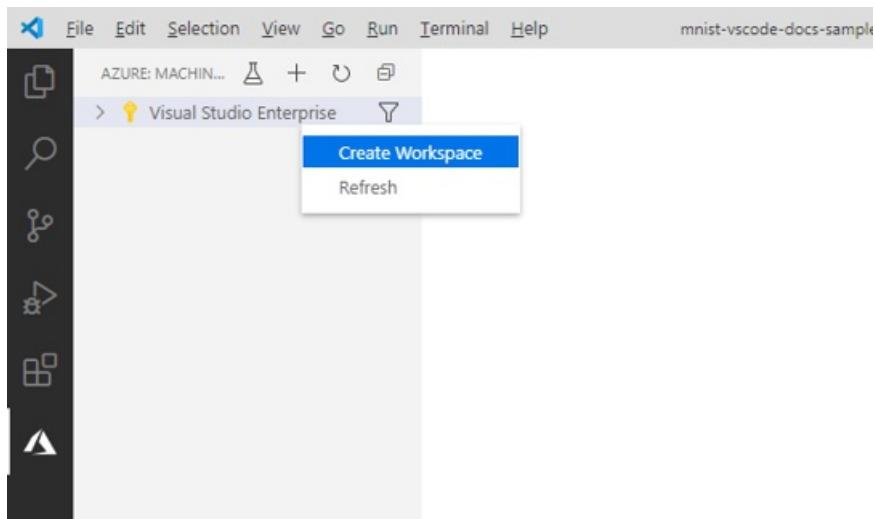
8 9 4 7 9 2 9 4 7 8 4 8 8 2 3 2 4 6 1 1 3 8 8 1 3 1 8 4 1 8


Get the code for this tutorial by downloading and unzipping the [VS Code Tools for AI repository](#) anywhere on your computer.

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



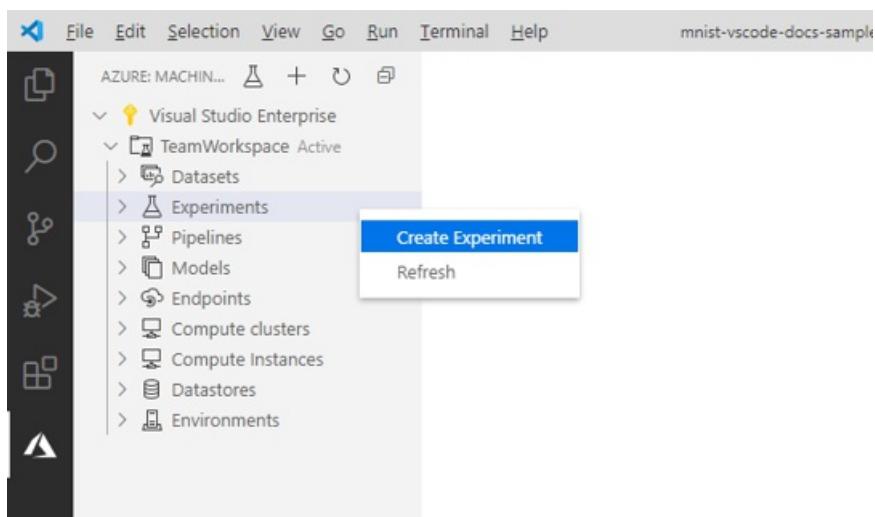
3. By default a name is generated containing the date and time of creation. In the text input box, change the name to "TeamWorkspace" and press **Enter**.
4. Select **Create a new resource group**.
5. Name your resource group "TeamWorkspace-rg" and press **Enter**.
6. Choose a location for your workspace. It's recommended to choose a location that is closest to the location you plan to deploy your model. For example, "West US 2".
7. When prompted to select the type of workspace, choose **basic**.

At this point, a request to Azure is made to create a new workspace in your account. After a few minutes, the new workspace appears in your subscription node.

Create an experiment

One or more experiments can be created in your workspace to track and analyze individual model training runs. Runs can be done in the Azure cloud or on your local machine.

1. On the Visual Studio Code activity bar, select the **Azure** icon. The Azure Machine Learning view appears.
2. Expand your subscription node.
3. Expand the **TeamWorkspace** node.
4. Right-click the **Experiments** node.
5. Select **Create Experiment** from the context menu.



6. Name your experiment "MNIST" and press **Enter** to create the new experiment.

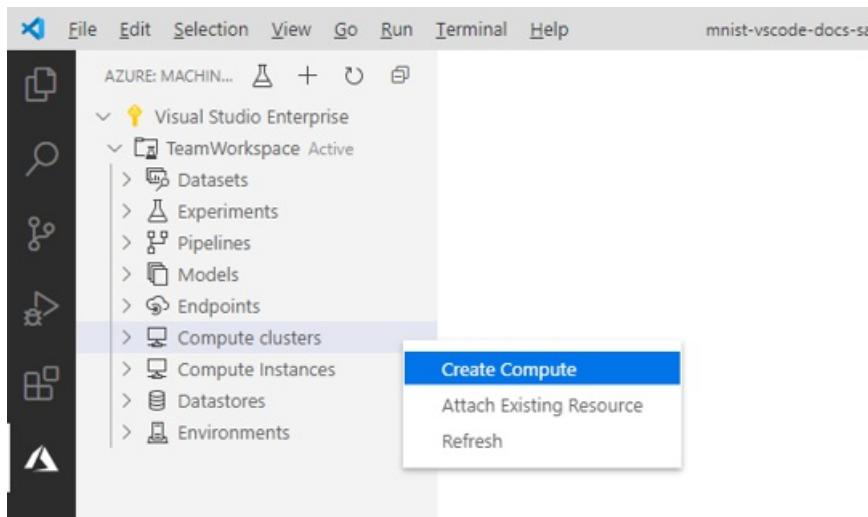
Like workspaces, a request is sent to Azure to create an experiment with the provided configurations. After a few minutes, the new experiment appears in the *Experiments* node of your workspace.

Configure Compute Targets

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

To create a compute target:

1. On the Visual Studio Code activity bar, select the **Azure** icon. The Azure Machine Learning view appears.
2. Expand your subscription node.
3. Expand the **TeamWorkspace** node.
4. Under the workspace node, right-click the **Compute clusters** node and choose **Create Compute**.



5. Select **Azure Machine Learning Compute (AmlCompute)**. Azure Machine Learning Compute is a managed-compute infrastructure that allows the user to easily create a single or multi-node compute that can be used with other users in your workspace.
6. Choose a VM size. Select **Standard_F2s_v2** from the list of options. The size of your VM has an impact on the amount of time it takes to train your models. For more information on VM sizes, see [sizes for Linux virtual machines in Azure](#).
7. Name your compute "TeamWkspc-com" and press **Enter** to create your compute.

A file appears in VS Code with content similar to the one below:

```
{
  "location": "westus2",
  "tags": {},
  "properties": {
    "computeType": "AmlCompute",
    "description": "",
    "properties": {
      "vmSize": "Standard_F2s_v2",
      "vmPriority": "dedicated",
      "scaleSettings": {
        "maxNodeCount": 4,
        "minNodeCount": 0,
        "nodeIdleTimeBeforeScaleDown": "PT120S"
      }
    }
  }
}
```

8. When satisfied with the configuration, open the command palette by selecting **View > Command Palette**.
9. Enter the following command into the command palette to save your run configuration file.

Azure ML: Save and Continue

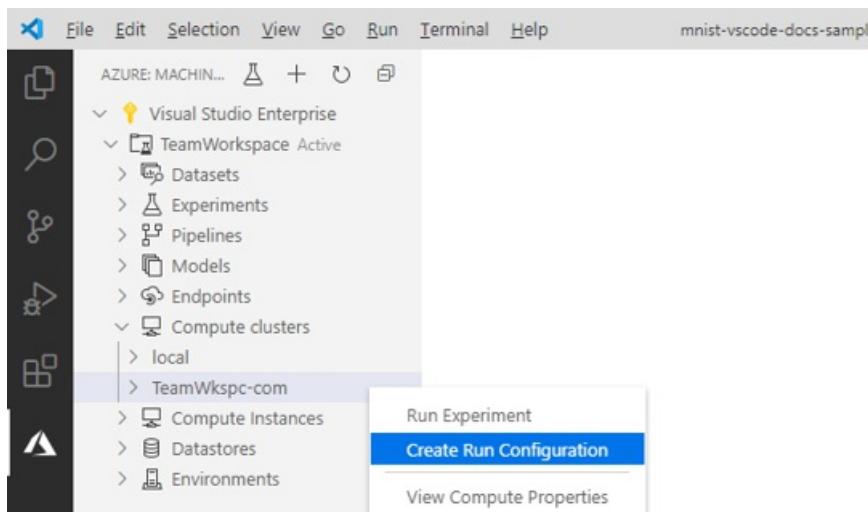
After a few minutes, the new compute target appears in the *Compute clusters* node of your workspace.

Create a run configuration

When you submit a training run to a compute target, you also submit the configuration needed to run the training job. For example, the script that contains the training code and the Python dependencies needed to run it.

To create a run configuration:

1. On the Visual Studio Code activity bar, select the **Azure** icon. The Azure Machine Learning view appears.
2. Expand your subscription node.
3. Expand the **TeamWorkspace > Compute clusters** node.
4. Under the compute node, right-click the **TeamWkspc-com** compute node and choose **Create Run Configuration**.



5. Name your run configuration "MNIST-rc" and press **Enter** to create your run configuration.

6. Then, select **Create new Azure ML Environment**. Environments define the dependencies required to run your scripts.

7. Name your environment "MNIST-env" and press **Enter**.

8. Select **Conda dependencies** file from the list.

9. Press **Enter** to browse the Conda dependencies file. In this case, the dependencies file is the `env.yml` file inside the `vscode-tools-for-ai/mnist-vscode-docs-sample` directory.

A file appears in VS Code with content similar to the one below:

```
{
  "name": "MNIST-env",
  "version": "1",
  "python": {
    "interpreterPath": "python",
    "userManagedDependencies": false,
    "condaDependencies": {
      "name": "vs-code-azure-ml-tutorial",
      "channels": [
        "defaults"
      ],
      "dependencies": [
        "python=3.6.2",
        "tensorflow=1.15.0",
        "pip",
        {
          "pip": [
            "azureml-defaults"
          ]
        }
      ]
    },
    "baseCondaEnvironment": null
  },
  "environmentVariables": {},
  "docker": {
    "baseImage": "mcr.microsoft.com/azureml/base:intelmpi2018.3-ubuntu16.04",
    "baseDockerfile": null,
    "baseImageRegistry": {
      "address": null,
      "username": null,
      "password": null
    },
    "enabled": false,
    "arguments": []
  },
  "spark": {
    "repositories": [],
    "packages": [],
    "precachePackages": true
  },
  "inferencingStackVersion": null
}
```

10. Once you're satisfied with your configuration, save it by opening the command palette and entering the following command:

```
Azure ML: Save and Continue
```

11. This sample does not use a dataset registered in Azure Machine Learning. Instead, it's loaded when `train.py` runs. When prompted to create a data reference for your training run, enter "n" into the prompt and press

Enter.

12. Press **Enter** to browse the script file to run on the compute. In this case, the script to train the model is the `train.py` file inside the `vscode-tools-for-ai/mnist-vscode-docs-sample` directory.

A file called `MNIST-rc.runconfig` appears in VS Code with content similar to the one below:

```
{  
    "script": "train.py",  
    "arguments": [],  
    "framework": "Python",  
    "communicator": "None",  
    "target": "TeamWkspc-com",  
    "environment": {  
        "name": "MNIST-env",  
        "version": "1",  
        "python": {  
            "interpreterPath": "python",  
            "userManagedDependencies": false,  
            "condaDependencies": {  
                "name": "vs-code-azure-ml-tutorial",  
                "channels": [  
                    "defaults"  
                ],  
                "dependencies": [  
                    "python=3.6.2",  
                    "tensorflow=1.15.0",  
                    "pip",  
                    {  
                        "pip": [  
                            "azureml-defaults"  
                        ]  
                    }  
                ]  
            },  
            "baseCondaEnvironment": null  
        },  
        "environmentVariables": {},  
        "docker": {  
            "baseImage": "mcr.microsoft.com/azureml/base:intelmpi2018.3-ubuntu16.04",  
            "baseDockerfile": null,  
            "baseImageRegistry": {  
                "address": null,  
                "username": null,  
                "password": null  
            },  
            "enabled": false,  
            "arguments": []  
        },  
        "spark": {  
            "repositories": [],  
            "packages": [],  
            "precachePackages": true  
        },  
        "inferencingStackVersion": null  
    },  
    "history": {  
        "outputCollection": true,  
        "snapshotProject": false,  
        "directoriesToWatch": [  
            "logs"  
        ]  
    }  
}
```

13. Once you're satisfied with your configuration, save it by opening the command palette and entering the

following command:

```
Azure ML: Save and Continue
```

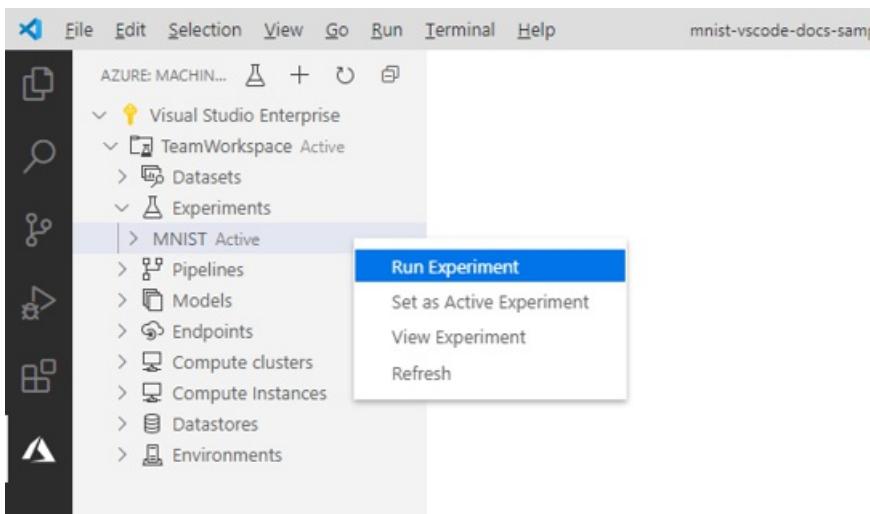
The `MNIST-rc` run configuration is added under the `TeamWkspc-com` compute node and the `MNIST-env` environment configuration is added under the `Environments` node.

Train the model

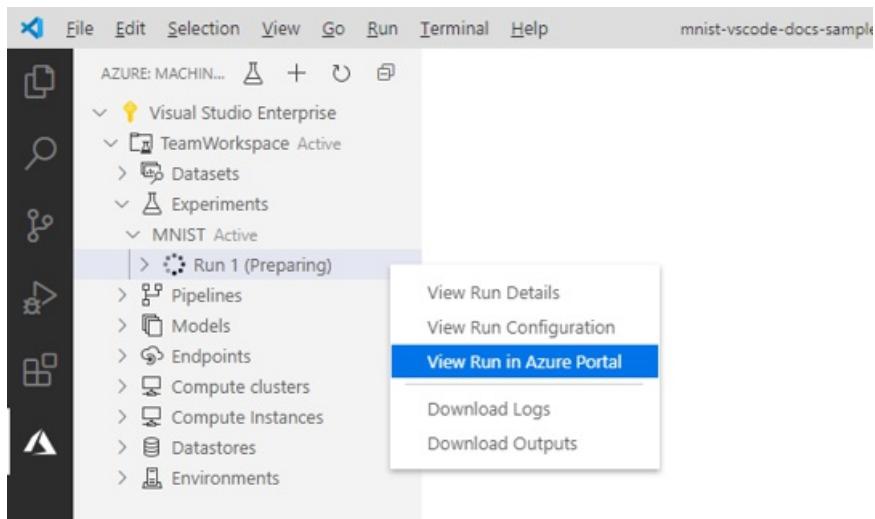
During the training process, a TensorFlow model is created by processing the training data and learning patterns embedded within it for each of the respective digits being classified.

To run an Azure Machine Learning experiment:

1. On the Visual Studio Code activity bar, select the **Azure** icon. The Azure Machine Learning view appears.
2. Expand your subscription node.
3. Expand the **TeamWorkspace > Experiments** node.
4. Right-click the **MNIST** experiment.
5. Select **Run Experiment**.



6. From the list of compute target options, select the `TeamWkspc-com` compute target.
7. Then, select the `MNIST-rc` run configuration.
8. 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 Run in Azure portal**.
9. When the dialog requesting to open an external website appears, select **Open**.



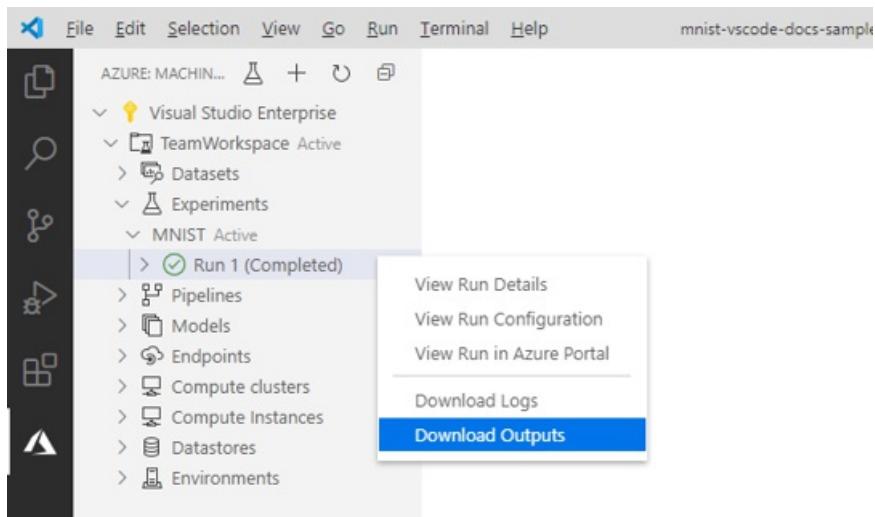
When the model is done training, the status label next to the run node updates to "Completed".

Register the model

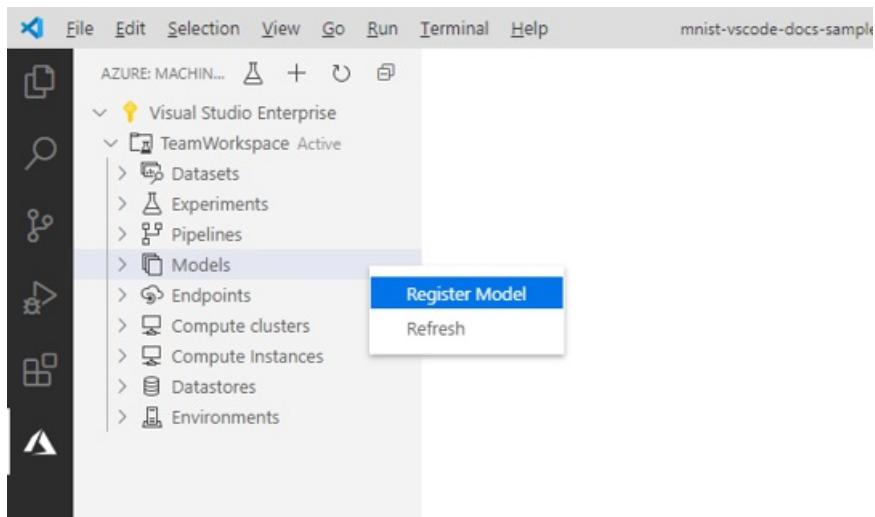
Now that you've trained your model, you can register it in your workspace.

To register your model:

1. On the Visual Studio Code activity bar, select the **Azure** icon. The Azure Machine Learning view appears.
2. Expand your subscription node.
3. Expand the **TeamWorkspace > Experiments > MNIST** node.
4. Get the model outputs generated from training the model. Right-click the **Run 1** run node and select **Download outputs**.



5. Choose the directory to save the downloaded outputs to. By default, the outputs are placed in the directory currently opened in Visual Studio Code.
6. Right-click the **Models** node and choose **Register Model**.



7. Name your model "MNIST-TensorFlow-model" and press **Enter**.
8. A TensorFlow model is made up of several files. Select **Model folder** as the model path format from the list of options.
9. Select the `azureml_outputs/Run_1/outputs/outputs/model` directory.

A file containing your model configurations appears in Visual Studio Code with similar content to the one below:

```
{
  "modelName": "MNIST-TensorFlow-model",
  "tags": {
    "": ""
  },
  "modelPath": "c:\\Dev\\vscode-tools-for-ai\\mnist-vscode-docs-
sample\\azureml_outputs\\Run_1\\outputs\\outputs\\model",
  "description": ""
}
```

10. Once you're satisfied with your configuration, save it by opening the command palette and entering the following command:

```
Azure ML: Save and Continue
```

After a few minutes, the model appears under the *Models* node.

Deploy the model

In Visual Studio Code, you can deploy your model as a web service to:

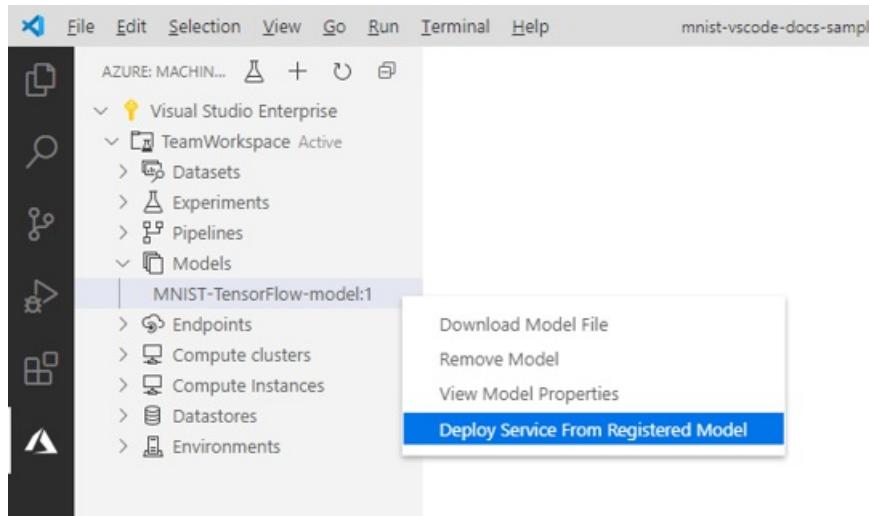
- Azure Container Instances (ACI).
- Azure Kubernetes Service (AKS).

You don't need to create an ACI container to test in advance, because ACI containers are created as needed. However, you do need to configure AKS clusters in advance. For more information on deployment options, see [deploy models with Azure Machine Learning](#).

To deploy a web service as an ACI :

1. On the Visual Studio Code activity bar, select the **Azure** icon. The Azure Machine Learning view appears.
2. Expand your subscription node.

3. Expand the TeamWorkspace > Models node.
4. Right-click the MNIST-TensorFlow-model and select Deploy Service from Registered Model.



5. Select Azure Container Instances.
6. Name your service "mnist-tensorflow-svc" and press Enter.
7. Choose the script to run in the container by pressing Enter in the input box and browsing for the `score.py` file in the `mnist-vscode-docs-sample` directory.
8. Provide the dependencies needed to run the script by pressing Enter in the input box and browsing for the `env.yml` file in the `mnist-vscode-docs-sample` directory.

A file containing your model configurations appears in Visual Studio Code with similar content to the one below:

```
{
  "name": "mnist-tensorflow-svc",
  "imageConfig": {
    "runtime": "python",
    "executionScript": "score.py",
    "dockerFile": null,
    "condaFile": "env.yml",
    "dependencies": [],
    "schemaFile": null,
    "enableGpu": false,
    "description": ""
  },
  "deploymentConfig": {
    "cpu_cores": 1,
    "memory_gb": 10,
    "tags": {
      "": ""
    },
    "description": ""
  },
  "deploymentType": "ACI",
  "modelIds": [
    "MNIST-TensorFlow-model:1"
  ]
}
```

9. Once you're satisfied with your configuration, save it by opening the command palette and entering the following command:

At this point, a request is sent to Azure to deploy your web service. This process takes several minutes. Once deployed, the new service appears under the *Endpoints* node.

Next steps

- For a walkthrough of how to train with Azure Machine Learning outside of Visual Studio Code, see [Tutorial: Train models with Azure Machine Learning](#).
- For a walkthrough of how to edit, run, and debug code locally, see the [Python hello-world tutorial](#).

Tutorial: Power BI integration - create the predictive model with a Notebook (part 1 of 2)

12/23/2020 • 6 minutes to read • [Edit Online](#)

In the first part of this tutorial, you train and deploy a predictive machine learning model using code in a Jupyter Notebook. In part 2, you'll then use the model to predict outcomes in Microsoft Power BI.

In this tutorial, you:

- Create a Jupyter Notebook
- Create an Azure Machine Learning compute instance
- Train a regression model using scikit-learn
- Deploy the model to a real-time scoring endpoint

There are three different ways to create and deploy the model you'll use in Power BI. This article covers Option A: Train and deploy models using Notebooks. This option shows a code-first authoring experience using Jupyter notebooks hosted in Azure Machine Learning studio.

You could instead use:

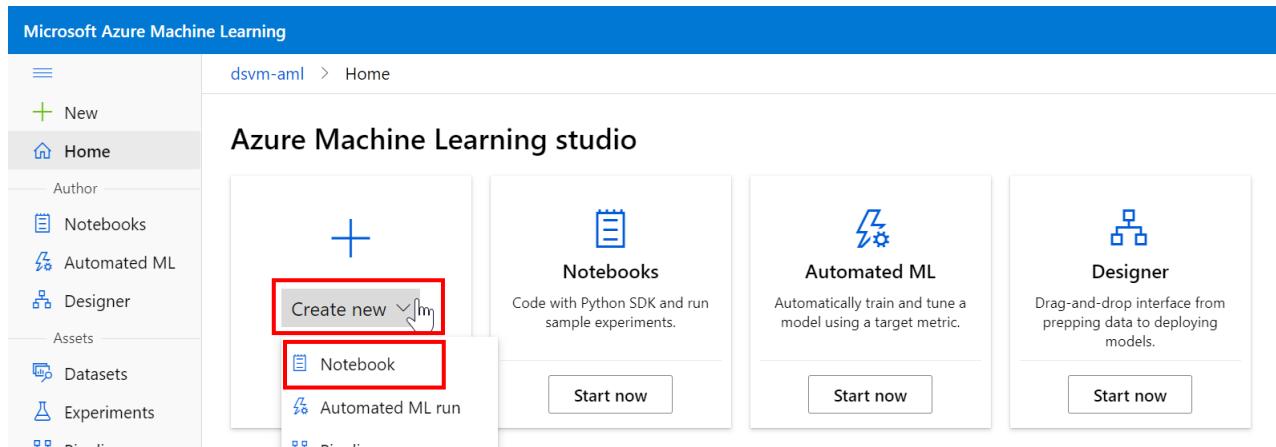
- [Option B: Train and deploy models using designer](#)- a low-code authoring experience using Designer (a drag-and-drop user interface).
- [Option C: Train and deploy models using automated ML](#) - a no-code authoring experience that fully automates the data preparation and model training.

Prerequisites

- An Azure subscription ([a free trial is available](#)).
- An Azure Machine Learning workspace. If you do not already have a workspace, follow [how to create an Azure Machine Learning Workspace](#).
- Introductory knowledge of the Python language and machine learning workflows.

Create a notebook and compute

In the [Azure Machine Learning Studio](#) homepage select **Create new** followed by **Notebook**:

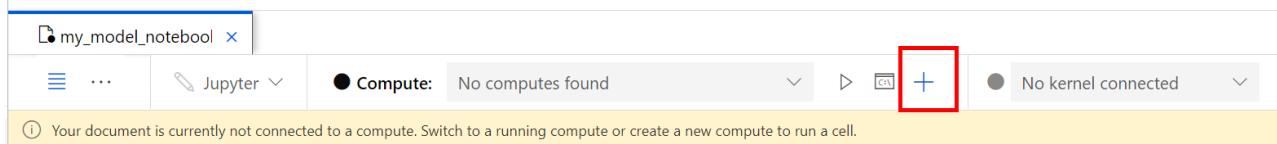


You are shown a dialog box to **Create a new file** enter:

1. A filename for your notebook (for example `my_model_notebook`)

2. Change the **File Type** to **Notebook**

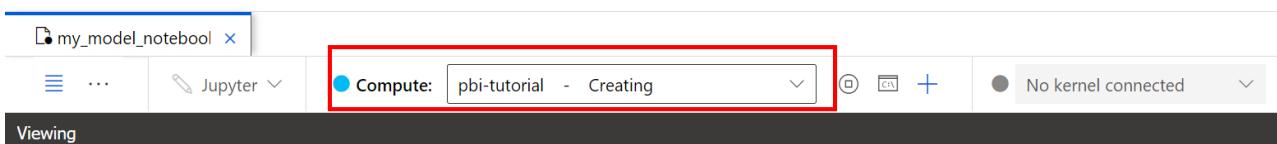
Select **Create**. Next, you need to create some compute and attach it to your notebook in order to run code cells. To do this, select the plus icon at the top of the notebook:



Next, on the **Create compute instance** page:

1. Choose a CPU virtual machine size - for the purposes of this tutorial a **Standard_D11_v2** (two cores, 14-GB RAM) will be fine.
2. Select **Next**.
3. On the **Configure Settings** page provide a valid **Compute name** (valid characters are upper and lower case letters, digits, and the - character).
4. Select **Create**.

You may notice on the notebook that the circle next to **Compute** has turned cyan, indicating the compute instance is being created:



NOTE

It can take around 2-4 minutes for the compute to be provisioned.

Once the compute is provisioned, you can use the notebook to execute code cells. For example, type into the cell:

```
import numpy as np  
  
np.sin(3)
```

Followed by **Shift-Enter** (or **Control-Enter** or select the play button next to the cell). You should see the following output:

The screenshot shows a Jupyter Notebook interface. At the top, there's a header bar with tabs for 'my_model_notebook' and 'Jupyter'. Below the header, the status bar indicates 'Compute: pbi-tutorial - Running'. The main area is titled 'pbi-tutorial · Jupyter kernel idle'. A code cell contains the following Python code:

```
1 import numpy as np
2
3 np.sin(3)
```

The output of the code cell is '0.1411200080598672'.

You are now ready to build a Machine Learning model!

Build a model using scikit-learn

In this tutorial, you use the [Diabetes dataset](#), which is made available in [Azure Open Datasets](#).

Import data

To import your data, copy-and-paste the code below into a new **code cell** in your notebook:

```
from azureml.opendatasets import Diabetes

diabetes = Diabetes.get_tabular_dataset()
X = diabetes.drop_columns("Y")
y = diabetes.keep_columns("Y")
X_df = X.to_pandas_dataframe()
y_df = y.to_pandas_dataframe()
X_df.info()
```

The `x_df` pandas data frame contains 10 baseline input variables (such as age, sex, body mass index, average blood pressure, and six blood serum measurements). The `y_df` pandas data frame is the target variable containing a quantitative measure of disease progression one year after baseline. There are a total of 442 records.

Train model

Create a new **code cell** in your notebook and copy-and-paste the code snippet below, which constructs a ridge regression model and serializes the model using Python's pickle format:

```
import joblib
from sklearn.linear_model import Ridge

model = Ridge().fit(X,y)
joblib.dump(model, 'sklearn_regression_model.pkl')
```

Register the model

In addition to the content of the model file itself, your registered model will also store model metadata - model description, tags, and framework information - that will be useful when managing and deploying models in your workspace. Using tags, for instance, you can categorize your models and apply filters when listing models in your workspace. Also, marking this model with the scikit-learn framework will simplify deploying it as a web service, as we'll see later.

Copy-and-paste the code below into a new **code cell** in your notebook:

```
import sklearn

from azureml.core import Workspace
from azureml.core import Model
from azureml.core.resource_configuration import ResourceConfiguration

ws = Workspace.from_config()

model = Model.register(workspace=ws,
                       model_name='my-sklearn-model',           # Name of the registered model in your
wspace.                                         workspace.
                       model_path='./sklearn_regression_model.pkl', # Local file to upload and register as a
model.                                         model.
                       model_framework=Model.Framework.SCIKITLEARN, # Framework used to create the model.
                       model_framework_version=sklearn.__version__, # Version of scikit-learn used to create
the model.
                       sample_input_dataset=X,
                       sample_output_dataset=y,
                       resource_configuration=ResourceConfiguration(cpu=2, memory_in_gb=4),
                       description='Ridge regression model to predict diabetes progression.',
                       tags={'area': 'diabetes', 'type': 'regression'})

print('Name:', model.name)
print('Version:', model.version)
```

You can also view the model in Azure Machine Learning Studio by navigating to **Endpoints** in the left hand-menu:

Name	Version	Experiment	Run ID	Created on	Tags	Pr
my-sklearn-model	1	--		Nov 5, 2020 2:53 PM	area : diabetes type : re...	
cifar10-model	7	cifar10-experiment	5b1c1d38-3ef9-4037-be94-7a9...	Sep 25, 2020 1:53 PM		
cifar10-model	6	cifar10-experiment	8fcaacde-ade0-4957-9162-87a...	Sep 24, 2020 11:56 AM		
cifar10-model	5	cifar10-experiment	43bfdf06-f770-4298-ac36-48ed...	Sep 23, 2020 8:07 PM		
iris-model	6	quickstart	c05aef52-00ee-47a3-ba53-0c88...	Sep 3, 2020 2:39 PM		

Define the scoring script

When deploying a model to be integrated into Microsoft Power BI, you need to define a Python *scoring script* and custom environment. The scoring script contains two functions:

- `init()` - this function is executed once the service starts. This function loads the model (note that the model is automatically downloaded from the model registry) and deserializes it.
- `run(data)` - this function is executed when a call is made to the service with some input data that needs scoring.

NOTE

We use Python decorators to define the schema of the input and output data, which is important for the Microsoft Power BI integration to work.

Copy-and-paste the code below into a new **code cell** in your notebook. The code snippet below has a cell magic that will write the code to a file named score.py.

```

%%writefile score.py

import json
import pickle
import numpy as np
import pandas as pd
import os
import joblib
from azureml.core.model import Model

from inference_schema.schema_decorators import input_schema, output_schema
from inference_schema.parameter_types.numpy_parameter_type import NumpyParameterType
from inference_schema.parameter_types.pandas_parameter_type import PandasParameterType


def init():
    global model
    # Replace filename if needed.
    path = os.getenv('AZUREML_MODEL_DIR')
    model_path = os.path.join(path, 'sklearn_regression_model.pkl')
    # Deserialize the model file back into a sklearn model.
    model = joblib.load(model_path)

input_sample = pd.DataFrame(data=[{
    "AGE": 5,
    "SEX": 2,
    "BMI": 3.1,
    "BP": 3.1,
    "S1": 3.1,
    "S2": 3.1,
    "S3": 3.1,
    "S4": 3.1,
    "S5": 3.1,
    "S6": 3.1
}])

# This is an integer type sample. Use the data type that reflects the expected result.
output_sample = np.array([0])

# To indicate that we support a variable length of data input,
# set enforce_shape=False
@input_schema('data', PandasParameterType(input_sample))
@output_schema(NumpyParameterType(output_sample))
def run(data):
    try:
        print("input_data....")
        print(data.columns)
        print(type(data))
        result = model.predict(data)
        print("result....")
        print(result)
    # You can return any data type, as long as it is JSON serializable.
    return result.tolist()
    except Exception as e:
        error = str(e)
        return error

```

Define the custom environment

Next we need to define the environment to score the model - we need to define in this environment the Python packages required by the scoring script (score.py) defined above such as pandas, scikit-learn, etc.

To define the environment, copy-and-paste the code below into a new **code cell** in your notebook:

```

from azureml.core.model import InferenceConfig
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies

environment = Environment('my-sklearn-environment')
environment.python.conda_dependencies = CondaDependencies.create(pip_packages=[
    'azureml-defaults',
    'inference-schema[numpy-support]',
    'joblib',
    'numpy',
    'pandas',
    'scikit-learn=={}'.format(sklearn.__version__)
])

inference_config = InferenceConfig(entry_script='./score.py', environment=environment)

```

Deploy the model

To deploy the model, copy-and-paste the code below into a new **code cell** in your notebook:

```

service_name = 'my-diabetes-model'

service = Model.deploy(ws, service_name, [model], inference_config, overwrite=True)
service.wait_for_deployment(show_output=True)

```

NOTE

It can take 2-4 minutes for the service to be deployed.

You should see the following output of a successfully deployed service:

```

Tips: You can try get_logs(): https://aka.ms/debugimage#dockerlog or local deployment:
https://aka.ms/debugimage#debug-locally to debug if deployment takes longer than 10 minutes.
Running..... .
Succeeded
ACI service creation operation finished, operation "Succeeded"

```

You can also view the service in Azure Machine Learning Studio by navigating to **Endpoints** in the left hand-menu:

Name	Description	Created on	Created by	Updated on	Compute type	Compute target
my-diabetes-model	--	November 12, 2020 4:16 PM	Sam Kemp	November 12, 2020 4:16 PM	AKS	infclust
pbi-test-real-time...	--	November 11, 2020 3:45 PM	Sam Kemp	November 11, 2020 3:45 PM	AKS	infclust

It is recommended that you test the webservice to ensure that it works as expected. Navigate back to your notebook by selecting **Notebooks** in the left-hand menu in Azure Machine Learning Studio. Copy-and-paste the code below into a new **code cell** in your notebook to test the service:

```
import json

input_payload = json.dumps({
    'data': X_df[0:2].values.tolist(),
    'method': 'predict' # If you have a classification model, you can get probabilities by changing this to
    'predict_proba'.
})

output = service.run(input_payload)

print(output)
```

The output should look like the following json structure: `{'predict': [[205.59], [68.84]]}`.

Next steps

In this tutorial, you saw how to build and deploy a model in such a way that they can be consumed by Microsoft Power BI. In the next part, you learn how to consume this model from a Power BI report.

[Tutorial: Consume model in Power BI](#)

Tutorial: Power BI integration - drag-and-drop to create the predictive model (part 1 of 2)

12/23/2020 • 5 minutes to read • [Edit Online](#)

In the first part of this tutorial, you train and deploy a predictive machine learning model using Azure Machine Learning designer - a low-code drag-and-drop user interface. In part 2, you'll then use the model to predict outcomes in Microsoft Power BI.

In this tutorial, you:

- Create an Azure Machine Learning compute instance
- Create an Azure Machine Learning inference cluster
- Create a dataset
- Train a regression model
- Deploy the model to a real-time scoring endpoint

There are three different ways to create and deploy the model you'll use in Power BI. This article covers Option B: Train and deploy models using designer. This option shows a low-code authoring experience using designer (a drag-and-drop user interface).

You could instead use:

- [Option A: Train and deploy models using Notebooks](#) - a code-first authoring experience using Jupyter notebooks hosted in Azure Machine Learning studio.
- [Option C: Train and deploy models using automated ML](#) - a no-code authoring experience that fully automates the data preparation and model training.

Prerequisites

- An Azure subscription ([a free trial is available](#)).
- An Azure Machine Learning workspace. If you do not already have a workspace, follow [how to create an Azure Machine Learning Workspace](#).
- Introductory knowledge of machine learning workflows.

Create compute for training and scoring

In this section, you create a *compute instance*, which is used for training machine learning models. Also, you create an *inference cluster* that will be used to host the deployed model for real-time scoring.

Log into the [Azure Machine Learning studio](#) and select **Compute** from the left-hand menu followed by **New**:

The screenshot shows the Microsoft Azure Machine Learning Compute page. On the left, there's a sidebar with various options like New, Home, Author, Notebooks, etc., and a 'Compute' option which is highlighted with a red box. The main area has tabs for 'Compute instances', 'Compute clusters', 'Inference clusters', and 'Attached compute'. The 'Compute instances' tab is selected and highlighted with a red box. Below it, there's a button '+ New' with a red box around it, followed by 'Refresh', 'Start', 'Stop', 'Restart', 'Delete', 'View quota', and a dropdown menu 'Instances I can use'. A table lists one instance: 'pbi-tutorial' (Status: Running, Application URI: JupyterLab Jupyter RStudio SSH, Virtual machine size: STANDARD_D11_V2, Created on: Nov 5, 2020 1:04 PM).

On the resulting **Create compute instance** screen, select a VM size (for this tutorial, select a **Standard_D11_v2**) followed by **Next**. In the Setting page, provide a valid name for your compute instance followed by selecting **Create**.

TIP

The compute instance can also be used to create and execute notebooks.

You can now see your compute instance **Status** is **Creating** - it will take around 4 minutes for the machine to be provisioned. While you are waiting, select **Inference Cluster** tab on the compute page followed by **New**:

The screenshot shows the Microsoft Azure Machine Learning Compute page again. The sidebar has the 'Compute' option highlighted with a red box. The main area has tabs for 'Compute instances', 'Compute clusters', and 'Inference clusters'. The 'Inference clusters' tab is selected and highlighted with a red box. Below it, there's a button '+ New' with a red box around it, followed by 'Refresh', 'Delete', and 'Detach'. A table lists one cluster: 'infclust' (Type: Kubernetes service, Created/Attached: Created, Provisioning state: Succeeded, Created on: Nov 11, 2020 3:32 PM).

In the resulting **Create inference cluster** page, select a region followed by a VM size (for this tutorial, select a **Standard_D11_v2**), then select **Next**. On the **Configure Settings** page:

1. Provide a valid compute name
2. Select **Dev-test** as the cluster purpose (creates a single node to host the deployed model)
3. Select **Create**

You can now see your inference cluster **Status** is **Creating** - it will take around 4 minutes for your single node cluster to deploy.

Create a dataset

In this tutorial, you use the [Diabetes dataset](#), which is made available in [Azure Open Datasets](#).

To create the dataset, in the left-hand menu select **Datasets**, then **Create Dataset** - you will see the following options:

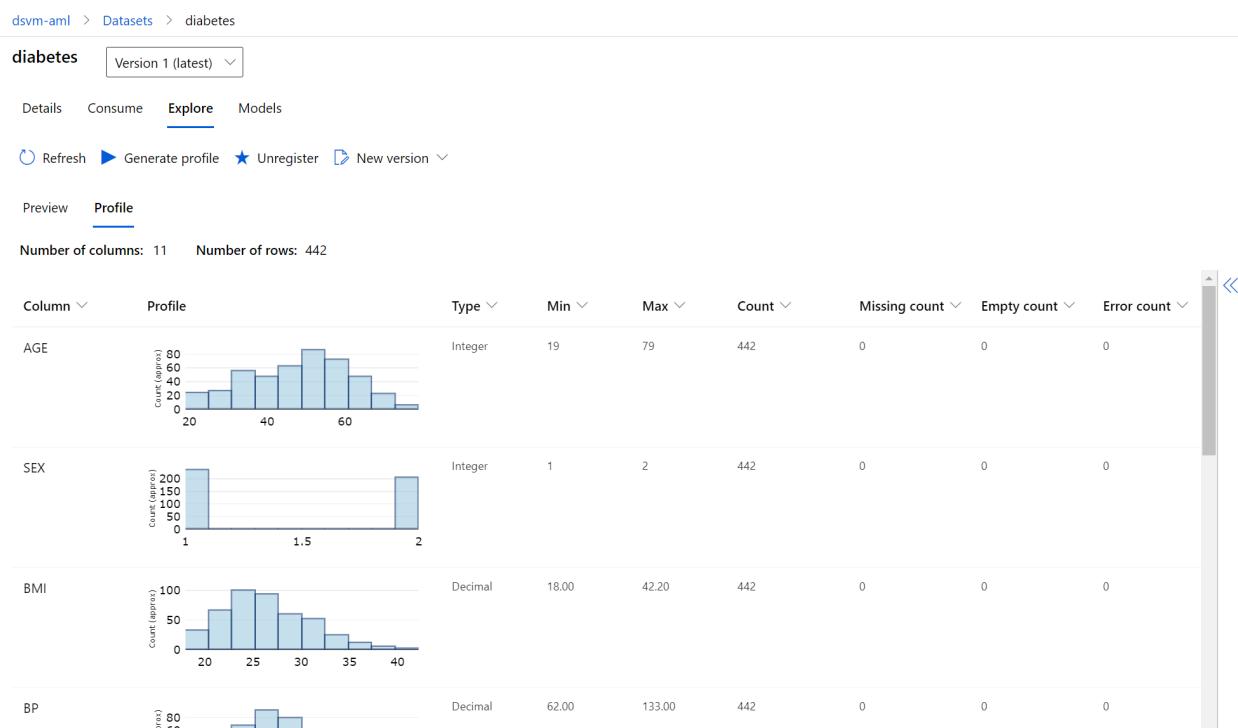
The screenshot shows the Microsoft Azure Machine Learning interface. The left sidebar has a 'Datasets' section with a red box around it. Under 'Create dataset', there is a red box around 'From Open Datasets'. A list of registered datasets is shown below, including 'diabetes'.

Version	Data source	Created on	Modified on	Properties
1	workspaceblobstore	Nov 11, 2020 3:28 PM	Nov 11, 2020 3:28 PM	File
1	URI	Oct 8, 2020 11:50 AM	Oct 8, 2020 11:50 AM	Tabular
1	stocks	Oct 6, 2020 3:02 PM	Oct 6, 2020 3:02 PM	File
1	workspaceblobstore	Sep 23, 2020 11:08 AM	Sep 23, 2020 11:08 AM	File
2	workspaceblobstore	Aug 28, 2020 3:45 PM	Sep 3, 2020 11:52 AM	Tabular
1	URI	Jul 23, 2020 12:22 PM	Jul 23, 2020 12:22 PM	File

Select **From Open Datasets**, and then in **Create dataset from Open Datasets** screen:

1. Search for *diabetes* using the search bar
2. Select **Sample: Diabetes**
3. Select **Next**
4. Provide a name for your dataset - *diabetes*
5. Select **Create**

You can explore the data by selecting the Dataset followed by **Explore**:



The data has 10 baseline input variables (such as age, sex, body mass index, average blood pressure, and six blood serum measurements), and one target variable named **Y** (a quantitative measure of diabetes progression one year after baseline).

Create a Machine Learning model using designer

Once you have created the compute and datasets, you can move on to creating the machine learning model using designer. In the Azure Machine Learning studio, select **Designer** followed by **New Pipeline**:

The screenshot shows the Azure Machine Learning studio interface with the 'Designer' tab selected in the left sidebar. A red box highlights the 'New pipeline' button on the canvas. Below the canvas, there are several prebuilt module samples: 'Easy-to-use prebuilt modules' (with a plus sign icon), 'Image Classification using DenseNet', 'Binary Classification using Vowpal Wabbit Model - Adu...', 'Wide & Deep based Recommendation - Restaur...', and 'Regression - Automobile Price Prediction (Basic)'. The 'Pipelines' section shows a table with one draft pipeline entry: 'Pipeline-Created-on-11-12-2020' (N/A type, Nov 12, 2020 1:59 PM, Created by Sam Kemp). A tooltip 'Select a pipeline from list to preview' is visible over the pipeline list.

You see a blank *canvas* where you can also see a **Settings menu**:

The screenshot shows the 'Pipeline-Authoring' interface for the pipeline 'Pipeline-Created-on-11-12-2020'. The left sidebar lists various asset categories. The main area shows pipeline settings. A red box highlights the 'Settings' section, which includes 'Default compute target' (warning: 'Select a compute target to run the pipeline') and 'No compute target selected' (button: 'Select compute target'). Another red box highlights the 'Draft details' section, which contains 'Draft name: Pipeline-Created-on-11-12-2020', 'Draft description (optional): Pipeline created on 20201112', and 'Created on'.

On the **Settings menu**, **Select compute target** and then select the compute instance you created earlier followed by **Save**. Rename your **Draft name** to something more memorable (for example *diabetes-model*) and enter a description.

Next, in listed assets expand **Datasets** and locate the **diabetes** dataset - drag-and-drop this module onto the canvas:

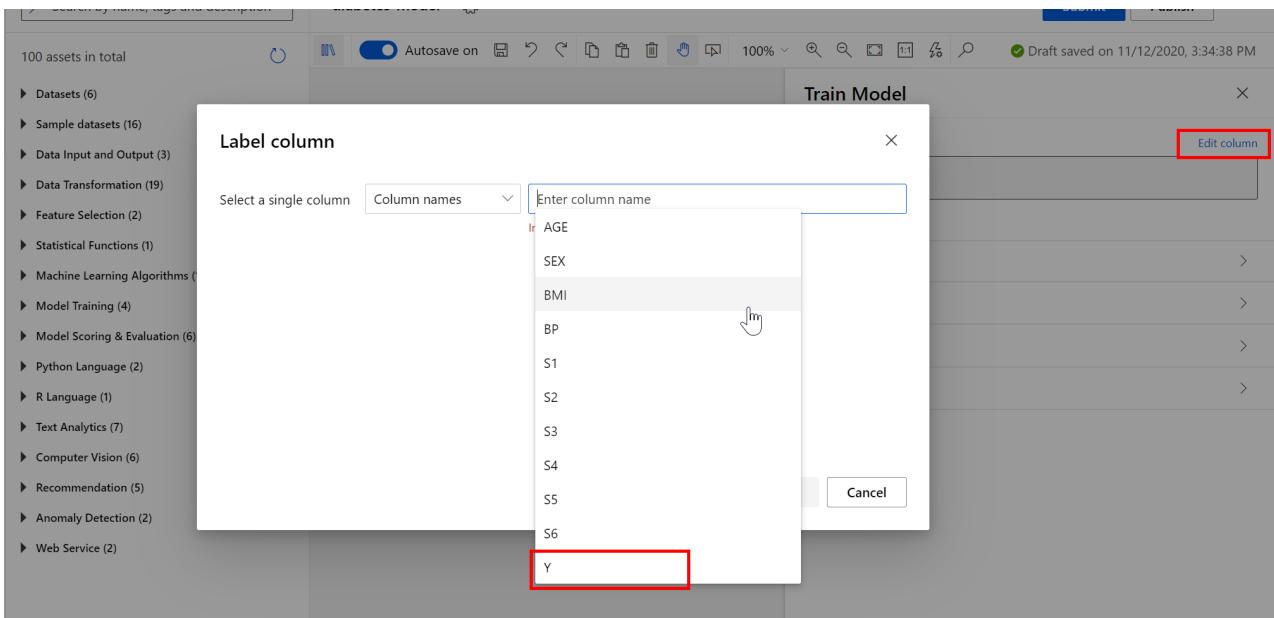
Next, drag-and-drop the following components on to the canvas:

1. Linear regression (located in **Machine Learning Algorithms**)
2. Train model (located in **Model Training**)

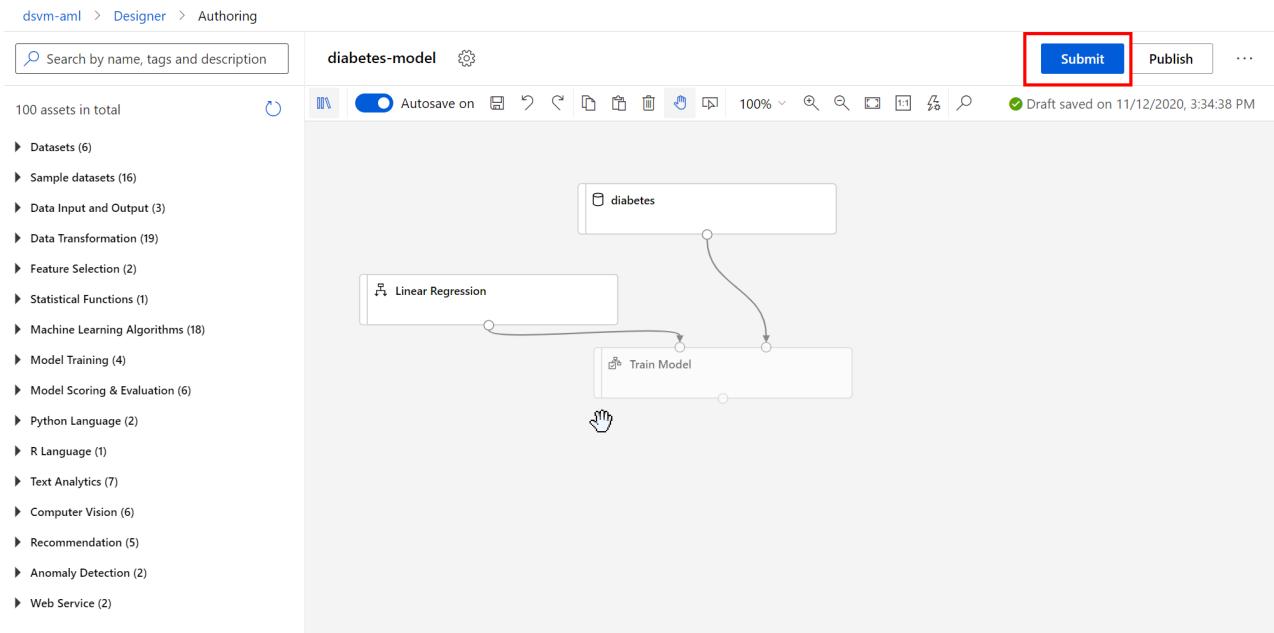
Your canvas should look like (notice that the top-and-bottom of the components has little circles called ports - highlighted in red below):

Next, you need to *wire* these components together. Select the port at the bottom of the **diabetes** dataset and drag it to the right-hand port at the top of the **Train model** component. Select the port at the bottom of the **Linear regression** component and drag onto the left-hand port at the top of the **Train model** port.

Choose the column in the dataset to be used as the label (target) variable to predict. Select the **Train model** component followed by **Edit column**. From the dialog box - Select the **Enter Column name** followed by **Y** in the drop-down list:



Select **Save**. Your machine learning *workflow* should look as follows:

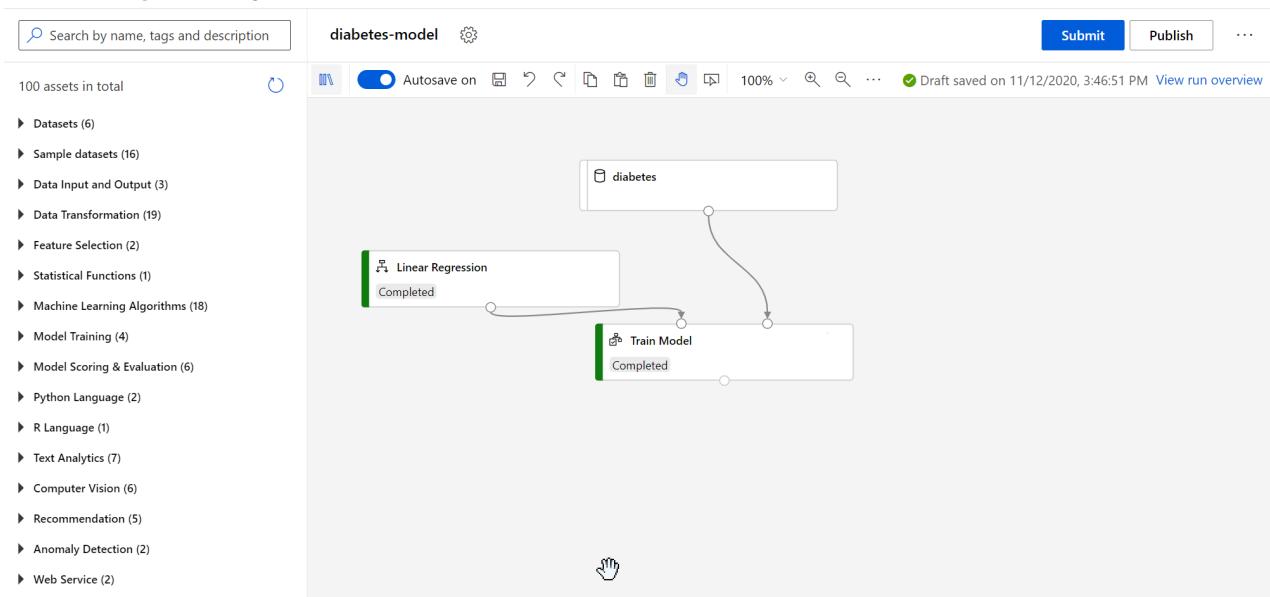


Select **Submit** and then **Create new** under experiment. Provide a name for the experiment followed by **Submit**.

NOTE

It should take around 5-minutes for your experiment to complete on the first run. Subsequent runs are much quicker - designer caches already run components to reduce latency.

When the experiment is completed, you see:



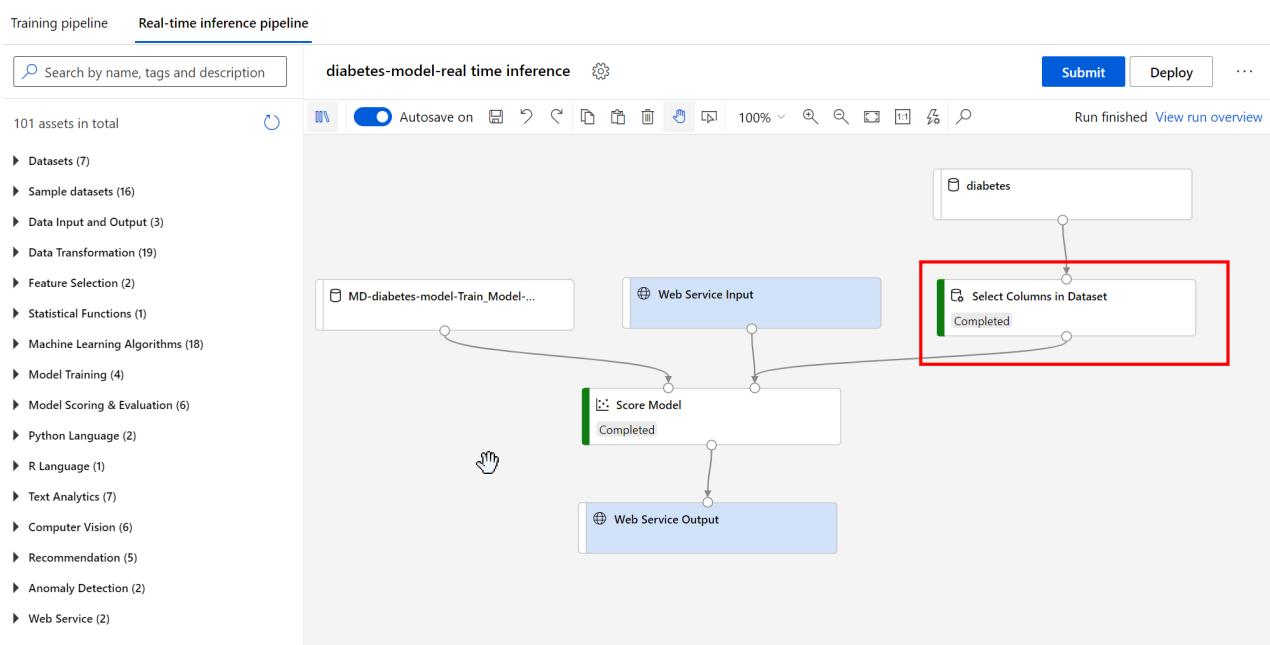
You can inspect the logs of the experiment by selecting **Train model** followed by **Outputs + logs**.

Deploy the model

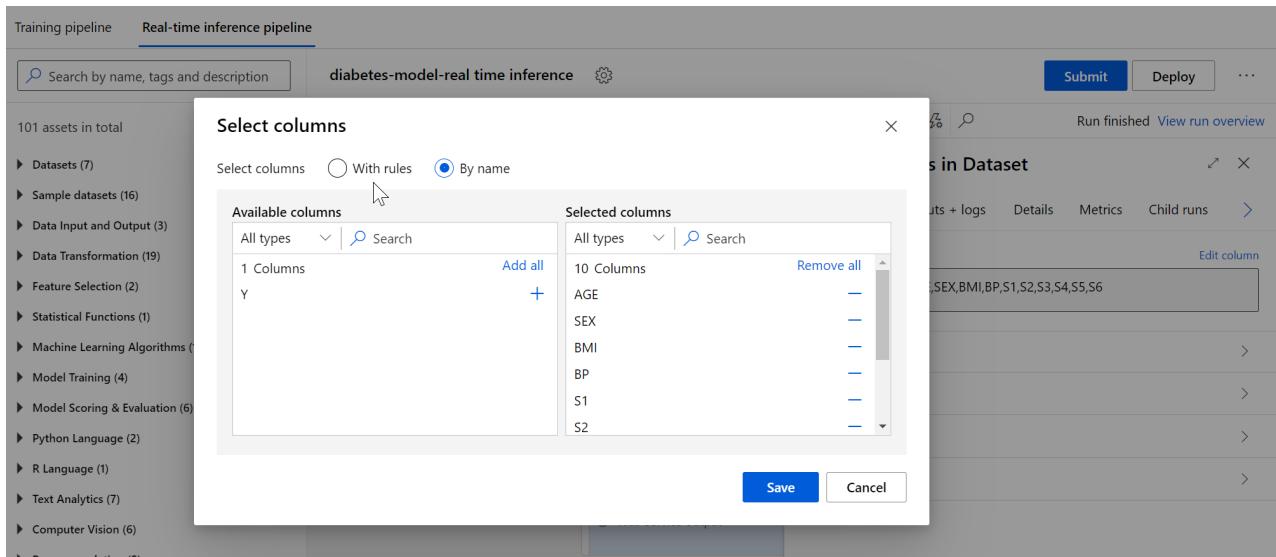
To deploy the model, select **Create Inference Pipeline** (located at the top of the canvas) followed by **Real-time inference pipeline**:



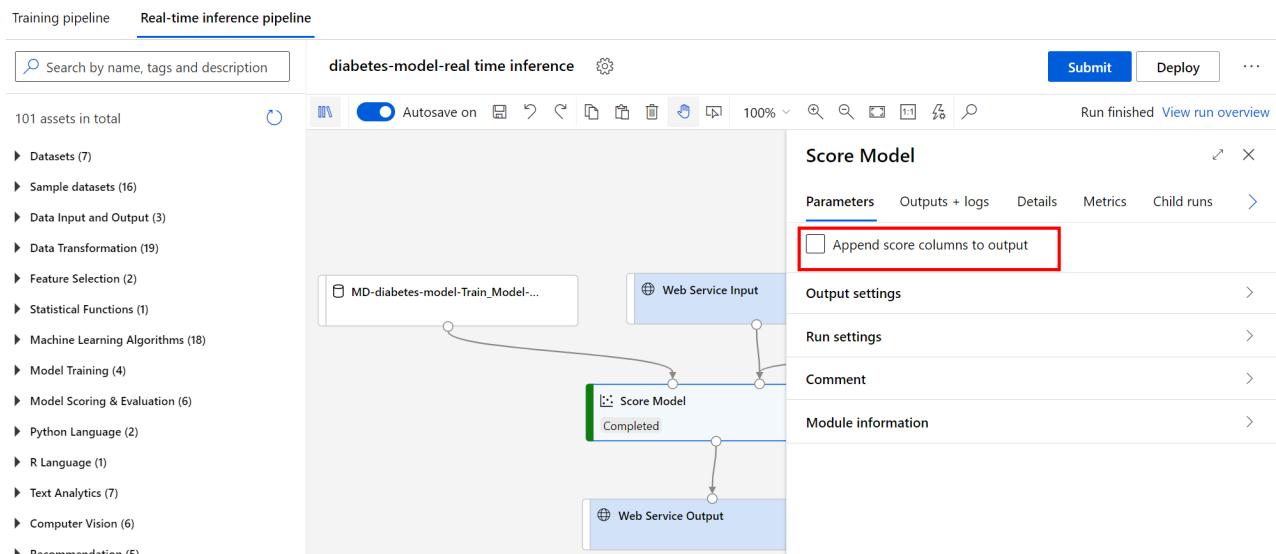
The pipeline condenses to just the components necessary to do the model scoring. When you score the data you will not know the target variable values, therefore we can remove **Y** from the dataset. To remove, add to the canvas a **Select columns in Dataset** component. Wire the component so the diabetes dataset is the input, and the results are the output into the **Score Model** component:



Select the **Select Columns in Dataset** component on the canvas followed by **Edit Columns**. In the Select columns dialog, select **By name** and then ensure all the input variables are selected but **not** the target:



Select **Save**. Finally, select the **Score Model** component and ensure the **Append score columns to output** checkbox is unchecked (only the predictions are sent back, rather than the inputs *and* predictions, reducing latency):



Select **Submit** at the top of the canvas.

When you have successfully run the inference pipeline, you can then deploy the model to your inference cluster. Select **Deploy**, which will show the **Set-up real-time endpoint** dialog box. Select **Deploy new real-time endpoint**, name the endpoint **my-diabetes-model**, select the inference you created earlier, select **Deploy**:

Training pipeline Real-time inference pipeline

Search by name, tags and description

101 assets in total

- Datasets (7)
- Sample datasets (16)
- Data Input and Output (3)
- Data Transformation (19)
- Feature Selection (2)
- Statistical Functions (1)
- Machine Learning Algorithms (18)
- Model Training (4)
- Model Scoring & Evaluation (6)
- Python Language (2)
- R Language (1)
- Text Analytics (7)
- Computer Vision (6)
- Recommendation (5)
- Anomaly Detection (2)
- Web Service (2)

Set up real-time endpoint

Deploy new real-time endpoint Replace an existing real-time endpoint

Real-time endpoint name *

Endpoint description (optional)

Compute target

Existing compute target(s)

Compute target name	Node count	Region	Status
infclust	1	uksouth	Succeeded

Deploy **Cancel**

Submit Deploy ...

Run finished View run overview

```
graph TD; A[Service Input] --> B[Select Columns in Data  
Completed]; B --> C[Score Model]; C --> D[Output]
```

Next steps

In this tutorial, you saw how to train and deploy a designer model. In the next part, you learn how to consume (score) this model from Power BI.

[Tutorial: Consume model in Power BI](#)

Tutorial: Power BI integration - create the predictive model using automated machine learning (part 1 of 2)

12/23/2020 • 3 minutes to read • [Edit Online](#)

In the first part of this tutorial, you train and deploy a predictive machine learning model using automated machine learning in the Azure Machine Learning studio. In part 2, you'll then use the best performing model to predict outcomes in Microsoft Power BI.

In this tutorial, you:

- Create an Azure Machine Learning compute cluster
- Create a dataset
- Create an automated ML run
- Deploy the best model to a real-time scoring endpoint

There are three different ways to create and deploy the model you'll use in Power BI. This article covers Option C: Train and deploy models using automated ML in the studio. This option shows a no-code authoring experience that fully automates the data preparation and model training.

You could instead use:

- [Option A: Train and deploy models using Notebooks](#) - a code-first authoring experience using Jupyter notebooks hosted in Azure Machine Learning studio.
- [Option B: Train and deploy models using designer](#) - a low-code authoring experience using Designer (a drag-and-drop user interface).

Prerequisites

- An Azure subscription ([a free trial is available](#)).
- An Azure Machine Learning workspace. If you do not already have a workspace, follow [how to create an Azure Machine Learning Workspace](#).

Create compute cluster

Automated ML automatically trains lots of different machine learning models to find the "best" algorithm and parameters. Azure Machine Learning parallelizes the execution of the model training over a compute cluster.

In the [Azure Machine Learning Studio](#), select **Compute** in the left-hand menu followed by **Compute Clusters** tab. Select **New**:

Name	Provisioning state	Virtual machine size	Created on	Idle nodes	Busy nodes	Unprovisioned nodes
cpu-cluster	Succeeded (0 nodes)	STANDARD_D2_V2	Sep 17, 2020 4:40 PM	0	0	4
gpu-cluster	Succeeded (0 nodes)	STANDARD_NC6	Jul 31, 2020 3:06 PM	0	0	1

In the **Create compute cluster** screen:

1. Select a VM size (for the purposes of this tutorial a **Standard_D11_v2** machine is fine).
2. Select **Next**
3. Provide a valid compute name
4. Keep **Minimum number of nodes** at 0
5. Change **Maximum number of nodes** to 4
6. Select **Create**

You can see that the status of your cluster has changed to **Creating**.

NOTE

When the cluster is created it will have 0 nodes, which means no compute costs are incurred. You only incur costs when the automated ML job runs. The cluster will scale back to 0 automatically for you after 120 seconds of idle time.

Create dataset

In this tutorial, you use the [Diabetes dataset](#), which is made available in [Azure Open Datasets](#).

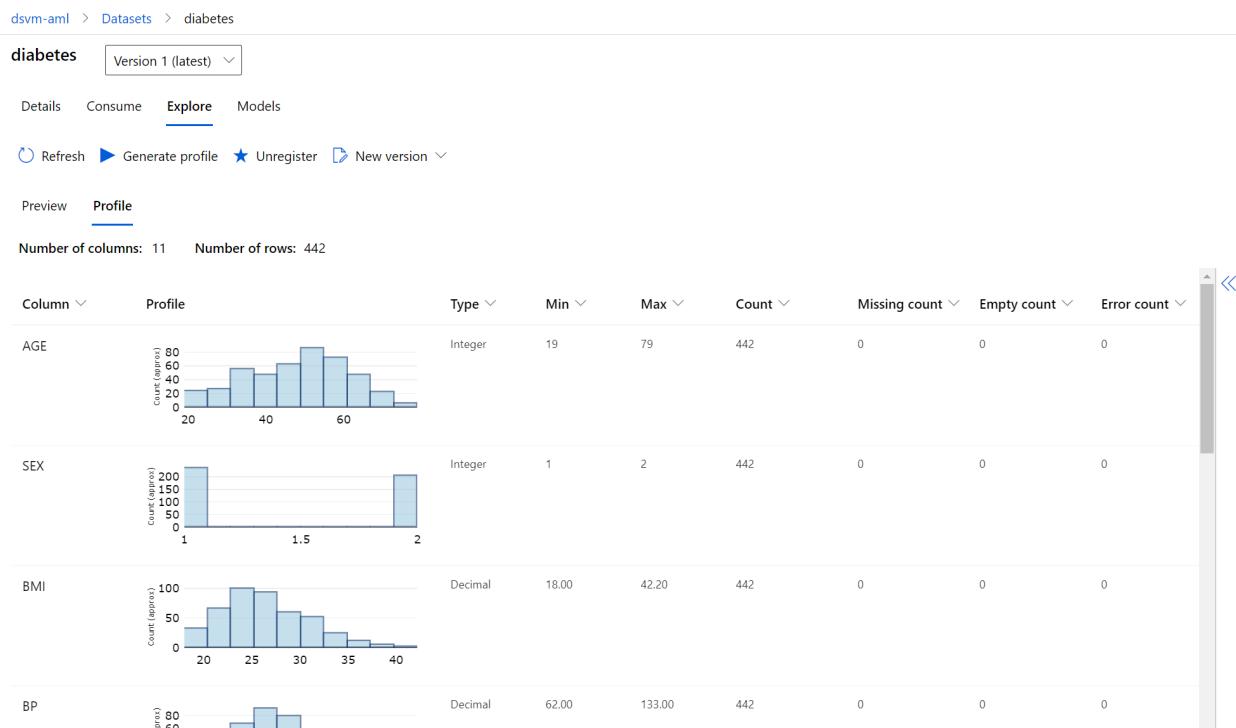
To create the dataset, select **Datasets** left-hand menu followed by **Create Dataset** - you will see the following options:

Version	Data source	Created on	Modified on	Properties
1	workspaceblobstore	Nov 11, 2020 3:28 PM	Nov 11, 2020 3:28 PM	File
1	URI	Oct 8, 2020 11:50 AM	Oct 8, 2020 11:50 AM	Tabular
1	stocks	Oct 6, 2020 3:02 PM	Oct 6, 2020 3:02 PM	File
1	workspaceblobstore	Sep 23, 2020 11:08 AM	Sep 23, 2020 11:08 AM	File
2	workspaceblobstore	Aug 28, 2020 3:45 PM	Sep 3, 2020 11:52 AM	Tabular
1	URI	Jul 23, 2020 12:22 PM	Jul 23, 2020 12:22 PM	File

Select From Open Datasets, and then in Create dataset from Open Datasets screen:

1. Search for *diabetes* using the search bar
2. Select **Sample: Diabetes**
3. Select **Next**
4. Provide a name for your dataset - *diabetes*
5. Select **Create**

You can explore the data by selecting the Dataset followed by **Explore**:



The data has 10 baseline input variables (such as age, sex, body mass index, average blood pressure, and six blood serum measurements), and one target variable named **Y** (a quantitative measure of diabetes progression one year after baseline).

Create automated ML run

In the [Azure Machine Learning Studio](#) select **Automated ML** in the left-hand menu followed by **New Automated ML Run**:

The screenshot shows the 'Automated ML' section of the Azure Machine Learning Studio. On the left sidebar, 'Automated ML' is highlighted with a red box. In the main area, there is a button '+ New Automated ML run' with a red box around it. Below this, a table lists 'Recent Automated ML runs':

Run	Run ID	Experiment	Status	Submitted time	Duration	Submitted by	Compute target	Tags
Run 81	AutoML_fe9d9373-54...	automl-pbi-test	Completed	Nov 13, 2020 10:27 AM	33m 54s	Sam Kemp	cpu-cluster	
Run 1	AutoML_c9ece312-4c...	sample	Completed	Oct 13, 2020 2:01 PM	29m 50s	Sam Kemp	cpu-cluster	

Next, select the **diabetes** dataset you created earlier and select **Next**:

Create a new Automated ML run

Select dataset

Select a dataset from the list below, or create a new dataset. Automated ML currently only supports tabular data for authoring runs.

+ Create dataset | Show supported datasets only | Search to filter items...

Dataset name	Dataset type	Created on	Modified on
<input checked="" type="checkbox"/> diabetes	Tabular	Oct 8, 2020 11:50 AM	Oct 8, 2020 11:50 AM
<input type="checkbox"/> iris	Tabular	Aug 28, 2020 3:45 PM	Sep 3, 2020 11:52 AM

Back Next Cancel

In the **Configure run** screen:

- Under **Experiment name**, select **Create new**
- Provide an experiment a name
- In the Target column field, select **Y**
- In the **Select compute cluster** field select the compute cluster you created earlier.

Your completed form should look similar to:

Create a new Automated ML run

Configure run

Configure the experiment. Select from existing experiments or define a new name, select the target column and the training compute to use. [Learn more on how to experiment](#)

Dataset
diabetes ([View dataset](#))

Experiment name * Create new

Select existing Create new

New experiment name

Target column * Y

Select compute cluster * cpu-cluster

Back Next Cancel

Finally, you need to select the machine learning task to perform, which is **Regression**:

Create a new Automated ML run

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.
- 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](#)

Select Finish.

IMPORTANT

It will take around 30 minutes for automated ML to finish training the 100 different models.

Deploy the best model

Once the automated ML run has completed, you can see the list of all the different machine learning models that have been tried by selecting the **Models** tab. The models are ordered in performance order - the best performing model will be shown first. When you select the best model, the **Deploy** button will be enabled:

dsvm-aml > Automated ML > automl-pbi-test > Run 81

Run 81 Completed

[Refresh](#) [Cancel](#)

Details	Data guardrails	Models	Outputs + logs	Child runs	Snapshot
Deploy	Download	Explain model			

[Search to filter items...](#)

Algorithm name	Explained	Spearman correlation ↓	Sampling ⓘ	Run	Created	Duration	Status
VotingEnsemble	View explanation	0.69077	100.00 %	Run 163	Nov 13, 2020 10:58 AM	2m 11s	Completed
StandardScalerWrapper, ExtremeRandomTrees		0.68694	100.00 %	Run 154	Nov 13, 2020 10:56 AM	49s	Completed
StandardScalerWrapper, GradientBoosting		0.68238	100.00 %	Run 124	Nov 13, 2020 10:48 AM	41s	Completed
StackEnsemble		0.67949	100.00 %	Run 164	Nov 13, 2020 10:58 AM	2m 17s	Completed
MaxAbsScaler, ElasticNet		0.67768	100.00 %	Run 111	Nov 13, 2020 10:45 AM	37s	Completed
MinMaxScaler, ExtremeRandomTrees		0.67722	100.00 %	Run 91	Nov 13, 2020 10:38 AM	49s	Completed
StandardScalerWrapper, ElasticNet		0.67689	100.00 %	Run 153	Nov 13, 2020 10:55 AM	39s	Completed
MinMaxScaler, ExtremeRandomTrees		0.67650	100.00 %	Run 107	Nov 13, 2020 10:45 AM	43s	Completed
MaxAbsScaler, ElasticNet		0.67573	100.00 %	Run 94	Nov 13, 2020 10:41 AM	42s	Completed
MinMaxScaler, ExtremeRandomTrees		0.67469	100.00 %	Run 123	Nov 13, 2020 10:48 AM	42s	Completed
RobustScaler, ExtremeRandomTrees		0.67458	100.00 %	Run 100	Nov 13, 2020 10:43 AM	44s	Completed
MaxAbsScaler, ElasticNet		0.67436	100.00 %	Run 92	Nov 13, 2020 10:39 AM	44s	Completed

Selecting **Deploy**, will present a **Deploy a model** screen:

1. Provide a name for your model service - use **diabetes-model**
2. Select **Azure Container Service**
3. Select **Deploy**

You should see a message that states the model has been deployed successfully.

Next steps

In this tutorial, you saw how to train and deploy a machine learning model using automated ML. In the next tutorial you are shown how to consume (score) this model from Power BI.

[Tutorial: Consume model in Power BI](#)

Explore Azure Machine Learning with Jupyter Notebooks

12/23/2020 • 2 minutes to read • [Edit Online](#)

NOTE

A community-driven repository of examples can be found at <https://github.com/Azure/azureml-examples>.

The [example Azure Machine Learning Notebooks repository](#) includes the latest Azure Machine Learning Python SDK samples. These Jupyter notebooks are designed to help you explore the SDK and serve as models for your own machine learning projects.

This article shows you how to access the repository from the following environments:

- [Azure Machine Learning compute instance](#)
- [Bring your own notebook server](#)
- [Data Science Virtual Machine](#)

NOTE

Once you've cloned the repository, you'll find tutorial notebooks in the **tutorials** folder and feature-specific notebooks in the **how-to-use-azureml** folder.

Get samples on Azure Machine Learning compute instance

The easiest way to get started with the samples is to complete the [Tutorial: Setup environment and workspace](#). Once completed, you'll have a dedicated notebook server pre-loaded with the SDK and the sample repository. No downloads or installation necessary.

Get samples on your notebook server

If you'd like to bring your own notebook server for local development, follow these steps:

1. Use the instructions at [Azure Machine Learning SDK](#) to install the Azure Machine Learning SDK for Python
2. Create an [Azure Machine Learning workspace](#).
3. Write a [configuration file](#) file (`aml_config/config.json`).
4. Clone [the GitHub repository](#).

```
git clone https://github.com/Azure/MachineLearningNotebooks.git
```

5. Start the notebook server from your cloned directory.

```
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](#).

Get samples on DSVM

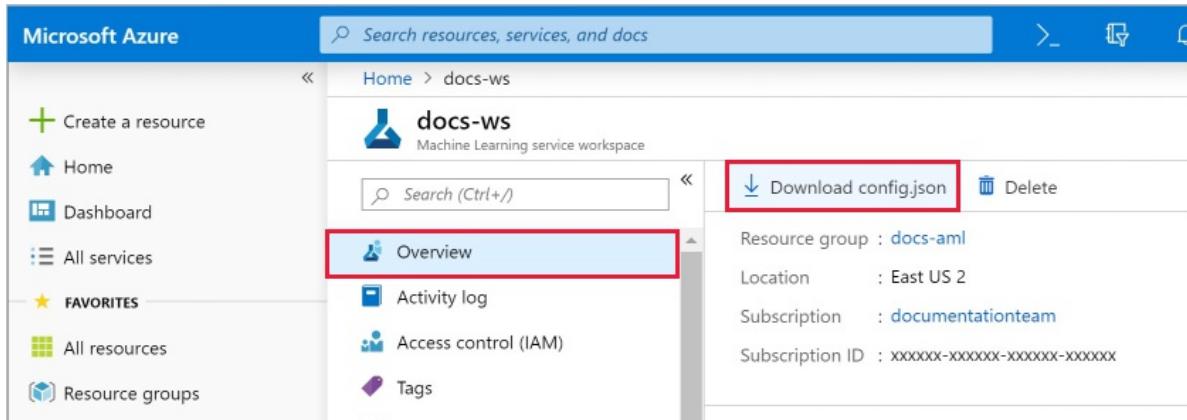
The Data Science Virtual Machine (DSVM) is a customized VM image built specifically for doing data science. If you [create a DSVM](#), the SDK and notebook server are installed and configured for you. However, you'll still need to create a workspace and clone the sample repository.

1. [Create an Azure Machine Learning workspace](#).
2. Clone [the GitHub repository](#).

```
git clone https://github.com/Azure/MachineLearningNotebooks.git
```

3. Add a workspace configuration file to the cloned directory using either of these methods:

- In the [Azure portal](#), select **Download config.json** from the **Overview** section of your workspace.



- Create a new workspace using code in the [configuration.ipynb](#) notebook in your cloned directory.
4. Start the notebook server from your cloned directory.

```
jupyter notebook
```

Next steps

Explore the [sample notebooks](#) to discover what Azure Machine Learning can do.

For more GitHub sample projects and examples, see these repos:

- [Azure/azureml-examples](#)
- [Microsoft/MLOps](#)
- [Microsoft/MLOpsPython](#)

Try these tutorials:

- [Train and deploy an image classification model with MNIST](#)
- [Prepare data and use automated machine learning to train a regression model with the NYC taxi data set](#)

Example pipelines & datasets for Azure Machine Learning designer

12/23/2020 • 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 run.

Depending on the sample pipeline and compute settings, runs 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 runs will take less time since the compute resources are already allocated. Additionally, the designer uses cached results for each module to further improve efficiency.

5. After the pipeline finishes running, you can review the pipeline and view the output for each module to learn more. Use the following steps to view module outputs:
 - a. Right-click the module 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 without documentation links by opening the samples and viewing the module comments instead.

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 module 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 without documentation links by opening the samples and viewing the module comments instead.

| [Image Classification using DenseNet](#) | Use computer vision modules to build image classification model based on PyTorch DenseNet.|

Recommender

Explore these built-in recommender samples. You can learn more about the samples without documentation links by opening the samples and viewing the module comments instead.

SAMPLE TITLE	DESCRIPTION
Wide & Deep based Recommendation - Restaurant Rating Prediction	Build a restaurant recommender engine from restaurant/user features and ratings.

SAMPLE TITLE	DESCRIPTION
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 without documentation links by opening the samples and viewing the module comments instead.

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 module 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	This data comes from the KDD Cup 2009 customer relationship prediction challenge (orange_small_train.data.zip). 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.
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	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."
Movie Ratings	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.
Weather Dataset	<p>Hourly land-based weather observations from NOAA (merged data from 201304 to 201310).</p> <p>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.</p> <p>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>
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>

Dataset Name	Dataset Description
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

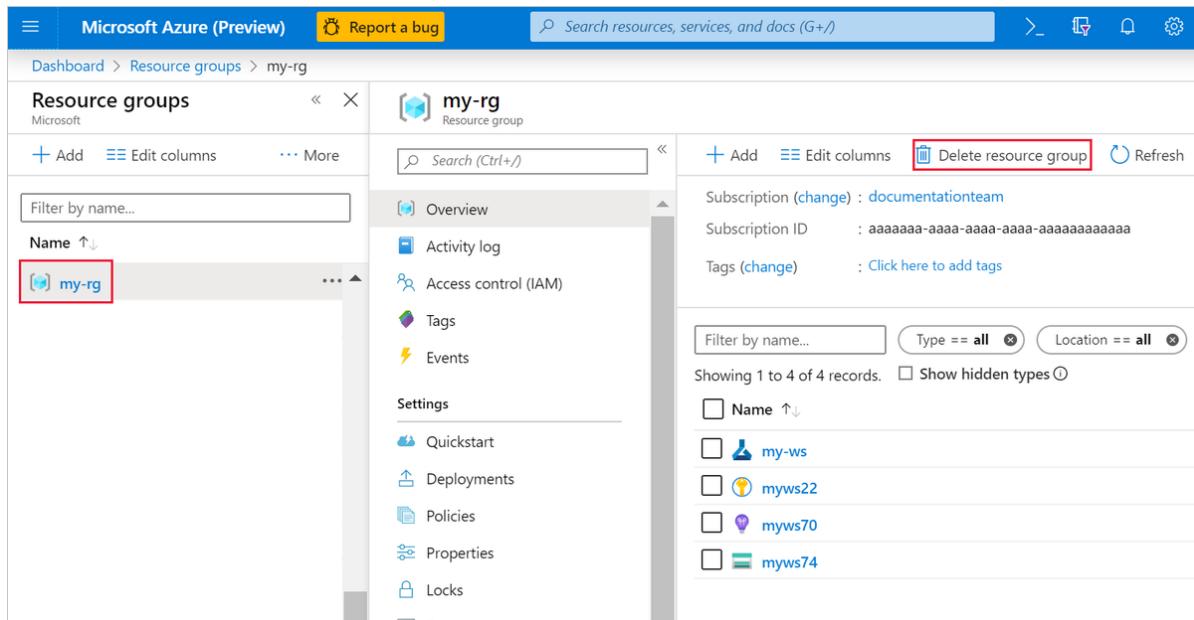
IMPORTANT

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.



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 interface. On the left sidebar, under 'Compute', the 'Compute' icon is highlighted with a red box. In the main area, the 'Inference Clusters' tab is selected. A table lists a single cluster named 'aks-compute'. The 'Delete' button next to the cluster name is also highlighted with a red box.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the Azure Machine Learning Datasets interface. On the left sidebar, the 'Datasets' icon is highlighted with a red box. In the main area, a specific dataset named 'TD-Sample_1:_Regression_-_Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1' is selected. 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

Learn the fundamentals of predictive analytics and machine learning with [Tutorial: Predict automobile price with the designer](#)

Plan and manage costs for Azure Machine Learning

12/23/2020 • 8 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. Finally, use cost-saving tips as you train your model with managed Azure Machine Learning compute clusters.

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.

When you train your machine learning models, use managed Azure Machine Learning compute clusters to take advantage of more cost-saving tips:

- Configure your training clusters for autoscaling
- Set quotas on your subscription and workspaces
- Set termination policies on your training run
- Use low-priority virtual machines (VM)
- Use an Azure Reserved VM Instance

Prerequisites

Cost analysis supports different kinds of Azure account types. 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 your 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 account. 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:

The screenshot shows the Azure Machine Learning Pricing calculator interface. At the top right, there's a search bar, a 'Portal' link, and a 'Free account' button. Below the header, a 'Your Estimate' section has a '+ Add' button. To the right, a red box highlights the 'Estimate total: \$167.17' text. The main area is titled 'Your Estimate' and shows 'Azure Machine Learning' selected. It displays 'Model Deployment with AKS, 1 D3 v2 (4 ...)' and a cost of '\$167.17'. On the left, there's a summary card for 'Azure Machine Learning' with fields for 'REGION:' (East US 2), 'USAGE TYPE:' (Model Deployment with AK), and 'INSTANCE:' (D3 v2: 4 vCPU(s), 14 GB RAM, \$0.229/hour). On the right, there are buttons for 'Clone' and 'Delete'. Below these, a 'More info' section includes links for 'Pricing details', 'Product details', and 'Documentation'. A 'Billing Option' section offers three choices: 'Pay as you go' (selected), '1 year reserved (~35% savings)', and '3 year reserved (~58% savings)'. At the bottom, it shows '1 Virtual' and '730 Hours' with a result of '= \$167.17 Per month'. A blue 'Chat with Sales' button is also present.

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](#)

Costs might accrue after resource deletion

When 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:

```
ws.delete(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 Monetary Credit with Azure Machine Learning

You can pay for Azure Machine Learning charges with your EA monetary commitment credit. However, you can't use EA monetary commitment credit to pay for charges for third party products and services including those from the Azure Marketplace.

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 when 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 or 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 these tips for containing costs on your machine learning compute resources.

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 run 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](#).

```
az ml computetarget create amlcompute --name testcluster --vm-size Standard_NC6 --min-nodes 0 --max-nodes 5 --idle-seconds-before-scaledown 300
```

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 run autetermination 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 run 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.

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.

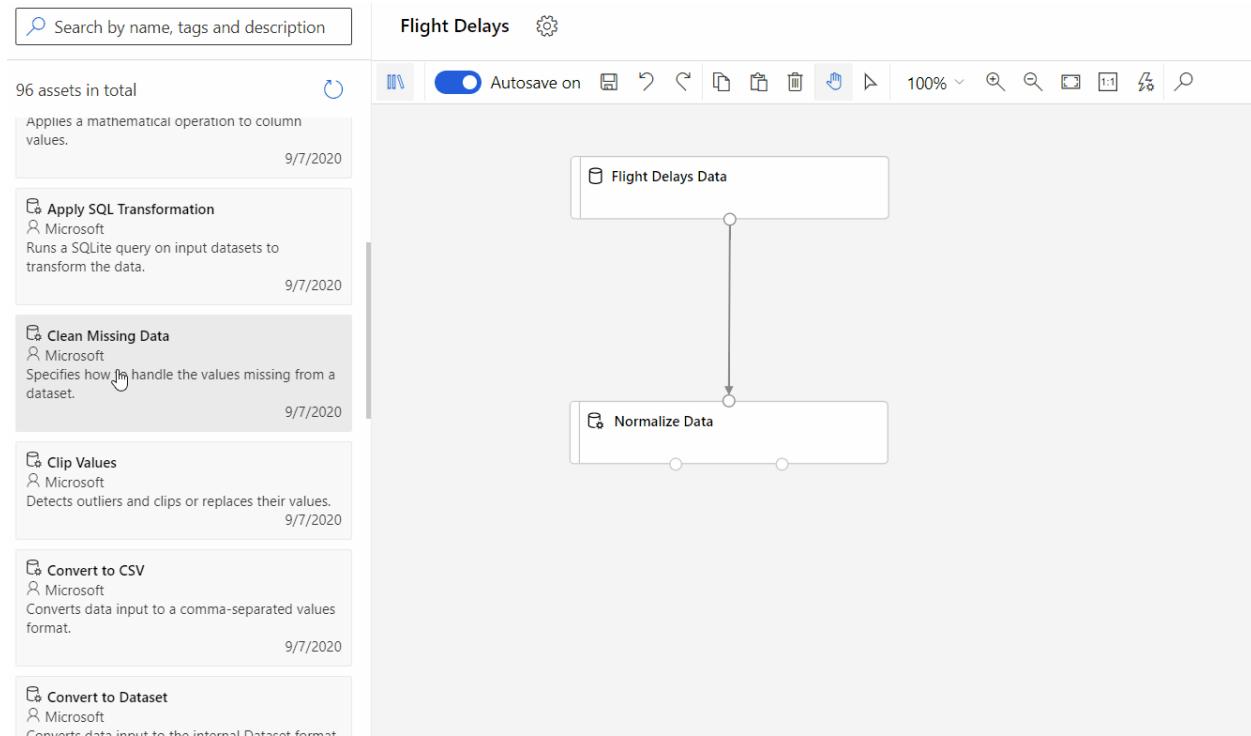
Next steps

- 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.

What is Azure Machine Learning designer?

12/23/2020 • 4 minutes to read • [Edit Online](#)

Azure Machine Learning designer lets you visually connect [datasets](#) and [modules](#) on an interactive canvas to create machine learning models. To learn how to get started with the designer, see [Tutorial: Predict automobile price with the designer](#)



The designer uses your Azure Machine Learning [workspace](#) to organize shared resources such as:

- [Pipelines](#)
- [Datasets](#)
- [Compute resources](#)
- [Registered models](#)
- [Published pipelines](#)
- [Real-time endpoints](#)

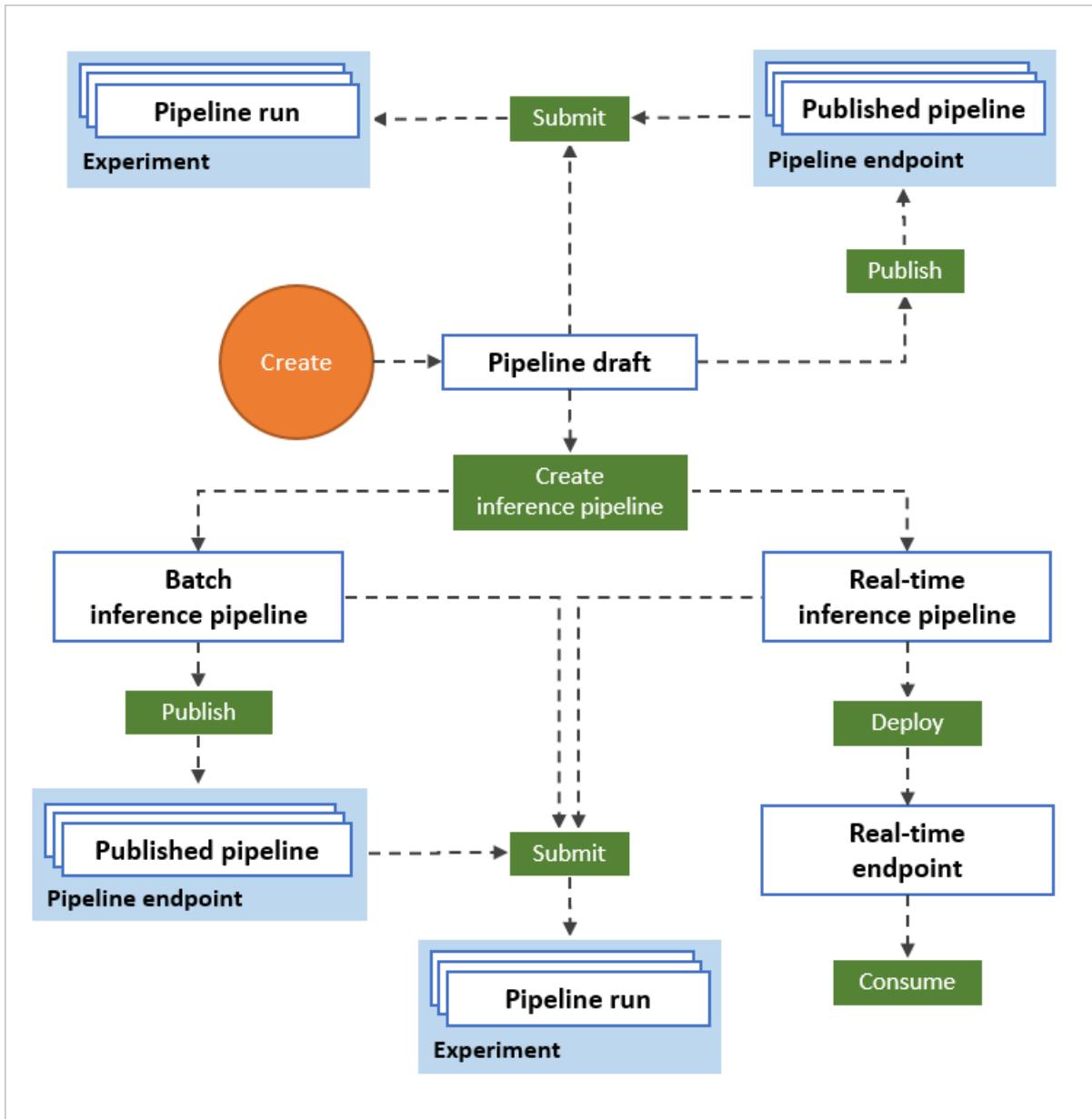
Model training and deployment

The designer gives you a visual canvas to build, test, and deploy machine learning models. With the designer you can:

- Drag-and-drop [datasets](#) and [modules](#) onto the canvas.
- Connect the modules 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 datasets.
 - Publish a [training pipeline](#) to reuse a single pipeline to train multiple models while changing parameters and datasets.
 - Publish a [batch inference pipeline](#) to make predictions on new data by using a previously trained

model.

- [Deploy](#) a real-time inference pipeline to a real-time endpoint to make predictions on new data in real-time.



Pipeline

A [pipeline](#) consists of datasets and analytical modules, 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 modules, configuring compute targets, creating parameters, and so on.

A valid pipeline has these characteristics:

- Datasets can only connect to modules.
- Modules can only connect to either datasets or other modules.
- All input ports for modules must have some connection to the data flow.
- All required parameters for each module must be set.

When you're ready to run your pipeline draft, you submit a pipeline run.

Pipeline run

Each time you run a pipeline, the configuration of the pipeline and its results are stored in your workspace as a **pipeline run**. You can go back to any pipeline run to inspect it for troubleshooting or auditing purposes. Clone a pipeline run to create a new pipeline draft for you to edit.

Pipeline runs are grouped into [experiments](#) to organize run history. You can set the experiment for every pipeline run.

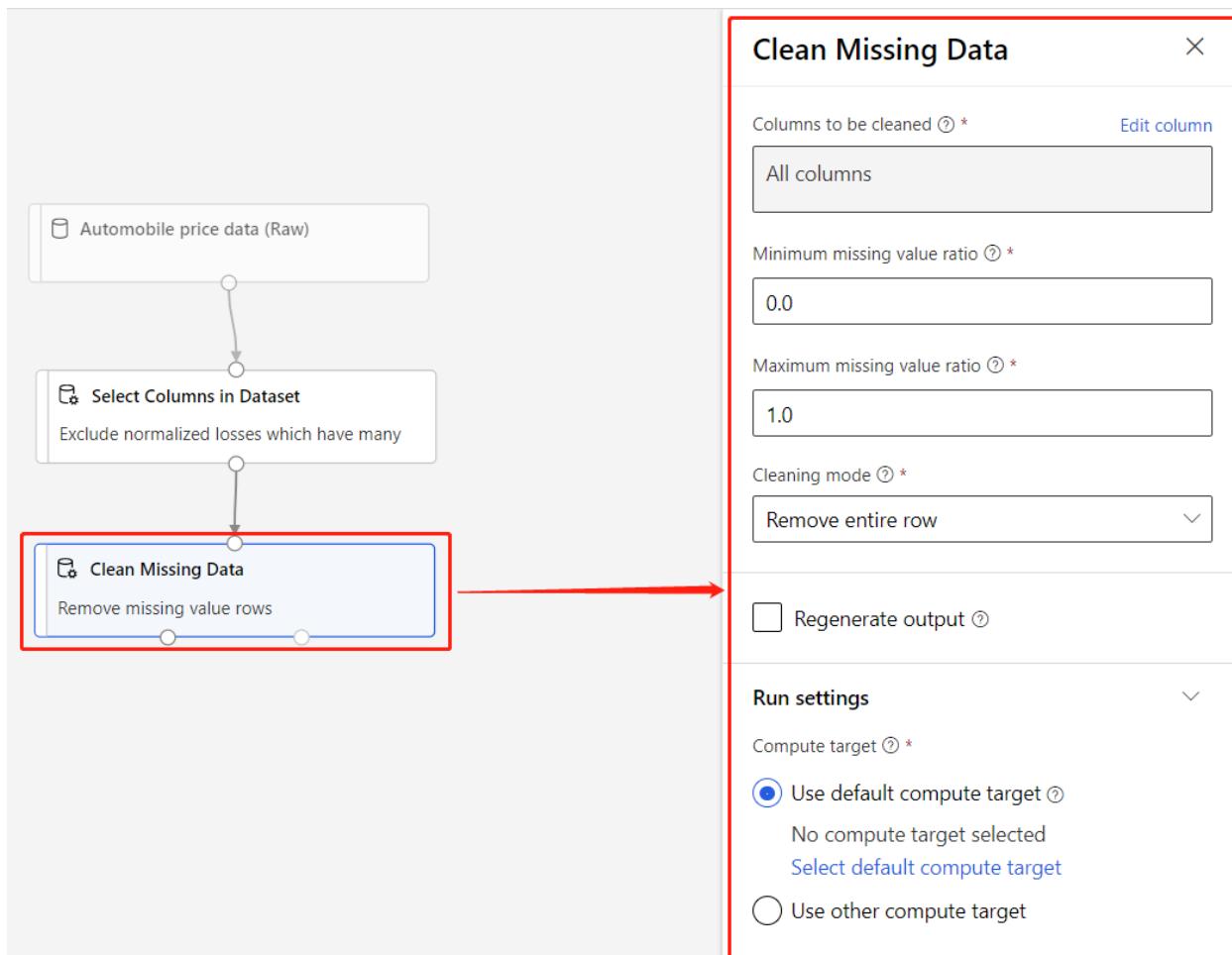
Datasets

A machine learning dataset makes it easy to access and work with your data. Several sample datasets are included in the designer for you to experiment with. You can [register](#) more datasets as you need them.

Module

A module is an algorithm that you can perform on your data. The designer has several modules ranging from data ingress functions to training, scoring, and validation processes.

A module may have a set of parameters that you can use to configure the module's internal algorithms. When you select a module on the canvas, the module'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 modules in the designer.



For some help navigating through the library of machine learning algorithms available, see [Algorithm & module reference overview](#). For help choosing an algorithm, see the [Azure Machine Learning Algorithm Cheat Sheet](#).

Compute resources

Use compute resources from your workspace to run your pipeline and host your deployed models as real-time endpoints or pipeline endpoints (for batch inference). The supported compute targets are:

COMPUTE TARGET	TRAINING	DEPLOYMENT
Azure Machine Learning compute	✓	
Azure Machine Learning compute instance	✓	
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 **real-time endpoint**. The real-time endpoint creates an interface between an external application and your scoring model. A call to a real-time endpoint returns prediction results to the application in real-time. To make a call to a real-time 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.

Real-time 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](#).

Publish

You can also publish a pipeline to a **pipeline endpoint**. Similar to a real-time endpoint, a pipeline endpoint lets you submit new pipeline runs 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 module.

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

5/7/2020 • 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 additional guidance, 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 module 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

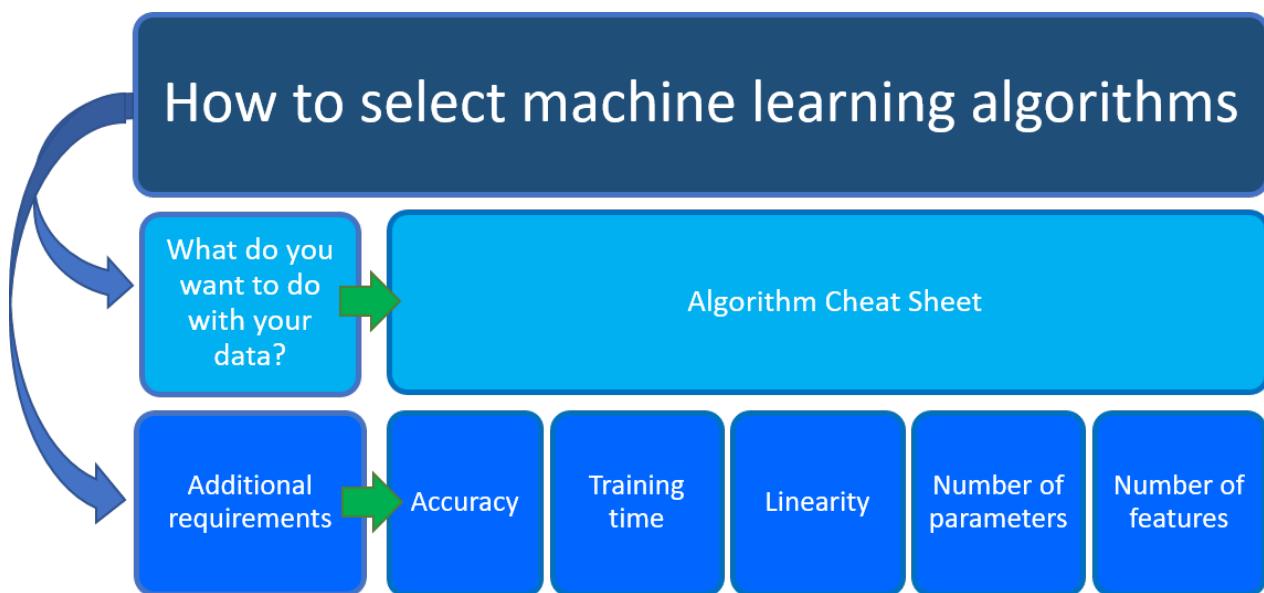
- See additional guidance on [How to select algorithms](#)
- [Learn about studio in Azure Machine Learning and the Azure portal.](#)
- [Tutorial: Build a prediction model in Azure Machine Learning designer.](#)
- [Learn about deep learning vs. machine learning.](#)

How to select algorithms for Azure Machine Learning

12/23/2020 • 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 module reference](#) for a complete list along with documentation about how each algorithm works and how to tune parameters to optimize the algorithm.

NOTE

To download the machine learning algorithm cheat sheet, go to [Azure Machine learning algorithm cheat sheet](#).

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 thread 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
Multiclass neural network	Good	Moderate	No	8	

ALGORITHM	ACCURACY	TRAINING TIME	LINEARITY	PARAMETERS	NOTES
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 module](#) computes a set of industry-standard evaluation metrics. You can use this module 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 module:

- 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 module](#).

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 module](#).
3. After training is completed, use the trained model with one of the [scoring modules](#) to make predictions on new data.

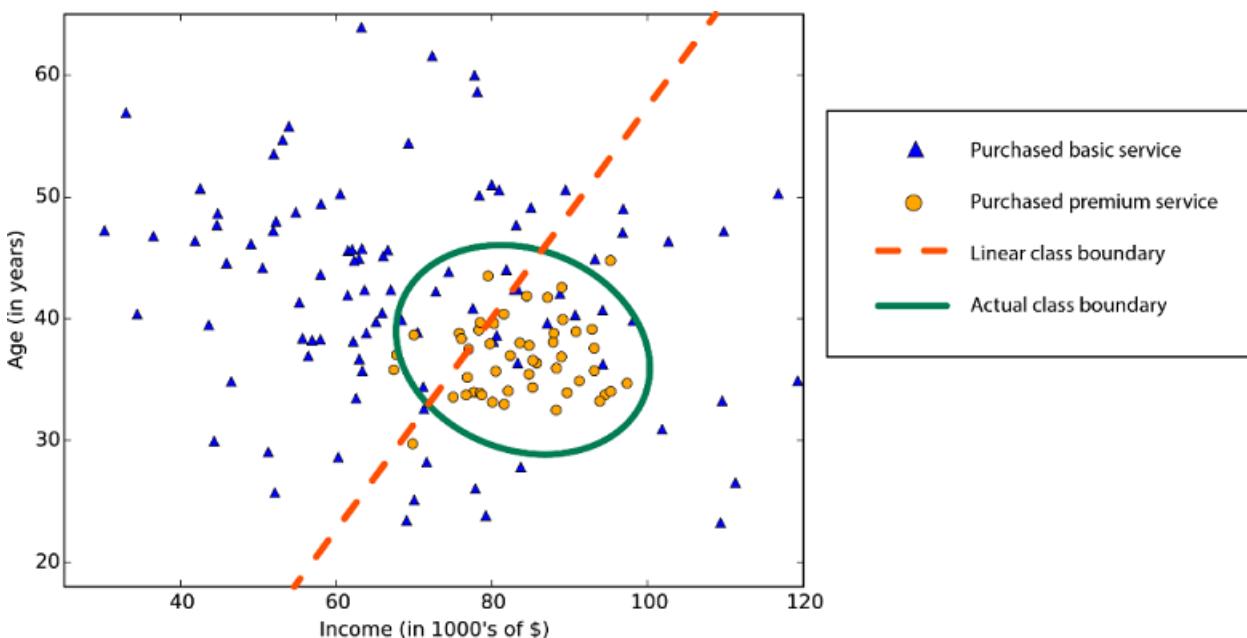
Linearity

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).

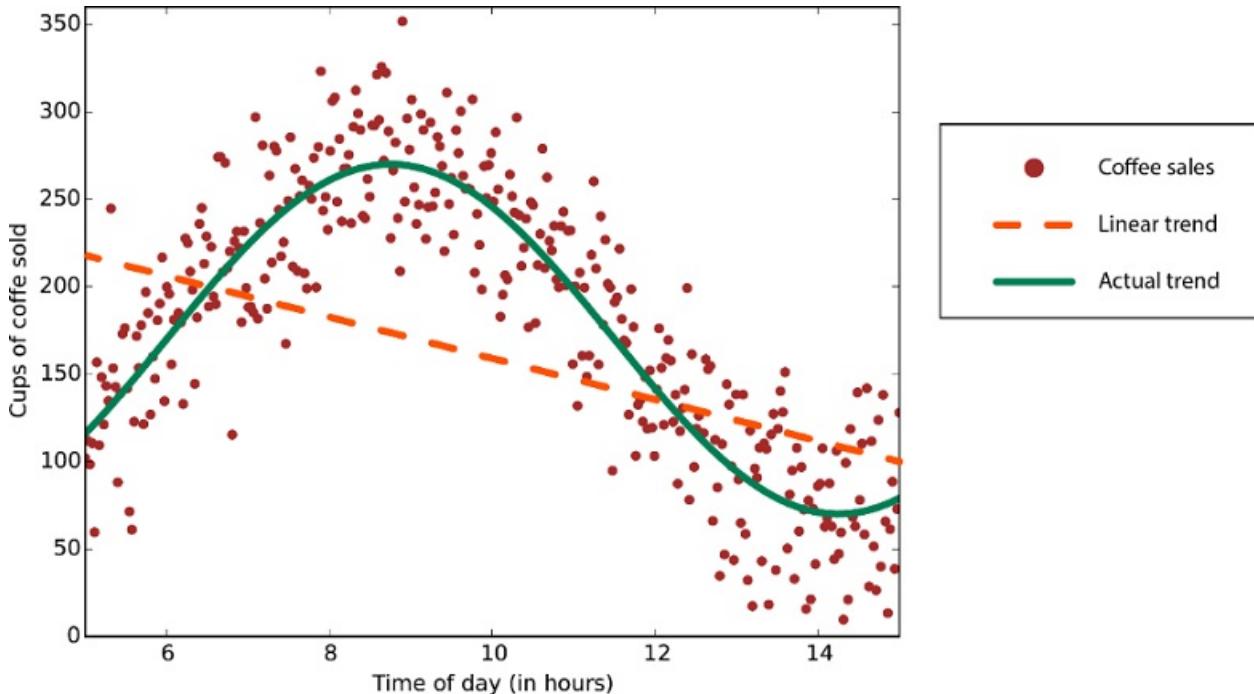
Lots of machine learning algorithms make use of linearity. In Azure Machine Learning designer, they include:

- [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 module](#) in Machine Learning designer: The goal of this module is to determine the optimum hyperparameters for a machine learning model. The module 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 module](#) in Machine Learning designer provides multiple feature selection algorithms to choose from. The module includes correlation

methods such as Pearson correlation and chi-squared values.

You can also use the [Permutation Feature Importance module](#) 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 module reference](#)
- To explore the relationship between deep learning, machine learning, and AI, see [Deep Learning vs. Machine Learning](#)

What is automated machine learning (AutoML)?

12/23/2020 • 12 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.

When to use AutoML: classify, regression, & forecast

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.

Data scientists, analysts, 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 [Beverage Production Forecast](#).

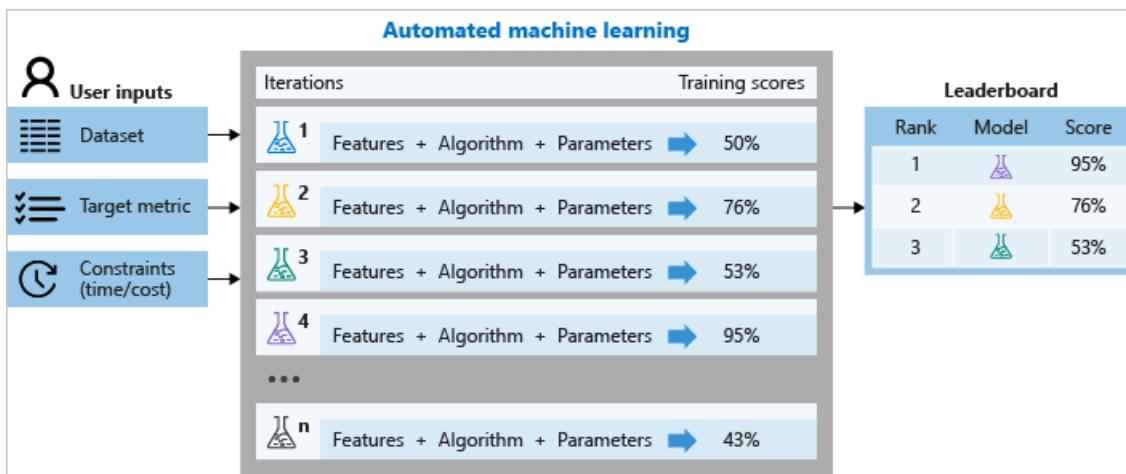
How automated ML works

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, or regression
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 compute target for model training**, such as your [local computer](#), [Azure Machine Learning Computes](#), [remote VMs](#), or [Azure Databricks](#). Learn about automated training [on a remote resource](#).
5. **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.
6. **Submit the training run.**
7. **Review the results**

The following diagram illustrates this process.



You can also inspect the logged run information, which [contains metrics](#) gathered during the run. The training run 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.

Learn how to use a [remote compute target](#).

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](#).

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.

Automatic featurization (standard)

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 will be applied to each model. Learn how AutoML helps [prevent over-fitting and imbalanced data](#) in your models.

SCALING & NORMALIZATION	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	This Scaler features by their quantile range

SCALING & NORMALIZATION	DESCRIPTION
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
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 ($\ 1$ or $\ 2$) equals one

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 run. 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.

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 run), 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 run, 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 runs• 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 runs• 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 run

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 runs/iterations in parallel	✓	

FEATURE	REMOTE	LOCAL
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 run	✓	
Forecasting	✓	✓
Create and run experiments in notebooks	✓	✓
Register and visualize experiment's info and metrics in UI	✓	✓
Data guardrails	✓	✓

Many models

The [Many Models Solution Accelerator](#) (preview) builds on Azure Machine Learning and enables you to use automated ML to train, operate, and manage hundreds or even thousands of machine learning models.

For example, building a model for **each instance or individual** in the following scenarios can lead to improved results:

- Predicting sales for each individual store
- Predictive maintenance for hundreds of oil wells
- Tailoring an experience for individual users.

AutoML in Azure Machine Learning

Azure Machine Learning offers two experiences for working with automated ML:

- For code experienced customers, [Azure Machine Learning Python SDK](#)
- For limited/no code experience customers, Azure Machine Learning studio at <https://ml.azure.com>

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, and forecasting	✓	✓

	THE PYTHON SDK	THE STUDIO WEB EXPERIENCE
Optimizes based on primary metric	✓	✓
Supports Azure ML compute as compute target	✓	✓
Configure forecast horizon, target lags & rolling window	✓	✓
Set exit criteria	✓	✓
Set concurrent iterations	✓	✓
Drop columns	✓	✓
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	✓	

Run control settings

These settings allow you to review and control your experiment runs and its child runs.

	THE PYTHON SDK	THE STUDIO WEB EXPERIENCE
Run summary table	✓	✓
Cancel runs & child runs	✓	✓
Get guardrails	✓	✓
Pause & resume runs	✓	

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: Automatically train a regression model with Azure Machine Learning Python SDK](#).
- **For a low or no-code experience**, see the [Tutorial: Create automated ML classification models with Azure Machine Learning studio](#).

How to articles provide additional detail into what functionality AutoML offers. For example,

- Configure the settings for automatic training experiments
 - In Azure Machine Learning studio, [use these steps](#).
 - With the Python SDK, [use these steps](#).
- Learn how to auto train using time series data, [with these steps](#).

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

12/23/2020 • 7 minutes to read • [Edit Online](#)

Over-fitting 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 over-fitting

Over-fitting 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 over-fitted. However, test accuracy should always be less than training accuracy, and the distinction for over-fit 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 over-fitting 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 over-fitting; 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 over-fitting

In the most egregious cases, an over-fitted model will assume that the feature value combinations seen during training will always result in the exact same output for the target.

The best way to prevent over-fitting 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 over-fitting. In settings other than automated ML, all six best-practices are worth following to avoid over-fitting models.

Best practices you implement

Using **more data** is the simplest and best possible way to prevent over-fitting, 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 over-fitting between your train and test sets, but there may be over-fitting present when compared to live test data.

Target leakage is a similar issue, where you may not see over-fitting 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.

Removing features can also help with over-fitting 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 over-fitting.

Best practices automated ML implements

Regularization is the process of minimizing a cost function to penalize complex and over-fitted 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 over-fitting. In simple terms, automated ML will vary how much a model is regulated and choose the best result.

Automated ML also implements explicit **model complexity limitations** to prevent over-fitting. 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 (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 over-fit, but by using many of them it reduces the probability that your final model is over-fit. 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 runs 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: Automatically train a regression model with Azure Machine Learning](#)
- Configure the settings for automatic training experiment:
 - In Azure Machine Learning studio, [use these steps](#).
 - With the Python SDK, [use these steps](#).

What is an Azure Machine Learning workspace?

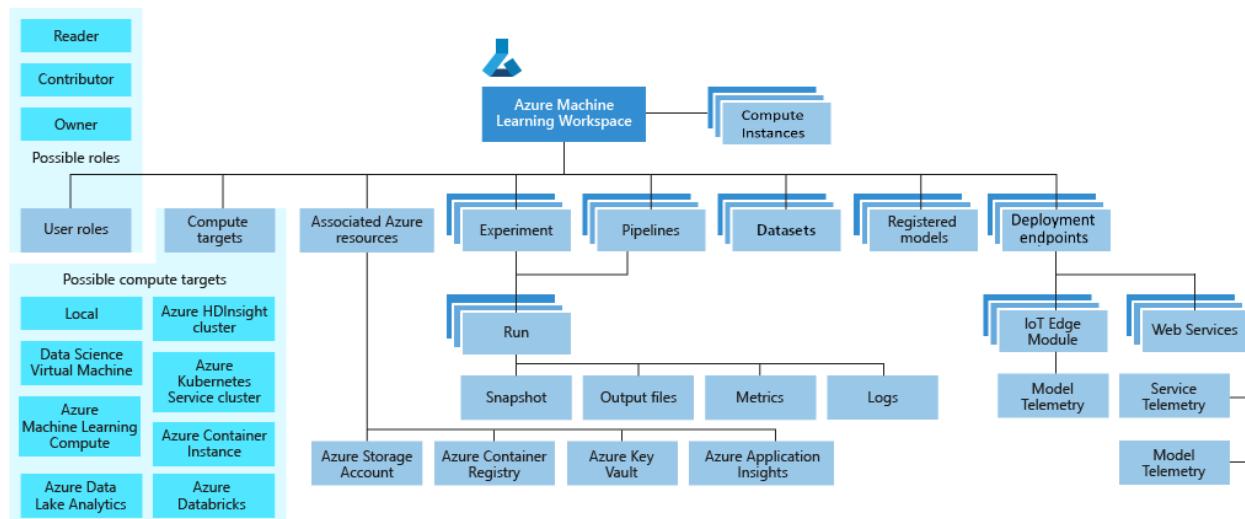
12/23/2020 • 5 minutes to read • [Edit Online](#)

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 Azure Container Instances, Azure Kubernetes Service, or to a field-programmable gate array (FPGA) as a REST-based HTTP endpoint. You can also deploy the model to an Azure IoT Edge device as a module.

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.
- [Experiments](#) are training runs you use to build your models.
- [Pipelines](#) are reusable workflows for training and retraining your model.
- [Datasets](#) 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 a [deployment endpoint](#).

Tools for workspace interaction

You can interact with your workspace in the following ways:

IMPORTANT

Tools 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](#).

- On the web:
 - [Azure Machine Learning studio](#)
 - [Azure Machine Learning designer](#)
- In any Python environment with the [Azure Machine Learning SDK for Python](#).
- In any R environment with the [Azure Machine Learning SDK for R \(preview\)](#).
- 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 experiment 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 experiments, 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 / R SDK	CLI	VS CODE
Create a workspace	✓		✓	✓	✓
Manage workspace access	✓			✓	
Create and manage compute resources	✓	✓	✓	✓	
Create a Notebook VM		✓			

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 the [Azure portal](#) for a point-and-click interface to walk you through each step.
- 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.

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 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 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 you use during training and when you deploy a model. To minimize costs, ACR is **lazy-loaded** until deployment images are created.
- **Azure Application Insights:** Stores monitoring information about your models.
- **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](#), [R SDK](#), or the [Azure Machine Learning CLI](#) [using an ARM template](#).

What happened to Enterprise edition

As of September 2020, all capabilities that were available in Enterprise edition workspaces are now also available in Basic edition workspaces. New Enterprise workspaces can no longer be created. Any SDK, CLI, or Azure Resource Manager calls that use the `sku` parameter will continue to work but a Basic workspace will be provisioned.

Beginning December 21st, all Enterprise Edition workspaces will be automatically set to Basic Edition, which has the same capabilities. No downtime will occur during this process. On January 1, 2021, Enterprise Edition will be formally retired.

In either editions, customers are responsible for the costs of Azure resources consumed and will not need to pay any additional charges for Azure Machine Learning. Please refer to the [Azure Machine Learning pricing page](#) for more details.

Next steps

To get started with Azure Machine Learning, see:

- [Azure Machine Learning overview](#)
- [Create and manage a workspace](#)
- [Tutorial: Get started with Azure Machine Learning in your development environment](#)
- [Tutorial: Get started creating your first ML experiment on a compute instance](#)
- [Tutorial: Get started with Azure Machine Learning with the R SDK](#)
- [Tutorial: Create your first classification model with automated machine learning](#)
- [Tutorial: Predict automobile price with the designer](#)

What are Azure Machine Learning environments?

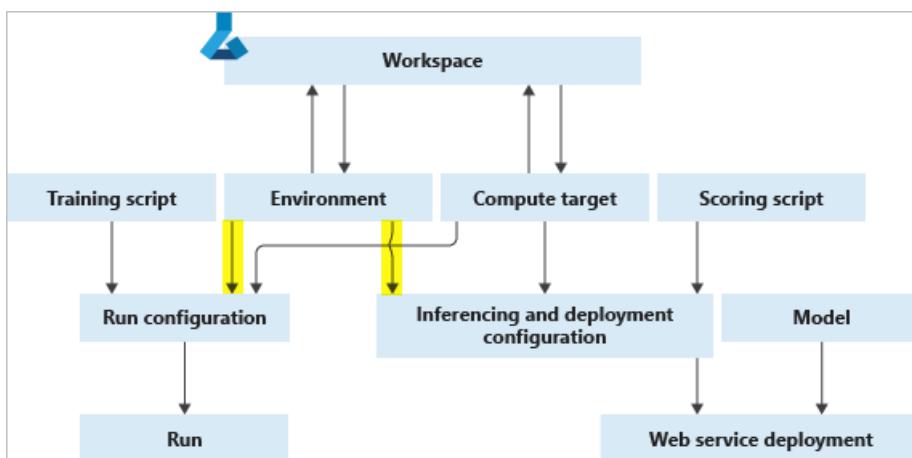
12/23/2020 • 6 minutes to read • [Edit Online](#)

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 run times (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 run configuration (for training) and your inference and deployment configuration (for web service deployments).



The environment, compute target and training script together form the run configuration: the full specification of a training run.

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. Conda doesn't check your environment or install anything for you. If you're defining your own environment, you must list `azureml-defaults` with version `>= 1.0.45` as a pip dependency. This package contains the functionality that's needed to host the model as a web service.

You use system-managed environments when you want `Conda` to manage the Python environment and the script dependencies for you. A new conda environment is built based on the `conda_dependencies` object. The Azure Machine Learning service assumes this type of environment by default, because of its usefulness on remote

compute targets that aren't manually configurable.

Create and manage environments

You can create environments by:

- Defining new `Environment` objects, either by using a curated environment or by defining your own dependencies.
- Using existing `Environment` objects from your workspace. This approach allows for consistency and reproducibility with your dependencies.
- Importing from an existing Anaconda environment definition.
- Using the Azure Machine Learning CLI
- [Using the VS Code extension](#)

For specific code samples, see the "Create an environment" section of [How to use environments](#). Environments are also easily managed through your workspace. They include the following functionality:

- Environments are automatically registered to your workspace when you submit an experiment. They can also be manually registered.
- You can fetch environments from your workspace to use for training or deployment, or to make edits to the environment definition.
- With versioning, you can see changes to your environments over time, which ensures reproducibility.
- You can build Docker images automatically from your environments.

For code samples, see the "Manage environments" section of [How to use environments](#).

Environment building, caching, and reuse

The Azure Machine Learning service builds environment definitions into Docker images and conda environments. It also caches the environments so they can be reused in subsequent training runs and service endpoint deployments. Running a training script remotely requires the creation of a Docker image whereas, a local run can use a Conda environment directly.

Submitting a run using an environment

When you first submit a remote run 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 run execution, the image is retrieved by the compute target from the relevant ACR.

For local runs, 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 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 run, the Azure Machine Learning service reuses the cached image from the Workspace ACR.

To view the details of a cached image, use [Environment.get_image_details](#) method.

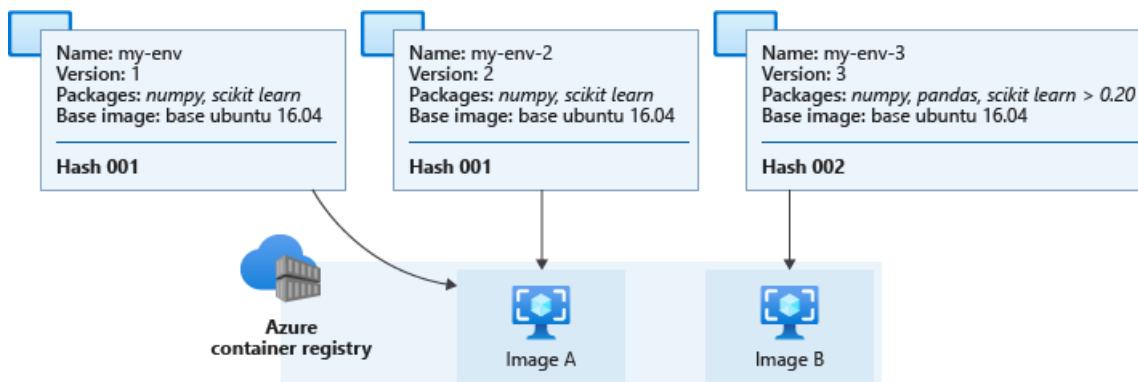
To determine whether to reuse a cached image or build a new one, the service computes a hash value from the environment definition and compares it to the hashes of existing environments. The hash is based on:

- Base image property value
- Custom docker steps property value
- List of Python packages in Conda definition
- List of packages in Spark definition

The hash doesn't depend on environment name or version - if you rename your environment or create a new environment with the exact properties and packages of an existing one, then the hash value remains the same. However, environment definition changes, such as adding or removing a Python package or changing the package version, cause the hash value to change. Changing the order of dependencies or channels in an environment will result in a new environment and thus require a new image build. It is important to note that any change to a curated environment will invalidate the hash and result in a new "non-curated" environment.

The computed hash value is compared to those in the Workspace and Global ACR (or on the compute target for local runs). If there is a match then the cached image is pulled, otherwise an image build is triggered. The duration to pull a cached image includes the download time whereas the duration to pull a newly built image includes both the build time and the download time.

The following diagram shows three environment definitions. Two of them have different names and versions, but identical base image and Python packages. But they have the same hash and thus correspond to the same cached image. The third environment has different Python packages and versions, and therefore corresponds to a different cached image.



IMPORTANT

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.

To update the package, specify a version number to force image rebuild, for example `numpy==1.18.1`. New dependencies, including nested ones, will be installed that might break a previously working scenario.

WARNING

The [Environment.build](#) method will rebuild the cached image, with possible side-effect of updating unpinned packages and breaking reproducibility for all environment definitions corresponding to that cached image.

Next steps

- Learn how to [create and use environments](#) in Azure Machine Learning.
- See the Python SDK reference documentation for the [environment class](#).
- See the R SDK reference documentation for [environments](#).

What is an Azure Machine Learning compute instance?

12/23/2020 • 8 minutes to read • [Edit Online](#)

An Azure Machine Learning compute instance is a managed cloud-based workstation for data scientists.

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 production grade model training, use an [Azure Machine Learning compute cluster](#) with multi-node scaling capabilities. For production grade model deployment, use [Azure Kubernetes Service cluster](#).

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.

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 - RStudio (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. You can also use VS Code with compute instances.
Managed & secure	Reduce your security footprint and add compliance with enterprise security requirements. Compute instances provide robust management policies and secure networking configurations such as: - Autoprovisioning from Resource Manager templates or Azure Machine Learning SDK - Azure role-based access control (Azure RBAC) - Virtual network support - SSH policy to enable/disable SSH access 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.

KEY BENEFITS	DESCRIPTION
--------------	-------------

You can [create a compute instance](#) yourself, or an administrator can [create a compute instance for you](#).

Tools and 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 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	CUDA cuDNN NVIDIA Blob FUSE
Intel MPI library	
Azure CLI	
Azure Machine Learning samples	
Docker	
Nginx	
NCCL 2.0	
Protobuf	
R TOOLS & ENVIRONMENTS	DETAILS
RStudio Server Open Source Edition (preview)	
R kernel	

R TOOLS & ENVIRONMENTS	DETAILS
Azure Machine Learning SDK for R	azuremlsdk SDK samples
PYTHON TOOLS & ENVIRONMENTS	DETAILS
Anaconda Python	
Jupyter and extensions	
Jupyterlab and extensions	
Azure Machine Learning SDK for Python from PyPI	Includes most of the azureml extra packages. To see the full list, open a terminal window on your compute instance and run <pre>conda list -n azureml_py36 azureml*</pre>
Other PyPI packages	<code>jupyter</code> <code>tensorboard</code> <code>nbconvert</code> <code>notebook</code> <code>Pillow</code>
Conda packages	<code>cython</code> <code>numpy</code> <code>ipykernel</code> <code>scikit-learn</code> <code>matplotlib</code> <code>tqdm</code> <code>joblib</code> <code>nodejs</code> <code>nb_conda_kernels</code>
Deep learning packages	<code>PyTorch</code> <code>TensorFlow</code> <code>Keras</code> <code>Horovod</code> <code>MLflow</code> <code>pandas-ml</code> <code>scrapbook</code>
ONNX packages	<code>keras2onnx</code> <code>onnx</code> <code>onnxconverter-common</code> <code>skl2onnx</code> <code>onnxmлttools</code>
Azure Machine Learning Python & R SDK samples	

Python packages are all installed in the **Python 3.6 - AzureML** environment.

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.

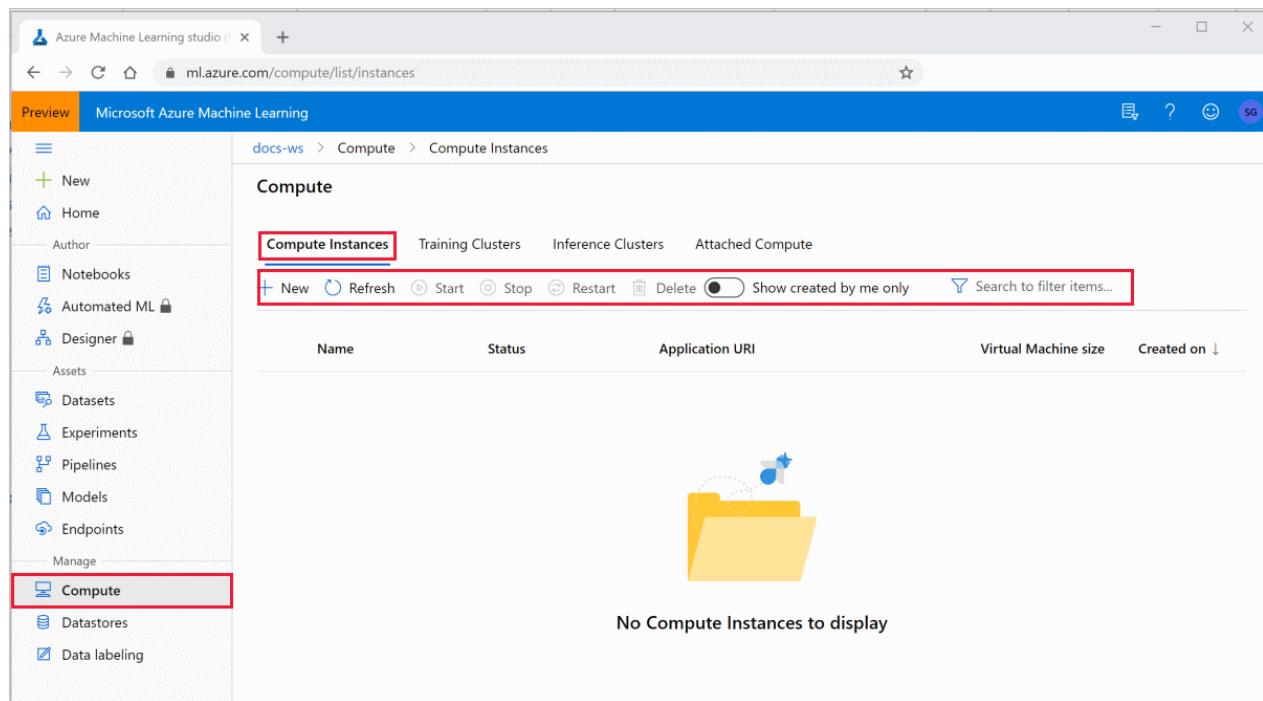
You can also clone the latest Azure Machine Learning samples to your folder under the user files directory in the workspace file share.

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.

You can use the `/tmp` directory on the compute instance for your temporary data. However, do not write large files of data on the OS disk of the compute instance. Use [datastores](#) instead. If you have installed JupyterLab git extension, it can also lead to slow down in compute instance performance.

Managing a compute instance

In your workspace in Azure Machine Learning studio, select **Compute**, then select **Compute Instance** on the top.



The screenshot shows the Azure Machine Learning studio interface. The left sidebar has a 'Compute' section highlighted with a red box. The main area shows the 'Compute' section with a 'Compute Instances' tab selected, also highlighted with a red box. Below the tabs is a toolbar with buttons for New, Refresh, Start, Stop, Restart, Delete, and Show created by me only, along with a search bar. A message 'No Compute Instances to display' is shown. The URL in the browser is ml.azure.com/compute/list/instances.

You can perform the following actions:

- [Create a compute instance](#).
- Refresh the compute instances tab.
- Start, stop, and restart a compute instance. You do pay for the instance whenever it is running. Stop the compute instance when you are not using it to reduce cost. Stopping a compute instance deallocates it. Then start it again when you need it. Please note stopping the compute instance stops the billing for compute hours but you will still be billed for disk, public IP, and standard load balancer.
- Delete a compute instance.

- Filter the list of compute instances to show only those you have created.

For each compute instance in your workspace that you can use, you can:

- Access Jupyter, JupyterLab, RStudio on the compute instance
- SSH into compute instance. SSH access is disabled by default but can be enabled at compute instance creation time. SSH access is through public/private key mechanism. The tab will give you details for SSH connection such as IP address, username, and port number.
- Get details about a specific compute instance such as IP address, and region.

[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, and can terminal in through Jupyter/JupyterLab/RStudio. Compute instance will have single-user log in and all actions will use that user's identity for Azure RBAC and attribution of experiment runs. 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*

Create a compute instance

In 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.

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:

- [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 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](#)

The data scientist can start, stop, and restart the compute instance. They can use the compute instance for:

- Jupyter
- JupyterLab
- RStudio
- Integrated notebooks

Compute target

Compute instances can be used as a [training compute target](#) similar to Azure Machine Learning compute training clusters.

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). Two jobs 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.

What happened to Notebook VM?

Compute instances are replacing the Notebook VM.

Any notebook files stored in the workspace file share and data in workspace data stores will be accessible from a compute instance. However, any custom packages previously installed on a Notebook VM will need to be reinstalled on the compute instance. Quota limitations, which apply to compute clusters creation will apply to compute instance creation as well.

New Notebook VMs cannot be created. However, you can still access and use Notebook VMs you have created, with full functionality. Compute instances can be created in same workspace as the existing Notebook VMs.

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?

12/23/2020 • 7 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 or IoT device 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 run. 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.

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	
Remote VM	Yes	Yes	
Azure Databricks	Yes (SDK local mode only)	Yes	

TRAINING TARGETS	AUTOMATED MACHINE LEARNING	MACHINE LEARNING PIPELINES	AZURE MACHINE LEARNING DESIGNER
Azure Data Lake Analytics		Yes	
Azure HDInsight		Yes	
Azure Batch		Yes	

Learn more about how to [submit a training run to a compute target](#).

Compute targets for inference

The following compute resources can be used to host your model deployment.

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	FPGA 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 Kubernetes Service (AKS)	Real-time inference	Yes (web service deployment)	Yes	Use for high-scale production deployments. Provides fast response time and autoscaling of the deployed service. Cluster autoscaling isn't supported through the Azure Machine Learning SDK. To change the nodes in the AKS cluster, use the UI for your AKS cluster in the Azure portal. Supported in the designer.
Azure Container Instances	Testing or development			Use for low-scale CPU-based workloads that require less than 48 GB of RAM. Supported in the designer.

COMPUTE TARGET	USED FOR	GPU SUPPORT	FPGA SUPPORT	DESCRIPTION
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 realtime inference.

NOTE

Although compute targets like local, Azure Machine Learning compute, 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 AKS.

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.
- Use single-node AKS clusters for dev/test of larger models.

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 one of the following deployment scenarios:

- As a *web service* that's used for real-time inference. Web service deployments use one of the following compute targets:
 - [Local computer](#)
 - [Azure Machine Learning compute instance](#)
 - [Azure Container Instances](#)
 - [Azure Kubernetes Service](#)
 - Azure Functions (preview). Deployment to Functions only relies on Azure Machine Learning to build the Docker container. From there, it's deployed by using Functions. For more information, see [Deploy a machine learning model to Azure Functions \(preview\)](#).
- As a *batch inference* endpoint that's used to periodically process batches of data. Batch inferences use [Azure Machine Learning compute clusters](#).
- To an *IoT device* (preview). Deployment to an IoT device only relies on Azure Machine Learning to build the Docker container. From there, it's deployed by using Azure IoT Edge. For more information, see [Deploy as an IoT Edge module \(preview\)](#).

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 CLI:
 - [Compute instance](#).
 - [Compute cluster](#).
- The [R SDK](#) (preview).
- An Azure Resource Manager template. For an example template, see [Create an Azure Machine Learning compute cluster](#).
- A machine learning [extension for the Azure CLI](#).

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	✓	
Autoscales each time you submit a run	✓	
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

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.
- Some VM series are restricted. To use a restricted series, contact support and request a quota increase for the series. For information on how to contact support, see [Azure support options](#).

See the following table to learn more about supported series and restrictions.

SUPPORTED VM SERIES	RESTRICTIONS
D	None.
Dv2	None.
Dv3	None.
DSv2	None.

SUPPORTED VM SERIES	RESTRICTIONS
DSv3	None.
FSv2	None.
HBv2	Requires approval.
HCS	Requires approval.
M	Requires approval.
NC	None.
NCsv2	Requires approval.
NCsv3	Requires approval.
NDs	Requires approval.
NDv2	Requires approval.
NV	None.
NVv3	Requires approval.

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](#)

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.

Next steps

Learn how to:

- [Use a compute target to train your model](#)
- [Deploy your model to a compute target](#)

Secure data access in Azure Machine Learning

12/23/2020 • 4 minutes to read • [Edit Online](#)

Azure Machine Learning makes it easy to connect to your data in the cloud. It provides an abstraction layer over the underlying storage service, so you can securely access and work with your data without having to write code specific to your storage type. Azure Machine Learning also provides the following data capabilities:

- Interoperability with Pandas and Spark DataFrames
- Versioning and tracking of data lineage
- Data labeling
- Data drift monitoring

Data workflow

When you're ready to use the data in your cloud-based storage solution, we recommend the following data delivery workflow. This workflow assumes you have an [Azure storage account](#) and data in a cloud-based storage service in Azure.

1. Create an [Azure Machine Learning datastore](#) to store connection information to your Azure storage.
 2. From that datastore, create an [Azure Machine Learning dataset](#) to point to a specific file(s) in your underlying storage.
 3. To use that dataset in your machine learning experiment you can either
 - a. Mount it to your experiment's compute target for model training.
- OR
- b. Consume it directly in Azure Machine Learning solutions like, automated machine learning (automated ML) experiment runs, machine learning pipelines, or the [Azure Machine Learning designer](#).
4. Create [dataset monitors](#) for your model output dataset to detect for data drift.
 5. If data drift is detected, update your input dataset and retrain your model accordingly.

The following diagram provides a visual demonstration of this recommended workflow.

Datastores

Azure Machine Learning datastores securely keep the connection information to your Azure storage, so you don't have to code it in your scripts. [Register and create a datastore](#) to easily connect to your storage account, and access the data in your underlying Azure storage service.

Supported cloud-based storage services in Azure that can be registered as datastores:

- Azure Blob Container
- Azure File Share
- Azure Data Lake
- Azure Data Lake Gen2
- Azure SQL Database

- Azure Database for PostgreSQL
- Databricks File System
- Azure Database for MySQL

Datasets

Azure Machine Learning datasets aren't copies of your data. By creating a dataset, you create a reference to the data in its storage service, along with a copy of its metadata.

Because datasets are lazily evaluated, and the data remains in its existing location, you

- Incur no extra storage cost.
- Don't risk unintentionally changing your original data sources.
- Improve ML workflow performance speeds.

To interact with your data in storage, [create a dataset](#) to package your data into a consumable object for machine learning tasks. Register the dataset to your workspace to share and reuse it across different experiments without data ingestion complexities.

Datasets can be created from local files, public urls, [Azure Open Datasets](#), or Azure storage services via datastores.

There are 2 types of datasets:

- A [FileDataset](#) references single or multiple files in your datastores or public URLs. If your data is already cleansed and ready to use in training experiments, you can [download or mount files](#) referenced by FileDatasets to your compute target.
- A [TabularDataset](#) represents data in a tabular format by parsing the provided file or list of files. You can load a TabularDataset into a pandas or Spark DataFrame for further manipulation and cleansing. For a complete list of data formats you can create TabularDatasets from, see the [TabularDatasetFactory class](#).

Additional datasets capabilities can be found in the following documentation:

- [Version and track](#) dataset lineage.
- [Monitor your dataset](#) to help with data drift detection.

Work with your data

With datasets, you can accomplish a number of machine learning tasks through seamless integration with Azure Machine Learning features.

- Create a [data labeling project](#).
- Train machine learning models:
 - [automated ML experiments](#)
 - the [designer](#)
 - [notebooks](#)
 - [Azure Machine Learning pipelines](#)
- Access datasets for scoring with [batch inference](#) in [machine learning pipelines](#).
- Set up a dataset monitor for [data drift](#) detection.

Data labeling

Labeling large amounts of data has often been a headache in machine learning projects. Those with a computer vision component, such as image classification or object detection, generally require thousands of images and corresponding labels.

Azure Machine Learning gives you a central location to create, manage, and monitor labeling projects. Labeling projects help coordinate the data, labels, and team members, allowing you to more efficiently manage the labeling tasks. Currently supported tasks are image classification, either multi-label or multi-class, and object identification using bounded boxes.

Create a [data labeling project](#), and output a dataset for use in machine learning experiments.

Data drift

In the context of machine learning, data drift is the change in model input data that leads to model performance degradation. It is one of the top reasons model accuracy degrades over time, thus monitoring data drift helps detect model performance issues.

See the [Create a dataset monitor](#) article, to learn more about how to detect and alert to data drift on new data in a dataset.

Next steps

- Create a dataset in Azure Machine Learning studio or with the Python SDK [using these steps](#).
- Try out dataset training examples with our [sample notebooks](#).

Data ingestion options for Azure Machine Learning workflows

12/23/2020 • 2 minutes to read • [Edit Online](#)

In this article, you learn the pros and cons of data ingestion options available with Azure Machine Learning.

Choose from:

- [Azure Data Factory](#) pipelines, specifically built to extract, load, and transform data
- [Azure Machine Learning Python SDK](#), providing a custom code solution for data ingestion tasks.
- a combination of both

Data ingestion is the process in which unstructured data is extracted from one or multiple sources and then prepared for training machine learning models. It's also time intensive, especially if done manually, and if you have large amounts of data from multiple sources. Automating this effort frees up resources and ensures your models use the most recent and applicable data.

Azure Data Factory

[Azure Data Factory](#) offers native support for data source monitoring and triggers for data ingestion pipelines.

The following table summarizes the pros and cons for using Azure Data Factory for your data ingestion workflows.

PROS	CONS
Specifically built to extract, load, and transform data.	Currently offers a limited set of Azure Data Factory pipeline tasks
Allows you to create data-driven workflows for orchestrating data movement and transformations at scale.	Expensive to construct and maintain. See Azure Data Factory's pricing page for more information.
Integrated with various Azure tools like Azure Databricks and Azure Functions	Doesn't natively run scripts, instead relies on separate compute for script runs
Natively supports data source triggered data ingestion	
Data preparation and model training processes are separate.	
Embedded data lineage capability for Azure Data Factory dataflows	
Provides a low code experience user interface for non-scripting approaches	

These steps and the following diagram illustrate Azure Data Factory's data ingestion workflow.

1. Pull the data from its sources
2. Transform and save the data to an output blob container, which serves as data storage for Azure Machine Learning

- With prepared data stored, the Azure Data Factory pipeline invokes a training Machine Learning pipeline that receives the prepared data for model training

Learn how to build a data ingestion pipeline for Machine Learning with [Azure Data Factory](#).

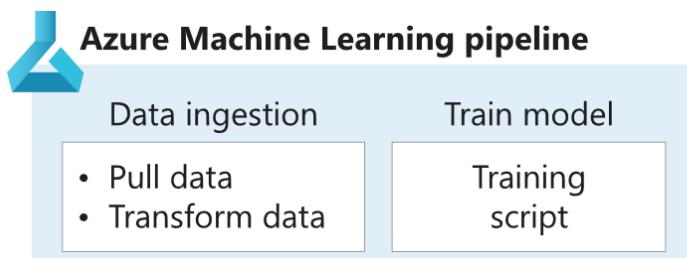
Azure Machine Learning Python SDK

With the [Python SDK](#), you can incorporate data ingestion tasks into an [Azure Machine Learning pipeline](#) step.

The following table summarizes the pros and con for using the SDK and an ML pipelines step for data ingestion tasks.

PROS	CONS
Configure your own Python scripts	Does not natively support data source change triggering. Requires Logic App or Azure Function implementations
Data preparation as part of every model training execution	Requires development skills to create a data ingestion script
Supports data preparation scripts on various compute targets, including Azure Machine Learning compute	Does not provide a user interface for creating the ingestion mechanism

In the following diagram, the Azure Machine Learning pipeline consists of two steps: data ingestion and model training. The data ingestion step encompasses tasks that can be accomplished using Python libraries and the Python SDK, such as extracting data from local/web sources, and data transformations, like missing value imputation. The training step then uses the prepared data as input to your training script to train your machine learning model.



Next steps

Follow these how-to articles:

- [Build a data ingestion pipeline with Azure Data Factory](#)
- [Automate and manage data ingestion pipelines with Azure Pipelines.](#)

Optimize data processing with Azure Machine Learning

12/23/2020 • 5 minutes to read • [Edit Online](#)

In this article, you learn about best practices to help you optimize data processing speeds locally and at scale.

Azure Machine Learning is integrated with open-source packages and frameworks for data processing. By using these integrations and applying the best practice recommendations in this article, you can improve your data processing speeds both locally and at scale.

Parquet and CSV file formats

Comma-separated values (csv) files are common file formats for data processing. However, parquet file formats are recommended for machine learning tasks.

[Parquet files](#) store data in a binary columnar format. This format is useful if splitting up the data into multiple files is needed. Also, this format allows you to target the relevant fields for your machine learning experiments. Instead of having to read in a 20-GB data file, you can decrease that data load, by selecting the necessary columns to train your ML model. Parquet files can also be compressed to minimize processing power and take up less space.

CSV files are commonly used to import and export data, since they're easy to edit and read in Excel. The data in CSVs are stored as strings in a row-based format, and the files can be compressed to lessen data transfer loads. Uncompressed CSVs can expand by a factor of about 2-10 and compressed CSVs can increase even further. So that 5-GB CSV in memory expands to well over the 8 GB of RAM you have on your machine. This compression behavior may increase data transfer latency, which isn't ideal if you have large amounts of data to process.

Pandas dataframe

[Pandas dataframes](#) are commonly used for data manipulation and analysis. [Pandas](#) works well for data sizes less than 1 GB, but processing times for [pandas](#) dataframes slow down when file sizes reach about 1 GB. This slowdown is because the size of your data in storage isn't the same as the size of data in a dataframe. For instance, data in CSV files can expand up to 10 times in a dataframe, so a 1-GB CSV file can become 10 GB in a dataframe.

[Pandas](#) is single threaded, meaning operations are done one at a time on a single CPU. You can easily parallelize workloads to multiple virtual CPUs on a single Azure Machine Learning compute instance with packages like [Modin](#) that wrap [Pandas](#) using a distributed backend.

To parallelize your tasks with [Modin](#) and [Dask](#), just change this line of code `import pandas as pd` to `import modin.pandas as pd`.

Dataframe: out of memory error

Typically an *out of memory* error occurs when your dataframe expands above the available RAM on your machine. This concept also applies to a distributed framework like [Modin](#) or [Dask](#). That is, your operation attempts to load the dataframe in memory on each node in your cluster, but not enough RAM is available to do so.

One solution is to increase your RAM to fit the dataframe in memory. We recommend your compute size and processing power contain two times the size of RAM. So if your dataframe is 10 GB, use a compute target with at least 20 GB of RAM to ensure that the dataframe can comfortably fit in memory and be processed.

For multiple virtual CPUs, vCPU, keep in mind that you want one partition to comfortably fit into the RAM each

vCPU can have on the machine. That is, if you have 16-GB RAM 4 vCPUs, you want about 2-GB dataframes per each vCPU.

Local vs remote

You may notice certain pandas dataframe commands perform faster when working on your local PC versus a remote VM you provisioned with Azure Machine Learning. Your local PC typically has a page file enabled, which allows you to load more than what fits in physical memory, that is your hard drive is being used as an extension of your RAM. Currently, Azure Machine Learning VMs run without a page file, therefore can only load as much data as physical RAM available.

For compute-heavy jobs, we recommend you pick a larger VM to improve processing speeds.

Learn more about the [available VM series and sizes](#) for Azure Machine Learning.

For RAM specifications, see the corresponding VM series pages such as, [Dv2-Dsv2 series](#) or [NC series](#).

Minimize CPU workloads

If you can't add more RAM to your machine, you can apply the following techniques to help minimize CPU workloads and optimize processing times. These recommendations pertain to both single and distributed systems.

TECHNIQUE	DESCRIPTION
Compression	<p>Use a different representation for your data, in a way that uses less memory and doesn't significantly impact the results of your calculation.</p> <p><i>Example:</i> Instead of storing entries as a string with about 10 bytes or more per entry, store them as a boolean, True or False, which you could store in 1 byte.</p>
Chunking	<p>Load data into memory in subsets (chunks), processing the data one subset at time, or multiple subsets in parallel. This method works best if you need to process all the data, but don't need to load all the data into memory at once.</p> <p><i>Example:</i> Instead of processing a full year's worth of data at once, load and process the data one month at a time.</p>
Indexing	<p>Apply and use an index, a summary that tells you where to find the data you care about. Indexing is useful when you only need to use a subset of the data, instead of the full set</p> <p><i>Example:</i> If you have a full year's worth of sales data sorted by month, an index helps you quickly search for the desired month that you wish to process.</p>

Scale data processing

If the previous recommendations aren't enough, and you can't get a virtual machine that fits your data, you can,

- Use a framework like [Spark](#) or [Dask](#) to process the data 'out of memory'. In this option, the dataframe is loaded into RAM partition by partition and processed, with the final result being gathered at the end.
- Scale out to a cluster using a distributed framework. In this option, data processing loads are split up and processed on multiple CPUs that work in parallel, with the final result gathered at the end.

Recommended distributed frameworks

The following table recommends distributed frameworks that are integrated with Azure Machine Learning based

on your code preference or data size.

EXPERIENCE OR DATA SIZE	RECOMMENDATION
If you're familiar with <code>Pandas</code>	<code>Modin</code> or <code>Dask</code> dataframe
If you prefer <code>Spark</code>	<code>PySpark</code>
For data less than 1 GB	<code>Pandas</code> locally or a remote Azure Machine Learning compute instance
For data larger than 10 GB	Move to a cluster using <code>Ray</code> , <code>Dask</code> , or <code>Spark</code>

You can create `Dask` clusters on Azure ML compute cluster with the `dask-cloudprovider` package. Or you can run `Dask` locally on a compute instance.

Next steps

- [Data ingestion options with Azure Machine Learning.](#)
- [Create and register datasets.](#)

Train models with Azure Machine Learning

12/23/2020 • 7 minutes to read • [Edit Online](#)

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
Run configuration	A typical way to train models is to use a training script and run configuration. The run configuration provides the information needed to configure the training environment used to train your model. You can specify your training script, compute target, and Azure ML environment in your run configuration and run a training job.
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 run 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.

- [Azure Machine Learning SDK for R \(preview\)](#): The SDK for R uses the reticulate package to bind to Azure Machine Learning's Python SDK. This allows you access to core objects and methods implemented in the Python SDK from any R environment.
- **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.

- **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 CLI to start a training run 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.

- [What is the Azure Machine Learning SDK for Python](#)
- [Install/update the SDK](#)
- [Configure a development environment for Azure Machine Learning](#)

Run configuration

A generic training job with Azure Machine Learning can be defined using the [ScriptRunConfig](#). The script run configuration is then used, along with your training script(s) to train a model on a compute target.

You may start with a run configuration 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 run configuration you use. A run also logs information about the training job, such as the inputs, outputs, and logs.

- [What is a run configuration?](#)
- [Tutorial: Train your first ML model](#)
- [Examples: Jupyter Notebook and Python examples of training models](#)
- [How to: Configure a training run](#)

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](#)
- [Tutorial: Use automated machine learning to predict taxi fares](#)
- [Examples: Jupyter Notebook examples for automated machine learning](#)
- [How to: Configure automated ML experiments in Python](#)
- [How to: Autotrain a time-series forecast model](#)
- [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. In a pipeline, you can train a model using automated machine learning or run configurations.

- [What are ML pipelines in Azure Machine Learning?](#)
- [Create and run machine learning pipelines with Azure Machine Learning SDK](#)
- [Tutorial: Use Azure Machine Learning Pipelines for batch scoring](#)
- [Examples: Jupyter Notebook examples for machine learning pipelines](#)
- [Examples: Pipeline with automated machine learning](#)

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 [Create & use software environments in Azure Machine Learning](#))
 - 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

If you choose to train on your local machine ("configure as local run"), you do not need to use Docker. You may use Docker locally if you choose (see the section [Configure ML pipeline](#) for an example).

R SDK (preview)

The R SDK enables you to use the R language with Azure Machine Learning. The SDK uses the reticulate package to bind to Azure Machine Learning's Python SDK. This gives you access to core objects and methods implemented in the Python SDK from any R environment.

For more information, see the following articles:

- [Tutorial: Create a logistic regression model](#)
- [Azure Machine Learning SDK for R reference](#)

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](#)

Many models solution accelerator

The [Many Models Solution Accelerator](#) (preview) builds on Azure Machine Learning and enables you to train, operate, and manage hundreds or even thousands of machine learning models.

For example, building a model for **each instance or individual** in the following scenarios can lead to improved results:

- Predicting sales for each individual store
- Predictive maintenance for hundreds of oil wells
- Tailoring an experience for individual users.

For more information, see the [Many Models Solution Accelerator](#) on GitHub.

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](#)

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 [Configure a training run](#).

Distributed training with Azure Machine Learning

12/23/2020 • 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

- Learn how to [use compute targets for model training](#) with the Python SDK.
- For a technical example, see the [reference architecture scenario](#).
- [Train ML models with TensorFlow](#).
- [Train ML models with PyTorch](#).

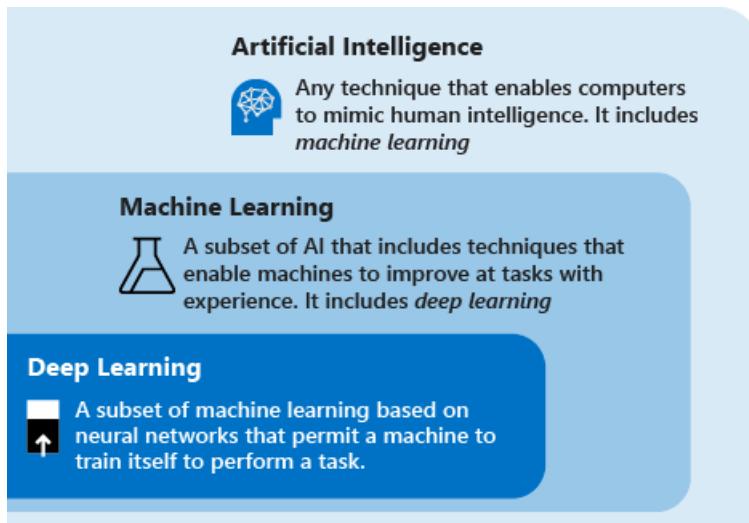
Deep learning vs. machine learning in Azure Machine Learning

12/23/2020 • 7 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.

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 contain 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 : [Classify images by using a Pytorch model](#).

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

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

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.

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](#)
- [Classify handwritten digits by using a Chainer model](#)

MLOps: Model management, deployment, and monitoring with Azure Machine Learning

12/23/2020 • 10 minutes to read • [Edit Online](#)

In this article, learn about how to use Azure Machine Learning to manage the lifecycle of your models. Azure Machine Learning uses a Machine Learning Operations (MLOps) approach. MLOps improves the quality and consistency of your machine learning solutions.

What is MLOps?

Machine Learning Operations (MLOps) is based on [DevOps](#) principles and practices that increase the efficiency of workflows. For example, 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

Azure Machine Learning provides the following MLOps capabilities:

- **Create reproducible ML pipelines.** Machine Learning pipelines allow you to define repeatable and reusable steps for your data preparation, training, and scoring processes.
- **Create reusable software 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 ML lifecycle.** The logged 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 ML lifecycle.** For example, experiment completion, model registration, model deployment, and data drift detection.
- **Monitor ML applications for operational and ML-related issues.** Compare model inputs between training and inference, explore model-specific metrics, and provide monitoring and alerts on your ML infrastructure.
- **Automate the end-to-end ML lifecycle with Azure Machine Learning and Azure Pipelines.** Using pipelines allows you to frequently update models, test new models, and continuously roll out new ML models alongside your other applications and services.

Create reproducible ML pipelines

Use ML pipelines from Azure Machine Learning to stitch together all of the steps involved in your model training process.

An ML pipeline can contain steps from data preparation to feature extraction to hyperparameter tuning to model evaluation. For more information, see [ML pipelines](#).

If you use the [Designer](#) to create your ML pipelines, you may at any time click the "..." at the top-right of the Designer page and then select **Clone**. Cloning your pipeline allows you to iterate your pipeline design without losing your old versions.

Create reusable software environments

Azure Machine Learning environments allow you to track and reproduce your projects' software dependencies as they evolve. Environments allow you to ensure that builds are reproducible without manual software configurations.

Environments describe the pip and Conda dependencies for your projects, and can be used for both training and deployment of models. For more information, see [What are Azure Machine Learning environments](#).

Register, package, and deploy models from anywhere

Register and track ML models

Model registration allows you to 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 is stored in multiple files, you can register them as a single model in your Azure 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. Additional metadata tags can be provided during registration. These tags are then used when searching for a model. Azure Machine Learning supports any model that can be loaded using Python 3.5.2 or higher.

TIP

You can also register models trained outside Azure Machine Learning.

You can't delete a registered model that is being used in an active deployment. For more information, see the register model section of [Deploy models](#).

IMPORTANT

When using `Filter by Tags` option on the Models page of Azure Machine Learning Studio, instead of using `TagName : TagValue` customers should use `TagName=TagValue` (without space)

Profile models

Azure Machine Learning can help you understand the CPU and memory requirements of the service that will be created when you deploy your model. Profiling tests the service that runs your model and returns information such as the CPU usage, memory usage, and response latency. It also provides a CPU and memory recommendation based on the resource usage. For more information, see the profiling section of [Deploy models](#).

Package and debug models

Before deploying a model into production, it is 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 [Deploy models](#) and [Troubleshooting deployments](#).

Convert and optimize models

Converting your model to [Open Neural Network Exchange](#) (ONNX) may improve performance. On average,

converting to ONNX can yield a 2x performance increase.

For more information on ONNX with Azure Machine Learning, see the [Create and accelerate ML models](#) article.

Use models

Trained machine learning models are deployed as web services in the cloud or locally. You can also deploy models to Azure IoT Edge devices. Deployments use CPU, GPU, or field-programmable gate arrays (FPGA) for inferencing. You can also use models from Power BI.

When using a model as a web service or IoT Edge device, you provide the following items:

- The model(s) that are used to score data submitted to the service/device.
- An entry script. This script accepts requests, uses the model(s) to score the data, and return a response.
- An Azure Machine Learning environment that describes the pip and Conda dependencies required by the model(s) and entry script.
- Any additional assets such as text, data, etc. that are required by the model(s) 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 deploying to Azure Kubernetes Service.

When the image is created, components required by Azure Machine Learning are also added. For example, assets needed to run the web service and interact with IoT Edge.

Batch scoring

Batch scoring is supported through ML pipelines. For more information, see [Batch predictions on big data](#).

Real-time web services

You can use your models in **web services** with the following compute targets:

- Azure Container Instance
- Azure Kubernetes Service
- Local development environment

To deploy the model as a web service, you must provide the following items:

- The model or ensemble of models.
- Dependencies required to use the model. For example, a script that accepts requests and invokes the model, conda dependencies, etc.
- Deployment configuration that describes how and where to deploy the model.

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

Controlled rollout

When deploying to Azure Kubernetes Service, 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 versions of the endpoint.
- Switch between endpoint versions by updating the traffic percentage in endpoint configuration.

For more information, see [Controlled rollout of ML models](#).

IoT Edge devices

You can use models with IoT devices through **Azure IoT Edge modules**. IoT Edge modules are deployed to a hardware device, which enables inference, or model scoring, on the device.

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

Analytics

Microsoft Power BI supports using machine learning models for data analytics. For more information, see [Azure](#)

Capture the governance data required for capturing the end-to-end ML lifecycle

Azure ML gives you the capability to track the end-to-end audit trail of all of your ML assets by using metadata.

- Azure ML [integrates with Git](#) to track information on which repository / branch / commit your code came from.
- [Azure ML 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 given input.
- Azure ML Run history stores a snapshot of the code, data, and computes used to train a model.
- The Azure ML Model Registry captures all of the metadata associated with your model (which experiment trained it, where it is being deployed, if its deployments are healthy).
- [Integration with Azure](#) allows you to act on events in the ML lifecycle. For example, model registration, deployment, data drift, and training (run) events.

TIP

While some information on models and datasets is automatically captured, you can add additional information by using [tags](#). When looking 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 referencing a dataset when registering a model, see the [Model](#) class reference.

Notify, automate, and alert on events in the ML lifecycle

Azure ML publishes key events to Azure EventGrid, which can be used to notify and automate on events in the ML lifecycle. For more information, please see [this document](#).

Monitor for operational & ML 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 may also be useful in training future versions of the model.

For more information, see [How to 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 in the face of such things as changes to a particular sensor, natural data changes such as seasonal effects, or features shifting in their relation to other features.

There is no universal answer to "How do I know if I should retrain?" but Azure ML event and monitoring tools previously discussed are good starting points for automation. Once you have decided to retrain, you should:

- Preprocess your data using a repeatable, automated process
- Train your new model
- Compare the outputs of your new model to those of your old model

- Use predefined criteria to choose whether to replace your old model

A theme of the above steps is that your retraining should be automated, not ad hoc. [Azure Machine Learning pipelines](#) are a good answer for creating workflows relating to data preparation, training, validation, and deployment. Read [Retrain models with Azure Machine Learning designer](#) to see how pipelines and the Azure Machine Learning designer fit into a retraining scenario.

Automate the ML 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, the Azure Pipeline will start a training run. The results of the run 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 [Azure Machine Learning extension](#) makes it easier to work with Azure Pipelines. It provides the following enhancements to Azure Pipelines:

- Enables workspace selection when defining 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 Azure Machine Learning, see the following links:

- [Continuous integration and deployment of ML models with Azure Pipelines](#)
- [Azure Machine Learning MLOps repository](#).
- [Azure 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:

- [How & where to deploy models with Azure Machine Learning](#)
- [Tutorial: Deploy an image classification model in ACI.](#)
- [End-to-end MLOps examples repo](#)
- [CI/CD of ML models with Azure Pipelines](#)
- Create clients that [consume a deployed model](#)
- [Machine learning at scale](#)
- [Azure AI reference architectures & best practices rep](#)

ONNX and Azure Machine Learning: Create and accelerate ML models

12/23/2020 • 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.

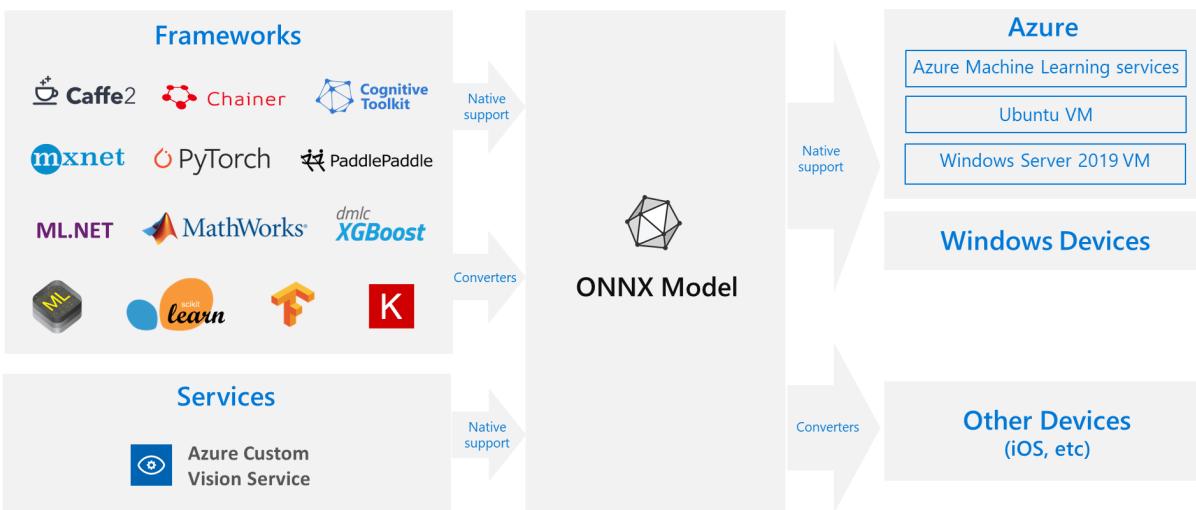
Microsoft and a community of partners created ONNX as an open standard for representing machine learning models. Models from [many frameworks](#) including TensorFlow, PyTorch, SciKit-Learn, Keras, Chainer, MXNet, MATLAB, and SparkML can be exported or converted to the standard ONNX format. Once the 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](#)

Open-source integration with Azure Machine Learning projects

12/23/2020 • 5 minutes to read • [Edit Online](#)

You can train, deploy, and manage the end-to-end machine learning process in Azure Machine Learning by using open-source Python machine learning libraries and platforms. Use development tools, like Jupyter Notebooks and Visual Studio Code, to leverage your existing models and scripts in Azure Machine Learning.

In this article, learn more about these open-source libraries and platforms.

Train open-source machine learning models

The machine learning training process involves the application of algorithms to your data in order to achieve a task or solve a problem. Depending on the problem, you may choose different algorithms that best fit the task and your data. For more information on the different branches of machine learning, see the [deep learning vs machine learning article](#) and the [machine learning algorithm cheat sheet](#).

Preserve data privacy using differential privacy

To train a machine learning model, you need data. Sometimes that data is sensitive, and it's important to make sure that the data is secure and private. Differential privacy is a technique of preserving the confidentiality of information in a dataset. To learn more, see the article on [preserving data privacy](#).

Open-source differential privacy toolkits like [SmartNoise](#) help you [preserve the privacy of data](#) in Azure Machine Learning solutions.

Classical machine learning: scikit-learn

For training tasks involving classical machine learning algorithms tasks such classification, clustering, and regression you might use something like Scikit-learn. To learn how to train a flower classification model, see the [how to train with Scikit-learn article](#).

Neural networks: PyTorch, TensorFlow, Keras

Open-source machine learning algorithms known as neural networks, a subset of machine learning, are useful for training deep learning models in Azure Machine Learning.

Open-source deep learning frameworks and how-to guides include:

- [PyTorch: Train a deep learning image classification model using transfer learning in PyTorch](#)
- [TensorFlow: Recognize handwritten digits using TensorFlow](#)
- [Keras: Build a neural network to analyze images using Keras](#)

Training a deep learning model from scratch often requires large amounts of time, data, and compute resources. You can shortcut the training process by using transfer learning. Transfer learning is a technique that applies knowledge gained from solving one problem to a different but related problem. This means you can take an existing model repurpose it. See the [deep learning article](#) to learn more about transfer learning.

Reinforcement learning: Ray RLLib

Reinforcement learning is an artificial intelligence technique that trains models using actions, states, and rewards: Reinforcement learning agents learn to take a set of predefined actions that maximize the specified rewards based on the current state of their environment.

The [Ray RLLib](#) project has a set features that allow for high scalability throughout the training process. The iterative

process is both time- and resource-intensive as reinforcement learning agents try to learn the optimal way of achieving a task. Ray RLLib also natively supports deep learning frameworks like TensorFlow and PyTorch.

To learn how to use Ray RLLib with Azure Machine Learning, see the [how to train a reinforcement learning model](#).

Monitor model performance: TensorBoard

Training a single or multiple models requires the visualization and inspection of desired metrics to make sure the model performs as expected. You can [use TensorBoard in Azure Machine Learning to track and visualize experiment metrics](#)

Frameworks for interpretable and fair models

Machine learning systems are used in different areas of society such as banking, education, and healthcare. As such, it's important for these systems to be accountable for the predictions and recommendations they make to prevent unintended consequences.

Open-source frameworks like [InterpretML](#) and Fairlearn (<https://github.com/fairlearn/fairlearn>) work with Azure Machine Learning to create more transparent and equitable machine learning models.

For more information on how to build fair and interpretable models, see the following articles:

- [Model interpretability in Azure Machine Learning](#)
- [Interpret and explain machine learning models](#)
- [Explain AutoML models](#)
- [Mitigate fairness in machine learning models](#)
- [Use Azure Machine Learning to assets model fairness](#)

Model deployment

Once models are trained and ready for production, you have to choose how to deploy it. Azure Machine Learning provides various deployment targets. For more information, see the [where and how to deploy article](#).

Standardize model formats with ONNX

After training, the contents of the model such as learned parameters are serialized and saved to a file. Each framework has its own serialization format. When working with different frameworks and tools, it means you have to deploy models according to the framework's requirements. To standardize this process, you can use the Open Neural Network Exchange (ONNX) format. ONNX is an open-source format for artificial intelligence 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.

For more information on ONNX and how to consume ONNX models, see the following articles:

- [Create and accelerate ML models with ONNX](#)
- [Use ONNX models in .NET applications](#)

Package and deploy models as containers

Container technologies such as Docker are one way to deploy models as web services. Containers provide a platform and resource agnostic way to build and orchestrate reproducible software environments. With these core technologies, you can use [preconfigured environments](#), [preconfigured container images](#) or custom ones to deploy your machine learning models to such as [Kubernetes clusters](#). For GPU intensive workflows, you can use tools like NVIDIA Triton Inference server to [make predictions using GPUs](#).

Secure deployments with homomorphic encryption

Securing deployments is an important part of the deployment process. To [deploy encrypted inferencing services](#), use the `encrypted-inference` open-source Python library. The `encrypted_inferencing` package provides bindings

based on [Microsoft SEAL](#), a homomorphic encryption library.

Machine Learning Operations (MLOps)

Machine Learning Operations (MLOps), commonly thought of as DevOps for machine learning allows you to build more transparent, resilient, and reproducible machine learning workflows. See the [what is MLOps article](#) to learn more about MLOps.

Using DevOps practices like continuous integration (CI) and continuous deployment (CD), you can automate the end-to-end machine learning lifecycle and capture governance data around it. You can define your [machine learning CI/CD pipeline in GitHub actions](#) to run Azure Machine Learning training and deployment tasks.

Capturing software dependencies, metrics, metadata, data and model versioning are an important part of the MLOps process in order to build transparent, reproducible, and auditable pipelines. For this task, you can [use MLFlow in Azure Machine Learning](#) as well as when [training machine learning models in Azure Databricks](#).

What are Azure Machine Learning pipelines?

12/23/2020 • 7 minutes to read • [Edit Online](#)

In this article, you learn how Azure Machine Learning pipelines help you build, optimize, and manage machine learning workflows. These workflows have a number of benefits:

- Simplicity
- Speed
- Repeatability
- Flexibility
- Versioning and tracking
- Modularity
- Quality assurance
- Cost control

These benefits become significant as soon as your machine learning project moves beyond pure exploration and into iteration. Even simple one-step pipelines can be valuable. Machine learning projects are often in a complex state, and it can be a relief to make the precise accomplishment of a single workflow a trivial process.

Which Azure pipeline technology should I use?

The Azure cloud provides several other pipelines, each with a different purpose. The following table lists the different pipelines and what they are 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 DevOps Pipelines	Jenkins	Code + Model -> App/Service	Most open and flexible activity support, approval queues, phases with gating

What can Azure ML pipelines do?

An Azure Machine Learning pipeline is an independently executable workflow of a complete machine learning task. Subtasks are encapsulated as a series of steps within the pipeline. An Azure Machine Learning pipeline can be as simple as one that calls a Python script, so *may* do just about anything. Pipelines *should* focus on machine learning tasks such as:

- Data preparation including importing, validating and cleaning, munging and transformation, normalization, and staging

- Training configuration including parameterizing arguments, filepaths, and logging / reporting configurations
- Training and validating efficiently and repeatedly. Efficiency might come from specifying specific data subsets, different hardware compute resources, distributed processing, and progress monitoring
- Deployment, including versioning, scaling, provisioning, and access control

Independent steps allow multiple data scientists to work on the same pipeline at the same time without over-taxing compute resources. Separate steps also make it easy to use different compute types/sizes for each step.

After the pipeline is designed, there is often more fine-tuning around the training loop of the pipeline. When you rerun a pipeline, the run jumps to the steps that need to be rerun, such as an updated training script. Steps that do not need to be rerun are skipped.

With pipelines, you may choose to use different hardware for different tasks. Azure coordinates the various [compute targets](#) you use, so your intermediate data seamlessly flows to downstream compute targets.

You can [track the metrics for your pipeline experiments](#) directly in Azure portal or your [workspace landing page \(preview\)](#). After a pipeline has been published, you can configure a REST endpoint, which allows you to rerun the pipeline from any platform or stack.

In short, all of the complex tasks of the machine learning lifecycle can be helped with pipelines. Other Azure pipeline technologies have their own strengths. [Azure Data Factory pipelines](#) excels at working with data and [Azure Pipelines](#) is the right tool for continuous integration and deployment. But if your focus is machine learning, Azure Machine Learning pipelines are likely to be the best choice for your workflow needs.

Analyzing dependencies

Many programming ecosystems have tools that orchestrate resource, library, or compilation dependencies. Generally, these tools use file timestamps to calculate dependencies. When a file is changed, only it and its dependents are updated (downloaded, recompiled, or packaged). Azure ML pipelines extend this concept. Like traditional build tools, pipelines calculate dependencies between steps and only perform the necessary recalculations.

The dependency analysis in Azure ML pipelines is more sophisticated than simple timestamps though. Every step may run in a different hardware and software environment. Data preparation might be a time-consuming process but not need to run on hardware with powerful GPUs, certain steps might require OS-specific software, you might want to use distributed training, and so forth.

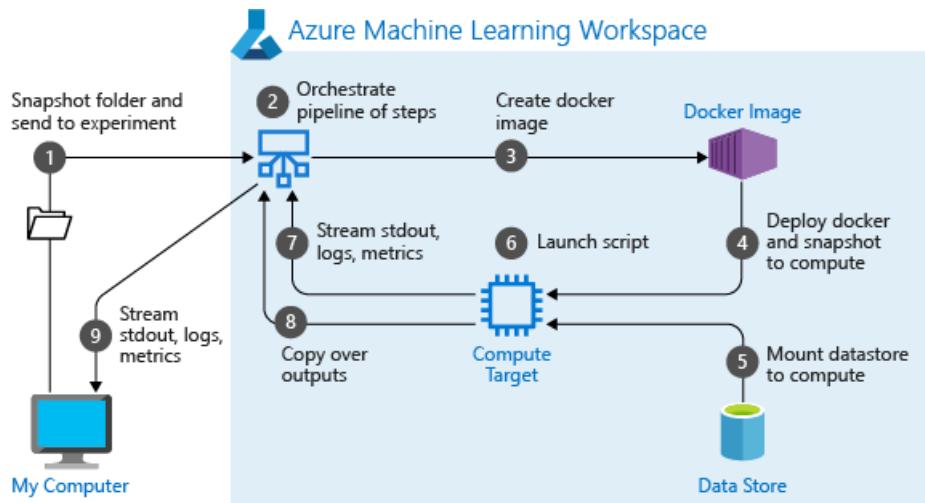
Azure Machine Learning automatically orchestrates all of the dependencies between pipeline steps. This orchestration might include spinning up and down Docker images, attaching and detaching compute resources, and moving data between the steps in a consistent and automatic manner.

Coordinating the steps involved

When you create and run a `Pipeline` object, the following high-level steps occur:

- For each step, the service calculates requirements for:
 - Hardware compute resources
 - OS resources (Docker image(s))
 - Software resources (Conda / virtualenv dependencies)
 - Data inputs
- The service determines the dependencies between steps, resulting in a dynamic execution graph
- When each node in the execution graph runs:
 - The service configures the necessary hardware and software environment (perhaps reusing existing resources)
 - The step runs, providing logging and monitoring information to its containing `Experiment` object
 - When the step completes, its outputs are prepared as inputs to the next step and/or written to storage

- Resources that are no longer needed are finalized and detached



Building pipelines with the Python SDK

In the [Azure Machine Learning Python SDK](#), a pipeline is a Python object defined in the `azureml.pipeline.core` module. A `Pipeline` object contains an ordered sequence of one or more `PipelineStep` objects. The `PipelineStep` class is abstract and the actual steps will be of subclasses such as `EstimatorStep`, `PythonScriptStep`, or `DataTransferStep`. The `ModuleStep` class holds a reusable sequence of steps that can be shared among pipelines. A `Pipeline` runs as part of an `Experiment`.

An Azure ML pipeline is associated with an Azure Machine Learning workspace and a pipeline step is associated with a compute target available within that workspace. For more information, see [Create and manage Azure Machine Learning workspaces in the Azure portal](#) or [What are compute targets in Azure Machine Learning?](#).

A simple Python Pipeline

This snippet shows the objects and calls needed to create and run a `Pipeline`:

```
ws = Workspace.from_config()
blob_store = Datastore(ws, "workspaceblobstore")
compute_target = ws.compute_targets["STANDARD_NC6"]
experiment = Experiment(ws, 'MyExperiment')

input_data = Dataset.File.from_files(
    DataPath(datastore, '20newsgroups/20news.pkl'))

output_data = PipelineData("output_data", datastore=blob_store)

input_named = input_data.as_named_input('input')

steps = [ PythonScriptStep(
    script_name="train.py",
    arguments=["--input", input_named.as_download(), "--output", output_data],
    inputs=[input_data],
    outputs=[output_data],
    compute_target=compute_target,
    source_directory="myfolder"
) ]

pipeline = Pipeline(workspace=ws, steps=steps)

pipeline_run = experiment.submit(pipeline)
pipeline_run.wait_for_completion()
```

The snippet starts with common Azure Machine Learning objects, a `Workspace`, a `Datastore`, a `ComputeTarget`,

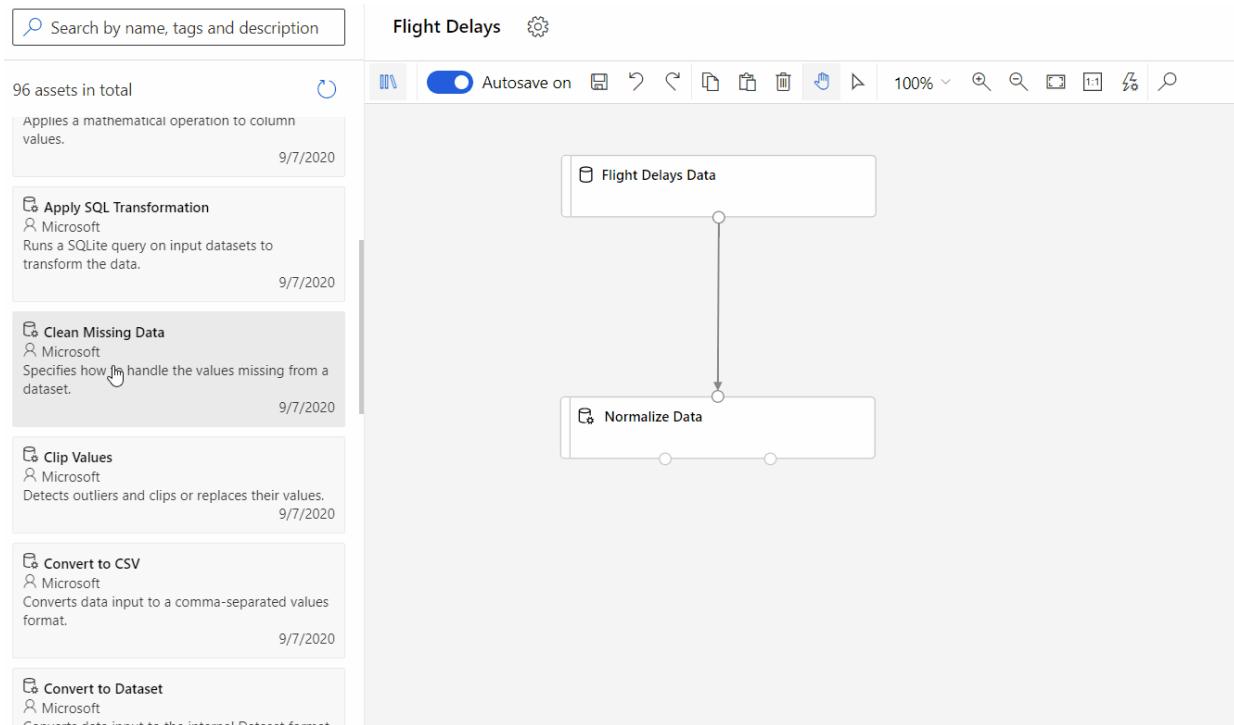
and an `Experiment`. Then, the code creates the objects to hold `input_data` and `output_data`. The array `steps` holds a single element, a `PythonScriptStep` that will use the data objects and run on the `compute_target`. Then, the code instantiates the `Pipeline` object itself, passing in the workspace and steps array. The call to `experiment.submit(pipeline)` begins the Azure ML pipeline run. The call to `wait_for_completion()` blocks until the pipeline is finished.

To learn more about connecting your pipeline to your data, see the articles [Data access in Azure Machine Learning](#) and [Moving data into and between ML pipeline steps \(Python\)](#).

Building pipelines with the designer

Developers who prefer a visual design surface can use the Azure Machine Learning designer to create pipelines. You can access this tool from the **Designer** selection on the homepage of your workspace. The designer allows you to drag and drop steps onto the design surface.

When you visually design pipelines, the inputs and outputs of a step are displayed visibly. You can drag and drop data connections, allowing you to quickly understand and modify the dataflow of your pipeline.



Key advantages

The key advantages of using pipelines for your machine learning workflows are:

KEY ADVANTAGE	DESCRIPTION
Unattended runs	Schedule steps to run in parallel or in sequence in a reliable and unattended manner. Data preparation and modeling can last days or weeks, and pipelines allow you to focus on other tasks while the process is running.
Heterogenous compute	Use multiple pipelines that are reliably coordinated across heterogeneous and scalable compute resources and storage locations. Make efficient use of available compute resources by running individual pipeline steps on different compute targets, such as HDInsight, GPU Data Science VMs, and Databricks.

KEY ADVANTAGE	DESCRIPTION
Reusability	Create pipeline templates for specific scenarios, such as retraining and batch-scoring. Trigger published pipelines from external systems via simple REST calls.
Tracking and versioning	Instead of manually tracking data and result paths as you iterate, use the pipelines SDK to explicitly name and version your data sources, inputs, and outputs. You can also manage scripts and data separately for increased productivity.
Modularity	Separating areas of concerns and isolating changes allows software to evolve at a faster rate with higher quality.
Collaboration	Pipelines allow data scientists to collaborate across all areas of the machine learning design process, while being able to concurrently work on pipeline steps.

Next steps

Azure ML pipelines are a powerful facility that begins delivering value in the early development stages. The value increases as the team and project grows. This article has explained how pipelines are specified with the Azure Machine Learning Python SDK and orchestrated on Azure. You've seen some simple source code and been introduced to a few of the `PipelineStep` classes that are available. You should have a sense of when to use Azure ML pipelines and how Azure runs them.

- Learn how to [create your first pipeline](#).
- Learn how to [run batch predictions on large data](#).
- See the SDK reference docs for [pipeline core](#) and [pipeline steps](#).
- Try out example Jupyter notebooks showcasing [Azure Machine Learning pipelines](#). Learn how to [run notebooks to explore this service](#).

Enterprise security and governance for Azure Machine Learning

12/23/2020 • 5 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:

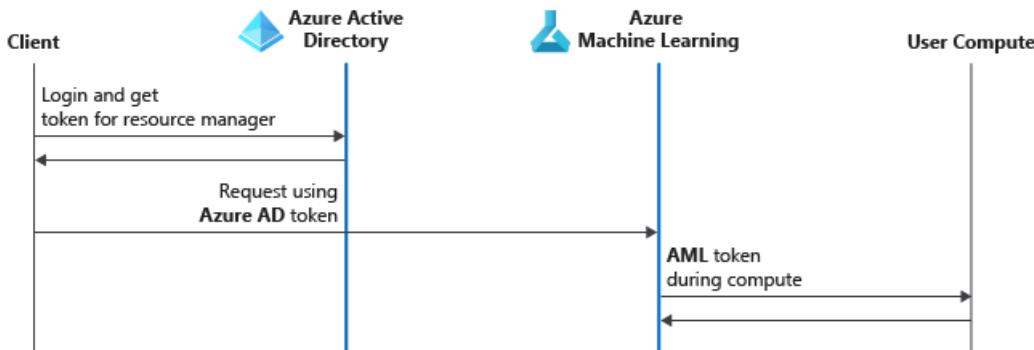
- Restrict access to resources and operations by user account or groups
- Restrict incoming and outgoing network communications
- Encrypt data in transit and at rest
- Scan for vulnerabilities
- Apply and audit configuration policies

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 run 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 attached 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
Resource group that contains the key vault (if different from the one that contains the workspace)	Contributor

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](#).

Azure Machine Learning also creates an additional application (the name starts with `aml-` or `Microsoft-AzureML-Support-App-`) with contributor-level access in your subscription for every workspace region. For example, if you have one workspace in East US and one in North Europe in the same subscription, you'll see two of these applications. These applications enable Azure Machine Learning to help you manage compute resources.

You can also configure your own managed identities for use with Azure Virtual Machines and Azure Machine Learning compute cluster. With a VM, the managed identity can be used to access your workspace from the SDK, instead of the individual user's Azure AD account. 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 [Authentication for Azure Machine Learning workspace](#).

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 web services (inference 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 [Configure authentication for models deployed as a web service](#).

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](#)
- [Use Azure AD managed identity with your web service](#)

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](#)
- [Secure inference environment](#)
- [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](#).

Vulnerability scanning

[Azure Security Center](#) 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 Security Center](#) and [Azure Kubernetes Services integration with Security Center](#).

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

- [Secure Azure Machine Learning web services with TLS](#)
- [Consume a Machine Learning model deployed as a web service](#)
- [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](#)

Azure security baseline for Azure Machine Learning

12/23/2020 • 37 minutes to read • [Edit Online](#)

The Azure Security Baseline for Microsoft Azure Machine Learning contains recommendations that will help you improve the security posture of your deployment. The baseline for this service is drawn from the [Azure Security Benchmark version 1.0](#), which provides recommendations on how you can secure your cloud solutions on Azure with our best practices guidance. For more information, see [Azure Security Baselines overview](#).

Network security

For more information, see the [Azure Security Benchmark: Network security](#).

1.1: Protect Azure resources within virtual networks

Guidance: Azure Machine Learning relies on other Azure services for compute resources. Compute resources (compute targets) are used to train and deploy models. You can create these compute targets in a virtual network. For example, you can use Azure Virtual Machine Learning compute instance to train a model and then deploy the model to Azure Kubernetes Service (AKS). You can secure your machine learning lifecycles by isolating Azure Machine Learning training and inference jobs within an Azure virtual network.

Azure Firewall can be used to control access to your Azure Machine Learning workspace and the public internet.

- [Virtual network isolation and privacy overview](#)
- [Use workspace behind Azure Firewall for Azure Machine Learning](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

1.2: Monitor and log the configuration and traffic of virtual networks, subnets, and NICs

Guidance: Azure Machine Learning relies on other Azure services for compute resources. Assign network security groups to the networks that are created as your Machine Learning deployment.

Enable network security group flow logs and send the logs to an Azure Storage account for auditing. You can also send the flow logs to a Log Analytics workspace and then use Traffic Analytics to provide insights into traffic patterns in your Azure cloud. Some advantages of Traffic Analytics are the ability to visualize network activity, identify hot spots and security threats, understand traffic flow patterns, and pinpoint network misconfigurations.

- [How to enable network security group flow logs](#)
- [How to enable and use Traffic Analytics](#)
- [Understand network security provided by Azure Security Center](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

1.3: Protect critical web applications

Guidance: You can enable HTTPS to secure communication with web services deployed by Azure Machine Learning. Web services are deployed on Azure Kubernetes Services (AKS) or Azure Container Instances (ACI) and secure the data submitted by clients. You can also use private IP with AKS to restrict scoring, so that only clients behind a virtual network can access the web service.

- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Virtual network isolation and privacy overview](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

1.4: Deny communications with known malicious IP addresses

Guidance: Enable DDoS Protection Standard on the virtual networks associated with your Machine Learning instance to guard against distributed denial-of-service (DDoS) attacks. Use Azure Security Center Integrated threat detection to detect communications with known malicious or unused Internet IP addresses.

Deploy Azure Firewall at each of the organization's network boundaries with threat intelligence-based filtering enabled and configured to "Alert and deny" for malicious network traffic.

- [How to configure DDoS protection](#)
- [Use workspace behind Azure Firewall for Azure Machine Learning](#)
- [For more information about the Azure Security Center threat detection](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

1.5: Record network packets

Guidance: For any VMs with the proper extension installed in your Azure Machine Learning services, you can enable Network Watcher packet capture to investigate anomalous activities.

- [How to create a Network Watcher instance](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

1.6: Deploy network-based intrusion detection/intrusion prevention systems (IDS/IPS)

Guidance: Deploy the firewall solution of your choice at each of your organization's network boundaries to detect and/or block malicious traffic.

Select an offer from Azure Marketplace that supports IDS/IPS functionality with payload inspection capabilities. When payload inspection is not a requirement, Azure Firewall threat intelligence can be used. Azure Firewall threat intelligence-based filtering is used to alert on and/or block traffic to and from known malicious IP addresses and domains. The IP addresses and domains are sourced from the Microsoft Threat Intelligence feed.

- [How to deploy Azure Firewall](#)
- [How to configure alerts with Azure Firewall](#)
- [Azure Marketplace](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

1.7: Manage traffic to web applications

Guidance: Not applicable; this recommendation is intended for web applications running on Azure App Service or compute resources.

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

1.8: Minimize complexity and administrative overhead of network security rules

Guidance: For resources that need access to your Azure Machine Learning account, use Virtual Network service tags to define network access controls on network security groups or Azure Firewall. You can use service tags in place of specific IP addresses when creating security rules. By specifying the service tag name (for example, AzureMachineLearning) in the appropriate source or destination field of a rule, you can allow or deny the traffic for the corresponding service. Microsoft manages the address prefixes encompassed by the service tag and automatically updates the service tag as addresses change.

Azure Machine Learning service documents a list of service tags for its compute targets within a virtual network that helps to minimize complexity, you can use it as guidelines in your network management.

- [For more information about using service tags](#)
- [Virtual network isolation and privacy overview](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

1.9: Maintain standard security configurations for network devices

Guidance: Define and implement standard security configurations for network resources associated with your Azure Machine Learning namespaces with Azure Policy. Use Azure Policy aliases in the "Microsoft.MachineLearning" and "Microsoft.Network" namespaces to create custom policies to audit or enforce the network configuration of your Machine Learning namespaces.

- [How to configure and manage Azure Policy](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

1.10: Document traffic configuration rules

Guidance: Use tags for network resources associated with your Azure Machine Learning deployment in order to logically organize them according to a taxonomy.

For a resource in your Azure Machine Learning virtual network that support the Description field, use it to document the rules that allow traffic to/from a network.

- [How to create and use tags](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

1.11: Use automated tools to monitor network resource configurations and detect changes

Guidance: Use Azure Activity Log to monitor network resource configurations and detect changes for network resources related to Azure Machine Learning. Create alerts within Azure Monitor that will trigger when changes to critical network resources take place.

- [How to view and retrieve Azure Activity Log events](#)
- [How to create alerts in Azure Monitor](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

Logging and monitoring

For more information, see the [Azure Security Benchmark: Logging and monitoring](#).

2.1: Use approved time synchronization sources

Guidance: Microsoft maintains the time source used for Azure resources such as Azure Machine Learning for timestamps in the logs.

Azure Security Center monitoring: Not Applicable

Responsibility: Microsoft

2.2: Configure central security log management

Guidance: Ingest logs via Azure Monitor to aggregate security data generated by Azure Machine Learning. In Azure Monitor, use Log Analytics workspaces to query and perform analytics, and use Azure Storage accounts for long term and archival storage. Alternatively, you may enable, and on-board data to Azure Sentinel or a third-party Security Incident and Event Management (SIEM).

- [How to configure diagnostic logs for Azure Machine Learning](#)
- [How to onboard Azure Sentinel](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

2.3: Enable audit logging for Azure resources

Guidance: Enable diagnostic settings on Azure resources for access to audit, security, and diagnostic logs. Activity logs, which are automatically available, include event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

You can also correlate Machine Learning service operation logs for security and compliance purposes.

- [How to collect platform logs and metrics with Azure Monitor](#)
- [Understand logging and different log types in Azure](#)
- [Enable logging in Azure Machine Learning](#)
- [Monitoring Azure Machine Learning](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

2.4: Collect security logs from operating systems

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for collecting and monitoring it.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For any compute resources that are owned by your organization, use Azure Security Center to monitor the operating system.

- [How to collect Azure Virtual Machine internal host logs with Azure Monitor](#)
- [Understand Azure Security Center data collection](#)

Azure Security Center monitoring: Yes

Responsibility: Shared

2.5: Configure security log storage retention

Guidance: In Azure Monitor, set the log retention period for Log Analytics workspaces associated with your Azure Machine Learning instances according to your organization's compliance regulations.

- [How to set log retention parameters](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

2.6: Monitor and review Logs

Guidance: Analyze and monitor logs for anomalous behavior and regularly review the results from your Azure Machine Learning. Use Azure Monitor and a Log Analytics workspace to review logs and perform queries on log data.

Alternatively, you can enable and on-board data to Azure Sentinel or a third-party SIEM.

- [How to perform queries for Azure Machine Learning in Log Analytics Workspaces](#)
- [Enable logging in Azure Machine Learning](#)
- [How to onboard Azure Sentinel](#)
- [Getting started with Log Analytics queries](#)
- [How to perform custom queries in Azure Monitor](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

2.7: Enable alerts for anomalous activities

Guidance: In Azure Monitor, configure logs related to Azure Machine Learning within the Activity Log, and Machine Learning diagnostic settings to send logs into a Log Analytics workspace to be queried or into a storage account for long-term archival storage. Use Log Analytics workspace to create alerts for anomalous activity found in security logs and events.

Alternatively, you may enable and on-board data to Azure Sentinel.

- [For more information on Azure Machine Learning alerts](#)
- [How to alert on Log Analytics workspace log data](#)
- [How to onboard Azure Sentinel](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

2.8: Centralize anti-malware logging

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for Antimalware deployment of Azure Machine Learning service.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, enable antimalware event collection for Microsoft Antimalware for Azure Cloud Services and Virtual Machines.

- [How to configure Microsoft Antimalware for a virtual machine](#)
- [How to configure the Microsoft Antimalware extension for cloud services](#)

- [Understand Microsoft Antimalware](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

2.9: Enable DNS query logging

Guidance: Not applicable; Azure Machine Learning does not process or produce DNS-related logs.

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

2.10: Enable command-line audit logging

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources are owned by your organization, use Azure Security Center to enable security event log monitoring for Azure virtual machines. Azure Security Center provisions the Log Analytics agent on all supported Azure VMs, and any new ones that are created if automatic provisioning is enabled. Or you can install the agent manually. The agent enables the process creation event 4688 and the commandline field inside event 4688. New processes created on the VM are recorded by event log and monitored by Security Center's detection services.

Azure Security Center monitoring: Yes

Responsibility: Customer

Identity and access control

For more information, see the [Azure Security Benchmark: Identity and access control](#).

3.1: Maintain an inventory of administrative accounts

Guidance: You can use the Identity and Access Management tab for a resource in the Azure portal to configure Azure role-based access control (Azure RBAC) and maintain inventory on Azure Machine Learning resources. The roles are applied to users, groups, service principals, and managed identities in Active Directory. You can use built-in roles or custom roles for individuals and groups.

Azure Machine Learning provides built-in roles for common management scenarios in Azure Machine Learning. An individual who has a profile in Azure Active Directory (Azure AD) can assign these roles to users, groups, service principals, or managed identities to grant or deny access to resources and operations on Azure Machine Learning resources.

You can also use the Azure AD PowerShell module to perform adhoc queries to discover accounts that are members of administrative groups.

- [Understand Azure role-based access control in Azure Machine Learning](#)
- [How to get a directory role in Azure Active Directory with PowerShell](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

3.2: Change default passwords where applicable

Guidance: Access management to Machine Learning resources is controlled through Azure Active Directory (Azure AD). Azure AD does not have the concept of default passwords.

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

3.3: Use dedicated administrative accounts

Guidance: Azure Machine Learning comes with three default roles when a new workspace is created, creating standard operating procedures around the use of owner accounts.

You can also enable a just-in-time access to administrative accounts by using Azure AD Privileged Identity Management and Azure Resource Manager.

- [To learn more Machine Learning default roles](#)
- [Learn more about Privileged Identity Management](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

3.4: Use single sign-on (SSO) with Azure Active Directory

Guidance: Machine Learning is integrated with Azure Active Directory, use Azure Active Directory SSO instead of configuring individual stand-alone credentials per-service. Use Azure Security Center identity and access recommendations.

- [Understand SSO with Azure AD](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

3.5: Use multi-factor authentication for all Azure Active Directory based access

Guidance: Enable Azure Active Directory Multi-Factor Authentication and follow Azure Security Center identity and access recommendations.

- [How to enable MFA in Azure](#)
- [How to monitor identity and access within Azure Security Center](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

3.6: Use dedicated machines (Privileged Access Workstations) for all administrative tasks

Guidance: Use a secure, Azure-managed workstation (also known as a Privileged Access Workstation, or PAW) for administrative tasks that require elevated privileges.

- [Understand secure, Azure-managed workstations](#)
- [How to enable Azure AD MFA](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

3.7: Log and alert on suspicious activities from administrative accounts

Guidance: Use Azure Active Directory security reports and monitoring to detect when suspicious or unsafe activity occurs in the environment. Use Azure Security Center to monitor identity and access activity.

- [How to identify Azure AD users flagged for risky activity](#)
- [How to monitor users' identity and access activity in Azure Security Center](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

3.8: Manage Azure resources only from approved locations

Guidance: Use Azure AD named locations to allow access only from specific logical groupings of IP address ranges or countries/regions.

- [How to configure Azure AD named locations](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

3.9: Use Azure Active Directory

Guidance: Use Azure Active Directory (Azure AD) as the central authentication and authorization system. Azure AD protects data by using strong encryption for data at rest and in transit. Azure AD also salts, hashes, and securely stores user credentials.

Role access can be scoped to multiple levels in Azure. For Machine Learning, roles can be managed at workspace level, for example, you have owner access to a workspace may not have owner access to the resource group that contains the workspace. This provides more granular access controls to separate roles within the same resource group.

- [Manage access to an Azure Machine Learning workspace](#)
- [How to create and configure an Azure AD instance](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

3.10: Regularly review and reconcile user access

Guidance: Azure AD provides logs to help discover stale accounts. In addition, use Azure AD identity and access reviews to efficiently manage group memberships, access to enterprise applications, and role assignments. User access can be reviewed on a regular basis to make sure only the right users have continued access.

Use Azure Active Directory (Azure AD) Privileged Identity Management (PIM) for generation of logs and alerts when suspicious or unsafe activity occurs in the environment.

- [Understand Azure AD reporting](#)
- [How to use Azure AD identity and access reviews](#)
- [Deploy Azure AD Privileged Identity Management \(PIM\)](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

3.11: Monitor attempts to access deactivated credentials

Guidance: You have access to Azure AD sign-in activity, audit, and risk event log sources, which allow you to integrate with any SIEM/monitoring tool.

You can streamline this process by creating diagnostic settings for Azure AD user accounts and sending the audit logs and sign-in logs to a Log Analytics workspace. You can configure desired alerts within Log Analytics workspace.

- [How to integrate Azure activity logs with Azure Monitor](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

3.12: Alert on account login behavior deviation

Guidance: Use Azure AD Identity Protection features to configure automated responses to detected suspicious actions related to user identities. You can also ingest data into Azure Sentinel for further investigation.

- [How to view Azure AD risky sign-ins](#)
- [How to configure and enable Identity Protection risk policies](#)
- [How to onboard Azure Sentinel](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

3.13: Provide Microsoft with access to relevant customer data during support scenarios

Guidance: Not applicable; Azure Machine Learning service doesn't support customer lockbox.

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

Data protection

For more information, see the [Azure Security Benchmark: Data protection](#).

4.1: Maintain an inventory of sensitive Information

Guidance: Use tags to assist in tracking Azure resources that store or process sensitive information.

- [How to create and use tags](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.2: Isolate systems storing or processing sensitive information

Guidance: Implement isolation using separate subscriptions and management groups for individual security domains such as environment type and data sensitivity level. You can restrict the level of access to your Azure resources that your applications and enterprise environments demand. You can control access to Azure resources via Azure RBAC.

- [How to create additional Azure subscriptions](#)
- [How to create management groups](#)
- [How to create and use tags](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.3: Monitor and block unauthorized transfer of sensitive information

Guidance: Use a third-party solution from Azure Marketplace in network perimeters to monitor for unauthorized transfer of sensitive information and block such transfers while alerting information security professionals.

For the underlying platform, which is managed by Microsoft, Microsoft treats all customer content as sensitive and guards against customer data loss and exposure. To ensure customer data within Azure remains secure, Microsoft has implemented and maintains a suite of robust data protection controls and capabilities.

- [Understand customer data protection in Azure](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.4: Encrypt all sensitive information in transit

Guidance: Web services deployed through Azure Machine Learning only support TLS version 1.2 that enforces data encryption in transit.

- [Use TLS to secure a web service through Azure Machine Learning](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.5: Use an active discovery tool to identify sensitive data

Guidance: Data identification, classification, and loss prevention features are not yet available for Azure Machine Learning. Implement a third-party solution if necessary for compliance purposes.

For the underlying platform, which is managed by Microsoft, Microsoft treats all customer content as sensitive and goes to great lengths to guard against customer data loss and exposure. To ensure customer data within Azure remains secure, Microsoft has implemented and maintains a suite of robust data protection controls and capabilities.

- [Understand customer data protection in Azure](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.6: Use Azure RBAC to manage access to resources

Guidance: Azure Machine Learning supports using Azure Active Directory (Azure AD) to authorize requests to Machine Learning resources. With Azure AD, you can use Azure role-based access control (Azure RBAC) to grant permissions to a security principal, which may be a user, or an application service principal.

- [Manage access to an Azure Machine Learning workspace](#)
- [Use Azure RBAC for Kubernetes authorization](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.7: Use host-based data loss prevention to enforce access control

Guidance: Not applicable; this guideline is intended for compute resources.

Microsoft manages the underlying infrastructure for Machine Learning and has implemented strict controls to prevent the loss or exposure of customer data.

- [Understand customer data protection in Azure](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Microsoft

4.8: Encrypt sensitive information at rest

Guidance: Azure Machine Learning stores snapshots, output, and logs in the Azure Blob 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. You can also encrypt data stored in Azure Blob storage with your own keys in Machine Learning service.

- [Azure Machine Learning data encryption at rest](#)
- [Understand encryption at rest in Azure](#)

- [How to configure customer managed encryption keys](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

4.9: Log and alert on changes to critical Azure resources

Guidance: Use Azure Monitor with the Azure Activity log to create alerts for when changes take place to production instances of Azure Machine Learning and other critical or related resources.

- [How to create alerts for Azure Activity Log events](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

Vulnerability management

For more information, see the [Azure Security Benchmark: Vulnerability management](#).

5.1: Run automated vulnerability scanning tools

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for vulnerability management of Azure Machine Learning service.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, follow the recommendations from Azure Security Center for performing vulnerability assessments on your Azure virtual machines, container images, and SQL servers.

- [How to implement Azure Security Center vulnerability assessment recommendations](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

5.2: Deploy automated operating system patch management solution

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for patch management of Azure Machine Learning service.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For any compute resources that are owned by your organization, use Azure Automation Update Management to ensure that the most recent security updates are installed on your Windows and Linux VMs. For Windows VMs, ensure Windows Update has been enabled and set to update automatically.

- [How to configure Update Management for virtual machines in Azure](#)
- [Understand Azure security policies monitored by Security Center](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

5.3: Deploy an automated patch management solution for third-party software titles

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use a third-party patch management solution. Customers already using Configuration Manager in their environment can also use System Center Updates Publisher, allowing them to publish custom updates into Windows Server Update Service. This allows Update Management to patch machines that use Configuration Manager as their update repository with third-party software.

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

5.4: Compare back-to-back vulnerability scans

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, follow recommendations from Azure Security Center for performing vulnerability assessments on your Azure virtual machines, container images, and SQL servers. Export scan results at consistent intervals and compare the results with previous scans to verify that vulnerabilities have been remediated. When using vulnerability management recommendations suggested by Azure Security Center, you can pivot into the selected solution's portal to view historical scan data.

- [How to implement Azure Security Center vulnerability assessment recommendations](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

5.5: Use a risk-rating process to prioritize the remediation of discovered vulnerabilities

Guidance: Not applicable; this guideline is intended for compute resources.

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

Inventory and asset management

For more information, see the [Azure Security Benchmark: Inventory and asset management](#).

6.1: Use automated asset discovery solution

Guidance: Use Azure Resource Graph to query for and discover resources (such as compute, storage, network, ports, and protocols etc.) in your subscriptions. Ensure appropriate (read) permissions in your tenant and enumerate all Azure subscriptions as well as resources in your subscriptions.

Although classic Azure resources can be discovered via Azure Resource Graph Explorer, it is highly recommended to create and use Azure Resource Manager resources going forward.

- [How to create queries with Azure Resource Graph Explorer](#)
- [How to view your Azure subscriptions](#)
- [Understand Azure RBAC](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

6.2: Maintain asset metadata

Guidance: Apply tags to Azure resources, adding metadata to logically organize according into a taxonomy.

- [How to create and use tags](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

6.3: Delete unauthorized Azure resources

Guidance: Use tagging, management groups, and separate subscriptions where appropriate, to organize and track assets. Reconcile inventory on a regular basis and ensure unauthorized resources are deleted from the subscription in a timely manner.

- [How to create additional Azure subscriptions](#)
- [How to create management groups](#)
- [How to create and use tags](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

6.4: Define and maintain an inventory of approved Azure resources

Guidance: Create an inventory of approved Azure resources and approved software for compute resources as per your organizational needs.

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

6.5: Monitor for unapproved Azure resources

Guidance: Use Azure Policy to put restrictions on the type of resources that can be created in customer subscriptions using the following built-in policy definitions:

- Not allowed resource types
- Allowed resource types

In addition, use the Azure Resource Graph to query/discover resources within the subscriptions.

- [How to configure and manage Azure Policy](#)
- [How to create queries with Azure Graph](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

6.6: Monitor for unapproved software applications within compute resources

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Azure Automation Change Tracking and Inventory to automate the collection of inventory information from your Windows and Linux VMs. Software name, version, publisher, and refresh time are available from the Azure portal. To get the software installation date and other information, enable guest-level diagnostics and direct the Windows Event Logs to Log Analytics workspace.

- [How to enable Azure Automation inventory collection for a VM](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

6.7: Remove unapproved Azure resources and software applications

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Azure Security Center's File Integrity Monitoring (FIM) to identify all software installed on VMs. Another option that can be used instead of or in conjunction with FIM is Azure Automation Change Tracking and Inventory to collect inventory from your Linux and Windows VMs.

You can implement your own process for removing unauthorized software. You can also use a third-party solution to identify unapproved software.

Remove Azure resources when they are no longer needed.

- [How to use File Integrity Monitoring](#)
- [Understand Azure Automation Change Tracking and Inventory](#)
- [How to enable Azure virtual machine inventory](#)
- [Azure resource group and resource deletion](#)

Azure Security Center monitoring: Yes

Responsibility: Shared

6.8: Use only approved applications

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Azure Security Center adaptive application controls to ensure that only authorized software executes and all unauthorized software is blocked from executing on Azure Virtual Machines.

- [How to use Azure Security Center adaptive application controls](#)

Azure Security Center monitoring: Yes

Responsibility: Shared

6.9: Use only approved Azure services

Guidance: Use Azure Policy to put restrictions on the type of resources that can be created in customer subscriptions using the following built-in policy definitions:

- Not allowed resource types
- Allowed resource types

In addition, use the Azure Resource Graph to query for and discover resources in the subscriptions.

- [How to configure and manage Azure Policy](#)
- [How to create queries with Azure Resource Graph](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

6.10: Maintain an inventory of approved software titles

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For any compute resources that are owned by your organization, use Azure Security Center adaptive application controls to specify which file types a rule may or may not apply to.

Implement a third-party solution if adaptive application controls don't meet the requirement.

- [How to use Azure Security Center adaptive application controls](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

6.11: Limit users' ability to interact with Azure Resource Manager

Guidance: Use Azure AD Conditional Access to limit users' ability to interact with Azure Resources Manager by configuring "Block access" for the "Microsoft Azure Management" App.

- [How to configure Conditional Access to block access to Azure Resources Manager](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

6.12: Limit users' ability to execute scripts in compute resources

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, depending on the type of scripts, you can use operating system-specific configurations or third-party resources to limit users' ability to execute scripts in Azure compute resources. You can also use Azure Security Center adaptive application controls to ensure that only authorized software executes and all unauthorized software is blocked from executing on Azure Virtual Machines.

- [How to control PowerShell script execution in Windows environments](#)
- [How to use Azure Security Center adaptive application controls](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

6.13: Physically or logically segregate high risk applications

Guidance: Not applicable; this recommendation is intended for web applications running on Azure App Service or compute resources.

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

Secure configuration

For more information, see the [Azure Security Benchmark: Secure configuration](#).

7.1: Establish secure configurations for all Azure resources

Guidance: Define and implement standard security configurations for your Azure Machine Learning service with Azure Policy. Use Azure Policy aliases in the "Microsoft.MachineLearning" namespace to create custom policies to audit or enforce the configuration of your Azure Machine Learning services.

Azure Resource Manager has the ability to export the template in JavaScript Object Notation (JSON), which should be reviewed to ensure that the configurations meet the security requirements for your organization.

You can also use the recommendations from Azure Security Center as a secure configuration baseline for your Azure resources.

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, and make sure secure configurations apply to code resources as part of your Machine Learning environment.

- [How to view available Azure Policy aliases](#)
- [Tutorial: Create and manage policies to enforce compliance](#)
- [Single and multi-resource export to a template in Azure portal](#)
- [Security recommendations - a reference guide](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

7.2: Establish secure operating system configurations

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for operating system secure configurations of Azure Machine Learning service.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Azure Security Center recommendations to maintain security configurations on all compute resources. Additionally, you can use custom operating system images or Azure Automation State configuration to establish the security configuration of the operating system required by your organization.

- [How to monitor Azure Security Center recommendations](#)
- [Security recommendations - a reference guide](#)
- [Azure Automation State Configuration Overview](#)
- [Upload a VHD and use it to create new Windows VMs in Azure](#)
- [Create a Linux VM from a custom disk with the Azure CLI](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

7.3: Maintain secure Azure resource configurations

Guidance: Use Azure Policy [deny] and [deploy if not exist] to enforce secure settings across your Azure resources. In addition, you can use Azure Resource Manager templates to maintain the security configuration of your Azure resources required by your organization.

- [Understand Azure Policy effects](#)
- [Create and manage policies to enforce compliance](#)
- [Azure Resource Manager templates overview](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

7.4: Maintain secure operating system configurations

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for operating system secure configurations of Azure Machine Learning service.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, follow recommendations from Azure Security Center on performing vulnerability assessments on your Azure compute resources. In addition, you may use Azure Resource Manager templates, custom operating system images, or Azure Automation State Configuration to maintain the security configuration of the operating system required by your organization. The Microsoft virtual machine templates combined with the Azure Automation State Configuration may assist in meeting and maintaining the security requirements.

Note that Azure Marketplace virtual machine Images published by Microsoft are managed and maintained by Microsoft.

- [How to implement Azure Security Center vulnerability assessment recommendations](#)
- [How to create an Azure Virtual Machine from an ARM template](#)
- [Azure Automation State Configuration Overview](#)
- [Create a Windows virtual machine in the Azure portal](#)
- [Information on how to download the VM template](#)

- [Sample script to upload a VHD to Azure and create a new VM](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

7.5: Securely store configuration of Azure resources

Guidance: If using custom Azure Policy definitions for your Machine Learning or related resources, use Azure Repos to securely store and manage your code.

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, and make sure secure configurations apply to code resources as part of your Machine Learning environment.

- [How to store code in Azure DevOps](#)
- [Azure Repos documentation](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

7.6: Securely store custom operating system images

Guidance: Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Azure role-based access control (Azure RBAC) to ensure that only authorized users can access your custom images. Use an Azure Shared Image Gallery you can share your images to different users, service principals, or Azure AD groups within your organization. Store container images in Azure Container Registry and use Azure RBAC to ensure that only authorized users have access.

- [Understand Azure RBAC](#)
- [Understand Azure RBAC for Container Registry](#)
- [How to configure Azure RBAC](#)
- [Shared Image Gallery overview](#)
- [Use Azure RBAC for Kubernetes authorization](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Not Applicable

7.7: Deploy configuration management tools for Azure resources

Guidance: Use Azure Policy aliases in the "Microsoft.MachineLearning" namespace to create custom policies to alert, audit, and enforce system configurations. Additionally, develop a process and pipeline for managing policy exceptions.

- [How to configure and manage Azure Policy](#)
- [How to use aliases](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

7.8: Deploy configuration management tools for operating systems

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for secure configuration deployment of Azure Machine Learning service.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Azure Automation State Configuration for Desired State Configuration (DSC) nodes in any cloud or on-premises datacenter. You can easily onboard machines, assign them declarative configurations, and view reports showing each machine's compliance to the desired state you specified.

- [How to enable Azure Automation State Configuration](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

7.9: Implement automated configuration monitoring for Azure resources

Guidance: Use Azure Security Center to perform baseline scans for your Azure Resources. Additionally, use Azure Policy to alert and audit Azure resource configurations.

- [How to remediate recommendations in Azure Security Center](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

7.10: Implement automated configuration monitoring for operating systems

Guidance: If the compute resource is owned by Microsoft, then Microsoft is responsible for automated secure configuration monitoring of Azure Machine Learning service.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Azure Security Center Compute & Apps and follow the recommendations for VMs and servers, and containers.

- [Understand Azure Security Center container recommendations](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

7.11: Manage Azure secrets securely

Guidance: Use managed identities in conjunction with Azure Key Vault to simplify secret management for your cloud applications.

Azure Machine Learning supports data store encryption with customer-managed keys, you need to manage key rotate and revoke per organization security and compliance requirements.

Use Azure Key Vault to pass secrets to remote runs securely instead of cleartext in your training scripts.

- [Azure Machine Learning customer-managed keys](#)
- [Use authentication credential secrets in Azure Machine Learning training runs](#)
- [How to use managed identities for Azure resources](#)
- [How to create a Key Vault](#)
- [How to authenticate to Key Vault](#)
- [How to assign a Key Vault access policy](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

7.12: Manage identities securely and automatically

Guidance: Azure Machine Learning supports both built-in roles and the ability to create custom roles. Use managed identities to provide Azure services with an automatically managed identity in Azure AD. Managed identities allow you to authenticate to any service that supports Azure AD authentication, including Key Vault, without any credentials in your code.

- [Manage access to an Azure Machine Learning workspace](#)
- [How to configure managed identities for Azure resources](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

7.13: Eliminate unintended credential exposure

Guidance: Implement Credential Scanner to identify credentials within code. Credential Scanner will also encourage moving discovered credentials to more secure locations such as Azure Key Vault.

- [How to setup Credential Scanner](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

Malware defense

For more information, see the [Azure Security Benchmark: Malware defense](#).

8.1: Use centrally managed antimalware software

Guidance: Microsoft anti-malware is enabled on the underlying host that supports Azure services (for example, Azure Machine Learning), however, it does not run on customer content.

Azure Machine Learning has varying support across different compute resources and even your own compute resources. For compute resources that are owned by your organization, use Microsoft Antimalware for Azure to continuously monitor and defend your resources. For Linux, use third-party antimalware solution. Also, use Azure Security Center's threat detection for data services to detect malware uploaded to storage accounts.

- [How to configure Microsoft Antimalware for Azure](#)
- [Threat protection in Azure Security Center](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

8.2: Pre-scan files to be uploaded to non-compute Azure resources

Guidance: Microsoft Anti-malware is enabled on the underlying host that supports Azure services (for example, Azure Machine Learning), however it does not run on customer content.

It is your responsibility to pre-scan any content being uploaded to non-compute Azure resources. Microsoft cannot access customer data, and therefore cannot conduct anti-malware scans of customer content on your behalf.

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

8.3: Ensure antimalware software and signatures are updated

Guidance: Microsoft anti-malware is enabled and maintained for the underlying host that supports Azure services (for example, Azure Machine learning), however, it does not run on customer content.

Azure Machine Learning has varying support across different compute resources and even your own compute

resources. For any compute resources that are owned by your organization, follow recommendations in Azure Security Center, Compute & Apps to ensure all endpoints are up to date with the latest signatures. For Linux, use third-party antimalware solution.

- [How to deploy Microsoft Antimalware for Azure](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

Data recovery

For more information, see the [Azure Security Benchmark: Data recovery](#).

9.1: Ensure regular automated back ups

Guidance: Data recovery management in Machine Learning service is through data managements on connected data stores. Ensure to follow up data recovery guidelines for connected stores to back up data per customer organization policies.

- [How to recover files from Azure Virtual Machine backup](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

9.2: Perform complete system backups and backup any customer-managed keys

Guidance: Data backup in Machine Learning service is through data managements on connected data stores. Enable Azure Backup for VMs and configure the desired frequency and retention periods. Back up customer-managed keys in Azure Key Vault.

- [How to recover files from Azure Virtual Machine backup](#)
- [How to restore Key Vault keys in Azure](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

9.3: Validate all backups including customer-managed keys

Guidance: Data backup validation in Machine Learning service is through data managements on connected data stores. Periodically perform data restoration of content in Azure Backup. Ensure that you can restore backed-up customer-managed keys.

- [How to recover files from an Azure virtual machine backup](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

9.4: Ensure protection of backups and customer-managed keys

Guidance: For on-premises backup, encryption at rest is provided using the passphrase you provide when backing up to Azure. Use Azure role-based access control to protect backups and customer-managed keys.

Enable soft delete and purge protection in Key Vault to protect keys against accidental or malicious deletion. If Azure Storage is used to store backups, enable soft delete to save and recover your data when blobs or blob snapshots are deleted.

- [Understand Azure RBAC](#)
- [How to enable soft delete and purge protection in Key Vault](#)

- [Soft delete for Azure Blob storage](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

Incident response

For more information, see the [Azure Security Benchmark: Incident response](#).

10.1: Create an incident response guide

Guidance: Develop an incident response guide for your organization. Ensure there are written incident response plans that define all the roles of personnel as well as the phases of incident handling and management from detection to post-incident review.

- [Guidance on building your own security incident response process](#)
- [Microsoft Security Response Center's Anatomy of an Incident](#)
- [Use NIST's Computer Security Incident Handling Guide to aid in the creation of your own incident response plan](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

10.2: Create an incident scoring and prioritization procedure

Guidance: Azure Security Center assigns a severity to each alert to help you prioritize which alerts should be investigated first. The severity is based on how confident Security Center is in the finding or the analytically used to issue the alert as well as the confidence level that there was malicious intent behind the activity that led to the alert.

Additionally, mark subscriptions using tags and create a naming system to identify and categorize Azure resources, especially those processing sensitive data. It's your responsibility to prioritize the remediation of alerts based on the criticality of the Azure resources and environment where the incident occurred.

- [Security alerts in Azure Security Center](#)
- [Use tags to organize your Azure resources](#)

Azure Security Center monitoring: Yes

Responsibility: Customer

10.3: Test security response procedures

Guidance: Conduct exercises to test your systems' incident response capabilities on a regular cadence to help protect your Azure resources. Identify weak points and gaps and then revise your response plan as needed.

- [NIST's publication--Guide to Test, Training, and Exercise Programs for IT Plans and Capabilities](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

10.4: Provide security incident contact details and configure alert notifications for security incidents

Guidance: Security incident contact information will be used by Microsoft to contact you if the Microsoft Security Response Center (MSRC) discovers that your data has been accessed by an unlawful or unauthorized party. Review incidents after the fact to ensure that issues are resolved.

- [How to set the Azure Security Center security contact](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

10.5: Incorporate security alerts into your incident response system

Guidance: Export your Azure Security Center alerts and recommendations using the continuous export feature to help identify risks to Azure resources. Continuous export allows you to export alerts and recommendations either manually or in an ongoing, continuous fashion. You can use the Azure Security Center data connector to stream the alerts to Azure Sentinel.

- [How to configure continuous export](#)
- [How to stream alerts into Azure Sentinel](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

10.6: Automate the response to security alerts

Guidance: Use workflow automation feature Azure Security Center to automatically trigger responses to security alerts and recommendations to protect your Azure resources.

- [How to configure workflow automation in Security Center](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Customer

Penetration tests and red team exercises

For more information, see the [Azure Security Benchmark: Penetration tests and red team exercises](#).

11.1: Conduct regular penetration testing of your Azure resources and ensure remediation of all critical security findings

Guidance: Follow the Microsoft Cloud Penetration Testing Rules of Engagement to ensure your penetration tests are not in violation of Microsoft policies. Use Microsoft's strategy and execution of Red Teaming and live site penetration testing against Microsoft-managed cloud infrastructure, services, and applications.

- [Penetration Testing Rules of Engagement](#)
- [Microsoft Cloud Red Teaming](#)

Azure Security Center monitoring: Not Applicable

Responsibility: Shared

Next steps

- See the [Azure security benchmark](#)
- Learn more about [Azure security baselines](#)

What is responsible machine learning? (preview)

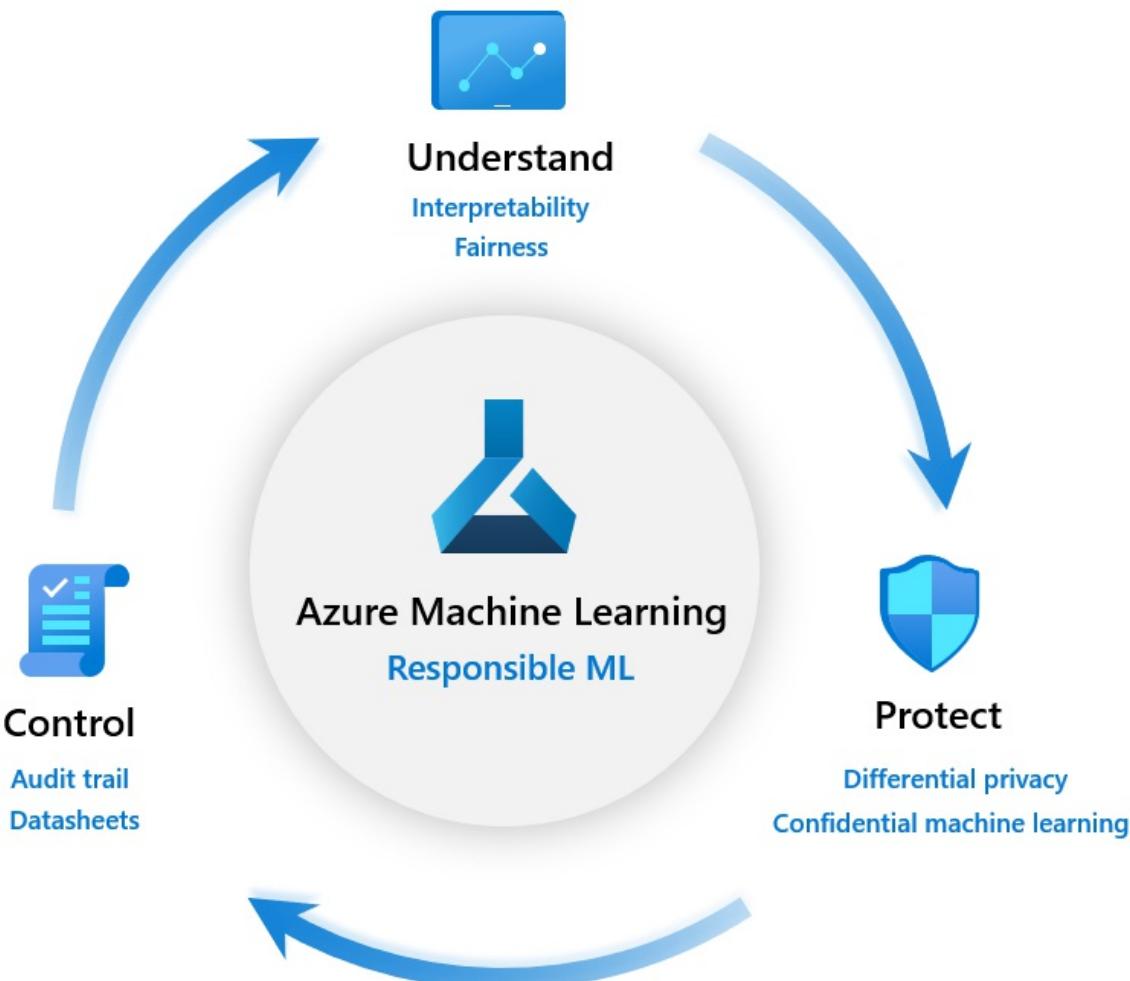
12/23/2020 • 4 minutes to read • [Edit Online](#)

In this article, you'll learn what responsible machine learning (ML) is and ways you can put it into practice with Azure Machine Learning.

Responsible machine learning principles

Throughout the development and use of AI systems, trust must be at the core. Trust in the platform, process, and models. At Microsoft, responsible machine learning encompasses the following values and principles:

- Understand machine learning models
 - Interpret and explain model behavior
 - Assess and mitigate model unfairness
- Protect people and their data
 - Prevent data exposure with differential privacy
 - Work with encrypted data using homomorphic encryption
- Control the end-to-end machine learning process
 - Document the machine learning lifecycle with datasheets



As artificial intelligence and autonomous systems integrate more into the fabric of society, it's important to

proactively make an effort to anticipate and mitigate the unintended consequences of these technologies.

Interpret and explain model behavior

Hard to explain or opaque-box systems can be problematic because it makes it hard for stakeholders like system developers, regulators, users, and business decision makers to understand why systems make certain decisions. Some AI systems are more explainable than others and there's sometimes a tradeoff between a system with higher accuracy and one that is more explainable.

To build interpretable AI systems, use [InterpretML](#), an open-source package built by Microsoft. [InterpretML](#) can be used inside of Azure Machine Learning to [interpret and explain your machine learning models](#), including automated machine learning models.

Mitigate fairness in machine learning models

As AI systems become more involved in the everyday decision-making of society, it's of extreme importance that these systems work well in providing fair outcomes for everyone.

Unfairness in AI systems can result in the following unintended consequences:

- Withholding opportunities, resources or information from individuals.
- Reinforcing biases and stereotypes.

Many aspects of fairness cannot be captured or represented by metrics. There are tools and practices that can improve fairness in the design and development of AI systems.

Two key steps in reducing unfairness in AI systems are assessment and mitigation. We recommend [FairLearn](#), an open-source package that can assess and mitigate the potential unfairness of AI systems. To learn more about fairness and the FairLearn package, see the [Fairness in ML article](#).

Prevent data exposure with differential privacy

When data is used for analysis, it's important that the data remains private and confidential throughout its use. Differential privacy is a set of systems and practices that help keep the data of individuals safe and private.

In traditional scenarios, raw data is stored in files and databases. When users analyze data, they typically use the raw data. This is a concern because it might infringe on an individual's privacy. Differential privacy tries to deal with this problem by adding "noise" or randomness to the data so that users can't identify any individual data points.

Implementing differentially private systems is difficult. [SmartNoise](#) is an open-source project that contains different components for building global differentially private systems. To learn more about differential privacy and the SmartNoise project, see the [preserve data privacy by using differential privacy and SmartNoise article](#).

Work on encrypted data with homomorphic encryption

In traditional cloud storage and computation solutions, the cloud needs to have unencrypted access to customer data to compute on it. This access exposes the data to cloud operators. Data privacy relies on access control policies implemented by the cloud and trusted by the customer.

Homomorphic encryption allows for computations to be done on encrypted data without requiring access to a secret (decryption) key. The results of the computations are encrypted and can be revealed only by the owner of the secret key. Using homomorphic encryption, cloud operators will never have unencrypted access to the data they're storing and computing on. Computations are performed directly on encrypted data. Data privacy relies on state-of-the-art cryptography, and the data owner controls all information releases. For more information on homomorphic encryption at Microsoft, see [Microsoft Research](#).

To get started with homomorphic encryption in Azure Machine Learning, use the [encrypted-inference](#) Python

bindings for [Microsoft SEAL](#). Microsoft SEAL is an open-source homomorphic encryption library that allows additions and multiplications to be performed on encrypted integers or real numbers. To learn more about Microsoft SEAL, see the [Azure Architecture Center](#) or the [Microsoft Research project page](#).

See the following sample to learn [how to deploy an encrypted inferencing web service in Azure Machine Learning](#).

Document the machine learning lifecycle with datasheets

Documenting the right information in the machine learning process is key to making responsible decisions at each stage. Datasheets are a way to document machine learning assets that are used and created as part of the machine learning lifecycle.

Models tend to be thought of as "opaque boxes" and often there is little information about them. Because machine learning systems are becoming more pervasive and are used for decision making, using datasheets is a step towards developing more responsible machine learning systems.

Some model information you might want to document as part of a datasheet:

- Intended use
- Model architecture
- Training data used
- Evaluation data used
- Training model performance metrics
- Fairness information.

See the following sample to learn how to use the Azure Machine Learning SDK to implement [datasheets for models](#).

Additional resources

- For more information, see the [responsible innovation toolkit](#) to learn about best practices.
- Learn more about the [ABOUT ML](#) set of guidelines for machine learning system documentation.

Model interpretability in Azure Machine Learning (preview)

12/23/2020 • 5 minutes to read • [Edit Online](#)

Overview of model interpretability

Interpretability is critical for data scientists, auditors, and business decision makers alike to ensure compliance with company policies, industry standards, and government regulations:

- Data scientists need the ability to explain their models to executives and stakeholders, so they can understand the value and accuracy of their findings. They also require interpretability to debug their models and make informed decisions about how to improve them.
- Legal auditors require tools to validate models with respect to regulatory compliance and monitor how models' decisions are impacting humans.
- Business decision makers need peace-of-mind by having the ability to provide transparency for end users. This allows them to earn and maintain trust.

Enabling the capability of explaining a machine learning model is important during two main phases of model development:

- During the training phase, as model designers and evaluators can use interpretability output of a model to verify hypotheses and build trust with stakeholders. They also use the insights into the model for debugging, validating model behavior matches their objectives, and to check for model unfairness or insignificant features.
- During the inferencing phase, as having transparency around deployed models empowers executives to understand "when deployed" how the model is working and how its decisions are treating and impacting people in real life.

Interpretability with Azure Machine Learning

The interpretability classes are made available through the following SDK package: (Learn how to [install SDK packages for Azure Machine Learning](#))

- `azureml.interpret`, contains functionalities supported by Microsoft.

Use `pip install azureml-interpret` for general use.

How to interpret your model

Using the classes and methods in the SDK, you can:

- Explain model prediction by generating feature importance values for the entire model and/or individual datapoints.
- Achieve model interpretability on real-world datasets at scale, during training and inference.
- Use an interactive visualization dashboard to discover patterns in data and explanations at training time

In machine learning, **features** are the data fields used to predict a target data point. For example, to predict credit risk, data fields for age, account size, and account age might be used. In this case, age, account size, and account age are **features**. Feature importance tells you how each data field affected the model's predictions. For example,

age may be heavily used in the prediction while account size and age do not affect the prediction values significantly. This process allows data scientists to explain resulting predictions, so that stakeholders have visibility into what features are most important in the model.

Learn about supported interpretability techniques, supported machine learning models, and supported run environments here.

Supported interpretability techniques

`azureml-interpret` uses the interpretability techniques developed in [Interpret-Community](#), an open source python package for training interpretable models and helping to explain blackbox AI systems. [Interpret-Community](#) serves as the host for this SDK's supported explainers, and currently supports the following interpretability techniques:

INTERPRETABILITY TECHNIQUE	DESCRIPTION	TYPE
SHAP Tree Explainer	SHAP 's tree explainer, which focuses on polynomial time fast SHAP value estimation algorithm specific to trees and ensembles of trees .	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 . TensorFlow models and Keras models using the TensorFlow backend are supported (there is also preliminary support for PyTorch)".	Model-specific
SHAP Linear Explainer	SHAP's Linear explainer computes SHAP values for a linear model , optionally accounting for inter-feature correlations.	Model-specific
SHAP Kernel Explainer	SHAP's Kernel explainer uses a specially weighted local linear regression to estimate SHAP values for any model .	Model-agnostic

INTERPRETABILITY TECHNIQUE	DESCRIPTION	TYPE
Mimic Explainer (Global Surrogate)	Mimic explainer is based on the idea of training global surrogate models to mimic blackbox models. A global surrogate model is an intrinsically interpretable model that is trained to approximate the predictions of any black box model as accurately as possible. Data scientists can interpret the surrogate model to draw conclusions about the black box model. You can use one of the following interpretable models as your surrogate model: LightGBM (LGBMExplainableModel), Linear Regression (LinearExplainableModel), Stochastic Gradient Descent explainable model (SGDExplainableModel), and Decision Tree (DecisionTreeExplainableModel).	Model-agnostic
Permutation Feature Importance Explainer (PFI)	Permutation Feature Importance is a technique used to explain classification and regression models that is 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 any underlying model but does not explain individual predictions.	Model-agnostic

Besides the interpretability techniques described above, we support another SHAP-based explainer, called `TabularExplainer`. Depending on the model, `TabularExplainer` uses one of the supported SHAP explainers:

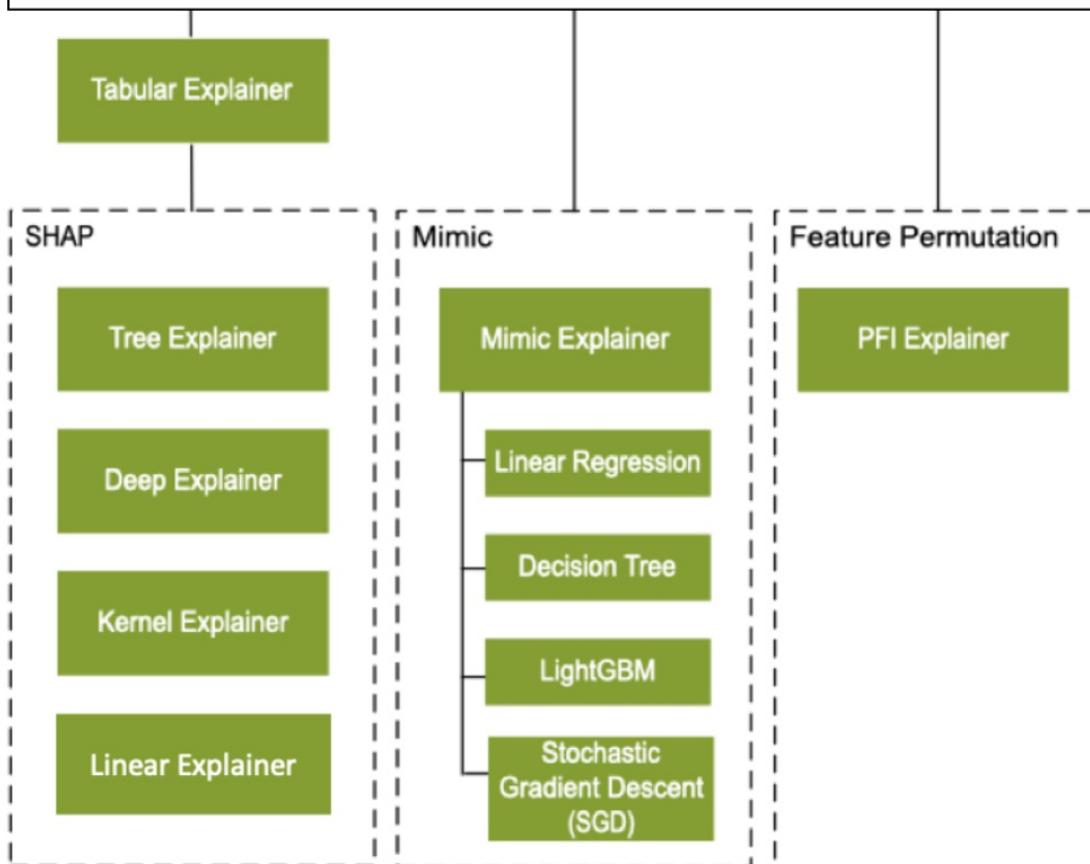
- `TreeExplainer` for all tree-based models
- `DeepExplainer` for DNN models
- `LinearExplainer` for linear models
- `KernelExplainer` for all other models

`TabularExplainer` has also made significant feature and performance enhancements over the direct SHAP Explainers:

- **Summarization of the initialization dataset.** In cases where speed of explanation is most important, we summarize the initialization dataset and generate a small set of representative samples, which speeds up the generation of overall and individual feature importance values.
- **Sampling the evaluation data set.** If the user passes in a large set of evaluation samples but does not actually need all of them to be evaluated, the sampling parameter can be set 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 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, the model must implement the prediction function `predict` or `predict_proba` that conforms to the Scikit convention. If your model does not support this, you can wrap your model 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 a pipeline is provided, the explanation function assumes that the running pipeline script returns a prediction. Using this wrapping technique, `azureml.interpret` can support models 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 run locally, The SDK functions will not contact any Azure services.

You can run explanation remotely on Azure Machine Learning Compute and log the explanation info into the Azure Machine Learning Run History Service. Once this information is logged, reports and visualizations from the explanation are readily available on Azure Machine Learning studio for user analysis.

Next steps

- See the [how-to](#) for enabling interpretability for models training both locally and on Azure Machine Learning remote compute resources.
- See the [sample notebooks](#) for additional scenarios.
- If you're interested in interpretability for text scenarios, see [Interpret-text](#), a related open source repo to [Interpret-Community](#), for interpretability techniques for NLP. `azureml.interpret` package does not currently support these techniques but you can get started with an [example notebook on text classification](#).

Mitigate fairness in machine learning models (preview)

12/23/2020 • 6 minutes to read • [Edit Online](#)

Learn about fairness in machine learning and how the [Fairlearn](#) open-source Python package can help you mitigate fairness issues in machine learning models. If you are not making an effort to understand fairness issues and to assess fairness when building machine learning models, you may build models that produce unfair results.

The following summary of the [user guide](#) for the Fairlearn open-source package, describes how to use it to assess the fairness of the AI systems that you are building. The Fairlearn open-source package can also offer options to help mitigate, or help to reduce, any fairness issues you observe. See the [how-to](#) and [sample notebooks](#) to enable fairness assessment of AI systems during training on Azure Machine Learning.

What is fairness in machine learning models?

NOTE

Fairness is a socio-technical challenge. Many aspects of fairness, such as justice and due process, are not captured in quantitative fairness metrics. Also, many quantitative fairness metrics can't all be satisfied simultaneously. The goal with the Fairlearn open-source package is to enable humans to assess different impact and mitigation strategies. Ultimately, it is up to the human users building artificial intelligence and machine learning models to make trade-offs that are appropriate to their scenario.

Artificial intelligence and machine learning systems can display unfair behavior. One way to define unfair behavior is by its harm, or impact on people. There are many types of harm that AI systems can give rise to. See the [NeurIPS 2017 keynote by Kate Crawford](#) to learn more.

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 much better at picking good candidates among a specific group of people than among other groups.
- Harm of quality-of-service: An AI system does not work as well for one group of people as it does for another. As an 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.

Fairness assessment and mitigation with Fairlearn

Fairlearn is an open-source Python package that allows machine learning systems developers to assess their systems' fairness and mitigate the observed fairness issues.

The Fairlearn open-source package has two components:

- Assessment Dashboard: A Jupyter notebook widget for assessing how a model's predictions affect different groups. It also enables comparing multiple models by using fairness and performance metrics.
- Mitigation Algorithms: A set of algorithms to mitigate unfairness in binary classification and regression.

Together, these components enabled data scientists and business leaders to navigate any trade-offs between fairness and performance, and to select the mitigation strategy that best fits their needs.

Assess fairness in machine learning models

In the Fairlearn open-source package, fairness is conceptualized through an approach known as **group fairness**, which asks: Which groups of individuals are at risk for experiencing harms? The relevant groups, also known as subpopulations, are defined through **sensitive features** or sensitive attributes. Sensitive features are passed to an estimator in the Fairlearn open-source package as a vector or a matrix called `sensitive_features`. The term suggests that the system designer should be sensitive to these features when assessing group fairness.

Something to be mindful of is whether these features contain privacy implications due to private data. But the word "sensitive" doesn't imply that these features shouldn't be used to make predictions.

NOTE

A fairness assessment is not a purely technical exercise. The Fairlearn open-source package can help you assess the fairness of a model, but it will not perform the assessment for you. The Fairlearn open-source package helps identify quantitative metrics to assess fairness, but developers must also perform a qualitative analysis to evaluate the fairness of their own models. The sensitive features noted above is an example of this kind of qualitative analysis.

During assessment phase, fairness is quantified through disparity metrics. **Disparity metrics** can evaluate and compare model's behavior across different groups either as ratios or as differences. The Fairlearn open-source package 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 different subgroups. Some examples include:
 - disparity in accuracy rate
 - disparity in error rate
 - disparity in precision
 - disparity in recall
 - disparity in MAE
 - many others
- Disparity in selection rate: This metric contains the difference in selection rate among different subgroups. An example of this is disparity in loan approval rate. Selection rate means the fraction of datapoints in each class classified as 1 (in binary classification) or distribution of prediction values (in regression).

Mitigate unfairness in machine learning models

Parity constraints

The Fairlearn open-source package includes a variety of unfairness mitigation algorithms. 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 behavior to be comparable across the groups that sensitive features define (e.g., different races). The mitigation algorithms in the Fairlearn open-source package use such parity constraints to mitigate the observed fairness issues.

NOTE

Mitigating unfairness in a model means reducing the unfairness, but this technical mitigation cannot eliminate this unfairness completely. The unfairness mitigation algorithms in the Fairlearn open-source package can provide suggested mitigation strategies to help reduce unfairness in a machine learning model, but they are not solutions to eliminate unfairness completely. There may be other parity constraints or criteria that should be considered for each particular developer's machine learning model. Developers using Azure Machine Learning must determine for themselves if the mitigation sufficiently eliminates any unfairness in their intended use and deployment of machine learning models.

The Fairlearn open-source package supports the following types of parity constraints:

PARITY 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 postprocessing and reduction unfairness mitigation algorithms:

- Reduction: These algorithms take a standard black-box machine learning estimator (e.g., a LightGBM model) and generate a set of retrained models using a sequence of re-weighted training datasets. For example, applicants of a certain gender might be up-weighted or down-weighted to retrain models and reduce disparities across different gender groups. Users can then pick a model that provides the best trade-off between accuracy (or other performance metric) and disparity, which generally would need to be based on business rules and cost calculations.
- Post-processing: These algorithms take an existing classifier and the sensitive feature as input. Then, they derive a transformation of the classifier's prediction to enforce the specified fairness constraints. The biggest advantage of threshold optimization is its simplicity and flexibility as it does not need to retrain the model.

ALGORITHM	DESCRIPTION	MACHINE LEARNING TASK	SENSITIVE FEATURES	SUPPORTED PARITY CONSTRAINTS	ALGORITHM TYPE
ExponentiatedGradient	Black-box approach to fair classification described in A Reductions Approach to Fair Classification	Binary classification	Categorical	Demographic parity, equalized odds	Reduction
GridSearch	Black-box approach described in A Reductions Approach to Fair Classification	Binary classification	Binary	Demographic parity, equalized odds	Reduction

ALGORITHM	DESCRIPTION	MACHINE LEARNING TASK	SENSITIVE FEATURES	SUPPORTED PARITY CONSTRAINTS	ALGORITHM TYPE
GridSearch	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
ThresholdOptimizer	Postprocessing algorithm based on the paper Equality of Opportunity in Supervised Learning . This technique takes as input an existing classifier and the sensitive feature, and 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 use the different components by checking out the Fairlearn's [GitHub](#), [user guide](#), [examples](#), and [sample notebooks](#).
- Learn [how to](#) enable fairness assessment of machine learning models in Azure Machine Learning.
- See the [sample notebooks](#) for additional fairness assessment scenarios in Azure Machine Learning.

Preserve data privacy by using differential privacy and the SmartNoise package (preview)

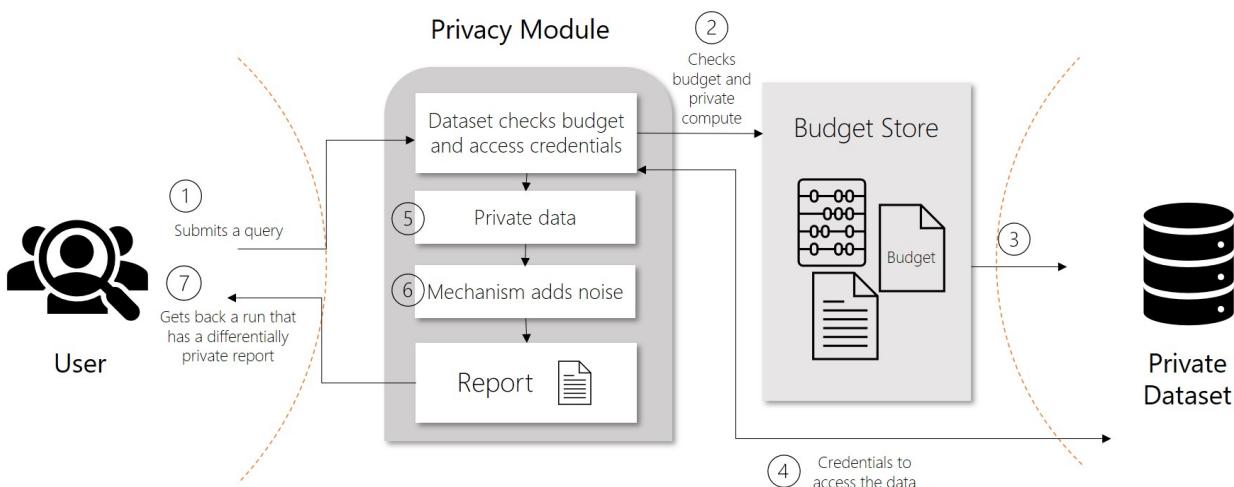
12/23/2020 • 4 minutes to read • [Edit Online](#)

Learn what differential privacy is and how the SmartNoise package can help you implement differentially private systems.

As the amount of data that an organization collects and uses for analyses increases, so do concerns of privacy and security. Analyses require data. Typically, the more data used to train models, the more accurate they are. When personal information is used for these analyses, it's especially important that the data remains private throughout its use.

How differential privacy works

Differential privacy is a set of systems and practices that help keep the data of individuals safe and private.



In traditional scenarios, raw data is stored in files and databases. When users analyze data, they typically use the raw data. This is a concern because it might infringe on an individual's privacy. Differential privacy tries to deal with this problem by adding "noise" or randomness to the data so that users can't identify any individual data points. At the least, such a system provides plausible deniability.

In differentially private systems, data is shared through requests called **queries**. When a user submits a query for data, operations known as **privacy mechanisms** add noise to the requested data. Privacy mechanisms return an *approximation of the data* instead of the raw data. This privacy-preserving result appears in a **report**. Reports consist of two parts, the actual data computed and a description of how the data was created.

Differential privacy metrics

Differential privacy tries to protect against the possibility that a user can produce an indefinite number of reports to eventually reveal sensitive data. A value known as **epsilon** measures how noisy or private a report is. Epsilon has an inverse relationship to noise or privacy. The lower the epsilon, the more noisy (and private) the data is.

Epsilon values are non-negative. Values below 1 provide full plausible deniability. Anything above 1 comes with a higher risk of exposure of the actual data. As you implement differentially private systems, you want to produce reports with epsilon values between 0 and 1.

Another value directly correlated to epsilon is **delta**. Delta is a measure of the probability that a report is not fully

private. The higher the delta, the higher the epsilon. Because these values are correlated, epsilon is used more often.

Privacy budget

To ensure privacy in systems where multiple queries are allowed, differential privacy defines a rate limit. This limit is known as a **privacy budget**. Privacy budgets are allocated an epsilon amount, typically between 1 and 3 to limit the risk of reidentification. As reports are generated, privacy budgets keep track of the epsilon value of individual reports as well as the aggregate for all reports. After a privacy budget is spent or depleted, users can no longer access data.

Reliability of data

Although the preservation of privacy should be the goal, there is a tradeoff when it comes to usability and reliability of the data. In data analytics, accuracy can be thought of as a measure of uncertainty introduced by sampling errors. This uncertainty tends to fall within certain bounds. **Accuracy** from a differential privacy perspective instead measures the reliability of the data, which is affected by the uncertainty introduced by the privacy mechanisms. In short, a higher level of noise or privacy translates to data that has a lower epsilon, accuracy, and reliability. Although the data is more private, because it's not reliable, the less likely it is to be used.

Implementing differentially private systems

Implementing differentially private systems is difficult. SmartNoise is an open-source project that contains different components for building global differentially private systems. SmartNoise is made up of the following top-level components:

- Core
- SDK

Core

The core library includes the following privacy mechanisms for implementing a differentially private system:

COMPONENT	DESCRIPTION
Analysis	A graph description of arbitrary computations.
Validator	A Rust library that contains a set of tools for checking and deriving the necessary conditions for an analysis to be differentially private.
Runtime	The medium to execute the analysis. The reference runtime is written in Rust but runtimes can be written using any computation framework such as SQL and Spark depending on your data needs.
Bindings	Language bindings and helper libraries to build analyses. Currently SmartNoise provides Python bindings.

SDK

The system library provides the following tools and services for working with tabular and relational data:

COMPONENT	DESCRIPTION
-----------	-------------

COMPONENT	DESCRIPTION
Data Access	<p>Library that intercepts and processes SQL queries and produces reports. This library is implemented in Python and supports the following ODBC and DBAPI data sources:</p> <ul style="list-style-type: none"> • PostgreSQL • SQL Server • Spark • Preston • Pandas
Service	<p>Execution service that provides a REST endpoint to serve requests or queries against shared data sources. The service is designed to allow composition of differential privacy modules that operate on requests containing different delta and epsilon values, also known as heterogeneous requests. This reference implementation accounts for additional impact from queries on correlated data.</p>
Evaluator	<p>Stochastic evaluator that checks for privacy violations, accuracy, and bias. The evaluator supports the following tests:</p> <ul style="list-style-type: none"> • Privacy Test - Determines whether a report adheres to the conditions of differential privacy. • Accuracy Test - Measures whether the reliability of reports falls within the upper and lower bounds given a 95% confidence level. • Utility Test - Determines whether the confidence bounds of a report are close enough to the data while still maximizing privacy. • Bias Test - Measures the distribution of reports for repeated queries to ensure they are not unbalanced

Next steps

[Preserve data privacy](#) in Azure Machine Learning.

To learn more about the components of SmartNoise, check out the GitHub repositories for [SmartNoise Core package](#), [SmartNoise SDK](#), and [SmartNoise samples](#).

Set up authentication for Azure Machine Learning resources and workflows

12/23/2020 • 11 minutes to read • [Edit Online](#)

Learn how to set up authentication to your Azure Machine Learning workspace. Authentication to your Azure Machine Learning workspace is based on **Azure Active Directory** (Azure AD) for most things. In general, there are three 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.
- **Managed identity:** When using the Azure Machine Learning SDK *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*.

IMPORTANT

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](#).

Prerequisites

- Create an [Azure Machine Learning workspace](#).
- [Configure your development environment](#) to install the Azure Machine Learning SDK, or use a [Azure Machine Learning compute instance](#) with the SDK already installed.

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 and 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. Install the Azure Machine Learning extension:

```
az extension add -n azure-cli-ml
```

3. 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
```

The output will be a JSON similar to the following. Take note of the `clientId`, `clientSecret`, and `tenantId` fields, as you will 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"  
}
```

4. 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 will need its value for the next step.

```
{  
    "accountEnabled": "True",  
    "addIns": [],  
    "appDisplayName": "ml-auth",  
    ...  
    ...  
    ...  
    "objectId": "your-sp-object-id",  
    "objectType": "ServicePrincipal"  
}
```

5. Allow the SP to access your Azure Machine Learning workspace. You will need your workspace name, and its resource group name for the `-w` and `-g` parameters, respectively. For the `--user` parameter, use the `objectId` value from the previous step. The `--role` parameter allows you to set the access role for the service principal. In the following example, the SP is assigned to the `owner` role.

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](#).

```
az ml workspace share -w your-workspace-name -g your-resource-group-name --user your-sp-object-id --role owner
```

This call does not produce any output on success.

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. Using a managed identity with a compute cluster is currently in preview.

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)**, **Add Role Assignment**, and select **Virtual Machine** from the **Assign Access To** dropdown. Finally, select your VM's identity.
3. Select the role to assign to this identity. For example, contributor or a custom role. For more information see, [Control access to resources](#).

Managed identity with compute cluster

For more information, see [Set up managed identity for compute cluster](#).

Use interactive authentication

IMPORTANT

Interactive authentication uses your browser, and requires cookies (including 3rd party cookies). If you have disabled cookies, you may receive an error such as "we couldn't sign you in." This error may also occur if you have enabled [Azure AD Multi-Factor Authentication](#).

Most examples in the documentation and samples use interactive authentication. For example, when using the SDK there are two function calls that will automatically prompt you with a UI-based authentication flow:

- Calling the `from_config()` function will issue the prompt.

```
from azureml.core import Workspace
ws = Workspace.from_config()
```

The `from_config()` function looks for a JSON file containing your workspace connection information.

- Using the `Workspace` constructor to provide subscription, resource group, and workspace information, will also prompt for interactive authentication.

```
ws = Workspace(subscription_id="your-sub-id",
               resource_group="your-resource-group-id",
               workspace_name="your-workspace-name"
              )
```

TIP

If you have access to multiple tenants, you may need to import the class and explicitly define what tenant you are targeting. Calling the constructor for `InteractiveLoginAuthentication` will also prompt you to login similar to the calls above.

```
from azureml.core.authentication import InteractiveLoginAuthentication
interactive_auth = InteractiveLoginAuthentication(tenant_id="your-tenant-id")
```

When using the Azure CLI, the `az login` command is used to authenticate the CLI session. For more information, see [Get started with Azure CLI](#).

TIP

If you are using the SDK from an environment where you have previously authenticated interactively using the Azure CLI, you can use the `AzureCliAuthentication` class to authenticate to the workspace using the credentials cached by the CLI:

```
from azureml.core.authentication import AzureCliAuthentication
cli_auth = AzureCliAuthentication()
ws = Workspace(subscription_id="your-sub-id",
               resource_group="your-resource-group-id",
               workspace_name="your-workspace-name",
               auth=cli_auth
              )
```

Use service principal authentication

To authenticate to your workspace from the SDK, using a service principal, use the `ServicePrincipalAuthentication` class constructor. Use the values you got when creating the service provider as the parameters. The `tenant_id` parameter maps to `tenantId` from above, `service_principal_id` maps to `clientId`, and `service_principal_password` maps to `clientSecret`.

```
from azureml.core.authentication import ServicePrincipalAuthentication

sp = ServicePrincipalAuthentication(tenant_id="your-tenant-id", # tenantID
                                    service_principal_id="your-client-id", # clientId
                                    service_principal_password="your-client-secret") # clientSecret
```

The `sp` variable now holds an authentication object that you use directly in the SDK. In general, it is a good idea to store the ids/secrets used above in environment variables as shown in the following code. Storing in environment variables prevents the information from being accidentally checked into a GitHub repo.

```
import os

sp = ServicePrincipalAuthentication(tenant_id=os.environ['AML_TENANT_ID'],
                                    service_principal_id=os.environ['AML_PRINCIPAL_ID'],
                                    service_principal_password=os.environ['AML_PRINCIPAL_PASS'])
```

For automated workflows that run in Python and use the SDK primarily, you can use this object as-is in most cases for your authentication. The following code authenticates to your workspace using the `auth` object you created.

```
from azureml.core import Workspace

ws = Workspace.get(name="ml-example",
                   auth=sp,
                   subscription_id="your-sub-id")
ws.get_details()
```

Use a service principal from the Azure CLI

You can use a service principal for Azure CLI commands. For more information, see [Sign in using a service principal](#).

Use a service principal with the REST API (preview)

The service principal can also be used to authenticate to the Azure Machine Learning [REST API](#) (preview). You use the Azure Active Directory [client credentials grant flow](#), which allow service-to-service calls for headless authentication in automated workflows. The examples are implemented with the [ADAL library](#) in both Python and

Node.js, but you can also use any open-source library that supports OpenID Connect 1.0.

NOTE

MSAL.js is a newer library than ADAL, but you cannot do service-to-service authentication using client credentials with MSAL.js, since it is primarily a client-side library intended for interactive/UI authentication tied to a specific user. We recommend using ADAL as shown below to build automated workflows with the REST API.

Node.js

Use the following steps to generate an auth token using Node.js. In your environment, run `npm install adal-node`. Then, use your `tenantId`, `clientId`, and `clientSecret` from the service principal you created in the steps above as values for the matching variables in the following script.

```
const adal = require('adal-node').AuthenticationContext;

const authorityHostUrl = 'https://login.microsoftonline.com/';
const tenantId = 'your-tenant-id';
const authorityUrl = authorityHostUrl + tenantId;
const clientId = 'your-client-id';
const clientSecret = 'your-client-secret';
const resource = 'https://management.azure.com/';

const context = new adal(authorityUrl);

context.acquireTokenWithClientCredentials(
  resource,
  clientId,
  clientSecret,
  (err, tokenResponse) => {
    if (err) {
      console.log(`Token generation failed due to ${err}`);
    } else {
      console.dir(tokenResponse, { depth: null, colors: true });
    }
  }
);
```

The variable `tokenResponse` is an object that includes the token and associated metadata such as expiration time. Tokens are valid for 1 hour, and can be refreshed by running the same call again to retrieve a new token. The following snippet is a sample response.

```
{
  tokenType: 'Bearer',
  expiresIn: 3599,
  expiresOn: 2019-12-17T19:15:56.326Z,
  resource: 'https://management.azure.com/',
  accessToken: "random-oauth-token",
  isMRRT: true,
  _clientId: 'your-client-id',
  _authority: 'https://login.microsoftonline.com/your-tenant-id'
}
```

Use the `accessToken` property to fetch the auth token. See the [REST API documentation](#) for examples on how to use the token to make API calls.

Python

Use the following steps to generate an auth token using Python. In your environment, run `pip install adal`. Then, use your `tenantId`, `clientId`, and `clientSecret` from the service principal you created in the steps above as values for the appropriate variables in the following script.

```
from adal import AuthenticationContext

client_id = "your-client-id"
client_secret = "your-client-secret"
resource_url = "https://login.microsoftonline.com"
tenant_id = "your-tenant-id"
authority = "{}{}".format(resource_url, tenant_id)

auth_context = AuthenticationContext(authority)
token_response = auth_context.acquire_token_with_client_credentials("https://management.azure.com/",
client_id, client_secret)
print(token_response)
```

The variable `token_response` is a dictionary that includes the token and associated metadata such as expiration time. Tokens are valid for 1 hour, and can be refreshed by running the same call again to retrieve a new token. The following snippet is a sample response.

```
{
  'tokenType': 'Bearer',
  'expiresIn': 3599,
  'expiresOn': '2019-12-17 19:47:15.150205',
  'resource': 'https://management.azure.com/',
  'accessToken': 'random-oauth-token',
  'isMRRT': True,
  '_clientId': 'your-client-id',
  '_authority': 'https://login.microsoftonline.com/your-tenant-id'
}
```

Use `token_response["accessToken"]` to fetch the auth token. See the [REST API documentation](#) for examples on how to use the token to make API calls.

Java

In Java, retrieve the bearer token using a standard REST call:

```

String tenantId = "your-tenant-id";
String clientId = "your-client-id";
String clientSecret = "your-client-secret";
String resourceManagerUrl = "https://management.azure.com";

HttpRequest tokenAuthenticationRequest = tokenAuthenticationRequest(tenantId, clientId, clientSecret,
resourceManagerUrl);

HttpClient client = HttpClient.newBuilder().build();
Gson gson = new Gson();
HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
if (response.statusCode == 200)
{
    body = gson.fromJson(body, AuthenticationBody.class);

    // ... etc ...
}
// ... etc ...

static HttpRequest tokenAuthenticationRequest(String tenantId, String clientId, String clientSecret, String
resourceManagerUrl){
    String authUrl = String.format("https://login.microsoftonline.com/%s/oauth2/token", tenantId);
    String clientIdParam = String.format("client_id=%s", clientId);
    String resourceParam = String.format("resource=%s", resourceManagerUrl);
    String clientSecretParam = String.format("client_secret=%s", clientSecret);

    String bodyString = String.format("grant_type=client_credentials&%s&%s&%s", clientIdParam, resourceParam,
clientSecretParam);

    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(authUrl))
        .POST(HttpRequest.BodyPublishers.ofString(bodyString))
        .build();
    return request;
}

class AuthenticationBody {
    String access_token;
    String token_type;
    int expires_in;
    String scope;
    String refresh_token;
    String id_token;

    AuthenticationBody() {}
}

```

The preceding code would have to handle exceptions and status codes other than `200 OK`, but shows the pattern:

- Use the client ID and secret to validate that your program should have access
- Use your tenant ID to specify where `login.microsoftonline.com` should be looking
- Use Azure Resource Manager as the source of the authorization token

Use managed identity authentication

To authenticate to the workspace from a VM or compute cluster that is configured with a managed identity, use the `MsiAuthentication` class. The following example demonstrates how to use this class to authenticate to a workspace:

```
from azureml.core.authentication import MsiAuthentication

msi_auth = MsiAuthentication()

ws = Workspace(subscription_id="your-sub-id",
               resource_group="your-resource-group-id",
               workspace_name="your-workspace-name",
               auth=msi_auth
              )
```

Next steps

- [How to use secrets in training.](#)
- [How to configure authentication for models deployed as a web service.](#)
- [Consume an Azure Machine Learning model deployed as a web service.](#)

Manage access to an Azure Machine Learning workspace

12/23/2020 • 12 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

An Azure Machine Learning workspace is an Azure resource. Like other Azure resources, when a new Azure Machine Learning workspace is created, it comes with three default roles. You can add users to the workspace and assign them to one of these built-in roles.

ROLE	ACCESS LEVEL
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.
Custom Role	Allows you to customize access to specific control or data plane operations within a workspace. For example, submitting a run, creating a compute, deploying a model or registering a dataset.

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](#).

Currently there are no additional built-in roles that are specific to Azure Machine Learning. For more information on built-in roles, see [Azure built-in roles](#).

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](#)

If you have installed the [Azure Machine Learning CLI](#), you can use CLI commands to assign roles to users:

```
az ml workspace share -w <workspace_name> -g <resource_group_name> --role <role_name> --user <user_corp_email_address>
```

The `user` field is the email address of an existing user in the instance of Azure Active Directory where the workspace parent subscription lives. Here is an example of how to use this command:

```
az ml workspace share -w my_workspace -g my_resource_group --role Contributor --user jdoe@contoson.com
```

NOTE

"az ml workspace share" command does not work for federated account by Azure Active Directory B2B. Please use Azure UI portal instead of command.

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.MachineLearningServices/works
paces/<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. Or, you can assign this role to a user by using the `az ml workspace share` CLI command:

```
az ml workspace share -w my_workspace -g my_resource_group --role "Data Scientist" --user jdoe@contoson.com
```

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.

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 <code>/</code> are relative paths to <code>Microsoft.MachineLearningServices/</code> :			
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>
Submitting any type of run	Not required	Not required	Owner, contributor, or custom role allowing: <code>"/workspaces/*/read", "/workspaces/environments/write", "/workspaces/experiments/runs/write", "/workspaces/metadata/artifacts/write", "/workspaces/metadata/snapshots/write", "/workspaces/environments/build/actions/write", "/workspaces/experiments/runs/submit", "/workspaces/environments/readSecrets"</code>
Publishing pipelines and endpoints	Not required	Not required	Owner, contributor, or custom role allowing: <code>"/workspaces/endpoints/pipelines/*", "/workspaces/pipelinedrafts/*", "/workspaces/modules/*"</code>
Deploying a registered model on an AKS/ACI resource	Not required	Not required	Owner, contributor, or custom role allowing: <code>"/workspaces/services/aks/write", "/workspaces/services/aci/write"</code>
Scoring against a deployed AKS endpoint	Not required	Not required	Owner, contributor, or custom role allowing: <code>"/workspaces/services/aks/score/actions/write", "/workspaces/services/aks/listkeys"</code> (when you are not using Azure Active Directory auth) OR <code>"/workspaces/read"</code> (when you are using token auth)
Accessing storage using interactive notebooks	Not required	Not required	Owner, contributor, or custom role allowing: <code>"/workspaces/computes/read", "/workspaces/notebooks/samples/read", "/workspaces/notebooks/storage/*", "/workspaces/listKeys/action"</code>

ACTIVITY	SUBSCRIPTION-LEVEL SCOPE	RESOURCE GROUP-LEVEL SCOPE	WORKSPACE-LEVEL SCOPE
Create new custom role	Owner, contributor, or custom role allowing <code>Microsoft.Authorization/roleDefinitions/write</code>	Not required	Owner, contributor, or custom role allowing: <code>/workspaces/computes/write</code>

TIP

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.

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](#)

MLflow operations

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	<code>Microsoft.MachineLearningServices/workspaces/experiments/read</code>
Create an experiment with a name , set a tag on an experiment, restore an experiment marked for deletion	<code>Microsoft.MachineLearningServices/workspaces/experiments/write</code>
Delete an experiment	<code>Microsoft.MachineLearningServices/workspaces/experiments/delete</code>
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	<code>Microsoft.MachineLearningServices/workspaces/experiments/runs/read</code>
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	<code>Microsoft.MachineLearningServices/workspaces/experiments/runs/write</code>
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	<code>Microsoft.MachineLearningServices/workspaces/models/read</code>
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	<code>Microsoft.MachineLearningServices/workspaces/models/write</code>
Delete a registered model along with all its version, delete specific versions of a registered model	<code>Microsoft.MachineLearningServices/workspaces/models/delete</code>

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 labeler

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/labels/write"
    ],
    "NotActions": [
        "Microsoft.MachineLearningServices/workspaces/labeling/projects/summary/read"
    ],
    "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/join/action` on the VNet resource.
 - `Microsoft.Network/virtualNetworks/subnet/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 models](#)
- [Resource provider operations](#)

Use Managed identities with Azure Machine Learning (preview)

12/23/2020 • 7 minutes to read • [Edit Online](#)

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 is 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.

IMPORTANT

Using managed identities to control access to resources with Azure Machine Learning is currently in preview. Preview functionality is provided "as-is", with no guarantee of support or service level agreement. For more information, see the [Supplemental terms of use for Microsoft Azure previews](#).

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- 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.

```
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 do not 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 run 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.

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

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

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.

```
az ml computetarget amlcompute identity show --name <cluster name> -w <workspace> -g <resource group>
```

Optionally, you can update the compute cluster to assign a user-assigned managed identity:

```
az ml computetarget amlcompute identity assign --name cpocluster \
-w $mlws -g $mlrg --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

```
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 run, specify the base image location in the [environment definition](#).

```
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

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 additional 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 <UAI resource ID>
```

The UAI resource ID is Azure resource ID of the user assigned identity, in the format

```
/subscriptions/<subscription ID>/resourceGroups/<resource group>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<UAI name>
```

3. Specify the external ACR and client ID of the **user-assigned managed identity** in workspace connections by using [Workspace.set_connection method](#):

```
workspace.set_connection(
    name="privateAcr",
    category="ACR",
    target = "<acr url>",
    authType = "RegistryConnection",
    value={"ResourceId": "<UAI resource id>", "ClientId": "<UAI 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:

```
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:

```
from azureml.core.container_registry import RegistryIdentity  
  
identity = RegistryIdentity()  
identity.resource_id= "<UAI resource ID>"  
identity.client_id="<UAI 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.

Next steps

- Learn more about [enterprise security in Azure Machine Learning](#).

Use Azure AD identity with your machine learning web service in Azure Kubernetes Service

12/23/2020 • 4 minutes to read • [Edit Online](#)

In this how-to, you learn how to assign an Azure Active Directory (Azure AD) identity to your deployed machine learning model in Azure Kubernetes Service. The [Azure AD Pod Identity](#) project allows applications to access cloud resources securely with Azure AD by using a [Managed Identity](#) and Kubernetes primitives. This allows your web service to securely access your Azure resources without having to embed credentials or manage tokens directly inside your `score.py` script. This article explains the steps to create and install an Azure Identity in your Azure Kubernetes Service cluster and assign the identity to your deployed web service.

Prerequisites

- The [Azure CLI extension for the Machine Learning service](#), the [Azure Machine Learning SDK for Python](#), or the [Azure Machine Learning Visual Studio Code extension](#).
- Access to your AKS cluster using the `kubectl` command. For more information, see [Connect to the cluster](#)
- An Azure Machine Learning web service deployed to your AKS cluster.

Create and install an Azure Identity

1. To determine if your AKS cluster is Kubernetes RBAC enabled, use the following command:

```
az aks show --name <AKS cluster name> --resource-group <resource group name> --subscription <subscription id> --query enableRbac
```

This command returns a value of `true` if Kubernetes RBAC is enabled. This value determines the command to use in the next step.

2. Install [Azure AD Pod Identity](#) in your AKS cluster.
3. [Create an Identity on Azure](#) following the steps shown in Azure AD Pod Identity project page.
4. [Deploy AzureIdentity](#) following the steps shown in Azure AD Pod Identity project page.
5. [Deploy AzureIdentityBinding](#) following the steps shown in Azure AD Pod Identity project page.
6. If the Azure Identity created in the previous step is not in the same node resource group for your AKS cluster, follow the [Role Assignment](#) steps shown in Azure AD Pod Identity project page.

Assign Azure Identity to web service

The following steps use the Azure Identity created in the previous section, and assign it to your AKS web service through a **selector label**.

First, identify the name and namespace of your deployment in your AKS cluster that you want to assign the Azure Identity. You can get this information by running the following command. The namespaces should be your Azure Machine Learning workspace name and your deployment name should be your endpoint name as shown in the portal.

```
kubectl get deployment --selector=isazuremlapp=true --all-namespaces --show-labels
```

Add the Azure Identity selector label to your deployment by editing the deployment spec. The selector value should be the one that you defined in step 5 of [Deploy AzureIdentityBinding](#).

```
apiVersion: "aadpodidentity.k8s.io/v1"
kind: AzureIdentityBinding
metadata:
  name: demo1-azure-identity-binding
spec:
  AzureIdentity: <a-idname>
  Selector: <label value to match>
```

Edit the deployment to add the Azure Identity selector label. Go to the following section under `/spec/template/metadata/labels`. You should see values such as `isazuremlapp: "true"`. Add the aad-pod-identity label like shown below.

```
kubectl edit deployment/<name of deployment> -n azureml-<name of workspace>
```

```
spec:
  template:
    metadata:
      labels:
        aadpodidbinding: "<value of Selector in AzureIdentityBinding>"
    ...
```

To verify that the label was correctly added, run the following command. You should also see the statuses of the newly created pods.

```
kubectl get pod -n azureml-<name of workspace> --show-labels
```

Once the pods are up and running, the web services for this deployment will now be able to access Azure resources through your Azure Identity without having to embed the credentials in your code.

Assign roles to your Azure Identity

[Assign your Azure Managed Identity with appropriate roles](#) to access other Azure resources. Ensure that the roles you are assigning have the correct [Data Actions](#). For example, the [Storage Blob Data Reader Role](#) will have read permissions to your Storage Blob while the generic [Reader Role](#) might not.

Use Azure Identity with your web service

Deploy a model to your AKS cluster. The `score.py` script can contain operations pointing to the Azure resources that your Azure Identity has access to. Ensure that you have installed your required client library dependencies for the resource that you are trying to access to. Below are a couple examples of how you can use your Azure Identity to access different Azure resources from your service.

Access Key Vault from your web service

If you have given your Azure Identity read access to a secret inside a [Key Vault](#), your `score.py` can access it using the following code.

```
from azure.identity import DefaultAzureCredential
from azure.keyvault.secrets import SecretClient

my_vault_name = "yourkeyvaultname"
my_vault_url = "https://{}.vault.azure.net/".format(my_vault_name)
my_secret_name = "sample-secret"

# This will use your Azure Managed Identity
credential = DefaultAzureCredential()
secret_client = SecretClient(
    vault_url=my_vault_url,
    credential=credential)
secret = secret_client.get_secret(my_secret_name)
```

IMPORTANT

This example uses the DefaultAzureCredential. To grant your identity access using a specific access policy, see [Assign a Key Vault access policy using the Azure CLI](#).

Access Blob from your web service

If you have given your Azure Identity read access to data inside a **Storage Blob**, your `score.py` can access it using the following code.

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

my_storage_account_name = "yourstorageaccountname"
my_storage_account_url = "https://{}.blob.core.windows.net/".format(my_storage_account_name)

# This will use your Azure Managed Identity
credential = DefaultAzureCredential()
blob_service_client = BlobServiceClient(
    account_url=my_storage_account_url,
    credential=credential
)
blob_client = blob_service_client.get_blob_client(container="some-container", blob="some_text.txt")
blob_data = blob_client.download_blob()
blob_data.readall()
```

Next steps

- For more information on how to use the Python Azure Identity client library, see the [repository](#) on GitHub.
- For a detailed guide on deploying models to Azure Kubernetes Service clusters, see the [how-to](#).

Virtual network isolation and privacy overview

12/23/2020 • 6 minutes to read • [Edit Online](#)

In this article, you learn how to use virtual networks (VNets) to secure network communication in Azure Machine Learning. This article uses an example scenario to show you how to configure a complete virtual network.

This article is part one of a five-part series that walks you through securing an Azure Machine Learning workflow. We highly recommend that you read through this overview article to understand the concepts first.

Here are the other articles in this series:

[1. VNet overview](#) > [2. Secure the workspace](#) > [3. Secure the training environment](#) > [4. Secure the inferencing environment](#) > [5. Enable studio functionality](#)

Prerequisites

This article assumes that you have familiarity with the following topics:

- [Azure Virtual Networks](#)
- [IP networking](#)
- [Azure Private Link](#)
- [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 table below compares how services access different parts of an Azure Machine Learning network both with a VNet 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
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 from your VNet to connect to Private Link on the workspace. The private endpoint connects the workspace to the vnet through several private IP addresses.
- **Associated resource** - Use service endpoints or private endpoints to connect to workspace resources like Azure storage, Azure Key Vault, and Azure Container Services.
 - **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 securely with private IP addresses.
- **Inferencing compute access** - Access Azure Kubernetes Services (AKS) compute clusters with private IP addresses.

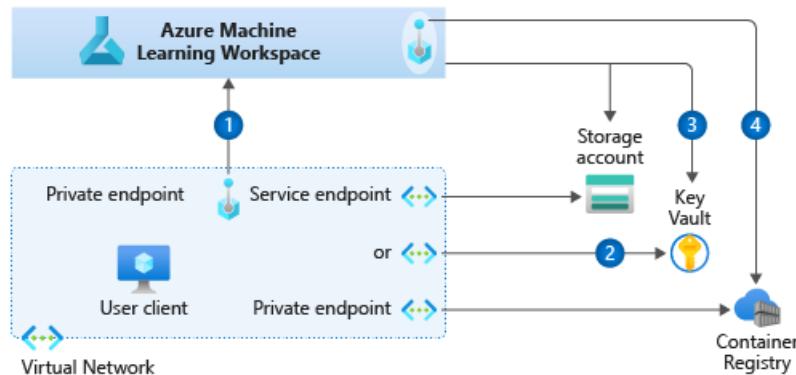
The next five sections show you how to secure the network scenario described above. To secure your network, you must:

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](#)

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.

1. Create a [Private Link-enabled workspace](#) to enable communication between your VNet and workspace.
2. Add Azure Key Vault to the virtual network with a [service endpoint](#) or a [private endpoint](#). Set Key Vault to "Allow trusted Microsoft services to bypass this firewall".
3. Add your Azure storage account to the virtual network with a [service endpoint](#) or a [private endpoint](#).
4. [Configure Azure Container Registry](#) to use a [private endpoint](#) and [enable subnet delegation](#) in Azure Container Instances.



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:

- Using an Azure Machine Learning workspace with private link is not available in the Azure Government or Azure China 21Vianet regions.
- All resources must be behind the same VNet. However, subnets within the same VNet are allowed.

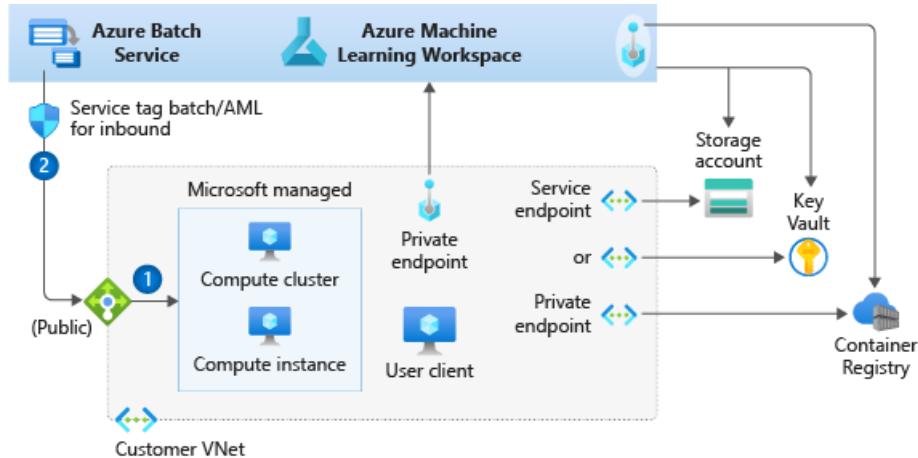
Secure the training environment

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. [Allow inbound communication from Azure Batch Service](#) so that Batch Service can submit jobs to your compute resources.

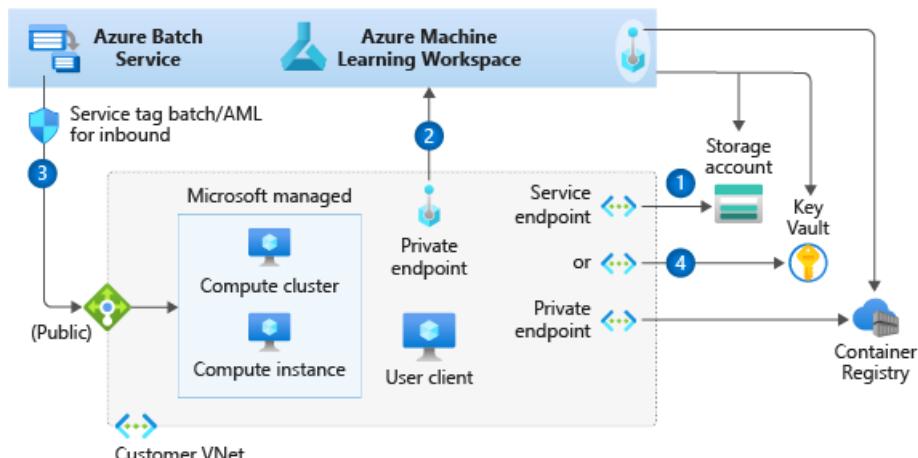


For detailed instructions on how to complete these steps, see [Secure a training environment](#).

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 services receives the job from the workspace and submits the training job to the compute environment through the public load balancer that's provisioned with the compute resource.
4. The compute resource receive the job and begins training. The compute resources accesses secure 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

In this section, you learn the options available for securing an inferencing environment. We recommend that you use Azure Kubernetes Services (AKS) clusters for high-scale, production deployments.

You have two options for AKS clusters in a virtual network:

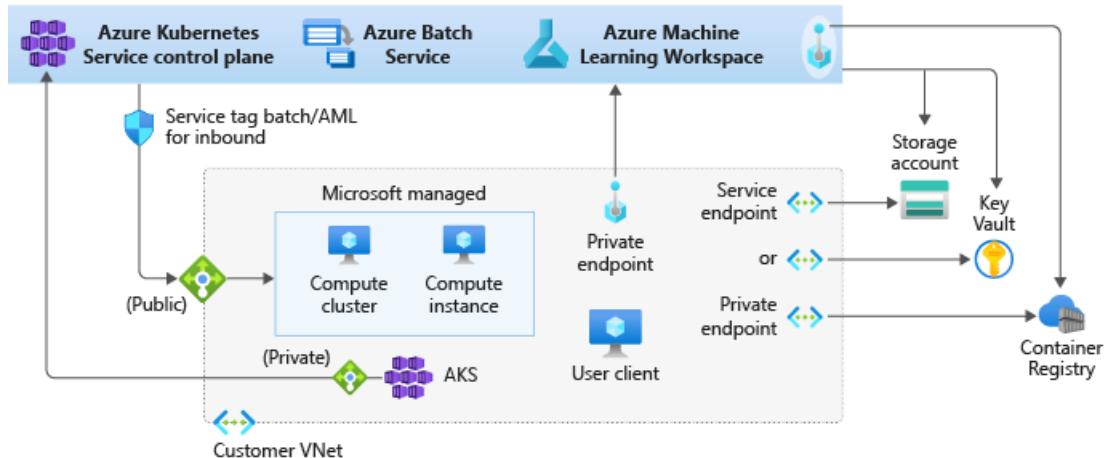
- Deploy or attach a default AKS cluster to your VNet.
- Attach a private AKS cluster to your VNet.

Default AKS clusters have a control plane with public IP addresses. You can add a default AKS cluster to your VNet during the deployment or attach a cluster after it's created.

Private AKS clusters have a control plane, which can only be accessed through private IPs. Private AKS clusters must be attached after the cluster is created.

For detailed instructions on how to add default and private clusters, see [Secure an inferencing environment](#).

The following network diagram shows a secured Azure Machine Learning workspace with a private AKS cluster attached to the virtual network.



Limitations

- AKS clusters must belong to the same VNet as the workspace and its associated resources.

Optional: enable studio functionality

[Secure the workspace](#) > [Secure the training environment](#) > [Secure the inferencing environment](#) > [Enable studio functionality](#) > [Configure firewall settings](#)

If your storage is in a VNet, you first must perform additional configuration steps to enable full functionality in [the studio](#). By default, the following feature 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 while inside of a VNet, see [Use Azure Machine Learning studio in a virtual network](#). The studio supports storage accounts using either service endpoints or private endpoints.

Limitations

- [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 labeling. Note, the non-default storage

account can be secured behind the virtual network.

Configure firewall settings

Configure your firewall to control access to your Azure Machine Learning workspace resources and the public internet. While we recommend Azure Firewall, you should be able to use other firewall products to secure your network. If you have questions about how to allow communication through your firewall, please consult the documentation for the firewall you are using.

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](#).

Next steps

This article is part one of a four-part virtual network series. See the rest of the articles to learn how to secure a virtual network:

- [Part 2: Virtual network overview](#)
- [Part 3: Secure the training environment](#)
- [Part 4: Secure the inferencing environment](#)
- [Part 5: Enable studio functionality](#)

Secure an Azure Machine Learning workspace with virtual networks

12/23/2020 • 8 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.

This article is part two of a five-part series that walks you through securing an Azure Machine Learning workflow. We highly recommend that you read through [Part one: VNet overview](#) to understand the overall architecture first.

See the other articles in this series:

[1. VNet overview](#) > [2. Secure the workspace](#) > [3. Secure the training environment](#) > [4. Secure the inferencing environment](#) > [5. Enable studio functionality](#)

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.
- 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/join/action" on the virtual network resource.
 - "Microsoft.Network/virtualNetworks/subnet/join/action" on the subnet resource.

For more information on Azure RBAC with networking, see the [Networking built-in roles](#)

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. Private Link helps reduce the risk of data exfiltration.

For more information on setting up a Private Link workspace, see [How to configure Private Link](#).

Secure Azure storage accounts with service endpoints

Azure Machine Learning supports storage accounts configured to use either service endpoints or private endpoints. In this section, you learn how to secure an Azure storage account using service endpoints. For private endpoints, see the next section.

IMPORTANT

You can place both the *default storage account* for Azure Machine Learning, or *non-default storage accounts* in a virtual network.

The default storage account is automatically provisioned when you create a workspace.

For non-default storage accounts, the `storage_account` parameter in the `Workspace.create()` function allows you to specify a custom storage account by Azure resource ID.

To use an Azure storage account for the workspace in a virtual network, use the following steps:

1. In the Azure portal, go to the storage service you want to use in your workspace.

The screenshot shows the Azure portal interface for a workspace named 'AMLSERVICEWS'. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main content area shows basic details: Resource group 'AMLSERVICERG', Location 'East US', and Subscription 'Aashish's Subscription'. To the right, under the 'Storage' section, the resource ID 'amlservicews4464378632' is highlighted with a red box. Other listed items include Registry ('amlservicews9093350076'), Key Vault ('amlservicews9171356082'), and Application Insights ('amlservicews3106698991').

2. On the storage service account page, select **Firewalls and virtual networks**.

The screenshot shows the Azure portal interface for a storage account named 'amlservicews4464378632'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Storage Explorer (preview), Settings (with sub-links for Access keys, Geo-replication, CORS, Configuration, Encryption, Shared access signature), and Firewalls and virtual networks. The 'Firewalls and virtual networks' link is highlighted with a red box. The main content area displays general account details: Resource group (change) to 'AMLSERVICERG', Status as Primary, Location as East US, and Subscription (change) to 'Aashish's Subscription'. It also shows sections for Services (Blobs and Queues) and a 'Tags (change)' section.

3. On the **Firewalls and virtual networks** page, do the following actions:

- Select **Selected networks**.
- Under **Virtual networks**, select the **Add existing virtual network** link. This action adds the virtual network where your compute resides (see step 1).

IMPORTANT

The storage account must be in the same virtual network and subnet as the compute instances or clusters used for training or inference.

- Select the **Allow trusted Microsoft services to access this storage account** check box. This does not give all Azure services access to your storage account.
 - Resources of some services, **registered in your subscription**, can access the storage account **in the same subscription** for select operations. For example, writing logs or creating backups.
 - Resources of some services can be granted explicit access to your storage account by **assigning an Azure role** to its system-assigned managed identity.

For more information, see [Configure Azure Storage firewalls and virtual networks](#).

IMPORTANT

When working with the Azure Machine Learning SDK, your development environment must be able to connect to the Azure Storage Account. When the storage account is inside a virtual network, the firewall must allow access from the development environment's IP address.

To enable access to the storage account, visit the **Firewalls and virtual networks** for the storage account *from a web browser on the development client*. Then use the **Add your client IP address** check box to add the client's IP address to the **ADDRESS RANGE**. You can also use the **ADDRESS RANGE** field to manually enter the IP address of the development environment. Once the IP address for the client has been added, it can access the storage account using the SDK.

amlservicews4464378632 - Firewalls and virtual networks

- Firewalls and virtual networks

i Firewall settings allowing access to storage services will remain in effect for up to a minute after saving updated settings restricting access.

Allow access from All networks Selected networks

Configure network security for your storage accounts. [Learn more](#).

Virtual networks

Secure your storage account with virtual networks. [+ Add existing virtual network](#) [+ Add new virtual network](#)

VIRTUAL NETWORK	SUBNET	ADDRESS RANGE	ENDPOINT STATUS	RESOURCE GROUP	SUBSCRIPTION
AMLVNetDemo	1			AMLServicRG	Aashish's Subscription
	default	172.32.0.0/16	✓ Enabled	AMLServicRG	Aashish's Subscription

Firewall

Add IP ranges to allow access from the internet or your on-premises networks. [Learn more](#).

Add your client IP address ('131.107.159.110') i

ADDRESS RANGE

Exceptions

Allow trusted Microsoft services to access this storage account i

Allow read access to storage logging from any network

Allow read access to storage metrics from any network

Secure Azure storage accounts with private endpoints

Azure Machine Learning supports storage accounts configured to use either service endpoints or private endpoints. If the storage account uses private endpoints, you must configure two private endpoints for your

default storage account:

1. A private endpoint with a **blob** target sub-resource.
2. A private endpoint with a **file** target sub-resource (fileshare).

Create a private endpoint

Basics **Resource** Configuration Tags 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](#)

Connection method i

Connect to an Azure resource in my directory.
 Connect to an Azure resource by resource ID or alias.

Subscription * i

Microsoft.Storage/storageAccounts

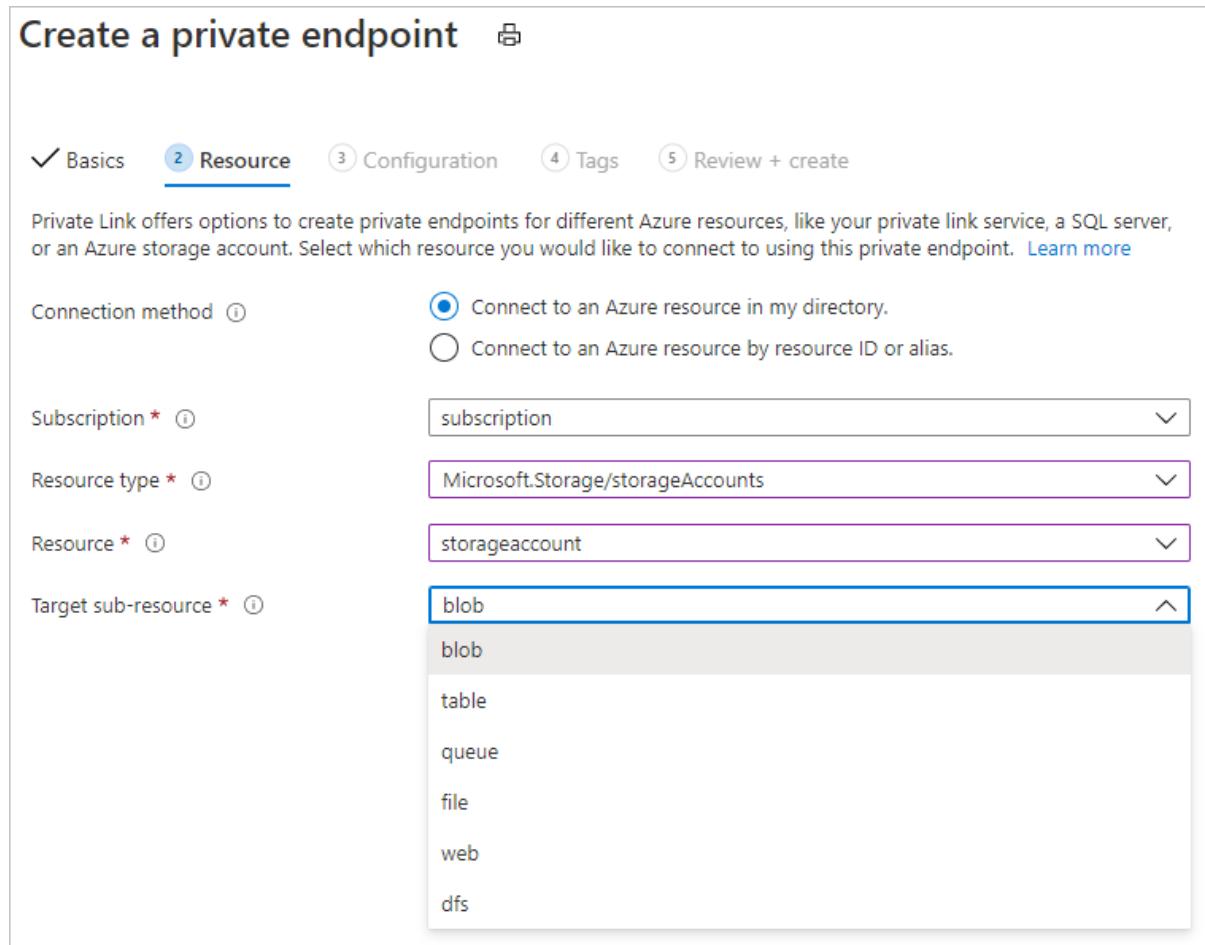
Resource type * i

storageaccount

Target sub-resource * i

blob

blob
table
queue
file
web
dfs



To configure a private endpoint for a storage account that is **not** the default storage, select the **Target sub-resource** type that corresponds to the storage account you want to add.

For more information, see [Use private endpoints for Azure Storage](#)

Secure datastores and datasets

In this section, you learn how to use datastore and datasets in the SDK experience with a virtual network. For more information on the studio experience, see [Use Azure Machine Learning studio in a virtual network](#).

To access data using the SDK, you must use the authentication method required by the individual service that the data is stored in. For example, if you register a datastore to access Azure Data Lake Store Gen2, you must still use a service principal as documented in [Connect to Azure storage services](#).

Disable data validation

By default, Azure Machine Learning performs data validity and credential checks when you attempt to access data using the SDK. If the data is behind a virtual network, Azure Machine Learning can't complete these checks. To avoid this, you must create datastores and datasets that skip validation.

Use datastores

Azure Data Lake Store Gen1 and Azure Data Lake Store Gen2 skip validation by default, so no further action is necessary. However, for the following services you can use similar syntax to skip datastore validation:

- Azure Blob storage
- Azure fileshare

- PostgreSQL
- Azure SQL Database

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)
```

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

To use Azure Machine Learning experimentation capabilities with Azure Key Vault behind a virtual network, use the following steps:

1. Go to the Key Vault that's associated with the workspace.
2. On the **Key Vault** page, in the left pane, select **Networking**.
3. On the **Firewalls and virtual networks** tab, do the following actions:
 - a. Under **Allow access from**, select **Private endpoint and selected networks**.
 - b. Under **Virtual networks**, select **Add existing virtual networks** to add the virtual network where your experimentation compute resides.
 - c. Under **Allow trusted Microsoft services to bypass this firewall?**, select **Yes**.

amlservicews9171356082 - Firewalls and virtual networks

Key vault

Save Discard

Allow access from:

All networks Selected networks

Configure network access control for your key vault.

Virtual networks:

Secure your key vault with virtual networks. [+ Add existing virtual networks](#) [+ Add](#)

VIRTUAL NETWORK	SUBNET	RESOURCE GROUP	SUBSCRIPTION
amlvnetdemo	default	amlservicerg	Aashish's Subscription

Firewall:

Add IPv4 ranges to allow access from the internet or your on-premises networks.

IPv4 ADDRESS OR CIDR

IPv4 address or CIDR ...

Exception:

Allow trusted Microsoft services to bypass this firewall? [?](#)

Yes No

Enable Azure Container Registry (ACR)

To use Azure Container Registry inside a virtual network, you must meet the following requirements:

- Your Azure Container Registry must be Premium version. For more information on upgrading, see [Changing SKUs](#).
- Your Azure Container Registry must be in the same virtual network and subnet as the storage account and compute targets used for training or inference.
- Your Azure Machine Learning workspace must contain an [Azure Machine Learning compute cluster](#).

When ACR is behind a virtual network, Azure Machine Learning cannot use it to directly build Docker images. Instead, the compute cluster is used to build the images.

- Before using ACR with Azure Machine Learning in a virtual network, you must open a support incident to enable this functionality. For more information, see [Manage and increase quotas](#).

Once those requirements are fulfilled, use the following steps to enable Azure Container Registry.

- Find the name of the Azure Container Registry for your workspace, using one of the following methods:

Azure portal

From the overview section of your workspace, the **Registry** value links to the Azure Container Registry.

Search (Ctrl+ /) Download config.json Delete

Overview

Activity log

Access control (IAM)

Workspace edition : Enterprise

Resource group : amlpl

Location : South Central US

Subscription : Visual Studio Ultimate with MSDN

Storage : sarxekmvapdhjno

Registry : amplclfa51398

Key Vault : kvrzekmvapdhjno

Application Insights : airxekmvapdhjno

Azure CLI

If you have [installed the Machine Learning extension for Azure CLI](#), you can use the `az ml workspace show` command to show the workspace information.

```
az ml workspace show -w yourworkspacename -g resourcegroupname --query 'containerRegistry'
```

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 [Configure network access for registry](#). When adding the virtual network, select the virtual network and subnet for your Azure Machine Learning resources.
- 3. Use the Azure Machine Learning Python SDK to configure a compute cluster to build docker images. The following code snippet demonstrates how to do this:

```
from azureml.core import Workspace
# Load workspace from an existing config file
ws = Workspace.from_config()
# Update the workspace to use an existing compute cluster
ws.update(image_build_compute = 'mycomputecluster')
```

IMPORTANT

Your storage account, compute cluster, and Azure Container Registry must all be in the same subnet of the virtual network.

For more information, see the [update\(\)](#) method reference.

- 4. Apply the following Azure Resource Manager template. This template enables your workspace to communicate with ACR.

```
{
"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
    "keyVaultArmId": {
        "type": "string"
    },
    "workspaceName": {
        "type": "string"
    },
    "containerRegistryArmId": {
        "type": "string"
    },
    "applicationInsightsArmId": {
        "type": "string"
    },
    "storageAccountArmId": {
        "type": "string"
    },
    "location": {
        "type": "string"
    }
},
"resources": [
    {
        "type": "Microsoft.MachineLearningServices/workspaces",
        "apiVersion": "2019-11-01",
        "name": "[parameters('workspaceName')]",
        "location": "[parameters('location')]",
        "identity": {
            "type": "SystemAssigned"
        },
        "sku": {
            "tier": "basic",
            "name": "basic"
        },
        "properties": {
            "sharedPrivateLinkResources": [
                {"Name": "Acr", "Properties": {"PrivateLinkResourceId": "[concat(parameters('containerRegistryArmId'), '/privateLinkResources/registry')]", "GroupId": "registry", "RequestMessage": "Approve", "Status": "Pending"}},
                {"keyVault": "[parameters('keyVaultArmId')]",
                 "containerRegistry": "[parameters('containerRegistryArmId')]",
                 "applicationInsights": "[parameters('applicationInsightsArmId')]",
                 "storageAccount": "[parameters('storageAccountArmId')]"
                }
            ]
        }
    }
]
```

This template creates a *private endpoint* for network access from the workspace to your ACR. The screenshot below shows an example of this private endpoint.

The screenshot shows the 'Networking' section of the Azure Container Registry for the 'mlacr' repository. The 'Private endpoint' tab is selected. A table lists two private endpoints:

Connection name	Connection state	Private endpoint	Description
mlacr.05c63e65ea9744159ab84dd30d396...	Approved	mlacr_17uUPh7TY4iQnGhvYpaLGv9kcZS...	Please Approve request
mlacr.c15af491ba304c698c7d906ff564596	Approved	mlacr-pe	Auto-Approved

Below the table, there are sections for 'Data endpoints' and 'Replications'. A yellow box highlights the 'IMPORTANT' note at the bottom.

IMPORTANT
Do not delete this endpoint! If you accidentally delete it, you can re-apply the template in this step to create a new one.

Next steps

This article is part one of a four-part virtual network series. See the rest of the articles to learn how to secure a virtual network:

- [Part 1: Virtual network overview](#)
- [Part 3: Secure the training environment](#)
- [Part 4: Secure the inferencing environment](#)
- [Part 5: Enable studio functionality](#)

Secure an Azure Machine Learning training environment with virtual networks

12/23/2020 • 11 minutes to read • [Edit Online](#)

In this article, you learn how to secure training environments with a virtual network in Azure Machine Learning.

This article is part three of a five-part series that walks you through securing an Azure Machine Learning workflow. We highly recommend that you read through [Part one: VNet overview](#) to understand the overall architecture first.

See the other articles in this series:

[1. VNet overview](#) > [Secure the workspace](#) > [3. Secure the training environment](#) > [4. Secure the inferencing environment](#) > [5. Enable studio functionality](#)

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/join/action" on the virtual network resource.
 - "Microsoft.Network/virtualNetworks/subnet/join/action" on the subnet resource.

For more information on Azure RBAC with networking, see the [Networking built-in roles](#)

Compute clusters & instances

To use either a [managed Azure Machine Learning compute target](#) or an [Azure Machine Learning compute instance](#) in a virtual network, the following network requirements must be met:

- The virtual network must be in the same subscription and region as the Azure Machine Learning workspace.
- The subnet that's specified for the compute instance or cluster must have enough unassigned IP addresses to accommodate the number of VMs that are targeted. If the subnet doesn't have enough unassigned IP addresses, a compute cluster will be partially allocated.
- Check to see whether your security policies or locks on the virtual network's subscription or resource group restrict permissions to manage the virtual network. If you plan to secure the virtual network by restricting traffic, leave some ports open for the compute service. For more information, see the [Required ports](#) section.
- If you're going to put multiple compute instances or clusters in one virtual network, you might need to request a quota increase for one or more of your resources.

- 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 Machine Learning compute instance or cluster.
- 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.
- When compute instance is deployed in a private link workspace it can be only be accessed from within virtual network. If you are using custom DNS or hosts file please add an entry for `<instance-name>.<region>.instances.azureml.ms` with private IP address of workspace private endpoint. For more information see the [custom DNS](#) article.

TIP

The Machine Learning compute instance or cluster automatically allocates additional 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
- One public IP address
- One load balancer

In the case of clusters these resources are deleted (and recreated) every time the cluster scales down to 0 nodes, however for an instance the resources are held onto till the instance is completely deleted (stopping does not remove the resources). These resources are limited by the subscription's [resource quotas](#).

Required ports

If you plan on securing the virtual network by restricting network traffic to/from the public internet, you must allow inbound communications from the Azure Batch service.

The Batch service adds network security groups (NSGs) at the level of network interfaces (NICs) that are attached to VMs. These NSGs automatically configure inbound and outbound rules to allow the following traffic:

- Inbound TCP traffic on ports 29876 and 29877 from a **Service Tag of BatchNodeManagement**.

Add inbound security rule

dsvm-vnet-nsg

Basic

* Source **Service Tag**: BatchNodeManagement

* Source service tag **BatchNodeManagement**

* Source port ranges *****

* Destination **Any**

* Destination port ranges **29876-29877**

* Protocol **Any**

* Action **Allow**

* Priority **1040**

* Name **Port_29876-29877**

Description

Add

The screenshot shows the 'Add inbound security rule' dialog box. The 'Source' section is highlighted with a red box, containing 'Service Tag' and 'BatchNodeManagement'. The 'Destination' section is also highlighted with a red box, containing 'Any' and '29876-29877'. A green checkmark is present next to '29876-29877'. The 'Protocol' section shows 'Any' selected. The 'Action' section shows 'Allow' selected. The 'Priority' section shows '1040'. The 'Name' section shows 'Port_29876-29877'. The 'Description' section is empty. At the bottom, there is a blue 'Add' button.

- (Optional) Inbound TCP traffic on port 22 to permit remote access. Use this port only if you want to connect by using SSH on the public IP.
- Outbound traffic on any port to the virtual network.
- Outbound traffic on any port to the internet.
- For compute instance inbound TCP traffic on port 44224 from a **Service Tag** of **AzureMachineLearning**.

IMPORTANT

Exercise caution if you modify or add inbound or outbound rules in Batch-configured NSGs. If an NSG blocks communication to the compute nodes, the compute service sets the state of the compute nodes to unusable.

You don't need to specify NSGs at the subnet level, because the Azure Batch service configures its own NSGs. However, if the subnet that contains the Azure Machine Learning compute has associated NSGs or a firewall, you must also allow the traffic listed earlier.

The NSG rule configuration in the Azure portal is shown in the following images:

dsvm-nsg | Inbound security rules

Network security group

Search (Ctrl+ /) Add Default rules Refresh

Priority	Name	Port	Protocol	Source	Destination	Action
1040	AzureBatch	29876-29877	TCP	BatchNodeManagem...	Any	Allow
1050	AzureMachineLearning	44224	TCP	AzureMachineLearning	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Settings
Inbound security rules
Outbound security rules

Outbound security rules

Add Default rules

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Limit outbound connectivity from the virtual network

If you don't want to use the default outbound rules and you do want to limit the outbound access of your virtual network, use the following steps:

- Deny outbound internet connection by using the NSG rules.
- For a **compute instance** or a **compute cluster**, limit outbound traffic to the following items:
 - Azure Storage, by using **Service Tag** of **Storage.RegionName**. Where `{RegionName}` is the name of an Azure region.
 - Azure Container Registry, by using **Service Tag** of **AzureContainerRegistry.RegionName**. Where `{RegionName}` is the name of an Azure region.
 - Azure Machine Learning, by using **Service Tag** of **AzureMachineLearning**
 - Azure Resource Manager, by using **Service Tag** of **AzureResourceManager**
 - Azure Active Directory, by using **Service Tag** of **AzureActiveDirectory**

The NSG rule configuration in the Azure portal is shown in the following image:

dsvm-nsg | Outbound security rules

Network security group

Search (Ctrl+ /) Add Default rules Refresh

Priority	Name	Port	Protocol	Source	Destination	Action
3700	AAD	Any	Any	Any	AzureActiveDirectory	Allow
3800	ARM	Any	Any	Any	AzureResourceManager	Allow
3850	AML	Any	Any	Any	AzureMachineLearning	Allow
3900	ACR	Any	Any	Any	AzureContainerRegistry.UKSouth	Allow
3950	Storage	Any	Any	Any	Storage.UKSouth	Allow
4000	DenyInternet	Any	Any	Any	Internet	Deny
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Settings
Outbound security rules
Network interfaces
Subnets
Properties

NOTE

If you plan on using default Docker images provided by Microsoft, and enabling user managed dependencies, you must also use the following [Service Tags](#):

- `MicrosoftContainerRegistry`
- `AzureFrontDoor.FirstParty`

This configuration is needed when you have code similar to the following snippets as part of your training scripts:

RunConfig training

```
# create a new runconfig object
run_config = RunConfiguration()

# configure Docker
run_config.environment.docker.enabled = True
# For GPU, use DEFAULT_GPU_IMAGE
run_config.environment.docker.base_image = DEFAULT_CPU_IMAGE
run_config.environment.python.user_managed_dependencies = True
```

Estimator training

```
est = Estimator(source_directory='.',
                 script_params=script_params,
                 compute_target='local',
                 entry_script='dummy_train.py',
                 user_managed=True)
run = exp.submit(est)
```

Forced tunneling

If you're using [forced tunneling](#) with Azure Machine Learning compute, you must allow communication with the public internet from the subnet that contains the compute resource. This communication is used for task scheduling and accessing Azure Storage.

There are two ways that you can accomplish this:

- Use a [Virtual Network NAT](#). A NAT gateway provides outbound internet connectivity for one or more subnets in your virtual network. For information, see [Designing virtual networks with NAT gateway resources](#).
- Add [user-defined routes \(UDRs\)](#) to the subnet that contains the compute resource. Establish a UDR for each IP address that's used by the Azure Batch service in the region where your resources exist. These UDRs enable the Batch service to communicate with compute nodes for task scheduling. Also add the IP address for the Azure Machine Learning service where the resources exist, as this is required for access to Compute Instances. To get a list of IP addresses of the Batch service and Azure Machine Learning service, use one of the following methods:
 - Download the [Azure IP Ranges and Service Tags](#) and search the file for `BatchNodeManagement.<region>` and `AzureMachineLearning.<region>`, where `<region>` is your Azure region.
 - Use the [Azure CLI](#) to download the information. The following example downloads the IP address information and filters out the information for the East US 2 region:

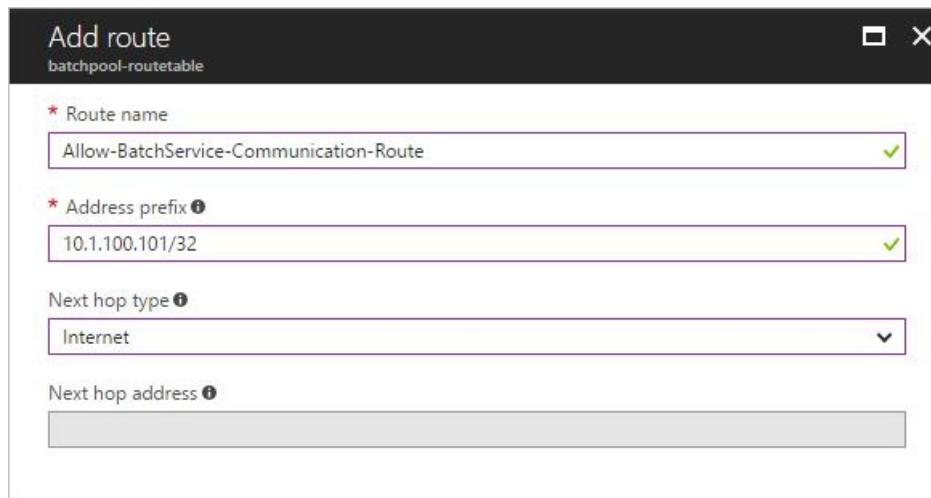
```
az network list-service-tags -l "East US 2" --query "values[?starts_with(id, 'Batch')] | [?properties.region=='eastus2']"  
az network list-service-tags -l "East US 2" --query "values[?starts_with(id, 'AzureMachineLearning')] | [?properties.region=='eastus2']"
```

TIP

If you are using the US-Virginia, US-Arizona regions, or China-East-2 regions, these commands return no IP addresses. Instead, use one of the following links to download a list of IP addresses:

- [Azure IP ranges and service tags for Azure Government](#)
- [Azure IP ranges and service tags for Azure China](#)

When you add the UDRs, define the route for each related Batch IP address prefix and set **Next hop type** to **Internet**. The following image shows an example of this UDR in the Azure portal:



IMPORTANT

The IP addresses may change over time.

In addition to any UDRs that you define, outbound traffic to Azure Storage must be allowed through your on-premises network appliance. Specifically, the URLs for this traffic are in the following forms:

<account>.table.core.windows.net , <account>.queue.core.windows.net , and
<account>.blob.core.windows.net .

For more information, see [Create an Azure Batch pool in a virtual network](#).

Create a compute cluster in a virtual network

To create a Machine Learning Compute cluster, use the following steps:

1. Sign in to [Azure Machine Learning studio](#), and then select your subscription and workspace.
2. Select **Compute** on the left.
3. Select **Training clusters** from the center, and then select **+**.
4. In the **New Training Cluster** dialog, expand the **Advanced settings** section.
5. To configure this compute resource to use a virtual network, perform the following actions in the **Configure virtual network** section:
 - a. In the **Resource group** drop-down list, select the resource group that contains the virtual network.

- b. In the **Virtual network** drop-down list, select the virtual network that contains the subnet.
 c. In the **Subnet** drop-down list, select the subnet to use.

New Training Cluster

(i) Customers should not include personal data or other sensitive information in fields marked with  because the content in these fields may be logged and shared across Microsoft systems to facilitate operations and troubleshooting. [Learn more](#)

Compute name * 

Compute name is required.

(i) Machine Learning Compute is a managed training environment consisting of one or more nodes. [Learn more](#).

Region * 

*

Virtual Machine size * 

*

Virtual Machine priority * 

Dedicated **Low Priority**

Minimum number of nodes * 

*

Maximum number of nodes * 

*

Idle seconds before scale down * 

*

Advanced settings

Configure virtual network 

Resource group



Virtual network



Subnet



(i) [Learn more about how to enable virtual network for training cluster](#)

You can also create a Machine Learning Compute cluster by using the Azure Machine Learning SDK. The following code creates a new Machine Learning Compute cluster in the `default` subnet of a virtual network named `mynetwork`:

```

from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

# The Azure virtual network name, subnet, and resource group
vnet_name = 'mynetwork'
subnet_name = 'default'
vnet_resourcegroup_name = 'mygroup'

# Choose a name for your CPU cluster
cpu_cluster_name = "cpucluster"

# Verify that cluster does not exist already
try:
    cpu_cluster = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print("Found existing cpucluster")
except ComputeTargetException:
    print("Creating new cpucluster")

# Specify the configuration for the new cluster
compute_config = AmlCompute.provisioning_configuration(vm_size="STANDARD_D2_V2",
                                                       min_nodes=0,
                                                       max_nodes=4,
                                                       vnet_resourcegroup_name=vnet_resourcegroup_name,
                                                       vnet_name=vnet_name,
                                                       subnet_name=subnet_name)

# Create the cluster with the specified name and configuration
cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

# Wait for the cluster to be completed, show the output log
cpu_cluster.wait_for_completion(show_output=True)

```

When the creation process finishes, you train your model by using the cluster in an experiment. For more information, see [Select and use a compute target for training](#).

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](#).

Access data in a Compute Instance notebook

If you're using notebooks on an Azure Compute instance, you must ensure that your notebook is running on a compute resource behind the same virtual network and subnet as your data.

You must configure your Compute Instance to be in the same virtual network during creation under **Advanced settings > Configure virtual network**. You cannot add an existing Compute Instance to a virtual network.

Azure Databricks

To use Azure Databricks in a virtual network with your workspace, the following requirements must be met:

- 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.
- 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.

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

IMPORTANT

Azure Machine Learning supports only virtual machines that are running Ubuntu.

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
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 [Limit outbound connectivity from the virtual network](#) section.

Attach the VM or HDInsight cluster

Attach the VM or HDInsight cluster to your Azure Machine Learning workspace. For more information, see [Set up compute targets for model training](#).

Next steps

This article is part three in a four-part virtual network series. See the rest of the articles to learn how to secure a virtual network:

- Part 1: Virtual network overview
- Part 2: Secure the workspace resources
- Part 4: Secure the inferencing environment
- Part 5: Enable studio functionality

Secure an Azure Machine Learning inferencing environment with virtual networks

12/23/2020 • 9 minutes to read • [Edit Online](#)

In this article, you learn how to secure inferencing environments with a virtual network in Azure Machine Learning.

This article is part four of a five-part series that walks you through securing an Azure Machine Learning workflow. We highly recommend that you read through [Part one: VNet overview](#) to understand the overall architecture first.

See the other articles in this series:

[1. VNet overview](#) > [Secure the workspace](#) > [3. Secure the training environment](#) > **4. Secure the inferencing environment** > [5. Enable studio functionality](#)

In this article you learn how to secure the following inferencing resources in a virtual network:

- Default Azure Kubernetes Service (AKS) cluster
- Private AKS cluster
- AKS cluster with private link
- Azure Container Instances (ACI)

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/join/action" on the virtual network resource.
 - "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 Kubernetes Service

To use an AKS cluster in a virtual network, the following network requirements must be met:

- Follow the prerequisites in [Configure advanced networking in Azure Kubernetes Service \(AKS\)](#).
- The AKS instance and the virtual network must be in the same region. If you secure the Azure Storage Account(s) used by the workspace in a virtual network, they must be in the same virtual network as the AKS instance too.

To add AKS in a virtual network to your workspace, use the following steps:

1. Sign in to [Azure Machine Learning studio](#), and then select your subscription and workspace.
2. Select **Compute** on the left.
3. Select **Inference clusters** from the center, and then select **+**.

4. In the **New Inference Cluster** dialog, select **Advanced** under Network configuration.
5. To configure this compute resource to use a virtual network, perform the following actions:
 - a. In the **Resource group** drop-down list, select the resource group that contains the virtual network.
 - b. In the **Virtual network** drop-down list, select the virtual network that contains the subnet.
 - c. In the **Subnet** drop-down list, select the subnet.
 - d. In the **Kubernetes Service address range** box, enter the Kubernetes service address range. This address range uses a Classless Inter-Domain Routing (CIDR) notation IP range to define the IP addresses that are available for the cluster. It must not overlap with any subnet IP ranges (for example, 10.0.0.0/16).
 - e. In the **Kubernetes DNS service IP address** box, enter the Kubernetes DNS service IP address. This IP address is assigned to the Kubernetes DNS service. It must be within the Kubernetes service address range (for example, 10.0.0.10).
 - f. In the **Docker bridge address** box, enter the Docker bridge address. This IP address is assigned to Docker Bridge. It must not be in any subnet IP ranges, or the Kubernetes service address range (for example, 172.17.0.1/16).

New Inference Cluster

(i) Customers should not include personal data or other sensitive information in fields marked with because the content in these fields may be logged and shared across Microsoft systems to facilitate operations and troubleshooting. [Learn more](#)

Compute name *

Kubernetes Service

[Create new](#) [Use existing](#)

Region *

Pick a region

Virtual Machine size *

Standard_D3_v2

Cluster purpose

Production Dev-test

Number of nodes *

3

Network configuration

[Basic](#) [Advanced](#)

Resource group *

Select or search by name

Virtual network *

Select or search by name

Subnet *

Select or search by name

(i) Learn more about how to enable virtual network for inference cluster

Kubernetes Service address range *

10.0.0.0/16

Kubernetes DNS Service IP address *

10.0.0.10

Docker Bridge address *

172.17.0.1/16

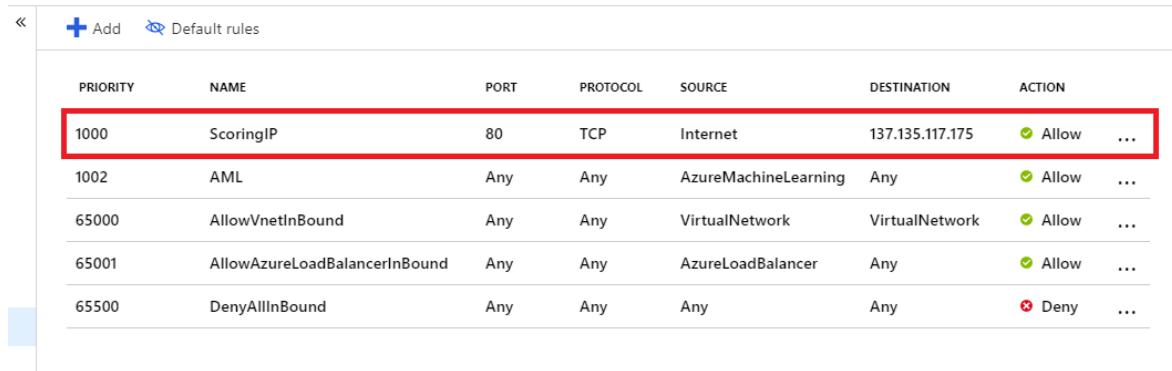
6. When you deploy a model as a web service to AKS, a scoring endpoint is created to handle inferencing requests. Make sure that the NSG group that controls the virtual network has an inbound security rule enabled for the IP address of the scoring endpoint if you want to call it from outside the virtual network.

To find the IP address of the scoring endpoint, look at the scoring URI for the deployed service. For information on viewing the scoring URI, see [Consume a model deployed as a web service](#).

IMPORTANT

Keep the default outbound rules for the NSG. For more information, see the default security rules in [Security groups](#).

- Inbound security rules



PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION	...
1000	ScoringIP	80	TCP	Internet	137.135.117.175	Allow	...
1002	AML	Any	Any	AzureMachineLearning	Any	Allow	...
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow	...
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow	...
65500	DenyAllInBound	Any	Any	Any	Any	Deny	...

IMPORTANT

The IP address shown in the image for the scoring endpoint will be different for your deployments. While the same IP is shared by all deployments to one AKS cluster, each AKS cluster will have a different IP address.

You can also use the Azure Machine Learning SDK to add Azure Kubernetes Service in a virtual network. If you already have an AKS cluster in a virtual network, attach it to the workspace as described in [How to deploy to AKS](#). The following code creates a new AKS instance in the `default` subnet of a virtual network named

```
mynetwork :
```

```
from azureml.core.compute import ComputeTarget, AksCompute

# Create the compute configuration and set virtual network information
config = AksCompute.provisioning_configuration(location="eastus2")
config.vnet_resourcegroup_name = "mygroup"
config.vnet_name = "mynetwork"
config.subnet_name = "default"
config.service_cidr = "10.0.0.0/16"
config.dns_service_ip = "10.0.0.10"
config.docker_bridge_cidr = "172.17.0.1/16"

# Create the compute target
aks_target = ComputeTarget.create(workspace=ws,
                                   name="myaks",
                                   provisioning_configuration=config)
```

When the creation process is completed, you can run inference, or model scoring, on an AKS cluster behind a virtual network. For more information, see [How to deploy to AKS](#).

For more information on using Role-Based Access Control with Kubernetes, see [Use Azure RBAC for Kubernetes authorization](#).

Network contributor role

IMPORTANT

If you create or attach an AKS cluster by providing a virtual network you previously created, you must grant the service principal (SP) or managed identity for your AKS cluster the *Network Contributor* role to the resource group that contains the virtual network.

To add the identity as network contributor, use the following steps:

1. To find the service principal or managed identity ID for AKS, use the following Azure CLI commands.

Replace `<aks-cluster-name>` with the name of the cluster. Replace `<resource-group-name>` with the name of the resource group that *contains the AKS cluster*:

```
az aks show -n <aks-cluster-name> --resource-group <resource-group-name> --query  
servicePrincipalProfile.clientId
```

If this command returns a value of `msi`, use the following command to identify the principal ID for the managed identity:

```
az aks show -n <aks-cluster-name> --resource-group <resource-group-name> --query identity.principalId
```

2. To find the ID of the resource group that contains your virtual network, use the following command.

Replace `<resource-group-name>` with the name of the resource group that *contains the virtual network*:

```
az group show -n <resource-group-name> --query id
```

3. To add the service principal or managed identity as a network contributor, use the following command.

Replace `<SP-or-managed-identity>` with the ID returned for the service principal or managed identity.

Replace `<resource-group-id>` with the ID returned for the resource group that contains the virtual network:

```
az role assignment create --assignee <SP-or-managed-identity> --role 'Network Contributor' --scope  
<resource-group-id>
```

For more information on using the internal load balancer with AKS, see [Use internal load balancer with Azure Kubernetes Service](#).

Secure VNet traffic

There are two approaches to isolate traffic to and from the AKS cluster to the virtual network:

- **Private AKS cluster:** This approach uses Azure Private Link to secure communications with the cluster for deployment/management operations.
- **Internal AKS load balancer:** This approach configures the endpoint for your deployments to AKS to use a private IP within the virtual network.

WARNING

Internal load balancer does not work with an AKS cluster that uses kubenet. If you want to use an internal load balancer and a private AKS cluster at the same time, configure your private AKS cluster with Azure Container Networking Interface (CNI). For more information, see [Configure Azure CNI networking in Azure Kubernetes Service](#).

Private AKS cluster

By default, AKS clusters have a control plane, or API server, with public IP addresses. You can configure AKS to use a private control plane by creating a private AKS cluster. For more information, see [Create a private Azure Kubernetes Service cluster](#).

After you create the private AKS cluster, [attach the cluster to the virtual network](#) to use with Azure Machine Learning.

IMPORTANT

Before using a private link enabled AKS cluster with Azure Machine Learning, you must open a support incident to enable this functionality. For more information, see [Manage and increase quotas](#).

Internal AKS load balancer

By default, AKS deployments use a [public load balancer](#). In this section, you learn how to configure AKS to use an internal load balancer. An internal (or private) load balancer is used where only private IPs are allowed as frontend. Internal load balancers are used to load balance traffic inside a virtual network

A private load balancer is enabled by configuring AKS to use an *internal load balancer*.

Enable private load balancer

IMPORTANT

You cannot enable private IP when creating the Azure Kubernetes Service cluster in Azure Machine Learning studio. You can create one with an internal load balancer when using the Python SDK or Azure CLI extension for machine learning.

The following examples demonstrate how to [create a new AKS cluster with a private IP/internal load balancer](#) using the SDK and CLI:

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

```

import azureml.core
from azureml.core.compute import AksCompute, ComputeTarget

# Verify that cluster does not exist already
try:
    aks_target = AksCompute(workspace=ws, name=aks_cluster_name)
    print("Found existing aks cluster")

except:
    print("Creating new aks cluster")

    # Subnet to use for AKS
    subnet_name = "default"
    # Create AKS configuration
    prov_config=AksCompute.provisioning_configuration(load_balancer_type="InternalLoadBalancer")
    # Set info for existing virtual network to create the cluster in
    prov_config.vnet_resourcegroup_name = "myvnetresourcegroup"
    prov_config.vnet_name = "myvnetname"
    prov_config.service_cidr = "10.0.0.0/16"
    prov_config.dns_service_ip = "10.0.0.10"
    prov_config.subnet_name = subnet_name
    prov_config.docker_bridge_cidr = "172.17.0.1/16"

    # Create compute target
    aks_target = ComputeTarget.create(workspace = ws, name = "myaks", provisioning_configuration =
prov_config)
    # Wait for the operation to complete
    aks_target.wait_for_completion(show_output = True)

```

When **attaching an existing cluster** to your workspace, you must wait until after the attach operation to configure the load balancer. For information on attaching a cluster, see [Attach an existing AKS cluster](#).

After attaching the existing cluster, you can then update the cluster to use an internal load balancer/private IP:

```

import azureml.core
from azureml.core.compute.aks import AksUpdateConfiguration
from azureml.core.compute import AksCompute

# ws = workspace object. Creation not shown in this snippet
aks_target = AksCompute(ws,"myaks")

# Change to the name of the subnet that contains AKS
subnet_name = "default"
# Update AKS configuration to use an internal load balancer
update_config = AksUpdateConfiguration(None, "InternalLoadBalancer", subnet_name)
aks_target.update(update_config)
# Wait for the operation to complete
aks_target.wait_for_completion(show_output = True)

```

Enable Azure Container Instances (ACI)

Azure Container Instances are dynamically created when deploying a model. To enable Azure Machine Learning to create ACI inside the virtual network, you must enable **subnet delegation** for the subnet used by the deployment.

WARNING

When using Azure Container Instances in a virtual network, the virtual network must be:

- In the same resource group as your Azure Machine Learning workspace.
- If your workspace has a **private endpoint**, the virtual network used for Azure Container Instances must be the same as the one used by the workspace private endpoint.

When using Azure Container Instances inside the virtual network, the Azure Container Registry (ACR) for your workspace cannot also be in the virtual network.

To use ACI in a virtual network to your workspace, use the following steps:

1. To enable subnet delegation on your virtual network, use the information in the [Add or remove a subnet delegation](#) article. You can enable delegation when creating a virtual network, or add it to an existing network.

IMPORTANT

When enabling delegation, use `Microsoft.ContainerInstance/containerGroups` as the **Delegate subnet to service** value.

2. Deploy the model using `AciWebService.deploy_configuration()`, use the `vnet_name` and `subnet_name` parameters. Set these parameters to the virtual network name and subnet where you enabled delegation.

Limit outbound connectivity from the virtual network

If you don't want to use the default outbound rules and you do want to limit the outbound access of your virtual network, you must allow access to Azure Container Registry. For example, make sure that your Network Security Groups (NSG) contains a rule that allows access to the `AzureContainerRegistry.RegionName` service tag where `{RegionName}` is the name of an Azure region.

Next steps

This article is part three in a four-part virtual network series. See the rest of the articles to learn how to secure a virtual network:

- [Part 1: Virtual network overview](#)
- [Part 2: Secure the workspace resources](#)
- [Part 3: Secure the training environment](#)
- [Part 5: Enable studio functionality](#)

Use Azure Machine Learning studio in an Azure virtual network

12/23/2020 • 6 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. In order to use those features in a virtual network, you must follow the steps in this article.

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.

This article is part five of a five-part series that walks you through securing an Azure Machine Learning workflow. We highly recommend that you read through the previous parts to set up a virtual network environment.

See the other articles in this series:

[1. VNet overview](#) > [2. Secure the workspace](#) > [3. Secure the training environment](#) > [4. Secure the inferencing environment](#) > [5. Enable studio functionality](#)

IMPORTANT

If your workspace is in a **sovereign cloud**, such as Azure Government or Azure China 21Vianet, integrated notebooks *do not* support using storage that is in a virtual network. Instead, you can use Jupyter Notebooks from a compute instance. For more information, see the [Access data in a Compute Instance notebook](#) section.

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 Private Link enabled](#).
- An existing [Azure storage account added your virtual network](#).

Configure data access in the studio

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 ([default storage account](#)).
- Submit an AutoML experiment ([default storage account](#)).
- Start a labeling project.

The studio supports reading data from the following datastore types in a virtual network:

- Azure Blob
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure SQL Database

Configure datastores to use workspace-managed identity

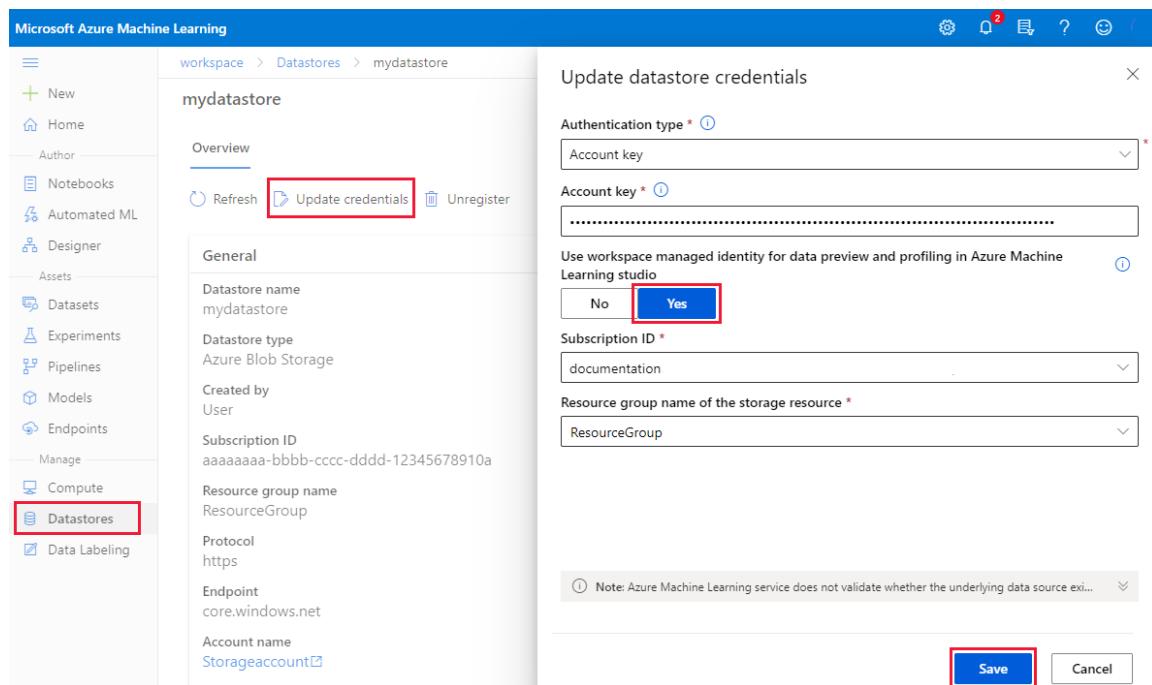
After you add an Azure storage account to your virtual network with either a [service endpoint](#) or [private endpoint](#), you must configure your datastore to use [managed identity](#) authentication. Doing so lets the studio access data in your storage account.

Azure Machine Learning uses [datastores](#) to connect to storage accounts. Use the following steps to configure a datastore to use managed identity:

1. In the studio, select **Datastores**.
2. To update an existing datastore, select the datastore and select **Update credentials**.

To create a new datastore, select **+** **New datastore**.

3. In the datastore settings, select **Yes** for **Use workspace managed identity for data preview and profiling in Azure Machine Learning studio**.



These steps add the workspace-managed identity as a **Reader** to the storage service using Azure RBAC. **Reader** access lets the workspace retrieve firewall settings to ensure that data doesn't leave the virtual network. Changes may take up to 10 minutes to take effect.

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 you must enable managed identity authentication for your workspace default storage accounts.

STORAGE ACCOUNT	NOTES
Workspace default blob storage	<p>Stores model assets from the designer. You must enable managed identity authentication on this storage account to deploy models in the designer.</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	<p>Stores AutoML experiment assets. You must enable managed identity authentication on this storage account to submit AutoML experiments.</p>

WARNING

There's a known issue where the default file store does not automatically create the `azureml-filestore` folder, which is required to submit AutoML experiments. This occurs when users bring an existing filestore to set as the default filestore during workspace creation.

To avoid this issue, you have two options: 1) Use the default filestore which is automatically created for you during workspace creation. 2) To bring your own filestore, make sure the filestore is outside of the VNet during workspace creation. After the workspace is created, add the storage account to the virtual network.

To resolve this issue, remove the filestore account from the virtual network then add it back to the virtual network.

Grant workspace managed identity Reader access to storage private link

If your Azure storage account uses a private endpoint, you must grant the workspace-managed identity **Reader** access to the private link. For more information, see the [Reader](#) built-in role.

If your storage account uses a service endpoint, you can skip this step.

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**.

Technical notes for managed identity

Using managed identity to access storage services impacts security considerations. This section describes the changes for each storage account type.

These considerations are unique to the **type of storage account** you are accessing.

Azure Blob storage

For **Azure Blob storage**, the workspace-managed identity is also added as a [Blob Data Reader](#) so that it can read data from blob storage.

Azure Data Lake Storage Gen2 access control

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, add the workspace-managed identity to the [Blob Data Reader](#) role. For more information, see [Azure role-based access control](#).

To use ACLs, the workspace-managed identity can be assigned access just like any other security principle. For more information, see [Access control lists on files and directories](#).

Azure Data Lake Storage Gen1 access control

Azure Data Lake Storage Gen1 only supports POSIX-style access control lists. You can assign the workspace-managed identity access to resources just like any other security principle. For more information, see [Access control in Azure Data Lake Storage Gen1](#).

Azure SQL Database contained user

To access data stored in an Azure SQL Database using 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](#).

Azure Machine Learning designer intermediate module output

You can specify the output location for any module in the designer. Use this to store intermediate datasets in separate location for security, logging, or auditing purposes. To specify output:

1. Select the module whose output you'd like to specify.
2. In the module settings pane that appears to the right, select **Output settings**.
3. Specify the datastore you want to use for each module output.

Make sure that you have access to the intermediate storage accounts in your virtual network. Otherwise, the pipeline will fail.

You should also [enable managed identity authentication](#) for intermediate storage accounts to visualize output data.

Next steps

This article is an optional part of a four-part virtual network series. See the rest of the articles to learn how to secure a virtual network:

- [Part 1: Virtual network overview](#)
- [Part 2: Secure the workspace resources](#)
- [Part 3: Secure the training environment](#)
- [Part 4: Secure the inferencing environment](#)

Configure Azure Private Link for an Azure Machine Learning workspace

12/23/2020 • 4 minutes to read • [Edit Online](#)

In this document, you learn how to use Azure Private Link with 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. Private Link helps reduce the risk of data exfiltration. To learn more about private endpoints, see the [Azure Private Link](#) article.

IMPORTANT

Azure Private Link does not effect 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.

You may encounter problems trying to access the private endpoint for your workspace if you are using Mozilla Firefox. This problem may be related to DNS over HTTPS in Mozilla. We recommend using Microsoft Edge or Google Chrome as a workaround.

Prerequisites

If you plan on using a private link enabled workspace with a customer-managed key, you must request this feature using a support ticket. For more information, see [Manage and increase quotas](#).

Limitations

Using an Azure Machine Learning workspace with private link is not available in the Azure Government regions or Azure China 21Vianet regions.

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](#).

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

The Azure Machine Learning Python SDK provides the [PrivateEndpointConfig](#) class, which can be used with

`Workspace.create()` to create a workspace with a private endpoint. This class requires an existing virtual network.

```
from azureml.core import Workspace
from azureml.core import PrivateEndPointConfig

pe = PrivateEndPointConfig(name='myprivateendpoint', vnet_name='myvnet', vnet_subnet_name='default')
ws = Workspace.create(name='myworkspace',
    subscription_id='<my-subscription-id>',
    resource_group='myresourcegroup',
    location='eastus2',
    private_endpoint_config=pe,
    private_endpoint_auto_approval=True,
    show_output=True)
```

Add a private endpoint to a workspace

Use one of the following methods to add a private endpoint to an existing workspace:

IMPORTANT

You must have an existing virtual network to create the private endpoint in. You must also [disable network policies for private endpoints](#) before adding the private endpoint.

WARNING

If you have any existing compute targets associated with this workspace, and they are not behind the same virtual network than the private endpoint is created in, they will not work.

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

```
from azureml.core import Workspace
from azureml.core import PrivateEndPointConfig

pe = PrivateEndPointConfig(name='myprivateendpoint', vnet_name='myvnet', vnet_subnet_name='default')
ws = Workspace.from_config()
ws.add_private_endpoint(private_endpoint_config=pe, private_endpoint_auto_approval=True, show_output=True)
```

For more information on the classes and methods used in this example, see [PrivateEndpointConfig](#) and [Workspace.add_private_endpoint](#).

Remove a private endpoint

Use one of the following methods to remove a private endpoint from a workspace:

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

Use [Workspace.delete_private_endpoint_connection](#) to remove a private endpoint.

```
from azureml.core import Workspace

ws = Workspace.from_config()
# get the connection name
_, _, connection_name = ws.get_details()['privateEndpointConnections'][0]['id'].rpartition('/')
ws.delete_private_endpoint_connection(private_endpoint_connection_name=connection_name)
```

Using a workspace over a private endpoint

Since communication to the workspace is only allowed from the virtual network, any development environments that use the workspace must be members of the virtual network. For example, a virtual machine in the virtual network.

IMPORTANT

To avoid temporary disruption of connectivity, Microsoft recommends flushing the DNS cache on machines connecting to the workspace after enabling Private Link.

For information on Azure Virtual Machines, see the [Virtual Machines documentation](#).

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](#).

How to use your workspace with a custom DNS server

12/23/2020 • 2 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 only covers how to find the fully qualified domain name (FQDN) and IP addresses for these entries it 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.

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](#)
- Optionally, [Azure CLI](#) or [Azure PowerShell](#).

FQDNs in use

These FQDNs are in use in the following regions: eastus, southcentralus and westus2.

The following list contains the fully qualified domain names (FQDN) used by your workspace:

- <workspace-GUID>.workspace.<region>.cert.azureml.ms
- <workspace-GUID>.workspace.<region>.api.azureml.ms
- <workspace-GUID>.workspace.<region>.experiments.azureml.net
- <workspace-GUID>.workspace.<region>.modelmanagement.azureml.net
- <workspace-GUID>.workspace.<region>.aether.ms
- ml-<workspace-name>-<region>-<workspace-guid>.notebooks.azure.net
- If you create a compute instance, you must also add an entry for <instance-name>.<region>.instances.azureml.ms with the private IP of the workspace private endpoint.

NOTE

Compute instances can be accessed only from within the virtual network.

These FQDNs are in use in all other regions

The following list contains the fully qualified domain names (FQDN) used by your workspace:

- <workspace-GUID>.workspace.<region>.cert.azureml.ms
- <workspace-GUID>.workspace.<region>.api.azureml.ms
- ml-<workspace-name>-<region>-<workspace-guid>.notebooks.azure.net
- <instance-name>.<region>.instances.azureml.ms

NOTE

Compute instances can be accessed only from within the virtual network.

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](#)

```
az network private-endpoint show --endpoint-name <endpoint> --resource-group <resource-group> --query  
'customDnsConfigs[*].{FQDN: fqdn, IPAddress: ipAddresses[0]}' --output table
```

The information returned from all methods is the same; a list of the FQDN and private IP address for the resources.

FQDN	IP ADDRESS
fb7e20a0-8891-458b-b969-55ddb3382f51.workspace.eastus.api.azureml.ms	10.1.0.5
ml-myworkspace-eastus-fb7e20a0-8891-458b-b969-55ddb3382f51.notebooks.azure.net	10.1.0.6

IMPORTANT

Some FQDNs are not shown in listed by the private endpoint, but are required by the workspace in eastus, southcentralus and westus2. These FQDNs are listed in the following table, and must also be added to your DNS server and/or an Azure Private DNS Zone:

- <workspace-GUID>.workspace.<region>.cert.azureml.ms
- <workspace-GUID>.workspace.<region>.experiments.azureml.net
- <workspace-GUID>.workspace.<region>.modelmanagement.azureml.net
- <workspace-GUID>.workspace.<region>.aether.ms
- If you have a compute instance, use <instance-name>.<region>.instances.azureml.ms , where <instance-name> is the name of your compute instance. Please use private IP address of workspace private endpoint. Please note compute instance can be accessed only from within the virtual network.

For all of these IP address, use the same address as the *.api.azureml.ms entries returned from the previous steps.

Next steps

For more information on using Azure Machine Learning with a virtual network, see the [virtual network overview](#).

For more information on integrating Private Endpoints into your DNS configuration, see [Azure Private Endpoint DNS configuration](#).

Use TLS to secure a web service through Azure Machine Learning

12/23/2020 • 8 minutes to read • [Edit Online](#)

This article shows you how to secure a web service that's deployed through Azure Machine Learning.

You use [HTTPS](#) to restrict access to web services and secure the data that clients submit. HTTPS helps secure communications between a client and a web service by encrypting communications between the two. Encryption uses [Transport Layer Security \(TLS\)](#). TLS is sometimes still referred to as *Secure Sockets Layer (SSL)*, which was the predecessor of TLS.

TIP

The Azure Machine Learning SDK uses the term "SSL" for properties that are related to secure communications. This doesn't mean that your web service doesn't use *TLS*. SSL is just a more commonly recognized term.

Specifically, web services deployed through Azure Machine Learning support TLS version 1.2 for AKS and ACI. For ACI deployments, if you are on older TLS version, we recommend re-deploying to get the latest TLS version.

TLS and SSL both rely on *digital certificates*, which help with encryption and identity verification. For more information on how digital certificates work, see the Wikipedia topic [Public key infrastructure](#).

WARNING

If you don't use HTTPS for your web service, data that's sent to and from the service might be visible to others on the internet.

HTTPS also enables the client to verify the authenticity of the server that it's connecting to. This feature protects clients against [man-in-the-middle attacks](#).

This is the general process to secure a web service:

1. Get a domain name.
2. Get a digital certificate.
3. Deploy or update the web service with TLS enabled.
4. Update your DNS to point to the web service.

IMPORTANT

If you're deploying to Azure Kubernetes Service (AKS), you can purchase your own certificate or use a certificate that's provided by Microsoft. If you use a certificate from Microsoft, you don't need to get a domain name or TLS/SSL certificate. For more information, see the [Enable TLS and deploy](#) section of this article.

There are slight differences when you secure s across [deployment targets](#).

Get a domain name

If you don't already own a domain name, purchase one from a *domain name registrar*. The process and price

differ among registrars. The registrar provides tools to manage the domain name. You use these tools to map a fully qualified domain name (FQDN) (such as www.contoso.com) to the IP address that hosts your web service.

Get a TLS/SSL certificate

There are many ways to get an TLS/SSL certificate (digital certificate). The most common is to purchase one from a *certificate authority* (CA). Regardless of where you get the certificate, you need the following files:

- A **certificate**. The certificate must contain the full certificate chain, and it must be "PEM-encoded."
- A **key**. The key must also be PEM-encoded.

When you request a certificate, you must provide the FQDN of the address that you plan to use for the web service (for example, www.contoso.com). The address that's stamped into the certificate and the address that the clients use are compared to verify the identity of the web service. If those addresses don't match, the client gets an error message.

TIP

If the certificate authority can't provide the certificate and key as PEM-encoded files, you can use a utility such as [OpenSSL](#) to change the format.

WARNING

Use *self-signed* certificates only for development. Don't use them in production environments. Self-signed certificates can cause problems in your client applications. For more information, see the documentation for the network libraries that your client application uses.

Enable TLS and deploy

To deploy (or redeploy) the service with TLS enabled, set the *ssl_enabled* parameter to "True" wherever it's applicable. Set the *ssl_certificate* parameter to the value of the *certificate* file. Set the *ssl_key* to the value of the *key* file.

Deploy on Azure Kubernetes Service

NOTE

The information in this section also applies when you deploy a secure web service for the designer. If you aren't familiar with using the Python SDK, see [What is the Azure Machine Learning SDK for Python?](#).

Both [AksCompute.provisioning_configuration\(\)](#) and [AksCompute.attach_configuration\(\)](#) return a configuration object that has an **enable_ssl** method, and you can use **enable_ssl** method to enable TLS.

You can enable TLS either with Microsoft certificate or a custom certificate purchased from CA.

- When you use a certificate from Microsoft, you must use the *leaf_domain_label* parameter. This parameter generates the DNS name for the service. For example, a value of "contoso" creates a domain name of "contoso<six-random-characters>.<azureregion>.cloudapp.azure.com", where <azureregion> is the region that contains the service. Optionally, you can use the *overwrite_existing_domain* parameter to overwrite the existing *leaf_domain_label*. The following example demonstrates how to create a configuration that enables an TLS with Microsoft certificate:

```

from azureml.core.compute import AksCompute

# Config used to create a new AKS cluster and enable TLS
provisioning_config = AksCompute.provisioning_configuration()

# Leaf domain label generates a name using the formula
# "<leaf-domain-label>#####.<azure-region>.cloudapp.azure.net"
# where "#####" is a random series of characters
provisioning_config.enable_ssl(leaf_domain_label = "contoso")

# Config used to attach an existing AKS cluster to your workspace and enable TLS
attach_config = AksCompute.attach_configuration(resource_group = resource_group,
                                                cluster_name = cluster_name)

# Leaf domain label generates a name using the formula
# "<leaf-domain-label>#####.<azure-region>.cloudapp.azure.net"
# where "#####" is a random series of characters
attach_config.enable_ssl(leaf_domain_label = "contoso")

```

IMPORTANT

When you use a certificate from Microsoft, you don't need to purchase your own certificate or domain name.

WARNING

If your AKS cluster is configured with an internal load balancer, using a Microsoft provided certificate is **not supported** and you must use custom certificate to enable TLS.

- When you use a custom certificate that you purchased, you use the `ssl_cert_pem_file`, `ssl_key_pem_file`, and `ssl_cname` parameters. The following example demonstrates how to use .pem files to create a configuration that uses a TLS/SSL certificate that you purchased:

```

from azureml.core.compute import AksCompute

# Config used to create a new AKS cluster and enable TLS
provisioning_config = AksCompute.provisioning_configuration()
provisioning_config.enable_ssl(ssl_cert_pem_file="cert.pem",
                             ssl_key_pem_file="key.pem", ssl_cname="www.contoso.com")

# Config used to attach an existing AKS cluster to your workspace and enable SSL
attach_config = AksCompute.attach_configuration(resource_group = resource_group,
                                                cluster_name = cluster_name)
attach_config.enable_ssl(ssl_cert_pem_file="cert.pem",
                        ssl_key_pem_file="key.pem", ssl_cname="www.contoso.com")

```

For more information about `enable_ssl`, see [AksProvisioningConfiguration.enable_ssl\(\)](#) and [AksAttachConfiguration.enable_ssl\(\)](#).

Deploy on Azure Container Instances

When you deploy to Azure Container Instances, you provide values for TLS-related parameters, as the following code snippet shows:

```
from azureml.core.webservice import AciWebservice

aci_config = AciWebservice.deploy_configuration(
    ssl_enabled=True, ssl_cert_pem_file="cert.pem", ssl_key_pem_file="key.pem", ssl_cname="www.contoso.com")
```

For more information, see [AciWebservice.deploy_configuration\(\)](#).

Update your DNS

For either AKS deployment with custom certificate or ACI deployment, you must update your DNS record to point to the IP address of scoring endpoint.

IMPORTANT

When you use a certificate from Microsoft for AKS deployment, you don't need to manually update the DNS value for the cluster. The value should be set automatically.

You can follow following steps to update DNS record for your custom domain name:

- Get scoring endpoint IP address from scoring endpoint URI, which is usually in the format of <http://104.214.29.152:80/api/v1/service//score>.
- Use the tools from your domain name registrar to update the DNS record for your domain name. The record must point to the IP address of scoring endpoint.
- After DNS record update, you can validate DNS resolution using `nslookup custom-domain-name` command. If DNS record is correctly updated, the custom domain name will point to the IP address of scoring endpoint.
- There can be a delay of minutes or hours before clients can resolve the domain name, depending on the registrar and the "time to live" (TTL) that's configured for the domain name.

Update the TLS/SSL certificate

TLS/SSL certificates expire and must be renewed. Typically this happens every year. Use the information in the following sections to update and renew your certificate for models deployed to Azure Kubernetes Service:

Update a Microsoft generated certificate

If the certificate was originally generated by Microsoft (when using the `leaf_domain_label` to create the service), use one of the following examples to update the certificate:

IMPORTANT

- If the existing certificate is still valid, use `renew=True` (SDK) or `--ssl-renew` (CLI) to force the configuration to renew it. For example, if the existing certificate is still valid for 10 days and you don't use `renew=True`, the certificate may not be renewed.
- When the service was originally deployed, the `leaf_domain_label` is used to create a DNS name using the pattern `<leaf-domain-label>#####.<azure-region>.cloudapp.azure.net`. To preserve the existing name (including the 6 digits originally generated), use the original `leaf_domain_label` value. Do not include the 6 digits that were generated.

Use the SDK

```

from azureml.core.compute import AksCompute
from azureml.core.compute.aks import AksUpdateConfiguration
from azureml.core.compute.aks import SslConfiguration

# Get the existing cluster
aks_target = AksCompute(ws, clustername)

# Update the existing certificate by referencing the leaf domain label
ssl_configuration = SslConfiguration(leaf_domain_label="myaks", overwrite_existing_domain=True, renew=True)
update_config = AksUpdateConfiguration(ssl_configuration)
aks_target.update(update_config)

```

Use the CLI

```
az ml computetarget update aks -g "myresourcegroup" -w "myresourceworkspace" -n "myaks" --ssl-leaf-domain-label "myaks" --ssl-overwrite-domain True --ssl-renew
```

For more information, see the following reference docs:

- [SslConfiguration](#)
- [AksUpdateConfiguration](#)

Update custom certificate

If the certificate was originally generated by a certificate authority, use the following steps:

1. Use the documentation provided by the certificate authority to renew the certificate. This process creates new certificate files.
2. Use either the SDK or CLI to update the service with the new certificate:

Use the SDK

```

from azureml.core.compute import AksCompute
from azureml.core.compute.aks import AksUpdateConfiguration
from azureml.core.compute.aks import SslConfiguration

# Read the certificate file
def get_content(file_name):
    with open(file_name, 'r') as f:
        return f.read()

# Get the existing cluster
aks_target = AksCompute(ws, clustername)

# Update cluster with custom certificate
ssl_configuration = SslConfiguration(cname="myaks", cert=get_content('cert.pem'),
key=get_content('key.pem'))
update_config = AksUpdateConfiguration(ssl_configuration)
aks_target.update(update_config)

```

Use the CLI

```
az ml computetarget update aks -g "myresourcegroup" -w "myresourceworkspace" -n "myaks" --ssl-cname "myaks"--ssl-cert-file "cert.pem" --ssl-key-file "key.pem"
```

For more information, see the following reference docs:

- [SslConfiguration](#)
- [AksUpdateConfiguration](#)

Disable TLS

To disable TLS for a model deployed to Azure Kubernetes Service, create an `SslConfiguration` with `status="Disabled"`, then perform an update:

```
from azureml.core.compute import AksCompute
from azureml.core.compute.aks import AksUpdateConfiguration
from azureml.core.compute.aks import SslConfiguration

# Get the existing cluster
aks_target = AksCompute(ws, clustername)

# Disable TLS
ssl_configuration = SslConfiguration(status="Disabled")
update_config = AksUpdateConfiguration(ssl_configuration)
aks_target.update(update_config)
```

Next steps

Learn how to:

- [Consume a machine learning model deployed as a web service](#)
- [Virtual network isolation and privacy overview](#)

Use workspace behind a Firewall for Azure Machine Learning

12/23/2020 • 6 minutes to read • [Edit Online](#)

In this article, learn 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](#).

WARNING

Access to data storage behind a firewall is only supported in code first experiences. Using the [Azure Machine Learning studio](#) to access data behind a firewall is not supported. To work with data storage on a private network with the studio, you must first [set up a virtual network](#) and [give the studio access to data stored inside of a virtual network](#).

Azure Firewall

When using Azure Firewall, use **destination network address translation (DNAT)** to create NAT rules for inbound traffic. For outbound traffic, create **network** and/or **application** rules. These rule collections are described in more detail in [What are some Azure Firewall concepts](#).

Inbound configuration

If you use an Azure Machine Learning **compute instance** or **compute cluster**, add a [user-defined routes \(UDRs\)](#) for the subnet that contains the Azure Machine Learning resources. This route forces traffic from the IP addresses of the `BatchNodeManagement` and `AzureMachineLearning` resources to the public IP of your compute instance and compute cluster.

These UDRs enable the Batch service to communicate with compute nodes for task scheduling. Also add the IP address for the Azure Machine Learning service where the resources exist, as this is required for access to Compute Instances. To get a list of IP addresses of the Batch service and Azure Machine Learning service, use one of the following methods:

- Download the [Azure IP Ranges and Service Tags](#) and search the file for `BatchNodeManagement.<region>` and `AzureMachineLearning.<region>`, where `<region>` is your Azure region.
- Use the [Azure CLI](#) to download the information. The following example downloads the IP address information and filters out the information for the East US 2 region:

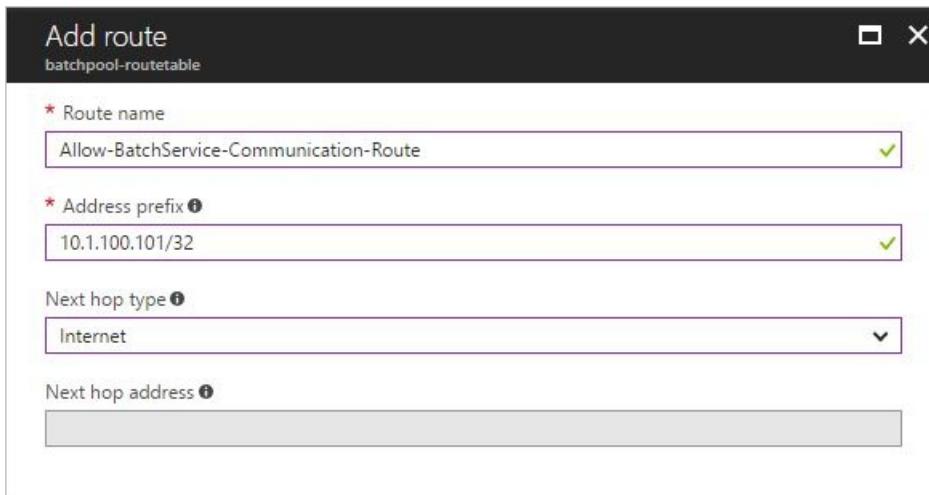
```
az network list-service-tags -l "East US 2" --query "values[?starts_with(id, 'Batch')] | [? properties.region=='eastus2']"  
az network list-service-tags -l "East US 2" --query "values[?starts_with(id, 'AzureMachineLearning')] | [?properties.region=='eastus2']"
```

TIP

If you are using the US-Virginia, US-Arizona regions, or China-East-2 regions, these commands return no IP addresses. Instead, use one of the following links to download a list of IP addresses:

- [Azure IP ranges and service tags for Azure Government](#)
- [Azure IP ranges and service tags for Azure China](#)

When you add the UDRs, define the route for each related Batch IP address prefix and set **Next hop type** to **Internet**. The following image shows an example of this UDR in the Azure portal:



IMPORTANT

The IP addresses may change over time.

For more information, see [Create an Azure Batch pool in a virtual network](#).

Outbound configuration

1. Add **Network rules**, allowing traffic **to** and **from** the following service tags:

- AzureActiveDirectory
- AzureMachineLearning
- AzureResourceManager
- Storage.region
- KeyVault.region
- ContainerRegistry.region

If you plan on using the default Docker images provided by Microsoft, and enabling user-managed dependencies, you must also add the following service tags:

- MicrosoftContainerRegistry.region
- AzureFrontDoor.FirstParty

For entries that contain `region`, replace with the Azure region that you are using. For example,
`keyvault.westus`.

For the **protocol**, select `TCP`. For the source and destination **ports**, select `*`.

2. Add **Application rules** for the following hosts:

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 is not 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.

For **Protocol:Port**, select use **http**, **https**.

For more information on configuring application rules, see [Deploy and configure Azure Firewall](#).

- To restrict access to models deployed to Azure Kubernetes Service (AKS), see [Restrict egress traffic in Azure Kubernetes Service](#).

Other firewalls

The guidance in this section is generic, as each firewall has its own terminology and specific configurations. If you have questions about how to allow communication through your firewall, please consult the documentation for the firewall you are using.

If not configured correctly, the firewall can cause problems using your workspace. There are a variety of host names that are used both by the Azure Machine Learning workspace. The following sections list hosts that are required for Azure Machine Learning.

Microsoft hosts

The hosts in this section are owned by Microsoft, and provide services required for the proper functioning of your workspace. The following tables list the host names for the Azure public, Azure Government, and Azure China 21Vianet regions.

General Azure hosts

REQUIRED FOR	AZURE PUBLIC	AZURE GOVERNMENT	AZURE CHINA 21VIANET
Azure Active Directory	login.microsoftonline.com	login.microsoftonline.us	login.chinacloudapi.cn
Azure portal	management.azure.com	management.azure.us	management.azure.cn

Azure Machine Learning hosts

REQUIRED FOR	AZURE PUBLIC	AZURE GOVERNMENT	AZURE CHINA 21VIANET
Azure Machine Learning studio	ml.azure.com	ml.azure.us	studio.ml.azure.cn

REQUIRED FOR	AZURE PUBLIC	AZURE GOVERNMENT	AZURE CHINA 21VIANET
API	*.azureml.ms	*.ml.azure.us	*.ml.azure.cn
Experimentation, History, Hyperdrive, labeling	*.experiments.azureml.net	*.ml.azure.us	*.ml.azure.cn
Model management	*.modelmanagement.azure.ml.net	*.ml.azure.us	*.ml.azure.cn
Pipeline	*.aether.ms	*.ml.azure.us	*.ml.azure.cn
Designer (studio service)	*.studioservice.azureml.com	*.ml.azure.us	*.ml.azure.cn
Integrated notebook	*.notebooks.azure.net	*.notebooks.usgovcloudapi.net	*.notebooks.chinacloudapi.cn
Integrated notebook	*.file.core.windows.net	*.file.core.usgovcloudapi.net	*.file.core.chinacloudapi.cn
Integrated notebook	*.dfs.core.windows.net	*.dfs.core.usgovcloudapi.net	*.dfs.core.chinacloudapi.cn
Integrated notebook	*.blob.core.windows.net	*.blob.core.usgovcloudapi.net	*.blob.core.chinacloudapi.cn
Integrated notebook	graph.microsoft.com	graph.microsoft.us	graph.chinacloudapi.cn
Integrated notebook	*.aznbcontent.net		

Azure Machine Learning compute instance and compute cluster hosts

REQUIRED FOR	AZURE PUBLIC	AZURE GOVERNMENT	AZURE CHINA 21VIANET
Compute cluster/instance	*.batchai.core.windows.net	*.batchai.core.usgovcloudapi.net	*.batchai.ml.azure.cn
Compute instance	*.instances.azureml.net	*.instances.azureml.us	*.instances.azureml.cn
Compute instance	*.instances.azureml.ms		

Associated resources used by Azure Machine Learning

REQUIRED FOR	AZURE PUBLIC	AZURE GOVERNMENT	AZURE CHINA 21VIANET
Azure Storage Account	core.windows.net	core.usgovcloudapi.net	core.chinacloudapi.cn
Azure Key Vault	vault.azure.net	vault.usgovcloudapi.net	vault.azure.cn
Azure Container Registry	azurecr.io	azurecr.us	azurecr.cn
Microsoft Container Registry	mcr.microsoft.com	mcr.microsoft.com	mcr.microsoft.com

TIP

If you plan on using federated identity, follow the [Best practices for securing Active Directory Federation Services](#) article.

Also, use the information in [forced tunneling](#) to add IP addresses for `BatchNodeManagement` and `AzureMachineLearning`.

For information on restricting access to models deployed to Azure Kubernetes Service (AKS), see [Restrict egress traffic in Azure Kubernetes Service](#).

Python hosts

The hosts in this section are used to install Python packages. They 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
<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 is not overwritten by user settings. If the index is overwritten, you must also allow <code>*.pythonhosted.org</code> .
<code>*pytorch.org</code>	Used by some examples based on PyTorch.
<code>*.tensorflow.org</code>	Used by some examples based on Tensorflow.

R hosts

The hosts in this section are used to install R packages. They 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
<code>cloud.r-project.org</code>	Used when installing CRAN packages.

IMPORTANT

Internally, the R SDK for Azure Machine Learning uses Python packages. So you must also allow Python hosts through the firewall.

Next steps

- [Tutorial: Deploy and configure Azure Firewall using the Azure portal](#)
- [Secure Azure ML experimentation and inference jobs within an Azure Virtual Network](#)

Increase Azure Machine Learning resiliency

12/23/2020 • 4 minutes to read • [Edit Online](#)

In this article, you'll learn how to make your Microsoft Azure Machine Learning resources more resilient by using high-availability configurations. You can configure the Azure services that Azure Machine Learning depends on for high availability. This article identifies the services you can configure for high availability, and links to additional information on configuring these resources.

NOTE

Azure Machine Learning itself does not offer a disaster recovery option.

Understand Azure services for Azure Machine Learning

Azure Machine Learning depends on multiple Azure services and has several layers. Some of these services are provisioned in your (customer) subscription. You're responsible for the high-availability configuration of these services. Other services are created in a Microsoft subscription and 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. You're responsible for configuring high-availability settings for these resources.
 - 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.
- **Additional data stores:** Azure Machine Learning can mount additional 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 which Azure services are managed by Microsoft, which are managed by you, and which are highly available by default.

SERVICE	MANAGED BY	HIGH AVAILABILITY BY DEFAULT
Azure Machine Learning infrastructure	Microsoft	

Service	Managed By	High Availability by Default
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	
Additional 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.

Associated resources

IMPORTANT

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).

Make sure to configure the high-availability settings of each resource by referring to the following documentation:

- **Azure Storage:** To configure high-availability settings, see [Azure Storage redundancy](#).
- **Key Vault:** Key Vault provides high availability by default and requires no user action. See [Azure Key Vault availability and redundancy](#).
- **Container Registry:** Choose the Premium registry option for geo-replication. See [Geo-replication in Azure Container Registry](#).
- **Application Insights:** Application Insights doesn't provide high-availability settings. To adjust the data-retention period and details, see [Data collection, retention, and storage in Application Insights](#).

Compute resources

Make sure to configure the high-availability settings of each resource by referring to the following documentation:

- **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](#).

Additional data stores

Make sure to configure the high-availability settings of each resource by referring to the following documentation:

- **Azure Blob container / Azure Files / Data Lake Storage Gen2**: Same as default storage.
- **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](#).

Azure Cosmos DB

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](#).

Next steps

To deploy Azure Machine Learning with associated resources with your high-availability settings, use an [Azure Resource Manager template](#).

Regenerate storage account access keys

12/23/2020 • 5 minutes to read • [Edit Online](#)

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 a workspace](#) article.
- The [Azure Machine Learning SDK](#).
- The [Azure Machine Learning CLI extension](#).

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.

Monitor Azure Machine Learning

12/23/2020 • 6 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 as web services or IoT Edge modules, see [Collect model data](#) and [Monitor with Application Insights](#).

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.

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, 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	Events from nodes within an Azure Machine Learning compute cluster.
AmlComputeJobEvent	Events from jobs running on Azure Machine Learning compute.

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.

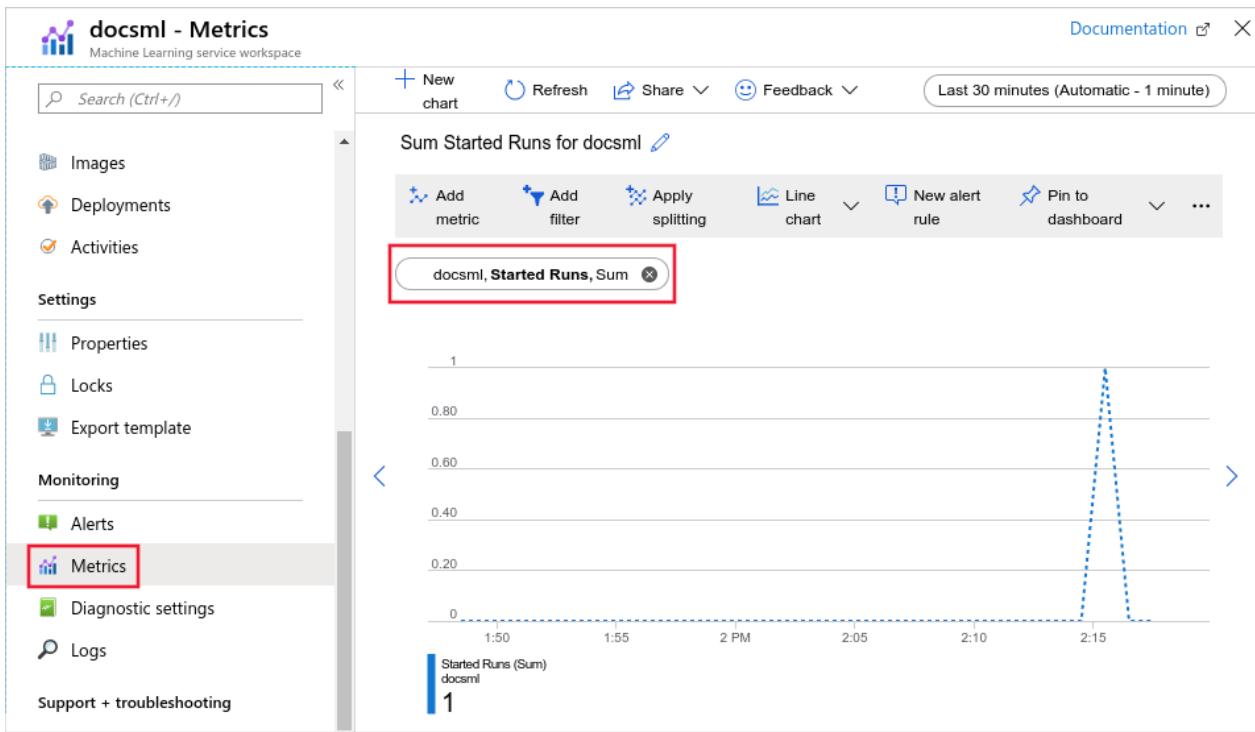
The metrics and logs you can collect are discussed in the following sections.

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](#).

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	Events from nodes within an Azure Machine Learning compute cluster.
AmlComputeJobEvent	Events from jobs running on Azure Machine Learning compute.

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 nodes allocated in the last eight days:

```
AmlComputeClusterNodeEvent  
| where TimeGenerated > ago(8d) and NodeAllocationTime > ago(8d)  
| distinct NodeId
```

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

12/23/2020 • 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 using Azure Policy

12/23/2020 • 2 minutes to read • [Edit Online](#)

Azure Policy is a governance tool that allows you to ensure that Azure resources are compliant with your policies. With Azure Machine Learning, you can assign the following policies:

- **Customer-managed key:** Audit or enforce whether workspaces must use a customer-managed key.
- **Private link:** Audit whether workspaces use a private endpoint to communicate with 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](#).

Built-in 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. For a complete list of the built-in policies for Azure Machine Learning, see [Built-in policies for Azure Machine Learning](#).

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](#).

Workspaces encryption with customer-managed key

Controls whether workspaces 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 workspaces 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.

Workspaces should use private link

Controls whether workspaces 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**. If you create a workspace without using private link, 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 Azure Security Center](#)

Create and manage Azure Machine Learning workspaces

12/23/2020 • 13 minutes to read • [Edit Online](#)

In this article, you'll 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 create and delete 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, [install the SDK](#).

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.MachineLearningService**. 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 does not currently support unicode characters in resource group names, use a resource group that does not contain these characters.

Create a workspace

- [Python](#)
- [Portal](#)
- **Default specification.** By default, dependent resources as well as the resource group will be created automatically. This code creates a workspace named `myworkspace` and a resource group named `myresourcegroup` in `eastus2`.

```
from azureml.core import Workspace

ws = Workspace.create(name='myworkspace',
                      subscription_id='<azure-subscription-id>',
                      resource_group='myresourcegroup',
                      create_resource_group=True,
                      location='eastus2'
                     )
```

Set `create_resource_group` to False if you have an existing Azure resource group that you want to use for the workspace.

- **Multiple tenants.** If you have multiple accounts, add the tenant ID of the Azure Active Directory you wish to use. Find your tenant ID from the [Azure portal](#) under **Azure Active Directory, External Identities**.

```
from azureml.core.authentication import InteractiveLoginAuthentication
from azureml.core import Workspace

interactive_auth = InteractiveLoginAuthentication(tenant_id="my-tenant-id")
ws = Workspace.create(name='myworkspace',
                      subscription_id='<azure-subscription-id>',
                      resource_group='myresourcegroup',
                      create_resource_group=True,
                      location='eastus2',
                      auth=interactive_auth
                     )
```

- **Sovereign cloud.** You'll need extra code to authenticate to Azure if you're working in a sovereign cloud.

```
from azureml.core.authentication import InteractiveLoginAuthentication
from azureml.core import Workspace

interactive_auth = InteractiveLoginAuthentication(cloud="<cloud name>") # for example,
cloud="AzureUSGovernment"
ws = Workspace.create(name='myworkspace',
                      subscription_id='<azure-subscription-id>',
                      resource_group='myresourcegroup',
                      create_resource_group=True,
                      location='eastus2',
                      auth=interactive_auth
                     )
```

- **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.

```

import os
from azureml.core import Workspace
from azureml.core.authentication import ServicePrincipalAuthentication

service_principal_password = os.environ.get("AZUREML_PASSWORD")

service_principal_auth = ServicePrincipalAuthentication(
    tenant_id=<tenant-id>,
    username=<application-id>,
    password=service_principal_password)

        auth=service_principal_auth,
        subscription_id='<azure-subscription-id>',
        resource_group='myresourcegroup',
        create_resource_group=False,
        location='eastus2',
        friendly_name='My workspace',
        storage_account='subscriptions/<azure-subscription-
id>/resourcegroups/myresourcegroup/providers/microsoft.storage/storageaccounts/mystorageacc
ount',
        key_vault='subscriptions/<azure-subscription-
id>/resourcegroups/myresourcegroup/providers/microsoft.keyvault/vaults/mykeyvault',
        app_insights='subscriptions/<azure-subscription-
id>/resourcegroups/myresourcegroup/providers/microsoft.insights/components/myappinsights',
        container_registry='subscriptions/<azure-subscription-
id>/resourcegroups/myresourcegroup/providers/microsoft.containerregistry/registries/myconta
inerregistry',
        exist_ok=False)

```

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](#)
- [Portal](#)

The Azure Machine Learning Python SDK provides the [PrivateEndpointConfig](#) class, which can be used with [Workspace.create\(\)](#) to create a workspace with a private endpoint. This class requires an existing virtual network.

IMPORTANT

Using a private endpoint with Azure Machine Learning workspace 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](#).

Multiple workspaces with private endpoint

When you create a private endpoint, a new Private DNS Zone named `privatelink.api.azureml.ms` is created. This contains a link to the virtual network. If you create multiple workspaces with private endpoints in the same resource group, only the virtual network for the first private endpoint may be

added to the DNS zone. To add entries for the virtual networks used by the additional workspaces/private endpoints, use the following steps:

1. In the [Azure portal](#), select the resource group that contains the workspace. Then select the Private DNS Zone resource named `privatelink.api.azureml.ms`
2. In the **Settings**, select **Virtual network links**.
3. Select **Add**. From the **Add virtual network link** page, provide a unique **Link name**, and then select the **Virtual network** to be added. Select **OK** to add the network link.

For more information, see [Azure Private Endpoint DNS configuration](#).

Vulnerability scanning

Azure Security Center provides unified security management and advanced threat protection across hybrid cloud workloads. You should allow Azure Security Center to scan your resources and follow its recommendations. For more, see [Azure Container Registry image scanning by Security Center](#) and [Azure Kubernetes Services integration with Security Center](#).

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 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.

IMPORTANT

The Cosmos DB instance is created in a Microsoft-managed resource group in **your subscription**, along with any resources it needs. This means that you are charged for this Cosmos DB instance. The managed resource group is named in the format `<AML Workspace Resource Group Name><GUID>`. If your Azure Machine Learning workspace uses a private endpoint, a virtual network is also created for the Cosmos DB instance. This VNet is used to secure communication between Cosmos DB and Azure Machine Learning.

- **Do not delete the resource group** that contains this Cosmos DB instance, or any of the resources automatically created in this group. If you need to delete the resource group, Cosmos DB instance, etc., you must delete the Azure Machine Learning workspace that uses it. The resource group, Cosmos DB instance, and other automatically created resources are deleted when the associated workspace is deleted.
- The default **Request Units** for this Cosmos DB account is set at **8000**.
- You **cannot provide your own VNet for use with the Cosmos DB instance** that is created. You also **cannot modify the virtual network**. For example, you cannot change the IP address range that it uses.

Use the following steps to provide your own key:

IMPORTANT

Before following these steps, you must first perform the following actions:

1. Authorize the **Machine Learning App** (in Identity and Access Management) with contributor permissions on your subscription.
2. 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

You do not need to manually create the Azure Cosmos DB instance, one will be created for you during workspace creation. This Azure Cosmos DB instance will be created in a separate resource group using a name based on this pattern: <your-workspace-resource-name>_<GUID> .

You cannot change this setting after workspace creation. If you delete the Azure Cosmos DB used by your workspace, you must also delete the workspace that is using it.

- [Python](#)
- [Portal](#)

Use `cmk_keyvault` and `resource_cmk_uri` to specify the customer managed key.

```
from azureml.core import Workspace
ws = Workspace.create(name='myworkspace',
                      subscription_id='<azure-subscription-id>',
                      resource_group='myresourcegroup',
                      create_resource_group=True,
                      location='eastus2'
                      cmk_keyvault='subscriptions/<azure-subscription-
id>/resourcegroups/myresourcegroup/providers/microsoft.keyvault/vaults/<keyvault-name>',
                      resource_cmk_uri='<key-identifier>'
                     )
```

Download a configuration file

If you will be creating a [compute instance](#), skip this step. The compute instance has already created a copy of this file for you.

- [Python](#)
- [Portal](#)

If you plan to use code on your local environment that references this workspace (`ws`), write the configuration file:

```
ws.write_config()
```

Place the file into the directory structure with your Python scripts or Jupyter Notebooks. It can be in the same directory, a subdirectory named `.azurerm/`, 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

In your Python code, you create a workspace object to connect to your workspace. This code will read the contents of the configuration file to find your workspace. You will get a prompt to sign in if you are

not already authenticated.

```
from azureml.core import Workspace  
  
ws = Workspace.from_config()
```

- **Multiple tenants.** If you have multiple accounts, add the tenant ID of the Azure Active Directory you wish to use. Find your tenant ID from the [Azure portal](#) under **Azure Active Directory, External Identities**.

```
from azureml.core.authentication import InteractiveLoginAuthentication  
from azureml.core import Workspace  
  
interactive_auth = InteractiveLoginAuthentication(tenant_id="my-tenant-id")  
ws = Workspace.from_config(auth=interactive_auth)
```

- **Sovereign cloud.** You'll need extra code to authenticate to Azure if you're working in a sovereign cloud.

```
from azureml.core.authentication import InteractiveLoginAuthentication  
from azureml.core import Workspace  
  
interactive_auth = InteractiveLoginAuthentication(cloud="<cloud name>") # for example,  
cloud="AzureUSGovernment"  
ws = Workspace.from_config(auth=interactive_auth)
```

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.

- [Python](#)
- [Portal](#)

Find your subscriptions in the [Subscriptions page in the Azure portal](#). Copy the ID and use it in the code below to see all workspaces available for that subscription.

```
from azureml.core import Workspace  
  
Workspace.list('<subscription-id>')
```

Delete a workspace

When you no longer need a workspace, delete it.

- [Python](#)
- [Portal](#)

Delete the workspace `ws`:

```
ws.delete(delete_dependent_resources=False, no_wait=False)
```

The default action is not to delete resources associated with the workspace, i.e., container registry, storage account, key vault, and application insights. Set `delete_dependent_resources` to True to delete these resources as well.

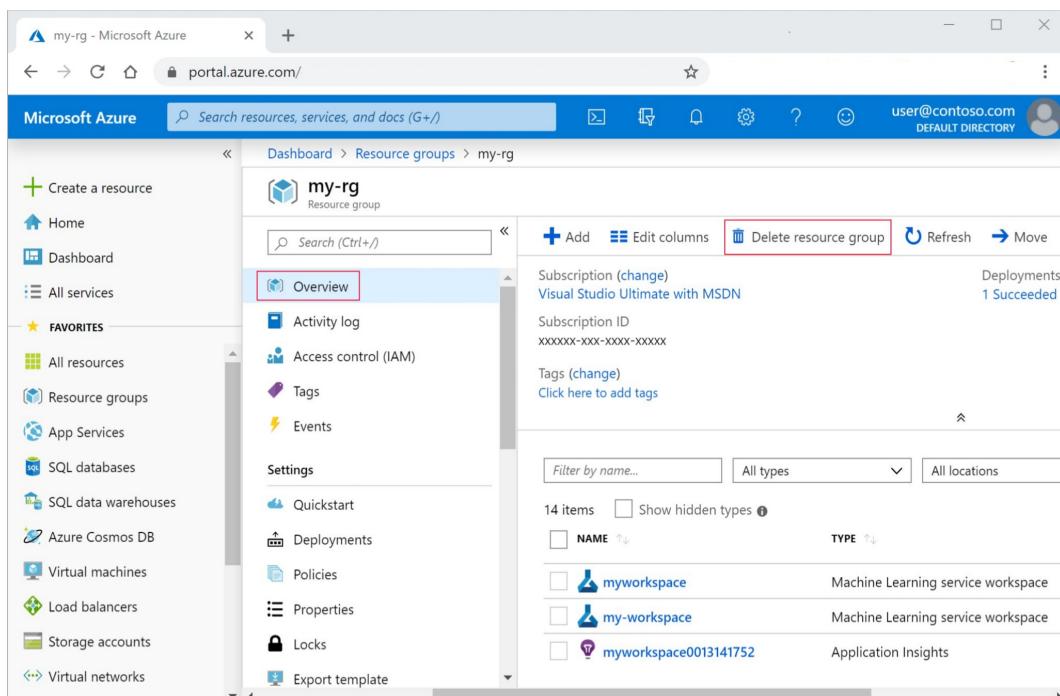
Clean up resources

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 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. On the left, there's a navigation menu with options like 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES' (which includes 'All resources', 'Resource groups', 'App Services', 'SQL databases', 'SQL data warehouses', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', 'Storage accounts', and 'Virtual networks'), and 'Deployments' (with 1 succeeded). The main area shows the 'Overview' tab selected. At the top right of this area, there are buttons for 'Add', 'Edit columns', 'Delete resource group' (which is highlighted with a red box), 'Refresh', and 'Move'. Below this, there's information about the subscription (Visual Studio Ultimate with MSDN), the subscription ID (XXXXXX-XXX-XXXX-XXXX), and tags. A table at the bottom lists 14 items, filtered by type, showing three entries: 'myworkspace' (Machine Learning service workspace), 'my-workspace' (Machine Learning service workspace), and 'myworkspace0013141752' (Application Insights).

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 (Datasets, Experiments, Computes, and so on) are available only in [Azure Machine Learning studio](#). They're *not* available from the Azure portal.

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.

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.

Examples

Examples of creating a workspace:

- Use Azure portal to [create a workspace and compute instance](#)
- Use Python SDK to [create a workspace in your own environment](#)

Next steps

Once you have a workspace, learn how to [Train and deploy a model](#).

Create a workspace for Azure Machine Learning with Azure CLI

12/23/2020 • 12 minutes to read • [Edit Online](#)

In this article, you learn how to create an Azure Machine Learning workspace using the Azure CLI. The Azure CLI provides commands for managing Azure resources. The machine learning extension to the CLI provides commands for working with Azure Machine Learning resources.

Prerequisites

- An **Azure subscription**. If you do not 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.MachineLearningService**. 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.

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](#).

Install the machine learning extension

To install the machine learning extension, use the following command:

```
az extension add -n azure-cli-ml
```

Create a workspace

The Azure Machine Learning workspace relies on the following Azure services or entities:

IMPORTANT

If you do not specify an existing Azure service, one will be created automatically during workspace creation. You must always specify a resource group. 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.

SERVICE	PARAMETER TO SPECIFY AN EXISTING INSTANCE
Azure resource group	<code>-g <resource-group-name></code>
Azure Storage Account	<code>--storage-account <service-id></code>
Azure Application Insights	<code>--application-insights <service-id></code>
Azure Key Vault	<code>--keyvault <service-id></code>
Azure Container Registry	<code>--container-registry <service-id></code>

Azure Container Registry (ACR) doesn't currently support unicode characters in resource group names. To mitigate this issue, use a resource group that does not contain these characters.

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:

TIP

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:

```
{
  "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](#).

Automatically create required resources

To create a new workspace where the **services are automatically created**, use the following command:

```
az ml workspace create -w <workspace-name> -g <resource-group-name>
```

NOTE

The workspace name is case-insensitive.

The output of this command is similar to the following JSON:

```
{
  "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>"
}
```

Virtual network and private endpoint

IMPORTANT

Using an Azure Machine Learning workspace with private link is not available in the Azure Government regions or Azure China 21Vianet regions.

If you want to restrict access to your workspace to a virtual network, you can use the following parameters:

- `--pe-name` : The name of the private endpoint that is created.
- `--pe-auto-approval` : Whether private endpoint connections to the workspace should be automatically approved.
- `--pe-resource-group` : The resource group to create the private endpoint in. Must be the same group that contains the virtual network.
- `--pe-vnet-name` : The existing virtual network to create the private endpoint in.
- `--pe-subnet-name` : The name of the subnet to create the private endpoint in. The default value is `default`.

For more information on using a private endpoint and virtual network with your workspace, see [Virtual network isolation and privacy overview](#).

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.

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.

Instead of using the Microsoft-managed key, you can use the provide your own key. Doing so creates the Azure Cosmos DB instance that stores metadata in your Azure subscription. Use the `--cmk-keyvault` parameter to specify the Azure Key Vault that contains the key, and `--resource-cmk-uri` to specify the URL of the key within the vault.

Before using the `--cmk-keyvault` and `--resource-cmk-uri` parameters, you must first perform the following actions:

1. Authorize the **Machine Learning App** (in Identity and Access Management) with contributor permissions on your subscription.
2. 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

You do not need to manually create the Azure Cosmos DB instance, one will be created for you during workspace creation. This Azure Cosmos DB instance will be created in a separate resource group using a name based on this pattern: `<your-resource-group-name>_<GUID>`.

IMPORTANT

The Cosmos DB instance is created in a Microsoft-managed resource group in **your subscription**, along with any resources it needs. This means that you are charged for this Cosmos DB instance. The managed resource group is named in the format `<AML Workspace Resource Group Name>_<GUID>`. If your Azure Machine Learning workspace uses a private endpoint, a virtual network is also created for the Cosmos DB instance. This VNet is used to secure communication between Cosmos DB and Azure Machine Learning.

- **Do not delete the resource group** that contains this Cosmos DB instance, or any of the resources automatically created in this group. If you need to delete the resource group, Cosmos DB instance, etc., you must delete the Azure Machine Learning workspace that uses it. The resource group, Cosmos DB instance, and other automatically created resources are deleted when the associated workspace is deleted.
- The default **Request Units** for this Cosmos DB account is set at **8000**.
- You cannot provide your own VNet for use with the Cosmos DB instance that is created. You also cannot modify the virtual network. For example, you cannot change the IP address range that it uses.

To limit the data that Microsoft collects on your workspace, use the `--hbi-workspace` parameter.

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](#).

Use existing resources

To create a workspace that uses existing resources, you must provide the ID for the resources. Use the following commands to get the ID for the services:

IMPORTANT

You don't have to specify all existing resources. You can specify one or more. For example, you can specify an existing storage account and the workspace will create the other resources.

- **Azure Storage Account:** `az storage account show --name <storage-account-name> --query "id"`

The response from this command is similar to the following text, and is the ID for your storage account:

```
"/subscriptions/<service-GUID>/resourceGroups/<resource-group-name>/providers/Microsoft.Storage/storageAccounts/<storage-account-name>"
```

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. You can use premium storage or hierarchical namespace with *non-default* storage accounts.

- **Azure Application Insights:**

1. Install the application insights extension:

```
az extension add -n application-insights
```

2. Get the ID of your application insight service:

```
az monitor app-insights component show --app <application-insight-name> -g <resource-group-name> --query "id"
```

The response from this command is similar to the following text, and is the ID for your application insights service:

```
"/subscriptions/<service-GUID>/resourceGroups/<resource-group-name>/providers/microsoft.insights/components/<application-insight-name>"
```

- **Azure Key Vault:** `az keyvault show --name <key-vault-name> --query "ID"`

The response from this command is similar to the following text, and is the ID for your key vault:

```
"/subscriptions/<service-GUID>/resourceGroups/<resource-group-name>/providers/Microsoft.KeyVault/vaults/<key-vault-name>"
```

- **Azure Container Registry:** `az acr show --name <acr-name> -g <resource-group-name> --query "id"`

The response from this command is similar to the following text, and is the ID for the container registry:

```
"/subscriptions/<service-GUID>/resourceGroups/<resource-group-name>/providers/Microsoft.ContainerRegistry/registries/<acr-name>"
```

IMPORTANT

The container registry must have the the [admin account](#) enabled before it can be used with an Azure Machine Learning workspace.

Once you have the IDs for the resource(s) that you want to use with the workspace, use the base

`az workspace create -w <workspace-name> -g <resource-group-name>` command and add the parameter(s) and ID(s) for the existing resources. For example, the following command creates a workspace that uses an existing container registry:

```
az ml workspace create -w <workspace-name> -g <resource-group-name> --container-registry
"/subscriptions/<service-GUID>/resourceGroups/<resource-group-name>/providers/Microsoft.ContainerRegistry/registries/<acr-name>"
```

The output of this command is similar to the following JSON:

```
{  
    "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>"  
}
```

List workspaces

To list all the workspaces for your Azure subscription, use the following command:

```
az ml workspace list
```

The output of this command is similar to the following JSON:

```
[  
  {  
    "resourceGroup": "myresourcegroup",  
    "subscriptionId": "<subscription-id>",  
    "workspaceName": "myml"  
  },  
  {  
    "resourceGroup": "anotherresourcegroup",  
    "subscriptionId": "<subscription-id>",  
    "workspaceName": "anotherml"  
  }  
]
```

For more information, see the [az ml workspace list](#) documentation.

Get workspace information

To get information about a workspace, use the following command:

```
az ml workspace show -w <workspace-name> -g <resource-group-name>
```

The output of this command is similar to the following JSON:

```
{  
  "applicationInsights": "/subscriptions/<service-GUID>/resourcegroups/<resource-group-name>/providers/microsoft.insights/components/application-insight-name>",  
  "creationTime": "2019-08-30T18:55:03.1807976+00:00",  
  "description": "",  
  "friendlyName": "",  
  "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>",  
  "tags": {},  
  "type": "Microsoft.MachineLearningServices/workspaces",  
  "workspaceid": "<GUID>"  
}
```

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 -w <workspace-name> -g <resource-group-name>
```

The output of this command is similar to the following JSON:

```
{  
  "applicationInsights": "/subscriptions/<service-GUID>/resourcegroups/<resource-group-name>/providers/microsoft.insights/components/application-insight-name>",  
  "creationTime": "2019-08-30T18:55:03.1807976+00:00",  
  "description": "",  
  "friendlyName": "",  
  "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>",  
  "tags": {},  
  "type": "Microsoft.MachineLearningServices/workspaces",  
  "workspaceid": "<GUID>"  
}
```

For more information, see the [az ml workspace update](#) documentation.

Share a workspace with another user

To share a workspace with another user on your subscription, use the following command:

```
az ml workspace share -w <workspace-name> -g <resource-group-name> --user <user> --role <role>
```

For more information on Azure role-based access control (Azure RBAC) with Azure Machine Learning, see [Manage users and roles](#).

For more information, see the [az ml workspace share](#) 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 -w <workspace-name> -g <resource-group-name>
```

For more information on changing keys, see [Regenerate storage access keys](#).

For more information, see the [az ml workspace sync-keys](#) documentation.

Delete a workspace

To delete a workspace after it is no longer needed, use the following command:

```
az ml workspace delete -w <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.

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.

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.

Create, run, and delete Azure ML resources using REST

12/23/2020 • 9 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 DELETE requests to clean up resources
- Use key-based authorization to score deployed models

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 curly 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 subsequent administrative requests. You'll do so by setting an Authorization header in all requests:

```
curl -h "Authentication: Bearer {your-access-token}" ...more args...
```

Note that 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=2019-11-01 -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. The `api-version` argument varies from service to service. For the Machine Learning Service, for instance, the current API version is `2019-11-01`. For storage accounts, it's `2019-06-01`. For key vaults, it's `2019-09-01`. 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, substituting `{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=2019-11-01 \  
-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/myworkspace02
75623111",
    "containerRegistry": null,
    "keyVault": "/subscriptions/12345abc-abbc-1b2b-1234-
57ab575a5a5a/resourcegroups/DeepLearningResourceGroup/providers/microsoft.keyvault/vaults/myworkspace2525649324
",
    "applicationInsights": "/subscriptions/12345abc-abbc-1b2b-1234-
57ab575a5a5a/resourcegroups/DeepLearningResourceGroup/providers/microsoft.insights/components/myworkspace205352
3719",
    "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 additional 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://{{regiona-api-server}}/history/v1.0/subscriptions/{{your-subscription-id}}/resourceGroups/{{your-resource-group}}/\\
providers/Microsoft.MachineLearningServices/workspaces/{{your-workspace-name}}/experiments?api-version=2019-11-01 \\
-H "Authorization:Bearer {{your-access-token}}"
```

Similarly, to retrieve registered models in your workspace, send:

```
curl https://{{regiona-api-server}}/modelmanagement/v1.0/subscriptions/{{your-subscription-id}}/resourceGroups/{{your-resource-group}}/\\
providers/Microsoft.MachineLearningServices/workspaces/{{your-workspace-name}}/models?api-version=2019-11-01 \\
-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/MLWorkspace/MyWorkspace/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=2019-11-01 \
-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 substitutions of `your-subscription-id`, `your-resource-group`, `your-workspace-name`, and `your-access-token`, substitute `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=2019-11-01' \
-H 'Authorization:Bearer {your-access-token}' \
-H 'Content-Type: application/json' \
-d '{
  "location": "{your-azure-location}",
  "properties": {
    "computeType": "AmlCompute",
    "properties": {
      "vmSize": "Standard_D1",
      "vmPriority": "Dedicated",
      "scaleSettings": {
        "maxNodeCount": 1,
        "minNodeCount": 0,
        "nodeIdleTimeBeforeScaleDown": "PT30M"
      }
    }
  },
  "userAccountCredentials": {
    "adminUserName": "{adminUserName}",
    "adminUserPassword": "{adminUserPassword}"
  }
}'
```

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 an experimental run

To start a run within an experiment, you need a zip folder containing your training script and related files, and a run definition JSON file. The zip folder must have the Python entry file in its root directory. As an example, zip a trivial Python program such as the following into a folder called `train.zip`.

```
# hello.py
# Entry file for run
print("Hello, REST!")
```

Save this next snippet as **definition.json**. Confirm the "Script" value matches the name of the Python file you just zipped up. Confirm the "Target" value matches the name of an available compute resource.

```
{  
    "Configuration":{  
        "Script":"hello.py",  
        "Arguments": [  
            "234"  
        ],  
        "SourceDirectoryDataStore":null,  
        "Framework":"Python",  
        "Communicator":"None",  
        "Target":"cpu-compute",  
        "MaxRunDurationSeconds":1200,  
        "NodeCount":1,  
        "Environment":{  
            "Python":{  
                "InterpreterPath":"python",  
                "UserManagedDependencies":false,  
                "CondaDependencies":{  
                    "name":"project_environment",  
                    "dependencies": [  
                        "python=3.6.2",  
                        {  
                            "pip": [  
                                "azureml-defaults"  
                            ]  
                        }  
                    ]  
                }  
            },  
            "Docker":{  
                "BaseImage":"mcr.microsoft.com/azureml/base:intelmpi2018.3-ubuntu16.04"  
            }  
        },  
        "History":{  
            "OutputCollection":true  
        }  
    }  
}
```

Post these files to the server using `multipart/form-data` content:

```
curl https://{{regiona-1-api-server}}/execution/v1.0/subscriptions/{{your-subscription-id}}/resourceGroups/{{your-resource-group}}/providers/Microsoft.MachineLearningServices/workspaces/{{your-workspace-name}}/experiments/{{your-experiment-name}}/startrun?api-version=2019-11-01 \  
-X POST \  
-H "Content-Type: multipart/form-data" \  
-H "Authorization: Bearer {{your-access-token}}" \  
-F projectZipFile=@train.zip \  
-F runDefinitionFile=@runDefinition.json
```

A successful POST request will generate a `200 OK` status, with a response body containing the identifier of the created run:

```
{  
    "runId": "my-first-experiment_1579642222877"  
}
```

You can monitor a run using the REST-ful pattern that should now be familiar:

```
curl 'https://{{regional-api-server}}/history/v1.0/subscriptions/{{your-subscription-id}}/resourceGroups/{{your-resource-group}}/providers/Microsoft.MachineLearningServices/workspaces/{{your-workspace-name}}/experiments/{{your-experiment-names}}/runs/{{your-run-id}}?api-version=2019-11-01' \
-H 'Authorization:Bearer {{your-access-token}}'
```

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=2019-11-01' \
-H 'Authorization:Bearer {{your-access-token}}'
```

Use REST to score a deployed model

While it's possible to deploy a model so that it authenticates with a service principal, most client-facing deployments use key-based authentication. You can find the appropriate key in your deployment's page within the **Endpoints** tab of Studio. The same location will show your endpoint's scoring URI. Your model's inputs must be modeled as a JSON array named `data`:

```
curl 'https://{{scoring-uri}}' \
-H 'Authorization:Bearer {{your-key}}' \
-H 'Content-Type: application/json' \
-d '{ "data" : [ {{model-specific-data-structure}} ] }'
```

Create a workspace using REST

Every Azure ML workspace has a dependency on four other Azure resources: a container registry with administration enabled, a key vault, an Application Insights resource, and a storage account. You cannot 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=2019-11-01' \
-H 'Authorization: Bearer {your-access-token}' \
-H 'Content-Type: application/json' \
-d '{
  "location": "{desired-region}",
  "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}"
  },
  "identity" : {
    "type" : "systemAssigned"
  }
}'

```

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 is a problem with one of your dependent resources (for instance, if you forgot to enable admin access on your container registry).

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.

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](#).

Use an Azure Resource Manager template to create a workspace for Azure Machine Learning

12/23/2020 • 16 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](#).
- Some scenarios require you to open a support ticket. These scenarios are:
 - [Private Link enabled workspace with a customer-managed key](#)
 - [Azure Container Registry for the workspace behind your virtual network](#)

For more information, see [Manage and increase quotas](#).

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.MachineLearningService](#). 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.

Workspace Resource Manager template

The Azure Resource Manager template used throughout this document can be found in the [201-machine-learning-advanced](#) 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. However, the container registry you use must have the **admin account** enabled. For information on enabling the admin account, see [Admin account](#).

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/201-machine-
learning-advanced/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/201-machine-
learning-advanced/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

The Cosmos DB instance is created in a Microsoft-managed resource group in **your subscription**, along with any resources it needs. This means that you are charged for this Cosmos DB instance. The managed resource group is named in the format <AML Workspace Resource Group Name><GUID>. If your Azure Machine Learning workspace uses a private endpoint, a virtual network is also created for the Cosmos DB instance. This VNet is used to secure communication between Cosmos DB and Azure Machine Learning.

- **Do not delete the resource group** that contains this Cosmos DB instance, or any of the resources automatically created in this group. If you need to delete the resource group, Cosmos DB instance, etc., you must delete the Azure Machine Learning workspace that uses it. The resource group, Cosmos DB instance, and other automatically created resources are deleted when the associated workspace is deleted.
- The default **Request Units** for this Cosmos DB account is set at **8000**.
- You **cannot provide your own VNet for use with the Cosmos DB instance** that is created. You also **cannot modify the virtual network**. For example, you cannot change the IP address range that it uses.

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](#).

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.

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/201-machine-
learning-advanced/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 runs.
- 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.

IMPORTANT

Using an Azure Machine Learning workspace with private link is not available in the Azure Government regions or Azure China 21Vianet regions.

- [Azure CLI](#)
- [Azure PowerShell](#)

```
az deployment group create \
    --name "exampledeployment" \
    --resource-group "examplegroup" \
    --template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-machine-
learning-advanced/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/201-machine-
learning-advanced/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/201-machine-learning-advanced/azuredeploy.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/201-machine-learning-advanced/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 **201-machine-learning-advanced** template from the dropdown.
2. Select **Select template** to use the template. 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.
3. Select **Review + create**.
4. 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.

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 run 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
```

Virtual network not linked to private DNS zone

When creating a workspace with a private endpoint, the template creates a Private DNS Zone named `privatelink.api.azureml.ms`. A **virtual network link** is automatically added to this private DNS zone. The link is only added for the first workspace and private endpoint you create in a resource group; if you create another virtual network and workspace with a private endpoint in the same resource group, the second virtual network may not get added to the private DNS zone.

To view the virtual network links that already exist for the private DNS zone, use the following Azure CLI command:

```
az network private-dns link vnet list --zone-name privatelink.api.azureml.ms --resource-group myresourcegroup
```

To add the virtual network that contains another workspace and private endpoint, use the following steps:

1. To find the virtual network ID for the network that you want to add, use the following command:

```
az network vnet show --name myvnet --resource-group myresourcegroup --query id
```

This command returns a value similar to `"/subscriptions/GUID/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/myvnet"`. Save this value and use it in the next step.

2. To add a virtual network link to the `privatelink.api.azureml.ms` Private DNS Zone, use the following command. For the `--virtual-network` parameter, use the output of the previous command:

```
az network private-dns link vnet create --name mylinkname --registration-enabled true --resource-group myresourcegroup --virtual-network myvirtualnetworkid --zone-name privatelink.api.azureml.ms
```

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](#)

Create and manage an Azure Machine Learning compute instance

12/23/2020 • 7 minutes to read • [Edit Online](#)

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 cannot 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
- Access the terminal window
- Install R or Python packages
- Create new environments or Jupyter kernels

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.

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- The [Azure CLI extension for Machine Learning service](#), [Azure Machine Learning Python SDK](#), or the [Azure Machine Learning Visual Studio Code extension](#).

Create

Time estimate: Approximately 5 minutes.

Creating a compute instance is a one time process for your workspace. You can reuse this 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 does not release quota to ensure you will be able to restart the compute instance. Please note it is not possible to change the virtual machine size of compute instance once it is created.

The following example demonstrates how to create a compute instance:

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

```

import datetime
import time

from azureml.core.compute import ComputeTarget, ComputeInstance
from azureml.core.compute_target import ComputeTargetException

# Choose a name for your instance
# Compute instance name should be unique across the azure region
compute_name = "ci{}".format(ws._workspace_id)[:10]

# Verify that instance does not exist already
try:
    instance = ComputeInstance(workspace=ws, name=compute_name)
    print('Found existing instance, use it.')
except ComputeTargetException:
    compute_config = ComputeInstance.provisioning_configuration(
        vm_size='STANDARD_D3_V2',
        ssh_public_access=False,
        # vnet_resourcegroup_name='<my-resource-group>',
        # vnet_name='<my-vnet-name>',
        # subnet_name='default',
        # admin_user_ssh_public_key='<my-sshkey>'
    )
    instance = ComputeInstance.create(ws, compute_name, compute_config)
    instance.wait_for_completion(show_output=True)

```

For more information on the classes, methods, and parameters used in this example, see the following reference documents:

- [ComputeInstance class](#)
- [ComputeTarget.create](#)
- [ComputeInstance.wait_for_completion](#)

You can also create a compute instance with an [Azure Resource Manager template](#).

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:

- [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*

The data scientist can start, stop, and restart the compute instance. They can use the compute instance for:

- Jupyter
- JupyterLab
- RStudio
- Integrated notebooks

Manage

Start, stop, restart and delete a compute instance. A compute instance does not automatically scale down, so make sure to stop the resource to prevent ongoing charges.

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

In the examples below, the name of the compute instance is `instance`

- Get status

```
# get_status() gets the latest status of the ComputeInstance target
instance.get_status()
```

- Stop

```
# stop() is used to stop the ComputeInstance
# Stopping ComputeInstance will stop the billing meter and persist the state on the disk.
# Available Quota will not be changed with this operation.
instance.stop(wait_for_completion=True, show_output=True)
```

- Start

```
# start() is used to start the ComputeInstance if it is in stopped state
instance.start(wait_for_completion=True, show_output=True)
```

- Restart

```
# restart() is used to restart the ComputeInstance
instance.restart(wait_for_completion=True, show_output=True)
```

- Delete

```
# delete() is used to delete the ComputeInstance target. Useful if you want to re-use the compute name
instance.delete(wait_for_completion=True, show_output=True)
```

[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, and can terminal in through Jupyter/JupyterLab/RStudio. Compute instance will have single-user log in and all actions will use that user's identity for Azure RBAC and attribution of experiment runs. 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](#)

Access the terminal window

Open the terminal window of your compute instance in any of these ways:

- 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.

Use the terminal window to install packages and create additional kernels.

Install packages

You can install packages directly in Jupyter Notebook or RStudio:

- 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.

Or you can install from a terminal window. Install Python packages into the **Python 3.6 - AzureML** environment. Install R packages into the **R** environment.

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` conda environment or **Python 3.6 - AzureML** kernel. This is needed for Jupyter/JupyterLab functionality

To add a new Jupyter kernel to the compute instance:

1. Create new terminal from Jupyter, JupyterLab or from notebooks pane or SSH into the compute instance
2. Use the terminal window to create a new environment. For example, the code below creates `newenv`:

```
conda create --name newenv
```

3. Activate the environment. For example, after creating `newenv`:

```
conda activate newenv
```

4. 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.

Next steps

- [Submit a training run](#)

Create an Azure Machine Learning compute cluster

12/23/2020 • 7 minutes to read • [Edit Online](#)

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](#), [Azure Machine Learning Python SDK](#), or the [Azure Machine Learning Visual Studio Code extension](#).

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 is created within your workspace region as 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. 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

- **Do not create multiple, simultaneous attachments to the same compute** from your workspace. For example, attaching one compute cluster to a workspace using two different names. Each new attachment will break the previous existing attachment(s).

If you want to re-attach a compute target, for example to change cluster configuration settings, you must first remove the existing attachment.

- 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](#).
- 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 cannot 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](#)
- [Azure CLI](#)
- [Studio](#)

To create a persistent Azure Machine Learning Compute resource in Python, specify the `vm_size` and `max_nodes` properties. Azure Machine Learning then uses smart defaults for the other properties.

- **vm_size:** The VM family of the nodes created by Azure Machine Learning Compute.
- **max_nodes:** The max number of nodes to autoscale up to when you run a job on Azure Machine Learning Compute.

```

from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

# Choose a name for your CPU cluster
cpu_cluster_name = "cpocluster"

# Verify that cluster does not exist already
try:
    cpu_cluster = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print('Found existing cluster, use it.')
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                           max_nodes=4)
    cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

cpu_cluster.wait_for_completion(show_output=True)

```

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.

Lower your compute cluster cost

You may also choose to use [low-priority VMs](#) to run some or all of your workloads. These VMs do not have guaranteed availability and may be preempted while in use. A preempted job is restarted, not resumed.

Use any of these ways to specify a low-priority VM:

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

```

compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                       vm_priority='lowpriority',
                                                       max_nodes=4)

```

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.
- [Python](#)
 - [Azure CLI](#)
 - [Studio](#)
- Configure managed identity in your provisioning configuration:
 - System assigned managed identity:

```
# configure cluster with a system-assigned managed identity
compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                       max_nodes=5,
                                                       identity_type="SystemAssigned",
                                                       )
```

- User-assigned managed identity:

```
# configure cluster with a user-assigned managed identity
compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                       max_nodes=5,
                                                       identity_type="UserAssigned",
                                                       identity_id=
                                                       ['/subscriptions/<subscription_id>/resourcegroups/<resource_group>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<user_assigned_identity>'])

cpu_cluster_name = "cpu-cluster"
cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)
```

- Add managed identity to an existing compute cluster

- System-assigned managed identity:

```
# add a system-assigned managed identity
cpu_cluster.add_identity(identity_type="SystemAssigned")
```

- User-assigned managed identity:

```
# add a user-assigned managed identity
cpu_cluster.add_identity(identity_type="UserAssigned",
                        identity_id=
                        ['/subscriptions/<subscription_id>/resourcegroups/<resource_group>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<user_assigned_identity>'])
```

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 is 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.

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.

For example, applying a delete lock to the resource group for your workspace will prevent scaling operations for Azure ML compute clusters.

For more information on locking resources, see [Lock resources to prevent unexpected changes](#).

Next steps

Use your compute cluster to:

- [Submit a training run](#)
- [Run batch inference.](#)

Create and attach an Azure Kubernetes Service cluster

12/23/2020 • 11 minutes to read • [Edit Online](#)

Azure Machine Learning can deploy trained machine learning models to Azure Kubernetes Service. However, you must first either **create** an Azure Kubernetes Service (AKS) cluster from your Azure ML workspace, or **attach** an existing AKS cluster. This article provides information on both creating and attaching a 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](#), [Azure Machine Learning Python SDK](#), or the [Azure Machine Learning Visual Studio Code extension](#).
- If you plan on using an Azure Virtual Network to secure communication between your Azure ML workspace and the AKS cluster, read the [Network isolation during training & inference](#) article.

Limitations

- If you need a **Standard Load Balancer(SLB)** deployed in your cluster instead of a Basic Load Balancer(BLB), create a cluster in the AKS portal/CLI/SDK and then **attach** it to the AML workspace.
- If you have an Azure Policy that restricts the creation of Public IP addresses, then AKS cluster creation will fail. AKS requires a Public IP for [egress traffic](#). The egress traffic article also provides guidance to lock down egress traffic from the cluster through the Public IP, except for a few fully qualified domain names. There are 2 ways to enable a Public IP:
 - The cluster can use the Public IP created by default with the BLB or SLB, Or
 - The cluster can be created without a Public IP and then a Public IP is configured with a firewall with a user defined route. For more information, see [Customize cluster egress with a user-defined-route](#).

The AML control plane does not talk to this Public IP. It talks to the AKS control plane for deployments.

- If you **attach** an AKS cluster, which has an [Authorized IP range enabled to access the API server](#), enable the AML control plane IP ranges for the AKS cluster. The AML control plane is deployed across paired regions and deploys inference pods on the AKS cluster. Without access to the API server, the inference pods cannot be deployed. Use the [IP ranges](#) for both the [paired regions](#) when enabling the IP ranges in an AKS cluster.

Authorized IP ranges only works with Standard Load Balancer.

- When **attaching** an AKS cluster, it must be in the same Azure subscription as your Azure Machine Learning workspace.
- If you want to use a private AKS cluster (using Azure Private Link), you must create the cluster first, and then **attach** it to the workspace. For more information, see [Create a private Azure Kubernetes Service cluster](#).
- The compute name for the AKS cluster MUST be unique within your Azure ML workspace.
 - Name is required and must be between 3 to 24 characters long.
 - Valid characters are upper and lower case letters, digits, and the - character.

- Name must start with a letter.
- Name needs to be unique across all existing computes within an Azure region. You will see an alert if the name you choose is not unique.
- If you want to deploy models to **GPU** nodes or **FPGA** nodes (or any specific SKU), then you must create a cluster with the specific SKU. There is no support for creating a secondary node pool in an existing cluster and deploying models in the secondary node pool.
- When creating or attaching a cluster, you can select whether to create the cluster for **dev-test** or **production**. If you want to create an AKS cluster for **development**, **validation**, and **testing** instead of production, set the **cluster purpose** to **dev-test**. If you do not specify the cluster purpose, a **production** cluster is created.

IMPORTANT

A **dev-test** cluster is not suitable for production level traffic and may increase inference times. Dev/test clusters also do not guarantee fault tolerance.

- When creating or attaching a cluster, if the cluster will be used for **production**, then it must contain at least **12 virtual CPUs**. The number of virtual CPUs can be calculated by multiplying the **number of nodes** in the cluster by the **number of cores** provided by the VM size selected. For example, if you use a VM size of "Standard_D3_v2", which has 4 virtual cores, then you should select 3 or greater as the number of nodes.

For a **dev-test** cluster, we recommend at least 2 virtual CPUs.

- The Azure Machine Learning SDK does not provide support scaling an AKS cluster. To scale the nodes in the cluster, use the UI for your AKS cluster in the Azure Machine Learning studio. You can only change the node count, not the VM size of the cluster. For more information on scaling the nodes in an AKS cluster, see the following articles:
 - [Manually scale the node count in an AKS cluster](#)
 - [Set up cluster autoscaler in AKS](#)

Azure Kubernetes Service version

Azure Kubernetes Service allows you to create a cluster using a variety of Kubernetes versions. For more information on available versions, see [supported Kubernetes versions in Azure Kubernetes Service](#).

When **creating** an Azure Kubernetes Service cluster using one of the following methods, you *do not have a choice in the version* of the cluster that is created:

- Azure Machine Learning studio, or the Azure Machine Learning section of the Azure portal.
- Machine Learning extension for Azure CLI.
- Azure Machine Learning SDK.

These methods of creating an AKS cluster use the **default** version of the cluster. *The default version changes over time* as new Kubernetes versions become available.

When **attaching** an existing AKS cluster, we support all currently supported AKS versions.

NOTE

There may be edge cases where you have an older cluster that is no longer supported. In this case, the attach operation will return an error and list the currently supported versions.

You can attach **preview** versions. 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. Support for using preview versions may be limited. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Available and default versions

To find the available and default AKS versions, use the [Azure CLI](#) command `az aks get-versions`. For example, the following command returns the versions available in the West US region:

```
az aks get-versions -l westus -o table
```

The output of this command is similar to the following text:

KubernetesVersion	Upgrades
1.18.6(preview)	None available
1.18.4(preview)	1.18.6(preview)
1.17.9	1.18.4(preview), 1.18.6(preview)
1.17.7	1.17.9, 1.18.4(preview), 1.18.6(preview)
1.16.13	1.17.7, 1.17.9
1.16.10	1.16.13, 1.17.7, 1.17.9
1.15.12	1.16.10, 1.16.13
1.15.11	1.15.12, 1.16.10, 1.16.13

To find the default version that is used when **creating** a cluster through Azure Machine Learning, you can use the `--query` parameter to select the default version:

```
az aks get-versions -l westus --query "orchestrators[?default == `true`].orchestratorVersion" -o table
```

The output of this command is similar to the following text:

```
Result
-----
1.16.13
```

If you'd like to **programmatically check the available versions**, use the [Container Service Client - List Orchestrators](#) REST API. To find the available versions, look at the entries where `orchestratorType` is `Kubernetes`. The associated `orchestrationVersion` entries contain the available versions that can be **attached** to your workspace.

To find the default version that is used when **creating** a cluster through Azure Machine Learning, find the entry where `orchestratorType` is `Kubernetes` and `default` is `true`. The associated `orchestratorVersion` value is the default version. The following JSON snippet shows an example entry:

```

...
{
    "orchestratorType": "Kubernetes",
    "orchestratorVersion": "1.16.13",
    "default": true,
    "upgrades": [
        {
            "orchestratorType": "",
            "orchestratorVersion": "1.17.7",
            "isPreview": false
        }
    ]
},
...

```

Create a new AKS cluster

Time estimate: Approximately 10 minutes.

Creating or attaching an AKS cluster is a one time process for your workspace. You can reuse this cluster for multiple deployments. If you delete the cluster or the resource group that contains it, you must create a new cluster the next time you need to deploy. You can have multiple AKS clusters attached to your workspace.

The following example demonstrates how to create a new AKS cluster using the SDK and CLI:

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

```

from azureml.core.compute import AksCompute, ComputeTarget

# Use the default configuration (you can also provide parameters to customize this).
# For example, to create a dev/test cluster, use:
# prov_config = AksCompute.provisioning_configuration(cluster_purpose = AksCompute.ClusterPurpose.DEV_TEST)
prov_config = AksCompute.provisioning_configuration()

# Example configuration to use an existing virtual network
# prov_config.vnet_name = "mynetwork"
# prov_config.vnet_resourcegroup_name = "mygroup"
# prov_config.subnet_name = "default"
# prov_config.service_cidr = "10.0.0.0/16"
# prov_config.dns_service_ip = "10.0.0.10"
# prov_config.docker_bridge_cidr = "172.17.0.1/16"

aks_name = 'myaks'
# Create the cluster
aks_target = ComputeTarget.create(workspace = ws,
                                   name = aks_name,
                                   provisioning_configuration = prov_config)

# Wait for the create process to complete
aks_target.wait_for_completion(show_output = True)

```

For more information on the classes, methods, and parameters used in this example, see the following reference documents:

- [AksCompute.ClusterPurpose](#)
- [AksCompute.provisioning_configuration](#)
- [ComputeTarget.create](#)
- [ComputeTarget.wait_for_completion](#)

Attach an existing AKS cluster

Time estimate: Approximately 5 minutes.

If you already have AKS cluster in your Azure subscription, and it is version 1.17 or lower, you can use it to deploy your image.

TIP

The existing AKS cluster can be in a Azure region other than your Azure Machine Learning workspace.

WARNING

Do not create multiple, simultaneous attachments to the same AKS cluster from your workspace. For example, attaching one AKS cluster to a workspace using two different names. Each new attachment will break the previous existing attachment(s).

If you want to re-attach an AKS cluster, for example to change TLS or other cluster configuration setting, you must first remove the existing attachment by using [AksCompute.detach\(\)](#).

For more information on creating an AKS cluster using the Azure CLI or portal, see the following articles:

- [Create an AKS cluster \(CLI\)](#)
- [Create an AKS cluster \(portal\)](#)
- [Create an AKS cluster \(ARM Template on Azure Quickstart templates\)](#)

The following example demonstrates how to attach an existing AKS cluster to your workspace:

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

```
from azureml.core.compute import AksCompute, ComputeTarget
# Set the resource group that contains the AKS cluster and the cluster name
resource_group = 'myresourcegroup'
cluster_name = 'myexistingcluster'

# Attach the cluster to your workgroup. If the cluster has less than 12 virtual CPUs, use the following instead:
# attach_config = AksCompute.attach_configuration(resource_group = resource_group,
#                                                 cluster_name = cluster_name,
#                                                 cluster_purpose = AksCompute.ClusterPurpose.DEV_TEST)
attach_config = AksCompute.attach_configuration(resource_group = resource_group,
                                                cluster_name = cluster_name)
aks_target = ComputeTarget.attach(ws, 'myaks', attach_config)

# Wait for the attach process to complete
aks_target.wait_for_completion(show_output = True)
```

For more information on the classes, methods, and parameters used in this example, see the following reference documents:

- [AksCompute.attach_configuration\(\)](#)
- [AksCompute.ClusterPurpose](#)
- [AksCompute.attach](#)

Detach an AKS cluster

To detach a cluster from your workspace, use one of the following methods:

WARNING

Using the Azure Machine Learning studio, SDK, or the Azure CLI extension for machine learning to detach an AKS cluster **does not delete the AKS cluster**. To delete the cluster, see [Use the Azure CLI with AKS](#).

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

```
aks_target.detach()
```

Troubleshooting

Update the cluster

Updates to Azure Machine Learning components installed in an Azure Kubernetes Service cluster must be manually applied.

You can apply these updates by detaching the cluster from the Azure Machine Learning workspace, and then reattaching the cluster to the workspace. If TLS is enabled in the cluster, you will need to supply the TLS/SSL certificate and private key when reattaching the cluster.

```
compute_target = ComputeTarget(workspace=ws, name=clusterWorkspaceName)
compute_target.detach()
compute_target.wait_for_completion(show_output=True)

attach_config = AksCompute.attach_configuration(resource_group=resourceGroup,
cluster_name=kubernetesClusterName)

## If SSL is enabled.
attach_config.enable_ssl(
    ssl_cert_pem_file="cert.pem",
    ssl_key_pem_file="key.pem",
    ssl_cname=sslCname)

attach_config.validate_configuration()

compute_target = ComputeTarget.attach(workspace=ws, name=args.clusterWorkspaceName,
attach_configuration=attach_config)
compute_target.wait_for_completion(show_output=True)
```

If you no longer have the TLS/SSL certificate and private key, or you are using a certificate generated by Azure Machine Learning, you can retrieve the files prior to detaching the cluster by connecting to the cluster using `kubectl` and retrieving the secret `azuremlfessl`.

```
kubectl get secret/azuremlfessl -o yaml
```

NOTE

Kubernetes stores the secrets in base-64 encoded format. You will need to base-64 decode the `cert.pem` and `key.pem` components of the secrets prior to providing them to `attach_config.enable_ssl`.

Webservice failures

Many webservice failures in AKS can be debugged by connecting to the cluster using `kubectl`. You can get the `kubeconfig.json` for an AKS cluster by running

```
az aks get-credentials -g <rg> -n <aks cluster name>
```

Next steps

- [Use Azure RBAC for Kubernetes authorization](#)
- [How and where to deploy a model](#)
- [Deploy a model to an Azure Kubernetes Service cluster](#)

Set up compute targets for model training and deployment

12/23/2020 • 10 minutes to read • [Edit Online](#)

Learn how to attach Azure compute resources to your Azure Machine Learning workspace. Then you can use these resources as training and inference [compute targets](#) in your machine learning tasks.

In this article, learn how to set up your workspace to use these compute resources:

- Your local computer
- Remote virtual machines
- Azure HDInsight
- Azure Batch
- Azure Databricks
- Azure Data Lake Analytics
- Azure Container Instance

To use compute targets managed by Azure Machine Learning, see:

- [Azure Machine Learning compute instance](#)
- [Azure Machine Learning compute cluster](#)
- [Azure Kubernetes Service 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](#), [Azure Machine Learning Python SDK](#), or the [Azure Machine Learning Visual Studio Code extension](#).

Limitations

- **Do not create multiple, simultaneous attachments to the same compute** from your workspace. For example, attaching one Azure Kubernetes Service cluster to a workspace using two different names. Each new attachment will break the previous existing attachment(s).

If you want to reattach a compute target, for example to change TLS or other cluster configuration setting, you must first remove the existing attachment.

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. You also use compute targets for model deployment as described in "[Where and how to deploy your models](#)".

Local computer

When you use your local computer for [training](#), there is no need to create a compute target. Just [submit the](#)

training run from your local machine.

When you use your local computer for inference, you must have Docker installed. To perform the deployment, use [LocalWebservice.deploy_configuration\(\)](#) to define the port that the web service will use. Then use the normal deployment process as described in [Deploy models with Azure Machine Learning](#).

Remote virtual machines

Azure Machine Learning also supports bringing your own compute resource and attaching it to your workspace. One such resource type is an arbitrary remote VM, as long as it's accessible from Azure Machine Learning. The resource can be an Azure VM, a remote server in your organization, or on-premises. Specifically, given the IP address and credentials (user name and password, or SSH key), you can use any accessible VM for remote runs.

You can use a system-built conda environment, an already existing Python environment, or a Docker container. To execute on a Docker container, you must have a Docker Engine running on the VM. This functionality is especially useful when you want a more flexible, cloud-based dev/experimentation environment than your local machine.

Use the Azure Data Science Virtual Machine (DSVM) as the Azure VM of choice for this scenario. This VM is a pre-configured data science and AI development environment in Azure. The VM offers a curated choice of tools and frameworks for full-lifecycle machine learning development. For more information on how to use the DSVM with Azure Machine Learning, see [Configure a development environment](#).

1. **Create:** Create a DSVM before using it to train your model. To create this resource, see [Provision the Data Science Virtual Machine for Linux \(Ubuntu\)](#).

WARNING

Azure Machine Learning only supports virtual machines that run **Ubuntu**. When you create a VM or choose an existing VM, you must select a VM that uses Ubuntu.

Azure Machine Learning also requires the virtual machine to have a **public IP address**.

2. **Attach:** To attach an existing virtual machine as a compute target, you must provide the resource ID, user name, and password for the virtual machine. The resource ID of the VM can be constructed using the subscription ID, resource group name, and VM name using the following string format:

```
/subscriptions/<subscription_id>/resourceGroups/<resource_group>/providers/Microsoft.Compute/virtualMachines/<vm_name>
```

```
from azureml.core.compute import RemoteCompute, ComputeTarget

# Create the compute config
compute_target_name = "attach-dsvm"

attach_config = RemoteCompute.attach_configuration(resource_id='<resource_id>',
                                                   ssh_port=22,
                                                   username='<username>',
                                                   password="<password>")

# Attach the compute
compute = ComputeTarget.attach(ws, compute_target_name, attach_config)

compute.wait_for_completion(show_output=True)
```

Or you can attach the DSVM to your workspace [using Azure Machine Learning studio](#).

WARNING

Do not create multiple, simultaneous attachments to the same DSVM from your workspace. Each new attachment will break the previous existing attachment(s).

3. **Configure:** Create a run configuration for the DSVM compute target. Docker and conda are used to create and configure the training environment on the DSVM.

```
from azureml.core import ScriptRunConfig
from azureml.core.environment import Environment
from azureml.core.conda_dependencies import CondaDependencies

# Create environment
myenv = Environment(name="myenv")

# Specify the conda dependencies
myenv.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'])

# If no base image is explicitly specified the default CPU image
# "azureml.core.runconfig.DEFAULT_CPU_IMAGE" will be used
# To use GPU in DSVM, you should specify the default GPU base Docker image or another GPU-enabled image:
# myenv.docker.enabled = True
# myenv.docker.base_image = azureml.core.runconfig.DEFAULT_GPU_IMAGE

# Configure the run configuration with the Linux DSVM as the compute target and the environment defined
# above
src = ScriptRunConfig(source_directory=".", script="train.py", compute_target=compute,
environment=myenv)
```

Azure HDInsight

Azure HDInsight is a popular platform for big-data analytics. The platform provides Apache Spark, which can be used to train your model.

1. **Create:** Create the HDInsight cluster before you use it to train your model. To create a Spark on HDInsight cluster, see [Create a Spark Cluster in HDInsight](#).

WARNING

Azure Machine Learning requires the HDInsight cluster to have a **public IP address**.

When you create the cluster, you must specify an SSH user name and password. Take note of these values, as you need them to use HDInsight as a compute target.

After the cluster is created, connect to it with the hostname <clustername>-ssh.azurehdinsight.net, where <clustername> is the name that you provided for the cluster.

2. **Attach:** To attach an HDInsight cluster as a compute target, you must provide the resource ID, user name, and password for the HDInsight cluster. The resource ID of the HDInsight cluster can be constructed using the subscription ID, resource group name, and HDInsight cluster name using the following string format:

```
/subscriptions/<subscription_id>/resourceGroups/<resource_group>/providers/Microsoft.HDInsight/clusters/<cluster_name>
```

```

from azureml.core.compute import ComputeTarget, HDInsightCompute
from azureml.exceptions import ComputeTargetException

try:
    # if you want to connect using SSH key instead of username/password you can provide parameters
    private_key_file and private_key_passphrase

    attach_config = HDInsightCompute.attach_configuration(resource_id='<resource_id>',
                                                            ssh_port=22,
                                                            username='<ssh-username>',
                                                            password='<ssh-pwd>')

    hdi_compute = ComputeTarget.attach(workspace=ws,
                                       name='myhdi',
                                       attach_configuration=attach_config)

except ComputeTargetException as e:
    print("Caught = {}".format(e.message))

    hdi_compute.wait_for_completion(show_output=True)

```

Or you can attach the HDInsight cluster to your workspace [using Azure Machine Learning studio](#).

WARNING

Do not create multiple, simultaneous attachments to the same HDInsight from your workspace. Each new attachment will break the previous existing attachment(s).

3. Configure: Create a run configuration for the HDI compute target.

```

from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies

# use pyspark framework
run_hdi = RunConfiguration(framework="pyspark")

# Set compute target to the HDI cluster
run_hdi.target = hdi_compute.name

# specify CondaDependencies object to ask system installing numpy
cd = CondaDependencies()
cd.add_conda_package('numpy')
run_hdi.environment.python.conda_dependencies = cd

```

Now that you've attached the compute and configured your run, the next step is to [submit the training run](#).

Azure Batch

Azure Batch is used to run large-scale parallel and high-performance computing (HPC) applications efficiently in the cloud. AzureBatchStep can be used in an Azure Machine Learning Pipeline to submit jobs to an Azure Batch pool of machines.

To attach Azure Batch as a compute target, you must use the Azure Machine Learning SDK and provide the following information:

- **Azure Batch compute name:** A friendly name to be used for the compute within the workspace
- **Azure Batch account name:** The name of the Azure Batch account
- **Resource Group:** The resource group that contains the Azure Batch account.

The following code demonstrates how to attach Azure Batch as a compute target:

```
from azureml.core.compute import ComputeTarget, BatchCompute
from azureml.exceptions import ComputeTargetException

# Name to associate with new compute in workspace
batch_compute_name = 'mybatchcompute'

# Batch account details needed to attach as compute to workspace
batch_account_name = "<batch_account_name>" # Name of the Batch account
# Name of the resource group which contains this account
batch_resource_group = "<batch_resource_group>"

try:
    # check if the compute is already attached
    batch_compute = BatchCompute(ws, batch_compute_name)
except ComputeTargetException:
    print('Attaching Batch compute...')
    provisioning_config = BatchCompute.attach_configuration(
        resource_group=batch_resource_group, account_name=batch_account_name)
    batch_compute = ComputeTarget.attach(
        ws, batch_compute_name, provisioning_config)
    batch_compute.wait_for_completion()
    print("Provisioning state:{}".format(batch_compute.provisioning_state))
    print("Provisioning errors:{}".format(batch_compute.provisioning_errors))

print("Using Batch compute:{}".format(batch_compute.cluster_resource_id))
```

WARNING

Do not create multiple, simultaneous attachments to the same Azure Batch from your workspace. Each new attachment will break the previous existing attachment(s).

Azure Databricks

Azure Databricks is an Apache Spark-based environment in the Azure cloud. It can be used as a compute target with an Azure Machine Learning pipeline.

Create an Azure Databricks workspace before using it. To create a workspace resource, see the [Run a Spark job on Azure Databricks](#) document.

To attach Azure Databricks as a compute target, provide the following information:

- **Databricks compute name:** The name you want to assign to this compute resource.
- **Databricks workspace name:** The name of the Azure Databricks workspace.
- **Databricks access token:** The access token used to authenticate to Azure Databricks. To generate an access token, see the [Authentication](#) document.

The following code demonstrates how to attach Azure Databricks as a compute target with the Azure Machine Learning SDK (**The Databricks workspace need to be present in the same subscription as your AML workspace**):

```

import os
from azureml.core.compute import ComputeTarget, DatabricksCompute
from azureml.exceptions import ComputeTargetException

databricks_compute_name = os.environ.get(
    "AML_DATABRICKS_COMPUTE_NAME", "<databricks_compute_name>")
databricks_workspace_name = os.environ.get(
    "AML_DATABRICKS_WORKSPACE", "<databricks_workspace_name>")
databricks_resource_group = os.environ.get(
    "AML_DATABRICKS_RESOURCE_GROUP", "<databricks_resource_group>")
databricks_access_token = os.environ.get(
    "AML_DATABRICKS_ACCESS_TOKEN", "<databricks_access_token>")

try:
    databricks_compute = ComputeTarget(
        workspace=ws, name=databricks_compute_name)
    print('Compute target already exists')
except ComputeTargetException:
    print('compute not found')
    print('databricks_compute_name {}'.format(databricks_compute_name))
    print('databricks_workspace_name {}'.format(databricks_workspace_name))
    print('databricks_access_token {}'.format(databricks_access_token))

# Create attach config
attach_config = DatabricksCompute.attach_configuration(resource_group=databricks_resource_group,
                                                       workspace_name=databricks_workspace_name,
                                                       access_token=databricks_access_token)

databricks_compute = ComputeTarget.attach(
    ws,
    databricks_compute_name,
    attach_config
)

databricks_compute.wait_for_completion(True)

```

For a more detailed example, see an [example notebook](#) on GitHub.

WARNING

Do not create multiple, simultaneous attachments to the same Azure Databricks from your workspace. Each new attachment will break the previous existing attachment(s).

Azure Data Lake Analytics

Azure Data Lake Analytics is a big data analytics platform in the Azure cloud. It can be used as a compute target with an Azure Machine Learning pipeline.

Create an Azure Data Lake Analytics account before using it. To create this resource, see the [Get started with Azure Data Lake Analytics](#) document.

To attach Data Lake Analytics as a compute target, you must use the Azure Machine Learning SDK and provide the following information:

- **Compute name:** The name you want to assign to this compute resource.
- **Resource Group:** The resource group that contains the Data Lake Analytics account.
- **Account name:** The Data Lake Analytics account name.

The following code demonstrates how to attach Data Lake Analytics as a compute target:

```
import os
from azureml.core.compute import ComputeTarget, AdlaCompute
from azureml.exceptions import ComputeTargetException

adla_compute_name = os.environ.get(
    "AML_ADLA_COMPUTE_NAME", "<adla_compute_name>")
adla_resource_group = os.environ.get(
    "AML_ADLA_RESOURCE_GROUP", "<adla_resource_group>")
adla_account_name = os.environ.get(
    "AML_ADLA_ACCOUNT_NAME", "<adla_account_name>")

try:
    adla_compute = ComputeTarget(workspace=ws, name=adla_compute_name)
    print('Compute target already exists')
except ComputeTargetException:
    print('compute not found')
    print('adla_compute_name {}'.format(adla_compute_name))
    print('adla_resource_id {}'.format(adla_resource_group))
    print('adla_account_name {}'.format(adla_account_name))
    # create attach config
    attach_config = AdlaCompute.attach_configuration(resource_group=adla_resource_group,
                                                       account_name=adla_account_name)

    # Attach ADLA
    adla_compute = ComputeTarget.attach(
        ws,
        adla_compute_name,
        attach_config
    )

    adla_compute.wait_for_completion(True)
```

For a more detailed example, see an [example notebook](#) on GitHub.

WARNING

Do not create multiple, simultaneous attachments to the same ADLA from your workspace. Each new attachment will break the previous existing attachment(s).

TIP

Azure Machine Learning pipelines can only work with data stored in the default data store of the Data Lake Analytics account. If the data you need to work with is in a non-default store, you can use a [DataTransferStep](#) to copy the data before training.

Azure Container Instance

Azure Container Instances (ACI) are created dynamically when you deploy a model. You cannot create or attach ACI to your workspace in any other way. For more information, see [Deploy a model to Azure Container Instances](#).

Azure Kubernetes Service

Azure Kubernetes Service (AKS) allows for a variety of configuration options when used with Azure Machine Learning. For more information, see [How to create and attach Azure Kubernetes Service](#).

Notebook examples

See these notebooks for examples of training with various compute targets:

- [how-to-use-azureml/training](#)
- [tutorials/img-classification-part1-training.ipynb](#)

Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service](#).

Next steps

- Use the compute resource to [configure and submit a training run](#).
- [Tutorial: Train a model](#) uses a managed compute target to train a model.
- 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](#)

Create compute targets for model training and deployment in Azure Machine Learning studio

12/23/2020 • 8 minutes to read • [Edit Online](#)

In this article, learn how to create and manage compute targets in Azure Machine studio. You can also create and manage compute targets with:

- Azure Machine Learning Learning SDK or CLI extension for Azure Machine Learning
 - [Compute instance](#)
 - [Compute cluster](#)
 - [Azure Kubernetes Service cluster](#)
 - [Other compute resources](#)
- The [VS Code extension](#) for Azure Machine Learning.

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. You can also create compute targets for model deployment as described in "[Where and how to deploy your models](#)".

View compute targets

To see all compute targets for your workspace, use the following steps:

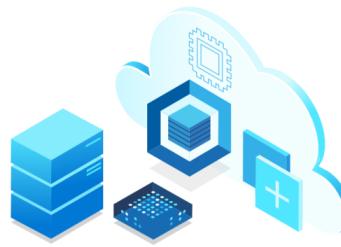
1. Navigate to [Azure Machine Learning studio](#).
2. Under **Manage**, select **Compute**.
3. Select tabs at the top to show each type of compute target.

Microsoft Azure Machine Learning

my-ws > Compute

Compute

Compute instances Compute clusters Inference clusters Attached compute



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](#)

Create

New Home Author Notebooks Automated ML Designer Assets Datasets Experiments Pipelines Models Endpoints Manage Compute Datastores Data Labeling

Create compute target

Follow the previous steps to view the list of compute targets. Then use these steps to create a compute target:

1. Select the tab at the top corresponding to the type of compute you will create.
2. If you have no compute targets, select **Create** in the middle of the page.

Microsoft Azure Machine Learning

my-ws > Compute

Compute

Compute instances Compute clusters Inference clusters Attached compute



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](#)

Create

New Home Author Notebooks Automated ML Designer Assets Datasets Experiments Pipelines Models Endpoints Manage Compute Datastores Data Labeling

3. If you see a list of compute resources, select **+ New** above the list.

Compute

Compute instances Compute clusters Inference clusters Attached compute

+ New Refresh Start Stop Restart Delete ... Search to filter items...

4. Fill out the form for your compute type:

- [Compute instance](#)
- [Compute clusters](#)
- [Inference clusters](#)
- [Attached compute](#)

1. Select **Create**.
2. View the status of the create operation by selecting the compute target from the list:

Compute

Compute instances	Compute clusters	Inference clusters	Attached compute
+ New Refresh Start Stop Restart Delete ... Search to filter items...			
Name	Status	Application URI	Virtual machine size
sdg-compute-instance	Creating		STANDARD_DS3_V2

Compute instance

Use the [steps above](#) to create the compute instance. Then fill out the form as follows:

New compute instance ×

Compute name *

Region *

Virtual machine type *

Virtual machine size *

Enable SSH access

Advanced settings

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

FIELD	DESCRIPTION
Compute name	<ul style="list-style-type: none"> • Name is required and must be between 3 to 24 characters long. • Valid characters are upper and lower case letters, digits, and the - character. • Name must start with a letter • Name needs to be unique across all existing computes within an Azure region. You will see an alert if the name you choose is not unique • If - character is used, then it needs to be followed by at least one letter later in the name
Virtual machine type	Choose CPU or GPU. This type cannot be changed after creation
Virtual machine size	Supported virtual machine sizes might be restricted in your region. Check the availability list
Enable/disable SSH access	SSH access is disabled by default. SSH access cannot be changed after creation. Make sure to enable access if you plan to debug interactively with VS Code Remote
Advanced settings	Optional. Configure a virtual network. Specify the Resource group , Virtual network , and Subnet to create the compute instance inside an Azure Virtual Network (vnet). For more information, see these network requirements for vnet.

Compute clusters

Create a single or multi node compute cluster for your training, batch inferencing or reinforcement learning workloads. Use the [steps above](#) to create the compute cluster. Then fill out the form as follows:

FIELD	DESCRIPTION
Compute name	<ul style="list-style-type: none"> • Name is required and must be between 3 to 24 characters long. • Valid characters are upper and lower case letters, digits, and the - character. • Name must start with a letter • Name needs to be unique across all existing computes within an Azure region. You will see an alert if the name you choose is not unique • If - character is used, then it needs to be followed by at least one letter later in the name
Virtual machine type	Choose CPU or GPU. This type cannot be changed after creation
Virtual machine priority	Choose Dedicated or Low priority . Low priority virtual machines are cheaper but don't guarantee the compute nodes. Your job may be preempted.
Virtual machine size	Supported virtual machine sizes might be restricted in your region. Check the availability list

FIELD	DESCRIPTION
Minimum number of nodes	Minimum number of nodes that you want to provision. If you want a dedicated number of nodes, set that count here. Save money by setting the minimum to 0, so you won't pay for any nodes when the cluster is idle.
Maximum number of nodes	Maximum number of nodes that you want to provision. The compute will autoscale to a maximum of this node count when a job is submitted.
Advanced settings	Optional. Configure a virtual network. Specify the Resource group , Virtual network , and Subnet to create the compute instance inside an Azure Virtual Network (vnet). For more information, see these network requirements for vnet. Also attach managed identities to grant access to resources

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.

During cluster creation or when editing compute cluster details, in the **Advanced settings**, toggle **Assign a managed identity** and specify a system-assigned identity or user-assigned identity.

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/')
```

Inference clusters

IMPORTANT

Using Azure Kubernetes Service with Azure Machine Learning has multiple configuration options. Some scenarios, such as networking, require additional setup and configuration. For more information on using AKS with Azure ML, see [Create and attach an Azure Kubernetes Service cluster](#).

Create or attach an Azure Kubernetes Service (AKS) cluster for large scale inferencing. Use the [steps above](#) to create the AKS cluster. Then fill out the form as follows:

FIELD	DESCRIPTION
Compute name	<ul style="list-style-type: none">• Name is required. Name must be between 2 to 16 characters.• Valid characters are upper and lower case letters, digits, and the - character.• Name must start with a letter• Name needs to be unique across all existing computes within an Azure region. You will see an alert if the name you choose is not unique• If - character is used, then it needs to be followed by at least one letter later in the name
Kubernetes Service	Select Create New and fill out the rest of the form. Or select Use existing and then select an existing AKS cluster from your subscription.
Region	Select the region where the cluster will be created
Virtual machine size	Supported virtual machine sizes might be restricted in your region. Check the availability list
Cluster purpose	Select Production or Dev-test
Number of nodes	The number of nodes multiplied by the virtual machine's number of cores (vCPUs) must be greater than or equal to 12.
Network configuration	Select Advanced to create the compute within an existing virtual network. For more information about AKS in a virtual network, see Network isolation during training and inference with private endpoints and virtual networks .
Enable SSL configuration	Use this to configure SSL certificate on the compute

Attached compute

To use compute targets created outside the Azure Machine Learning workspace, you must attach them. Attaching a compute target makes it available to your workspace. Use **Attached compute** to attach a compute target for **training**. Use **Inference clusters** to attach an AKS cluster for **inferencing**.

Use the [steps above](#) to attach a compute. Then fill out the form as follows:

1. Enter a name for the compute target.
2. Select the type of compute to attach. Not all compute types can be attached from Azure Machine Learning studio. The compute types that can currently be attached for training include:
 - A remote VM
 - Azure Databricks (for use in machine learning pipelines)

- Azure Data Lake Analytics (for use in machine learning pipelines)
 - Azure HDInsight
3. 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](#)

4. Select **Attach**.

Next steps

After a target is created and attached to your workspace, you use it in your [run configuration](#) with a `ComputeTarget` object:

```
from azureml.core.compute import ComputeTarget  
myvm = ComputeTarget(workspace=ws, name='my-vm-name')
```

- Use the compute resource to [submit a training run](#).
- [Tutorial: Train a model](#) uses a managed compute target to train a model.
- 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](#)

Set up a Python development environment for Azure Machine Learning

12/23/2020 • 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>"
}
```

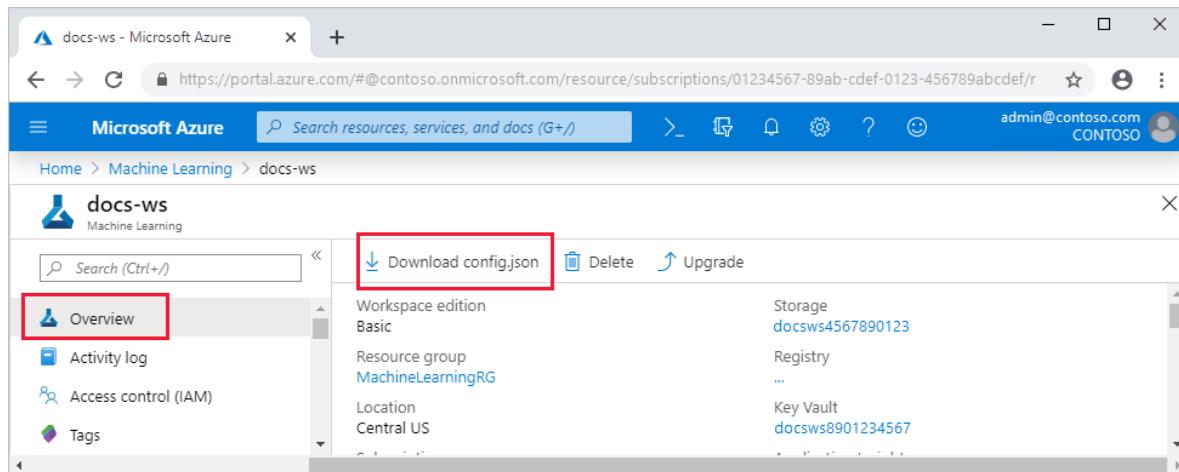
This JSON file must be in the directory structure that contains your Python scripts or Jupyter Notebooks. It can be in the same directory, a subdirectory named `.azureml`, or in a parent directory.

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.

```
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.

NOTE

A community-driven repository of examples can be found at <https://github.com/Azure/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, you can manage your [Azure Machine Learning resources](#), [run and debug experiments](#), and [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 [configure an Azure Machine Learning compute instance as a remote Jupyter Notebook server](#).

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:linuxdsvmubuntu: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 a model](#) on Azure Machine Learning with the MNIST dataset.
- See the [Azure Machine Learning SDK for Python reference](#).

How to run Jupyter Notebooks in your workspace

12/23/2020 • 8 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.

See how you can:

- Create Jupyter Notebooks in your workspace
- Run an experiment from a notebook
- Change the notebook environment
- Find details of the compute instances used to run your notebooks

Prerequisites

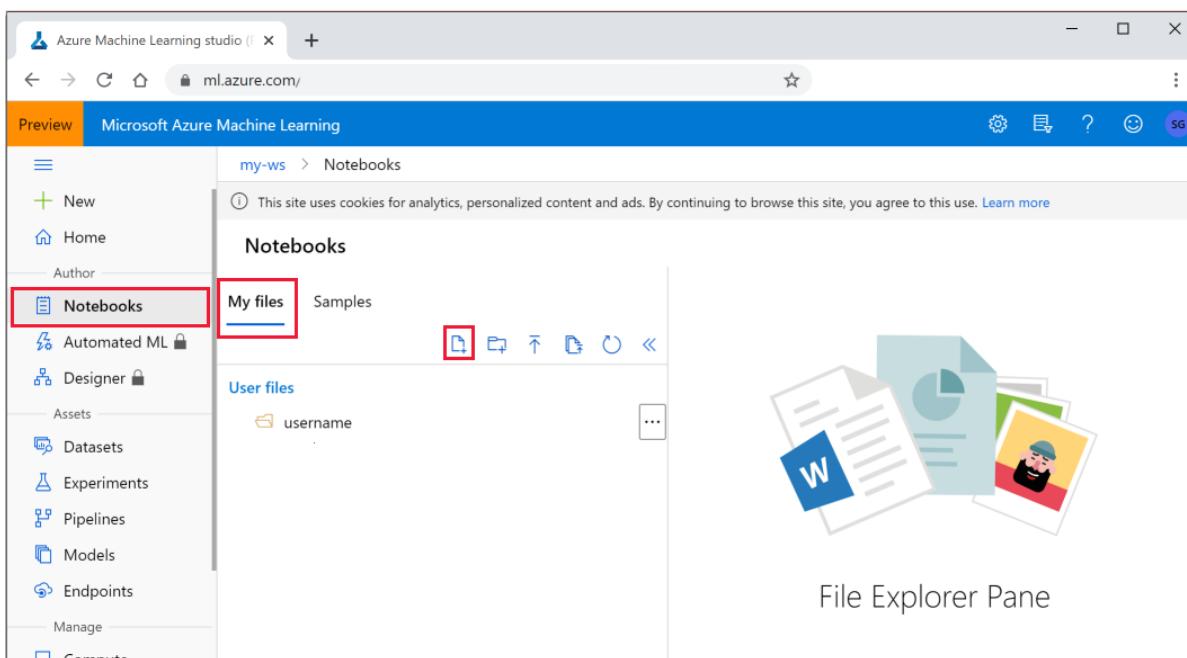
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A Machine Learning workspace. See [Create an Azure Machine Learning workspace](#).

Create notebooks

In your Azure Machine Learning workspace, create a new Jupyter notebook and start working. The newly created notebook is stored in the default workspace storage. This notebook can be shared with anyone with access to the workspace.

To create a new notebook:

1. Open your workspace in [Azure Machine Learning studio](#).
2. On the left side, select **Notebooks**.
3. Select the **Create new** file icon above the list **User files** in the **My files** section.



4. Name the file.
5. For Jupyter Notebook Files, select **Notebook** as the file type.

6. Select a file directory.

7. Select **Create**.

You can create text files as well. Select **Text** as the file type and add the extension to the name (for example, myfile.py or myfile.txt)

You can also upload folders and files, including notebooks, with the tools at the top of the Notebooks page. Notebooks and most text file types display in the preview section. No preview is available for most other file types.

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](#).

Clone samples

Your workspace contains a **Samples** folder with notebooks designed to help you explore the SDK and serve as examples for your own machine learning projects. You can clone these notebooks into your own folder on your workspace storage container.

For an example, see [Tutorial: Create your first ML experiment](#).

Use files from Git and version my files

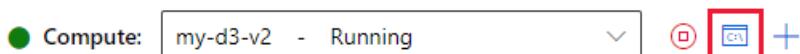
You can access all Git operations by using a terminal window. All Git files and folders will be stored in your workspace file system.

NOTE

Add your files and folders anywhere under the `~/cloudfiles/code/Users` folder so they will be visible in all your Jupyter environments.

To access the terminal:

1. Open your workspace in [Azure Machine Learning studio](#).
2. On the left side, select **Notebooks**.
3. Select any notebook located in the **User files** section on the left-hand side. If you don't have any notebooks there, first [create a notebook](#)
4. Select a **Compute** target or create a new one and wait until it's running.
5. Select the **Open terminal** icon.



6. If you don't see the icon, select the ... to the right of the compute target and then select **Open terminal**.



Learn more about [cloning Git repositories into your workspace file system](#).

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.

Share notebooks and other files

Copy and paste the URL to share a notebook or file. Only other users of the workspace can access this URL. Learn more about [granting access to your workspace](#).

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.

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.

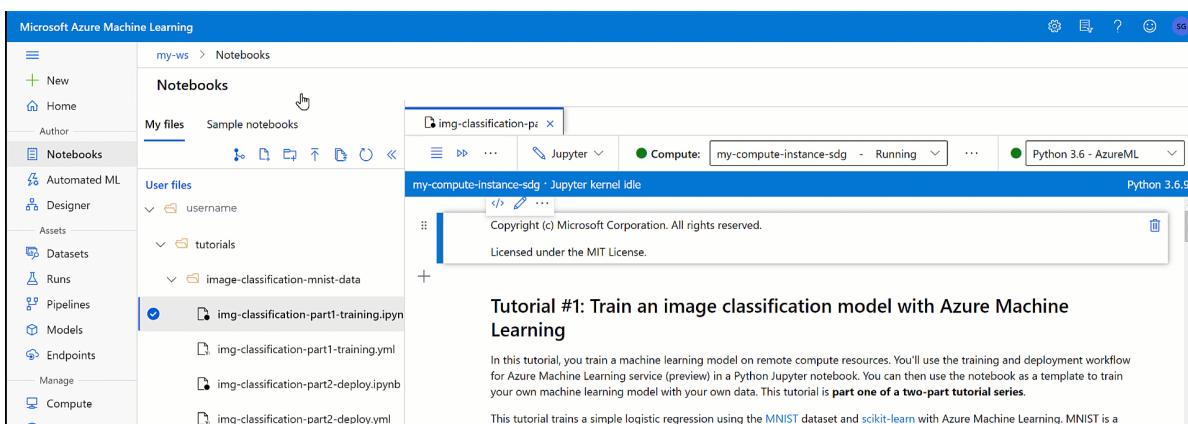
When a compute instance is running, you can also use code completion, powered by [Intellisense](#), in any Python Notebook.

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, this may be located under the ... menu item in your toolbar.
2. While in focus mode, return to the standard view by selecting **Standard view**.



Use IntelliSense

[IntelliSense](#) is a code-completion aid that includes a number of features: List Members, Parameter Info, Quick Info, and Complete Word. These features help you to learn more about the code you're using, keep track of the parameters you're typing, and add calls to properties and methods with only a few keystrokes.

When typing code, use `Ctrl+Space` to trigger IntelliSense.

Clean your notebook (preview)

IMPORTANT

The gather feature 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](#).

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.

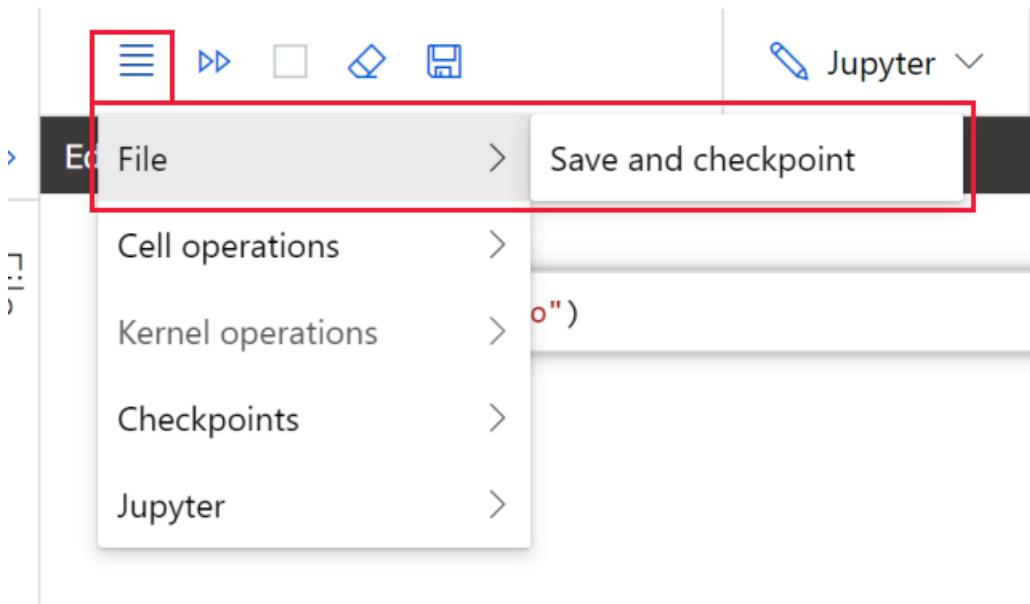
The new notebook contains only code cells, with all cells required to produce the same results as the cell you selected for gathering.

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.

Notebooks



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.

Useful keyboard shortcuts

KEYBOARD	ACTION
Shift+Enter	Run a cell
Ctrl+Space	Activate IntelliSense
Ctrl+M(Windows)	Enable/disable tab trapping in notebook.
Ctrl+Shift+M(Mac & Linux)	Enable/disable tab trapping in notebook.
Tab (when tab trap enabled)	Add a '\t' character (indent)
Tab (when tab trap disabled)	Change focus to next focusable item (delete cell button, run button, etc.)

Delete a notebook

You *can't* delete the **Samples** notebooks. These notebooks are part of the studio and are updated each time a new SDK is published.

You *can* delete **User files** notebooks 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).
- From any Notebook toolbar, select **Open terminal** to access the terminal window for the compute instance.
- In either Jupyter or JupyterLab with their tools.

Run an experiment

To run an experiment from a Notebook, you first connect to a running **compute instance**. If you don't have a compute instance, use these steps to create one:

1. Select + in the Notebook toolbar.
2. Name the Compute and choose a **Virtual Machine Size**.
3. Select **Create**.
4. The compute instance is connected to the Notebook automatically and you can now run your cells.

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.

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

The Notebook will automatically find all Jupyter kernels installed on the connected compute instance. To add a kernel to the compute instance:

1. Select **Open terminal** in the Notebook toolbar.
2. Use the terminal window to create a new environment. For example, the code below creates `newenv` :

```
conda create -y --name newenv
```

3. Activate the environment. For example, after creating `newenv` :

```
conda activate newenv
```

4. Install pip and ipykernel package to the new environment and create a kernel for that conda env

```
conda install -y pip
conda install -y ipykernel
python -m ipykernel install --user --name newenv --display-name "Python (newenv)"
```

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)

Any of the [available Jupyter Kernels](#) can be installed.

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](#).

Next steps

- [Run your first experiment](#)
- [Backup your file storage with snapshots](#)

Create & use software environments in Azure Machine Learning

12/23/2020 • 13 minutes to read • [Edit Online](#)

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 configuring development environments, see [here](#).

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-Minimal* environment contains a minimal set of packages to enable run tracking and asset uploading. You can use it as a starting point for your own environment.
- The *AzureML-Tutorial* 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-Minimal")
```

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.

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

When you 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.

The `DockerSection` of the Azure Machine Learning `Environment` class allows you to finely customize and control the guest operating system on which you run your training. The `arguments` variable can be used to specify extra arguments to pass to the Docker run command.

```
# Creates the environment inside a Docker container.  
myenv.docker.enabled = True
```

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 16.04 or greater.
- Conda 4.5.# or greater.
- Python 3.5+.

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.

```
# Specify docker steps as a string.
dockerfile = r"""
FROM mcr.microsoft.com/azureml/base:intelmpi2018.3-ubuntu16.04
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/base:intelmpi2018.3-ubuntu16.04
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.

```
myenv = Environment.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

```

You can also add environment variables to your environment. These then become available using `os.environ.get` in your training script.

```
myenv.environment_variables = {"MESSAGE": "Hello from Azure Machine Learning"}
```

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.

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", 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`.

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

This [article](#) provides information about how to install a Conda environment as a kernel in a notebook.

[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 CLI

The [Azure Machine Learning CLI](#) mirrors most of the functionality of the Python SDK. You can use it to create and manage environments. The commands that we discuss in this section demonstrate fundamental functionality.

The following command scaffolds the files for a default environment definition in the specified directory. These files are JSON files. They work like the corresponding class in the SDK. You can use the files to create new environments that have custom settings.

```
az ml environment scaffold -n myenv -d myenvdir
```

Run the following command to register an environment from a specified directory.

```
az ml environment register -d myenvdir
```

Run the following command to list all registered environments.

```
az ml environment list
```

Download a registered environment by using the following command.

```
az ml environment download -n myenv -d downloaddir
```

Next steps

- To use a managed compute target to train a model, see [Tutorial: Train a model](#).
- 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

12/23/2020 • 3 minutes to read • [Edit Online](#)

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://<my-org>.pkgs.visualstudio.com",
    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://<my-org>.pkgs.visualstudio.com/<my-project>/_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

12/23/2020 • 3 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 runs 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 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 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 datastore.** 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 specifying a datastore as your source directory, and uploading files to your datastore in the [Access data from your datastores](#) article.
- **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 run. 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 takes precedence.</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 runs 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 run, and they're not propagated back to the client machine automatically. Therefore, the changes made during the first experiment run don't and shouldn't affect those in the second.

When writing changes, we recommend writing files to an Azure Machine Learning datastore. See [Access data from your datastores](#).

If you don't require a datastore, 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 run history, so that you have access to them once your run is finished.

- For output such as status messages or scoring results, write files to the `./outputs` folder, so they are persisted as artifacts in run history. Be mindful of the number and size of files written to this folder, as latency may occur when the contents are uploaded to run history. If latency is a concern, writing files to a datastore is recommended.
- To save written file as logs in run 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 run.

Next steps

- Learn more about [accessing data from your datastores](#).
- Learn more about [Create compute targets for model training and deployment](#)

Connect to an Azure Machine Learning compute instance in Visual Studio Code (preview)

12/23/2020 • 3 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 Jupyter Notebook server. This option allows you to set a compute instance as a remote Jupyter Notebook server.
- [Visual Studio Code remote development](#). Visual Studio Code remote development allows you to use a container, remote machine, or the Windows Subsystem for Linux (WSL) as a full-featured development environment.

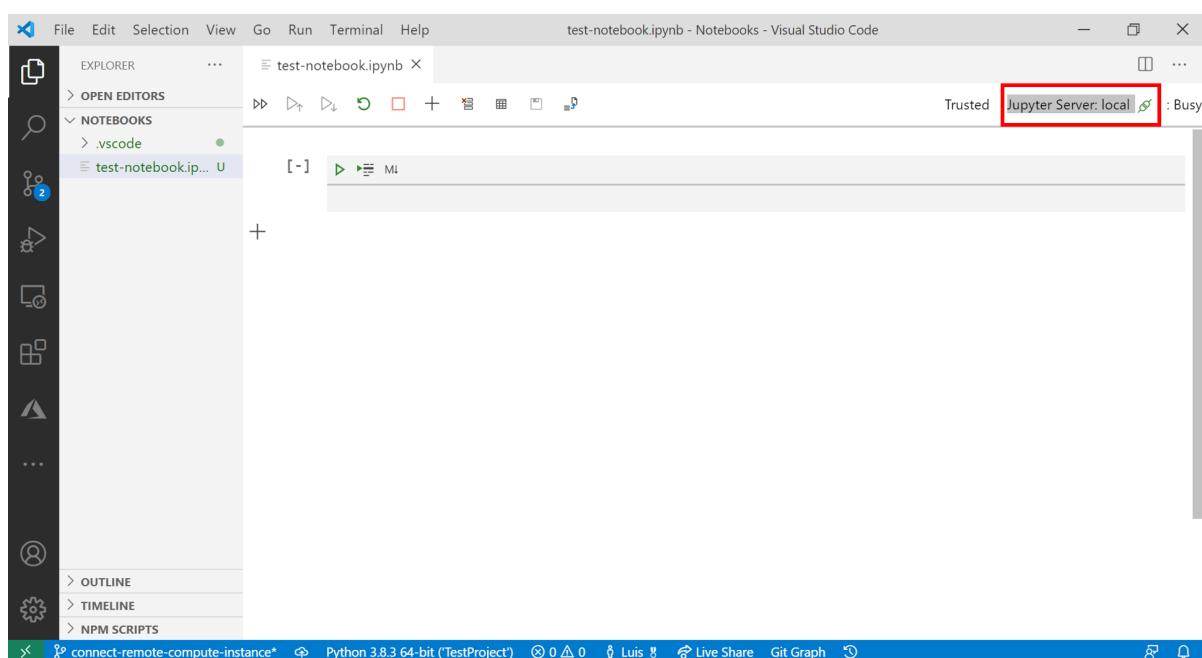
Configure compute instance as remote notebook server

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](#).

Configure compute instance remote development

For a full-featured remote development experience, you'll need a few prerequisites:

- [Visual Studio Code Remote SSH extension](#).
- SSH-enabled compute instance. For more information, see the [Create a compute instance guide](#).

NOTE

On Windows platforms, you must [install an OpenSSH compatible SSH client](#) if one is not already present. PuTTY is not supported on Windows since the ssh command must be in the path.

Get the IP and SSH port for your compute instance

1. Go to the Azure Machine Learning studio at <https://ml.azure.com/>.
2. Select your [workspace](#).
3. Click the [Compute Instances](#) tab.
4. In the **Application URI** column, click the **SSH** link of the compute instance you want to use as a remote compute.
5. In the dialog, take note of the IP Address and SSH port.
6. Save your private key to the `~/.ssh/` directory on your local computer; for instance, open an editor for a new file and paste the key in:

Linux:

```
vi ~/.ssh/id_azmlcitest_rsa
```

Windows:

```
notepad C:\Users\<username>\.ssh\id_azmlcitest_rsa
```

The private key will look somewhat like this:

```
-----BEGIN RSA PRIVATE KEY-----  
  
MIIEpAIBAAKCAQEAr99EPm0P4CaTPT2KtBt+kpN3rmsNNE5dS0vmGwxIXq4vAWXD  
.....  
ewMtLnDgXWYJo0IyQ91yn0dxbFoVuuGNdDoBykUZPQfeHDONy2Raw==  
  
-----END RSA PRIVATE KEY-----
```

7. Change permissions on file to make sure only you can read the file.

```
chmod 600 ~/.ssh/id_azmlcitest_rsa
```

Add instance as a host

Open the file `~/.ssh/config` (Linux) or `C:\Users\<username>.ssh\config` (Windows) in an editor and add a new entry similar to the content below:

```
Host azmlci1  
HostName 13.69.56.51  
Port 50000  
User azureuser  
IdentityFile ~/.ssh/id_azmlcitest_rsa
```

Here some details on the fields:

FIELD	DESCRIPTION
Host	Use whatever shorthand you like for the compute instance
HostName	This is the IP address of the compute instance
Port	This is the port shown on the SSH dialog above
User	This needs to be <code>azureuser</code>
IdentityFile	Should point to the file where you saved the private key

Now, you should be able to ssh to your compute instance using the shorthand you used above, `ssh azmlci1`.

Connect VS Code to the instance

1. Click the Remote-SSH icon from the Visual Studio Code activity bar to show your SSH configurations.

2. Right-click the SSH host configuration you just created.

3. Select **Connect to Host in Current Window**.

From here on, you are entirely working on the compute instance and you can now edit, debug, use git, use extensions, etc. -- just like you can with your local Visual Studio Code.

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

12/23/2020 • 5 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.

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.

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]
```

- 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

- 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.

- 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

- Copy the SSH Git clone URL from the Git repo.

- 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:nThbg6kXUpJWGi7E1IGOCspRomTxdCARLviKw6E5SY8.  
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 run 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 run. This information is stored in the following properties for the training run:

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 run was submitted.
<code>mlflow.source.git.branch</code>	<code>git symbolic-ref --short HEAD</code>	The active branch when the run was submitted.
<code>azureml.git.commit</code>	<code>git rev-parse HEAD</code>	The commit hash of the code that was submitted for the run.
<code>mlflow.source.git.commit</code>	<code>git rev-parse HEAD</code>	The commit hash of the code that was submitted for the run.
<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 runs 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 run. You can view this information using the Azure portal, Python SDK, and CLI.

Azure portal

1. From the [studio portal](#), select your workspace.
2. Select **Experiments**, and then select one of your experiments.
3. Select one of the runs from the **RUN NUMBER** 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_9d3568242c6bfe9631879915768deaf",  
    "ProcessInfoFile": "azureml-logs/process_info.json",  
    "ProcessStatusFile": "azureml-logs/process_status.json"  
}
```

Python SDK

After submitting a training run, a `Run` object is returned. The `properties` attribute of this object contains the logged git information. For example, the following code retrieves the commit hash:

```
run.properties['azureml.git.commit']
```

CLI

The `az ml run` CLI command can be used to retrieve the properties from a run. For example, the following command returns the properties for the last run in the experiment named `train-on-amlcompute`:

```
az ml run list -e train-on-amlcompute --last 1 -w myworkspace -g myresourcegroup --query '[].properties'
```

For more information, see the [az ml run](#) reference documentation.

Next steps

- [Use compute targets for model training](#)

Troubleshoot environment image builds

12/23/2020 • 5 minutes to read • [Edit Online](#)

Learn how to troubleshoot issues with Docker environment image builds and package installations.

Prerequisites

- An [Azure subscription](#). Try the [free or paid version of Azure Machine Learning](#).
- The [Azure Machine Learning SDK](#).
- The [Azure CLI](#).
- The [CLI extension for Azure Machine Learning](#).
- To debug locally, you must have a working Docker installation on your local system.

Docker Image Build Failures

For the majority of image build failures, the root cause can be found in the image build log. You can find the image build log from the Azure Machine Learning portal (20_image_build_log.txt) or from your ACR tasks runs logs

In most cases, it is easier to reproduce errors locally. Check the kind of error and try one of the following

`setup tools` :

- Install conda dependency locally `conda install suspicious-dependency==X.Y.Z`
- Install pip dependency locally `pip install suspicious-dependency==X.Y.Z`
- Try to materialize the entire environment `conda create -f conda-specification.yml`

IMPORTANT

Make sure that platform and interpreter on your local compute match the ones on the remote.

Timeout

Timeout issues can happen for various network issues:

- Low internet bandwidth
- Server issues
- Large dependency that can't be downloaded with the given conda or pip timeout settings

Messages similar to the below will indicate the issue:

```
('Connection broken: OSSError("(104, \\'ECONNRESET\\')")', OSSError("(104, \'ECONNRESET\')"))
```

```
ReadTimeoutError("HTTPSConnectionPool(host='****', port=443): Read timed out. (read timeout=15)",)
```

Possible solutions:

- Try a different source for the dependency if available such as mirrors, blob storage, or other python feeds.
- Update conda or pip. If a custom docker file is used, update the timeout settings.
- Some pip versions have known issues. Consider adding a specific version of pip into environment dependencies

Package Not Found

This is the most common case for image build failures.

- Conda package could not be found `ResolvePackageNotFound: - not-existing-conda-package`

- Specified pip package or version could not be found

```
ERROR: Could not find a version that satisfies the requirement invalid-pip-package (from versions: none)
ERROR: No matching distribution found for invalid-pip-package
```

- Bad nested pip dependency

```
ERROR: No matching distribution found for bad-backage==0.0 (from good-package==1.0)
```

Check the package exists on the specified sources. Use [pip search](#) to verify pip dependencies.

```
pip search azureml-core
```

For conda dependencies, use [conda search](#).

```
conda search conda-forge::numpy
```

For more options:

- `pip search -h`
- `conda search -h`

Installer Notes

Make sure that the required distribution exists for the specified platform and Python interpreter version.

For pip dependencies, navigate to [https://pypi.org/project/\[PROJECT NAME\]/\[VERSION\]/#files](https://pypi.org/project/[PROJECT_NAME]/[VERSION]/#files) to see if required version is available. For example, <https://pypi.org/project/azureml-core/1.11.0/#files>

For conda dependencies, check package on the channel repository. For channels maintained by Anaconda, Inc., check [here](#).

Pip Package Update

During install or update of a pip package the resolver may need to update an already installed package to satisfy the new requirements. Uninstall can fail for various reasons related to pip version or the way the dependency was installed. The most common scenario is that a dependency installed by conda could not be uninstalled by pip. For this scenario, consider uninstalling the dependency using `conda remove mypackage`.

```
Attempting uninstall: mypackage
Found existing installation: mypackage X.Y.Z
ERROR: Cannot uninstall 'mypackage'. It is a distutils installed project and thus we cannot accurately
determine which files belong to it which would lead to only a partial uninstall.
```

Installer issues

Certain installer versions have issues in the package resolvers that can lead to a build failure.

If a custom base image or dockerfile is used, we recommend using conda version 4.5.4 or higher.

Pip package is required to install pip dependencies and if a version is not specified in the environment the latest version will be used. We recommend using a known version of pip to avoid transient issues or breaking changes that can be caused by the latest version of the tool.

Consider pinning the pip version in your environment if you see any of the messages below:

```
Warning: you have pip-installed dependencies in your environment file, but you do not list pip itself as one of
your conda dependencies. Conda may not use the correct pip to install your packages, and they may end up in the
wrong place. Please add an explicit pip dependency. I'm adding one for you, but still nagging you.
```

```
Pip subprocess error: ERROR: THESE PACKAGES DO NOT MATCH THE HASHES FROM THE REQUIREMENTS FILE. If you have updated the package versions, update the hashes as well. Otherwise, examine the package contents carefully; someone may have tampered with them.
```

In addition, pip installation can be stuck in an infinite loop if there are unresolvable conflicts in the dependencies. If working locally, downgrade the pip version to < 20.3. In a conda environment created from a YAML file, this issue will only be seen if conda-forge is highest priority channel. To mitigate the issue, explicitly specify pip < 20.3 (!=20.3 or =20.2.4 pin to other version) as a conda dependency in the conda specification file.

Service Side Failures

Unable to pull image from MCR/Address could not be resolved for Container Registry.

Possible issues:

- Path name to container registry may not be resolving correctly. Check that image names use double slashes and the direction of slashes on Linux vs Windows hosts is correct.
- If the ACR behind a Vnet is using a private endpoint in [an unsupported region](#), configure the ACR behind a VNet using the service endpoint (Public access) from the portal and retry.
- After putting the ACR behind a VNet, ensure that the [ARM template](#) is run. This enables the workspace to communicate with the ACR instance.

401 error from workspace ACR

Resynchronize storage keys using [ws.sync_keys\(\)](#)

Environment keeps throwing "Waiting for other Conda operations to finish..." Error

When an image build is ongoing, conda is locked by the SDK client. If the process crashed or was canceled incorrectly by the user - conda stays in the locked state. To resolve this, manually delete the lock file.

Custom docker image not in registry

Check if the [correct tag](#) is used and that `user_managed_dependencies = True`.

`Environment.python.user_managed_dependencies = True` disables Conda and uses the user's installed packages.

Common VNet issues

1. Check that the storage account, compute cluster, and Azure Container Registry are all in the same subnet of the virtual network.
2. When ACR is behind a VNet, it can't directly be used to build images. The compute cluster needs to be used to build images.
3. Storage may need to be placed behind a VNet when:
 - Using inferencing or private wheel
 - Seeing 403 (not authorized) service errors
 - Can't get image details from ACR/MCR

Image build fails when trying to access network protected storage

- ACR tasks do not work behind the VNet. If the user has their ACR behind the VNet, they need to use the compute cluster to build an image.
- Storage should be behind VNet in order to be able to pull dependencies from it.

Cannot run experiments when storage has network security enabled

When using default Docker images and enabling user-managed dependencies, you must use the MicrosoftContainerRegistry and AzureFrontDoor.FirstParty [service tags](#) to allowlist MCR and its dependencies.

See [enabling VNET](#) for more.

Creating an ICM

When creating/assigning an ICM to Metastore, please include the CSS support ticket so that we can better

understand the issue.

Next steps

- [Train a machine learning model to categorize flowers](#)
- [Train a machine learning model using a custom Docker image](#)

Create a data labeling project and export labels

12/23/2020 • 14 minutes to read • [Edit Online](#)

Labeling voluminous data in machine learning projects is often a headache. Projects that have a computer-vision component, such as image classification or object detection, generally require labels for thousands of images.

Azure Machine Learning data labeling gives you a central place to create, manage, and monitor labeling projects. Use it to coordinate data, labels, and team members to efficiently manage labeling tasks. Machine Learning supports image classification, either multi-label or multi-class, and object identification with bounded boxes.

Data labeling tracks progress and maintains the queue of incomplete labeling tasks.

You are able to start and stop the project and control the labeling progress. You can review the labeled data and export labeled in COCO format or as an Azure Machine Learning dataset.

IMPORTANT

Only image classification and object identification labeling projects are currently supported. Additionally, the data images must be available in an Azure blob datastore. (If you do not have an existing datastore, you may upload images during project creation.)

In this article, you'll learn how to:

- Create a project
- Specify the project's data and structure
- Run and monitor the project
- Export the labels

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 labeling project

Labeling projects are administered from Azure Machine Learning. You use the [Labeling projects](#) 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. For an example of using a datastore, see [Tutorial: Create your first image classification labeling project](#).

To create a project, select **Add project**. Give the project an appropriate name and select **Labeling task type**.

- 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.

Select **Next** when you're ready 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** 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 images. If your dataset has more, only the first 500,000 images 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 **File** as the **Dataset type**. Only file dataset types are supported.
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. 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. Choose "File" as the **Dataset type**.
4. *Optional:* Select **Advanced settings** to customize the datastore, container, and path to your data.
5. Select **Browse** to select the local files to upload.
6. Provide a description of your dataset.
7. Select **Next**.
8. Confirm the details. Select **Back** to modify the settings or **Create** to create the dataset.

The data gets uploaded to the default blob store ("workspaceblobstore") of your Machine Learning workspace.

Configure incremental refresh

If you plan to add new images to your dataset, use incremental refresh to add these new images to your project. When **incremental refresh** is enabled, the dataset is checked periodically for new images 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 images.

To add more images to your project, use [Azure Storage Explorer](#) to upload to the appropriate folder in the blob storage.

Check the box for **Enable incremental refresh** when you want your project to continually monitor for new data in the datastore. This data will be pulled into your project once a day when enabled.

Uncheck this box if you do not want new images that appear in the datastore to be added to your project.

You can find the timestamp for the latest refresh in the **Incremental refresh** section of **Details** tab for your project.

Specify label classes

On the **Label classes** page, specify the set of classes to categorize your data. Do this carefully, because your labelers' accuracy and speed will be 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 3 or 4 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 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?

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?
- How to label the object that is partially shown in the image?
- How to label the object that partially covered by other object?

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 labeling

The **ML assisted labeling** page lets you trigger automatic machine learning models to accelerate the labeling task. At the beginning of your labeling project, the images 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 images are of a single class, then approximately 80% of the data used to train the model will be of that class. This training does not include active learning.

Select *Enable ML assisted labeling* and specify a GPU to enable assisted labeling, which consists of two phases:

- Clustering
- Prelabeling

The exact number of labeled images necessary to start assisted labeling is not a fixed number. This can vary significantly from one labeling project to another. For some projects, it is sometimes possible to see prelabel or cluster tasks after 300 images 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 images. 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 labelling 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 image classification starts to group together similar images. 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 which are nearest neighbors in the space are used for clustering tasks.

The clustering phase does not appear for object detection models.

Prelabeling

After enough image labels are submitted, a classification model is used to predict image 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 image. 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 images to determine its accuracy at a variety of 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. Images with predictions more confident than this threshold are used for pre-labeling.

Initialize the 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**.

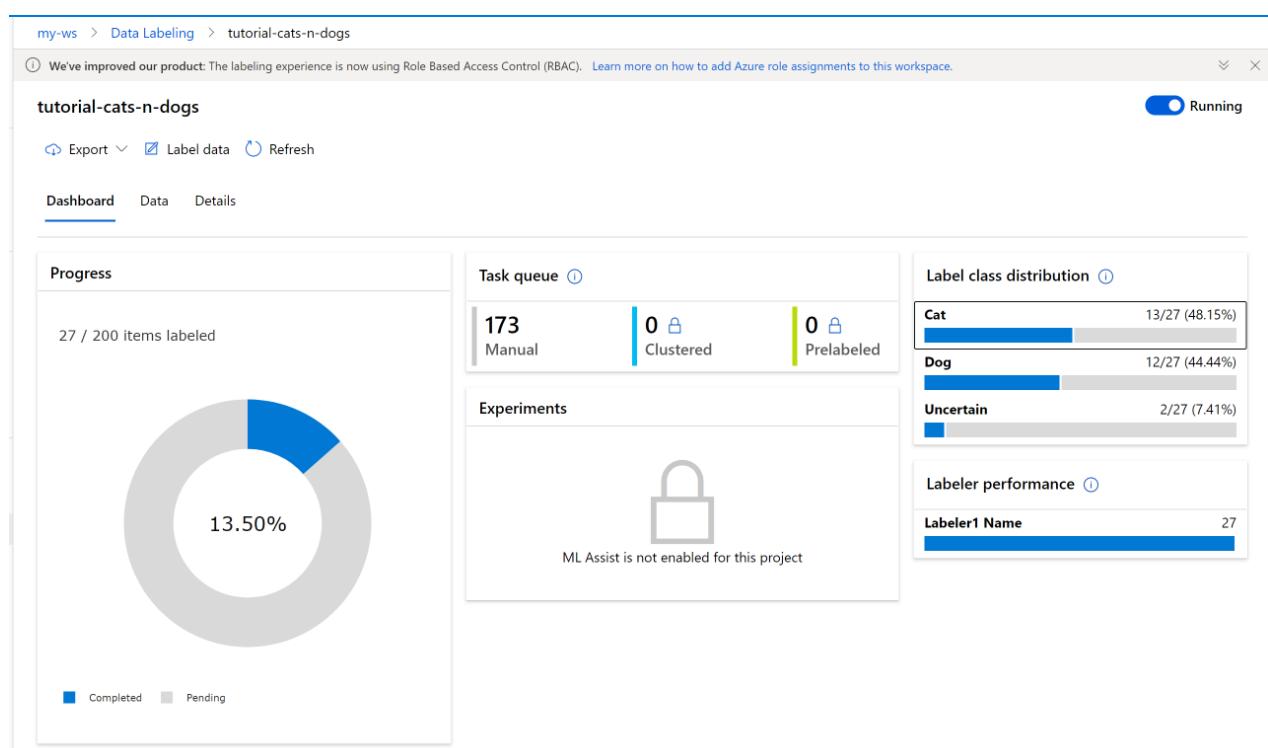
Run and monitor the project

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 and how many are not yet done. Items pending may be:

- Not yet added to a task
- Included in a task that is assigned to a labeler but not yet completed
- In the queue of tasks yet to be assigned

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 hand 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. 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 details of your project. In this tab you can:

- View project details and input datasets
- Enable incremental 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 access to your workspace can label data in your project. You can also customize the permissions for your labelers so that they can access labeling but not other parts of the workspace or your labeling project. For more details, see [Manage access to an Azure Machine Learning workspace](#), and learn how to create the [labeler custom role](#).

Add new label class to a project

During the labeling process, you may find that additional labels are needed to classify your images. For example, you may want to add an "Unknown" or "Other" label to indicate confusing images.

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 interface. On the left, there's a sidebar with various options like 'New', 'Home', 'Notebooks', 'Automated ML', 'Designer', 'Datasets', 'Experiments', 'Pipelines', 'Models', 'Endpoints', 'Compute', 'Datastores', and 'Data Labeling'. The 'Data Labeling' option is highlighted with a red box. The main area shows a project named 'tutorial-cats-n-dogs'. At the top right, there's a toggle switch labeled 'Paused' with a red box around it. Below that, there are 'Export' and 'Refresh' buttons. The 'Details' tab is selected. Under 'Project details', there's a 'Label classes' section which is also highlighted with a red box. This section contains three entries: 'Cat', 'Dog', and 'Uncertain'. There's also a 'Label Name' input field. Other sections visible include 'Input datasets', 'Incremental refresh (optional)', 'Intermediate label storage', 'Instructions', and 'ML assisted labeling (optional) (preview)'.

6. In the form, add your new label and choose how to proceed. Since you've changed the available labels for an image, 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 have added all new labels, at the top right of the page toggle **Paused** to **Running** to restart the project.

Export the labels

You can export the label data for Machine Learning experimentation at any time. Image labels can be exported in [COCO format](#) or as an [Azure Machine Learning dataset with labels](#). Use the **Export** button on the **Project details** page of your labeling project.

The COCO file is created in the default blob store of the Azure Machine Learning workspace in a folder within `export/coco`. You can access the exported Azure Machine Learning dataset 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's a sidebar with navigation links: New, Home, Author, Notebooks, Automated ML, Designer, Assets, Datasets (which is selected and highlighted with a red border), Experiments, Pipelines, Models, Endpoints, Manage, Compute, Environments, Datastores, and Data labeling. The main content area has a header 'labeling_ws > Datasets > ML0-2019-10-30 00:34:49' and a sub-header 'ML0-2019-10-30 00:34:49 Version 1 (latest)'. Below this are tabs for Details, Explore, and Models, with 'Details' being the active tab. Under 'Details', there are sections for Attributes (Properties: Tabular, Labeled; Description: LabeledDs_ML0 Of Type ImageClassification, Sourced From workspaceblobstore), Datastore (workspaceblobstore), Relative path (/export/dataset/d90699e5-aaaa-aaaa-aaaa-0a785321bac5/18d84ce2-f316-aaaa-aaaa-1ba0fa9ca83c/48a5eb2b-aaaa-aaaa-83ae-315c0ec5896b/LabeledDatasetJsonLines.json), Profile (No profile generated), Current version (1), Latest version (1), Created time (Oct 29, 2019 2:34 PM), and Modified time (Oct 29, 2019 2:34 PM). To the right, there are sections for Tags (No data), Sample usage (with a code snippet in Python using azureml.core to import Workspace, Dataset), and a large grayed-out area for preview.

Troubleshooting

Use these tips if you see any of these issues.

ISSUE	RESOLUTION
Only datasets created on blob datastores can be used.	This is a known limitation of the current release.
After creation, the project shows "Initializing" for a long time.	Manually refresh the page. Initialization should proceed at roughly 20 datapoints per second. The lack of autorefresh is a known issue.
When reviewing images, newly labeled images are not shown.	To load all labeled images, choose the First button. The First button will take you back to the front of the list, but loads all labeled data.
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

- [Tutorial: Create your first image classification labeling project.](#)
- Label images for [image classification or object detection](#)
- Learn more about [Azure Machine Learning and Machine Learning Studio \(classic\)](#)

Tag images in a labeling project

12/23/2020 • 8 minutes to read • [Edit Online](#)

After your project administrator [creates a 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 workspace

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. Select **Data labeling** on the left-hand side to find the project.

Understand the labeling task

In the table of data labeling projects, select **Label 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**.

Common features of the labeling task

In all image-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.

In 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.

Screenshot of the Microsoft Azure Machine Learning Data Labeling interface showing a multi-class labeling task for animals.

The interface includes a navigation bar with "Preview" and "All Projects" tabs, and a title "Animal Classes (Multiclass)". Below the title are "Instructions" and "Tasks" buttons, with "Tasks" being active.

A "Select all (2 selected)" button is present. On the right, there is a "Tags" sidebar with checkboxes for "Mammal" (checked), "Bird", "Bony fish", "Cartilaginous fish", "Reptile", "Anthozoan", and "Other/Unknown".

Four image cards are displayed:

- Image 1: A lizard on rocks. Tag status: Unselected (radio button).
- Image 2: A brown bear sitting on a rock. Tag status: Selected (radio button with checkmark).
- Image 3: An orca breaching the water. Tag status: Selected (radio button with checkmark).
- Image 4: A shark swimming underwater. Tag status: Unselected (radio button).

A green button labeled "Cartilaginous fish X" is located below the fourth image card.

A "Submit" button is 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.

Assisted machine learning (preview)

IMPORTANT

Assisted machine learning 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](#).

Machine learning algorithms may be triggered. If these algorithms are enabled in your project, you may see the following:

- After some amount of images 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. Images 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 images for which the model is confident, these images might still be incorrectly prelabeled. When you see these labels, correct any wrong labels before submitting the page.
- For object detection models, you may see bounding boxes and labels already present. Correct any that are incorrect 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.

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, click 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. Clicking on "Land" will clear that tag from the two

selected images.

All Projects > Photo labels (Multilabel)

Photo labels (Multilabel)

Instructions Tasks

Select all (2 selected)

Tags

- Land
- Ocean
- Closeup
- Wideangle

Image	Tags
	<input type="radio"/> Closeup X Ocean X
	<input type="radio"/> Closeup X Ocean X
	<input checked="" type="radio"/> Wideangle X Land X
	<input checked="" type="radio"/> Wideangle X Land X

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. Click 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 double-headed arrow (1), a square with a diagonal line, a document, a folder, a lock, a magnifying glass, a sun, a circle, and a hand. A red circle with the number '2' is placed over the hand icon. To the right of the toolbar is a search bar labeled 'Search tags' and a zoom slider set to '100%'. On the far right, there is a 'Tags' section with a red circle containing the number '1'. Below the tags are five options: '(#1) Shark', '(#2) Dog', '(#3) Cat', '(#4) Bird' (which is selected, indicated by a blue dot), and '(#5) Other'. The main area displays a photograph of a blue-footed booby standing on a rock. A dashed black bounding box surrounds the entire bird. A red circle with the number '3' is placed at the bottom right corner of the bounding box. At the bottom left of the main area is a 'Submit' button.

To delete a bounding box, click 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. Click 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

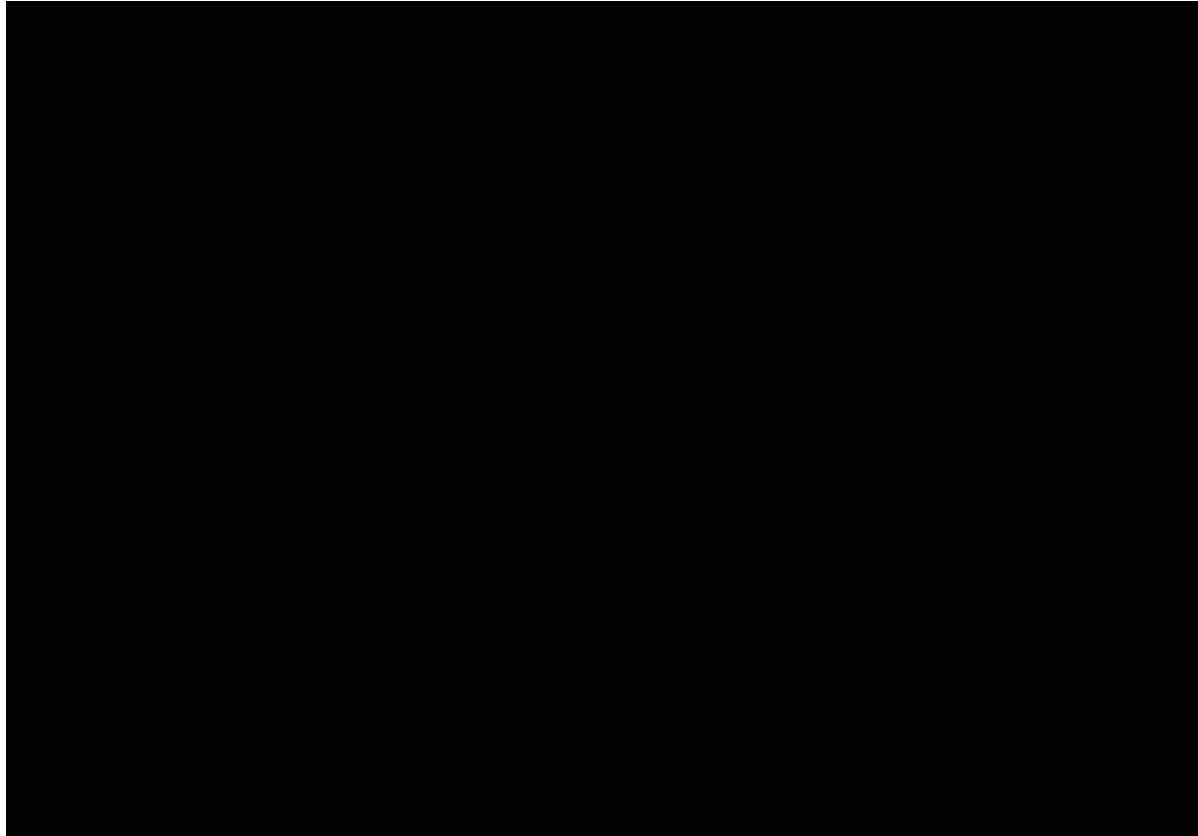
If your project is of type "Instance Segmentation (Polygon)," you'll specify one or more polygons in the image and

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. Click for each point in the polygon. When you have completed the shape, double click to finish.



To delete a polygon, click 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, click on 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. Click on 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.

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](#)

Create and explore Azure Machine Learning dataset with labels

12/23/2020 • 3 minutes to read • [Edit Online](#)

In this article, you'll learn how to export the data labels from an Azure Machine Learning data labeling project and load them into popular formats such as, a pandas dataframe for data exploration or a Torchvision dataset for image transformation.

What are datasets with labels

We refer to Azure Machine Learning datasets with labels as labeled datasets. These specific dataset types of labeled datasets are only created as an output of Azure Machine Learning data labeling projects. Create a data labeling project with [these steps](#). Machine Learning supports data labeling projects for image classification, either multi-label or multi-class, and object identification together with bounded boxes.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- The [Azure Machine Learning SDK for Python](#), or access to [Azure Machine Learning studio](#).
 - Install the `azure-contrib-dataset` package
- A Machine Learning workspace. See [Create an Azure Machine Learning workspace](#).
- Access to an Azure Machine Learning data labeling project. If you don't have a labeling project, create one with [these steps](#).

Export data labels

When you complete a data labeling project, you can export the label data from a labeling project. Doing so, allows you to capture both the reference to the data and its labels, and export them in [COCO format](#) or as an Azure Machine Learning dataset. Use the **Export** button on the **Project details** page of your labeling project.

COCO

The COCO file is created in the default blob store of the Azure Machine Learning workspace in a folder within `export/coco`.

Azure Machine Learning dataset

You can access the exported Azure Machine Learning dataset in the **Datasets** section of your Azure Machine Learning studio. 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's a sidebar with various options like 'New', 'Home', 'Notebooks', 'Automated ML', 'Designer', 'Assets', and 'Datasets'. The 'Datasets' option is highlighted with a red border. The main area displays a dataset named 'ML0-2019-10-30 00:34:49'. At the top, it says 'Version 1 (latest)'. Below that, there are tabs for 'Details', 'Explore', and 'Models', with 'Details' being the active tab. Under 'Details', there are sections for 'Attributes' (Properties: Tabular, Labeled; Description: LabeledDs_ML0 Of Type ImageClassification, Sourced From: Datastore workspaceblobstore; Relative path: /export/dataset/d90699e5-aaaa-aaaa-aaaa-0a785321bac5/18d84ce2-f316-aaaa-aaaa-1ba0fa9ca83c/48a5eb2b-aaaa-aaa-83ae-315c0ec5896b/LabeledDatasetJsonLines.json), 'Profile' (No profile generated), 'Current version' (1), 'Latest version' (1), 'Created time' (Oct 29, 2019 2:34 PM), and 'Modified time' (Oct 29, 2019 2:34 PM). To the right, there are sections for 'Tags' (No data) and 'Sample usage' (a code snippet for loading the dataset into a pandas DataFrame).

Explore labeled datasets

Load your labeled datasets into a pandas dataframe or Torchvision dataset to leverage popular open-source libraries for data exploration, as well as PyTorch provided libraries for image transformation and training.

Pandas dataframe

You can load labeled datasets into a pandas dataframe with the `to_pandas_dataframe()` method from the `azureml-contrib-dataset` class. Install the class with the following shell command:

```
pip install azureml-contrib-dataset
```

NOTE

The `azureml.contrib` namespace changes frequently, as we work to improve the service. As such, anything in this namespace should be considered as a preview, and not fully supported by Microsoft.

Azure Machine Learning offers the following file handling options for file streams when converting to a pandas dataframe.

- Download: Download your data files to a local path.
- Mount: Mount your data files to a mount point. Mount only works for Linux-based compute, including Azure Machine Learning notebook VM and Azure Machine Learning Compute.

In the following code, the `animal_labels` dataset is the output from a labeling project previously saved to the workspace.

```

import azureml.core
import azureml.contrib.dataset
from azureml.core import Dataset, Workspace
from azureml.contrib.dataset import FileHandlingOption

# get animal_labels dataset from the workspace
animal_labels = Dataset.get_by_name(workspace, 'animal_labels')
animal_pd = animal_labels.to_pandas_dataframe(file_handling_option=FileHandlingOption.DOWNLOAD,
target_path='./download/', overwrite_download=True)

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

#read images from downloaded path
img = mpimg.imread(animal_pd.loc[0,'image_url'])
imgplot = plt.imshow(img)

```

Torchvision datasets

You can load labeled datasets into Torchvision dataset with the [to_torchvision\(\)](#) method also from the `azureml-contrib-dataset` class. To use this method, you need to have [PyTorch](#) installed.

In the following code, the `animal_labels` dataset is the output from a labeling project previously saved to the workspace.

```

import azureml.core
import azureml.contrib.dataset
from azureml.core import Dataset, Workspace
from azureml.contrib.dataset import FileHandlingOption

from torchvision.transforms import functional as F

# get animal_labels dataset from the workspace
animal_labels = Dataset.get_by_name(workspace, 'animal_labels')

# load animal_labels dataset into torchvision dataset
pytorch_dataset = animal_labels.to_torchvision()
img = pytorch_dataset[0][0]
print(type(img))

# use methods from torchvision to transform the img into grayscale
pil_image = F.to_pil_image(img)
gray_image = F.to_grayscale(pil_image, num_output_channels=3)

imgplot = plt.imshow(gray_image)

```

Next steps

- See the [dataset with labels notebook](#) for complete training sample.

Data ingestion with Azure Data Factory

12/23/2020 • 4 minutes to read • [Edit Online](#)

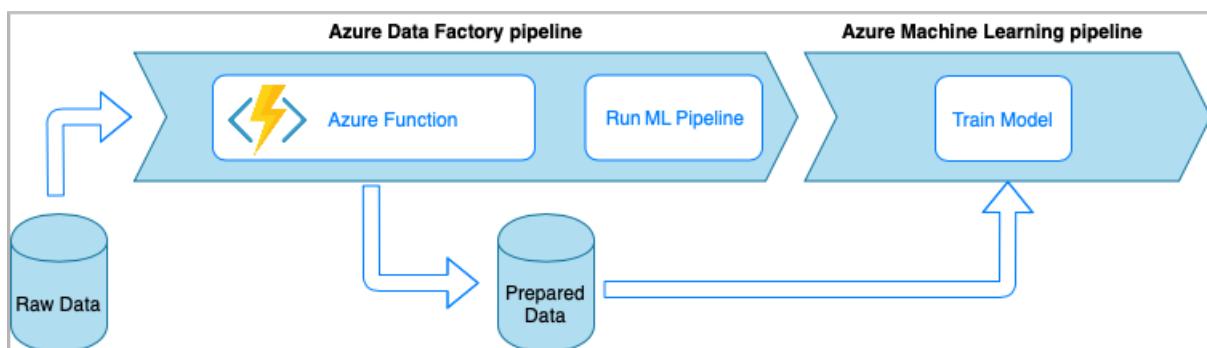
In this article, you learn about the available options for building a data ingestion pipeline with Azure Data Factory (ADF). This pipeline is used to ingest data for use with Azure Machine Learning. Azure Data Factory allows you to easily extract, transform, and load (ETL) data. Once the data has been transformed and loaded into storage, it can be used to train your machine learning models.

Simple data transformation can be handled with native ADF activities and instruments such as [data flow](#). When it comes to more complicated scenarios, the data can be processed with some custom code. For example, Python or R code.

There are several common techniques of using Azure Data Factory to transform data during ingestion. Each technique has pros and cons that determine if it is a good fit for a specific use case:

TECHNIQUE	PROS	CONS
ADF + Azure Functions	Low latency, serverless compute Stateful functions Reusable functions	Only good for short running processing
ADF + custom component	Large-scale parallel computing Suited for heavy algorithms	Wrapping code into an executable Complexity of handling dependencies and IO
ADF + Azure Databricks notebook	Apache Spark Native Python environment	Can be expensive Creating clusters initially takes time and adds latency

ADF with Azure functions



Azure Functions allows you to run small pieces of code (functions) without worrying about application infrastructure. In this option, the data is processed with custom Python code wrapped into an Azure Function.

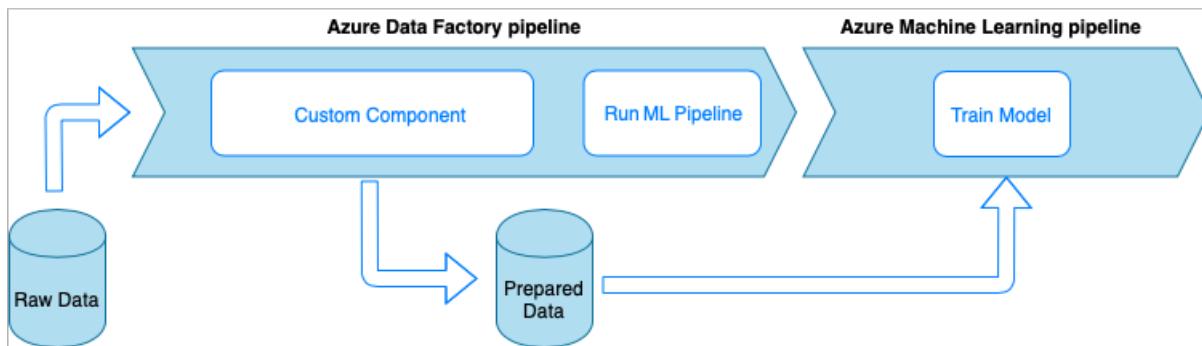
The function is invoked with the [ADF Azure Function activity](#). This approach is a good option for lightweight data transformations.

- Pros:
 - The data is processed on a serverless compute with a relatively low latency
 - ADF pipeline can invoke a [Durable Azure Function](#) that may implement a sophisticated data transformation flow
 - The details of the data transformation are abstracted away by the Azure Function that can be reused and

invoked from other places

- Cons:
 - The Azure Functions must be created before use with ADF
 - Azure Functions is good only for short running data processing

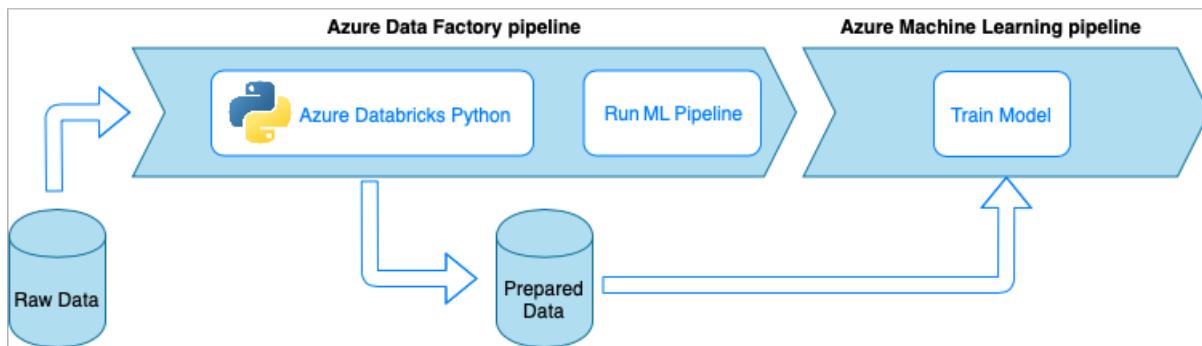
ADF with Custom Component Activity



In this option, the data is processed with custom Python code wrapped into an executable. It is invoked with an [ADF Custom Component activity](#). This approach is a better fit for large data than the previous technique.

- Pros:
 - The data is processed on [Azure Batch](#) pool, which provides large-scale parallel and high-performance computing
 - Can be used to run heavy algorithms and process significant amounts of data
- Cons:
 - Azure Batch pool must be created before use with ADF
 - Over engineering related to wrapping Python code into an executable. Complexity of handling dependencies and input/output parameters

ADF with Azure Databricks Python notebook



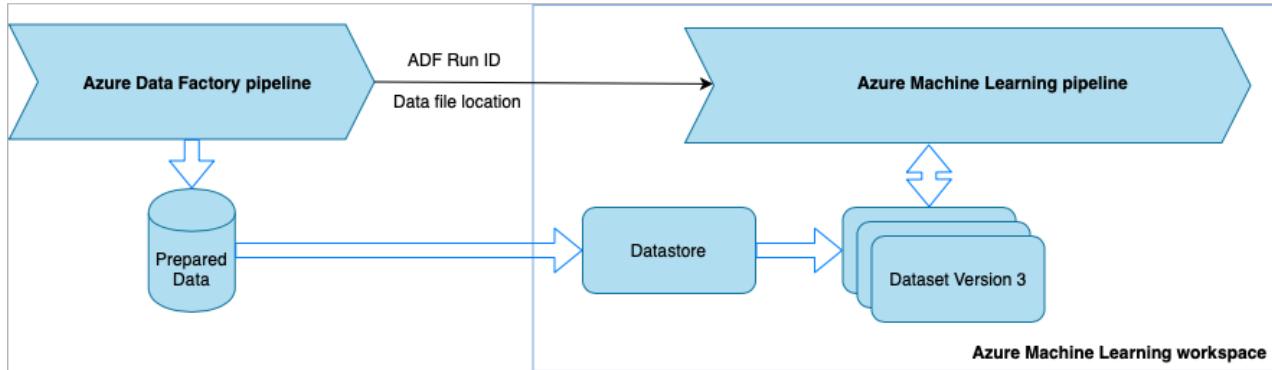
[Azure Databricks](#) is an Apache Spark-based analytics platform in the Microsoft cloud.

In this technique, the data transformation is performed by a [Python notebook](#), running on an Azure Databricks cluster. This is probably, the most common approach that leverages the full power of an Azure Databricks service. It is designed for distributed data processing at scale.

- Pros:
 - The data is transformed on the most powerful data processing Azure service, which is backed up by Apache Spark environment
 - Native support of Python along with data science frameworks and libraries including TensorFlow, PyTorch, and scikit-learn
 - There is no need to wrap the Python code into functions or executable modules. The code works as is.
- Cons:

- Azure Databricks infrastructure must be created before use with ADF
- Can be expensive depending on Azure Databricks configuration
- Spinning up compute clusters from "cold" mode takes some time that brings high latency to the solution

Consuming data in Azure Machine Learning pipelines



The transformed data from the ADF pipeline is saved to data storage (such as Azure Blob). Azure Machine Learning can access this data using [datastores](#) and [datasets](#).

Each time the ADF pipeline runs, the data is saved to a different location in storage. To pass the location to Azure Machine Learning, the ADF pipeline calls an Azure Machine Learning pipeline. When calling the ML pipeline, the data location and run ID are sent as parameters. The ML pipeline can then create a datastore/dataset using the data location.

TIP

Datasets [support versioning](#), so the ML pipeline can register a new version of the dataset that points to the most recent data from the ADF pipeline.

Once the data is accessible through a datastore or dataset, you can use it to train an ML model. The training process might be part of the same ML pipeline that is called from ADF. Or it might be a separate process such as experimentation in a Jupyter notebook.

Since datasets support versioning, and each run from the pipeline creates a new version, it's easy to understand which version of the data was used to train a model.

Next steps

- [Run a Databricks notebook in Azure Data Factory](#)
- [Access data in Azure storage services](#)
- [Train models with datasets in Azure Machine Learning](#)
- [DevOps for a data ingestion pipeline](#)

DevOps for a data ingestion pipeline

12/23/2020 • 11 minutes to read • [Edit Online](#)

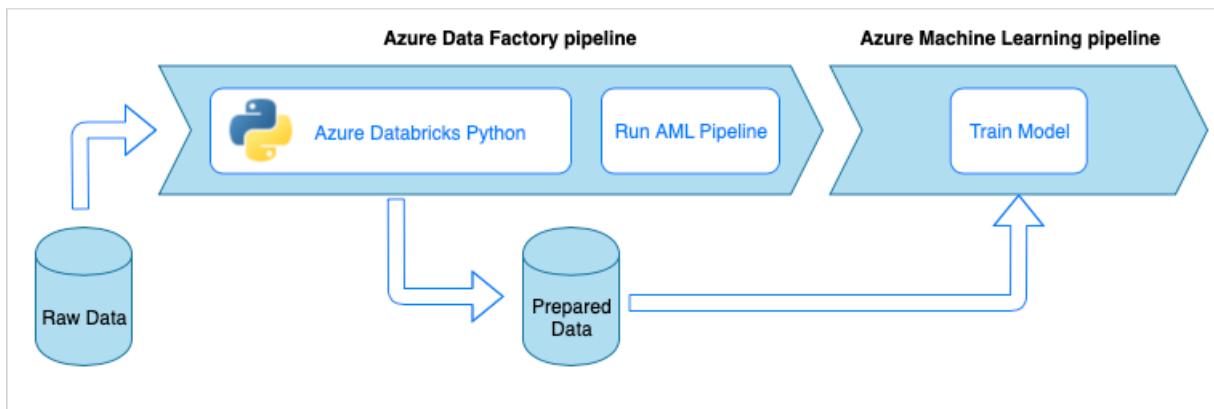
In most scenarios, a data ingestion solution is a composition of scripts, service invocations, and a pipeline orchestrating all the activities. In this article, you learn how to apply DevOps practices to the development lifecycle of a common data ingestion pipeline that prepares data for machine learning model training. The pipeline is built using the following Azure services:

- **Azure Data Factory**: Reads the raw data and orchestrates data preparation.
- **Azure Databricks**: Runs a Python notebook that transforms the data.
- **Azure Pipelines**: Automates a continuous integration and development process.

Data ingestion pipeline workflow

The data ingestion pipeline implements the following workflow:

1. Raw data is read into an Azure Data Factory (ADF) pipeline.
2. The ADF pipeline sends the data to an Azure Databricks cluster, which runs a Python notebook to transform the data.
3. The data is stored to a blob container, where it can be used by Azure Machine Learning to train a model.



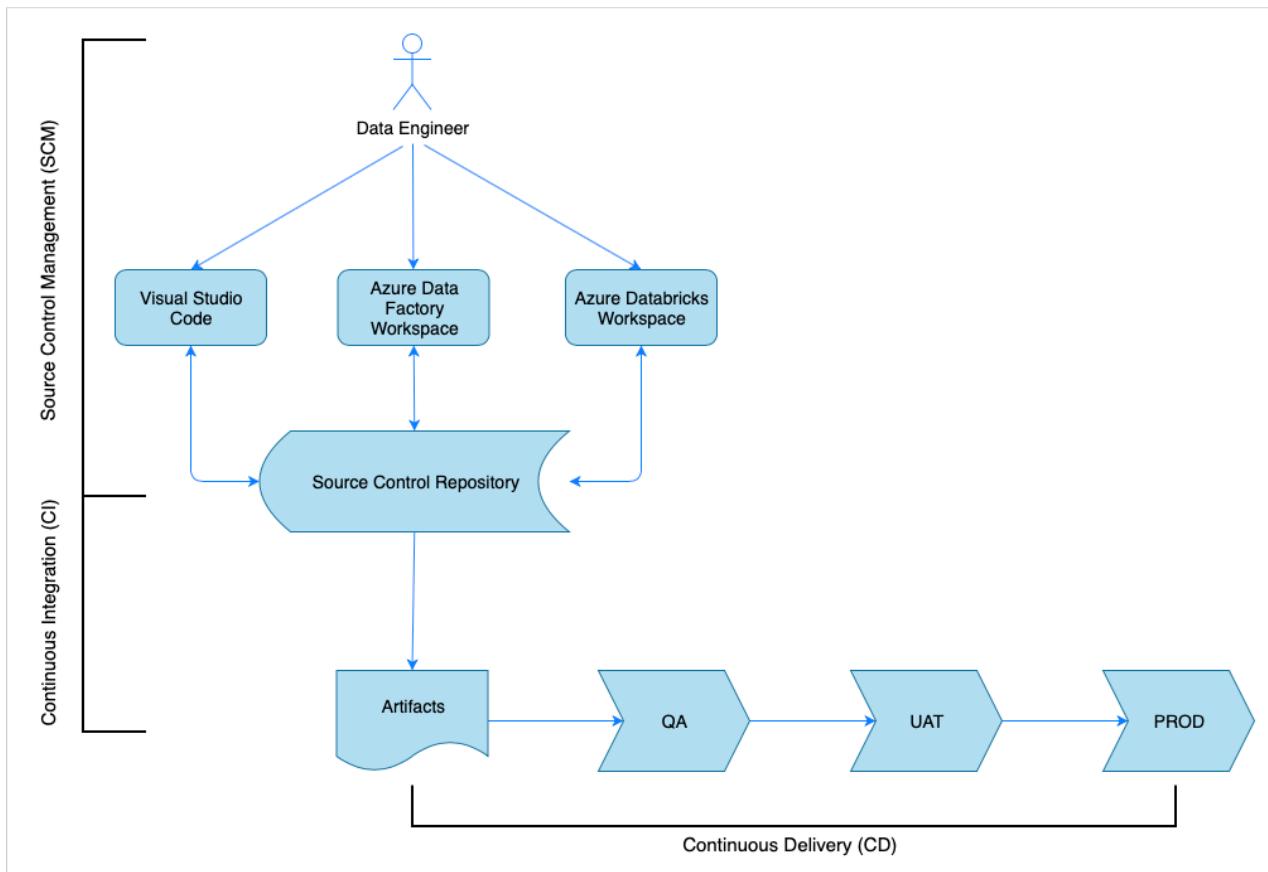
Continuous integration and delivery overview

As with many software solutions, there is a team (for example, Data Engineers) working on it. They collaborate and share the same Azure resources such as Azure Data Factory, Azure Databricks, and Azure Storage accounts. The collection of these resources is a Development environment. The data engineers contribute to the same source code base.

A continuous integration and delivery system automates the process of building, testing, and delivering (deploying) the solution. The Continuous Integration (CI) process performs the following tasks:

- Assembles the code
- Checks it with the code quality tests
- Runs unit tests
- Produces artifacts such as tested code and Azure Resource Manager templates

The Continuous Delivery (CD) process deploys the artifacts to the downstream environments.



This article demonstrates how to automate the CI and CD processes with [Azure Pipelines](#).

Source control management

Source control management is needed to track changes and enable collaboration between team members. For example, the code would be stored in an Azure DevOps, GitHub, or GitLab repository. The collaboration workflow is based on a branching model. For example, [GitFlow](#).

Python Notebook Source Code

The data engineers work with the Python notebook source code either locally in an IDE (for example, [Visual Studio Code](#)) or directly in the Databricks workspace. Once the code changes are complete, they are merged to the repository following a branching policy.

TIP

We recommend storing the code in `.py` files rather than in `.ipynb` Jupyter Notebook format. It improves the code readability and enables automatic code quality checks in the CI process.

Azure Data Factory Source Code

The source code of Azure Data Factory pipelines is a collection of JSON files generated by an Azure Data Factory workspace. Normally the data engineers work with a visual designer in the Azure Data Factory workspace rather than with the source code files directly.

To configure the workspace to use a source control repository, see [Author with Azure Repos Git integration](#).

Continuous integration (CI)

The ultimate goal of the Continuous Integration process is to gather the joint team work from the source code and prepare it for the deployment to the downstream environments. As with the source code management this process is different for the Python notebooks and Azure Data Factory pipelines.

Python Notebook CI

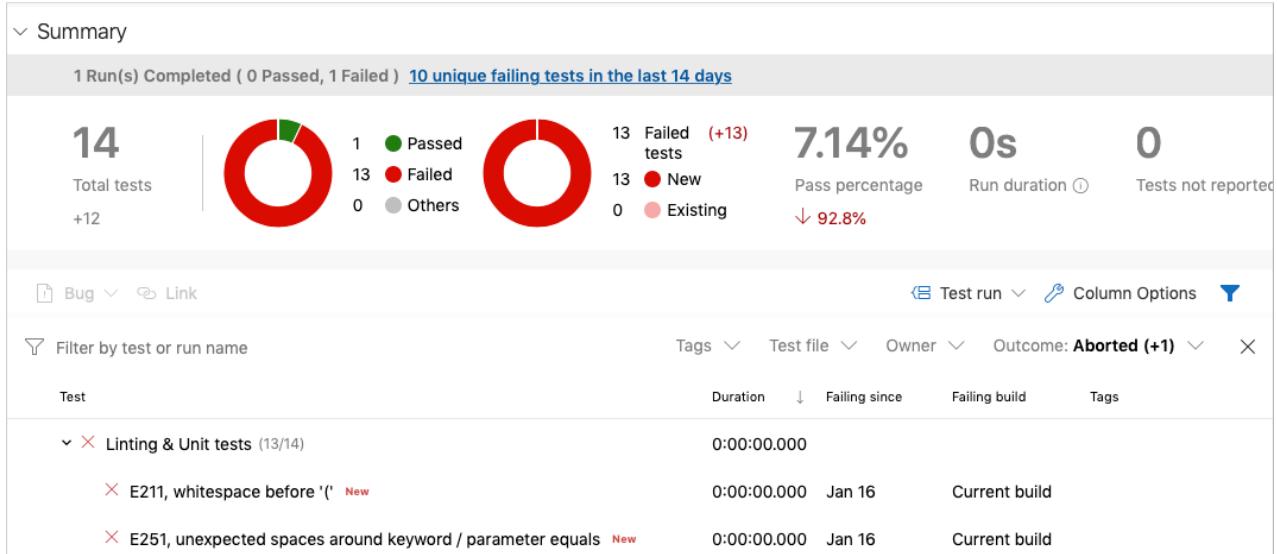
The CI process for the Python Notebooks gets the code from the collaboration branch (for example, *master* or *develop*) and performs the following activities:

- Code linting
- Unit testing
- Saving the code as an artifact

The following code snippet demonstrates the implementation of these steps in an Azure DevOps *yml* pipeline:

```
steps:  
- script: |  
  flake8 --output-file=$(Build.BinariesDirectory)/lint-testresults.xml --format junit-xml  
  workingDirectory: '$(Build.SourcesDirectory)'  
  displayName: 'Run flake8 (code style analysis)'  
  
- script: |  
  python -m pytest --junitxml=$(Build.BinariesDirectory)/unit-testresults.xml $(Build.SourcesDirectory)  
  displayName: 'Run unit tests'  
  
- task: PublishTestResults@2  
  condition: succeededOrFailed()  
  inputs:  
    testResultsFiles: ' $(Build.BinariesDirectory)/*-testresults.xml'  
    testRunTitle: 'Linting & Unit tests'  
    failTaskOnFailedTests: true  
  displayName: 'Publish linting and unit test results'  
  
- publish: $(Build.SourcesDirectory)  
  artifact: di-notebooks
```

The pipeline uses [flake8](#) to do the Python code linting. It runs the unit tests defined in the source code and publishes the linting and test results so they're available in the Azure Pipeline execution screen:



If the linting and unit testing is successful, the pipeline will copy the source code to the artifact repository to be used by the subsequent deployment steps.

Azure Data Factory CI

CI process for an Azure Data Factory pipeline is a bottleneck for a data ingestion pipeline. There's no continuous integration. A deployable artifact for Azure Data Factory is a collection of Azure Resource Manager templates. The only way to produce those templates is to click the *publish* button in the Azure Data Factory workspace.

1. The data engineers merge the source code from their feature branches into the collaboration branch, for

example, *master* or *develop*.

2. Someone with the granted permissions clicks the *publish* button to generate Azure Resource Manager templates from the source code in the collaboration branch.
3. The workspace validates the pipelines (think of it as of linting and unit testing), generates Azure Resource Manager templates (think of it as of building) and saves the generated templates to a technical branch *adf_publish* in the same code repository (think of it as of publishing artifacts). This branch is created automatically by the Azure Data Factory workspace.

For more information on this process, see [Continuous integration and delivery in Azure Data Factory](#).

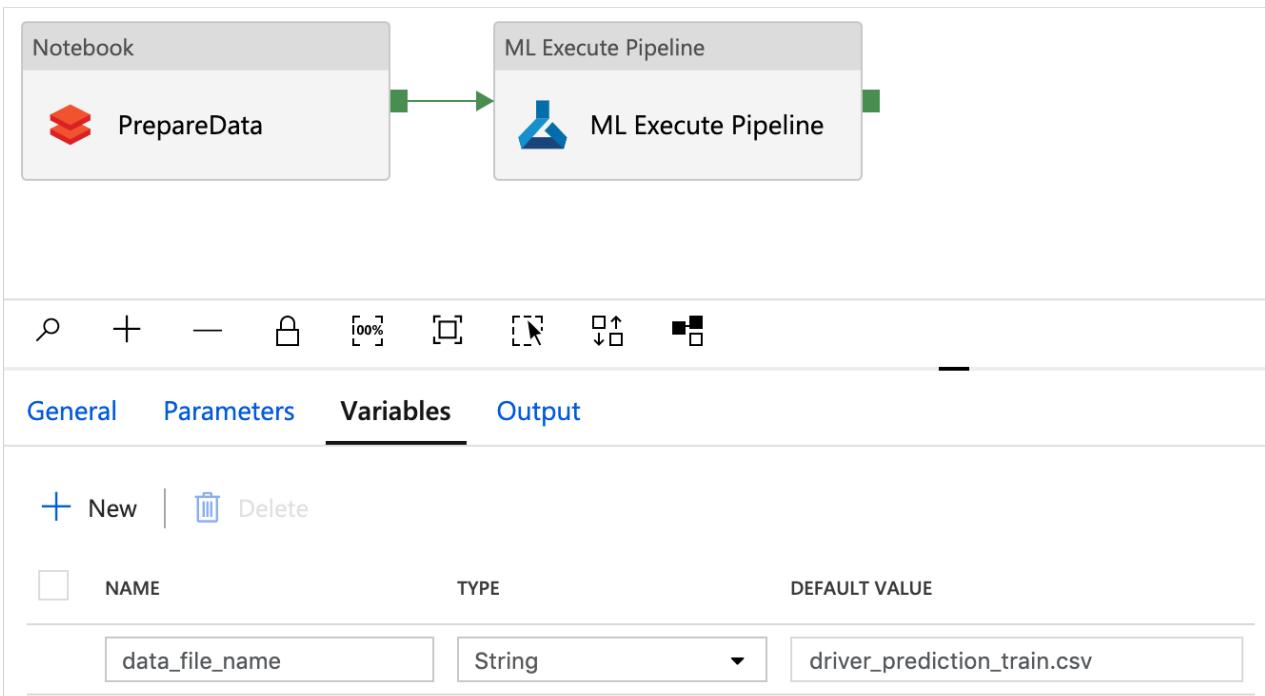
It's important to make sure that the generated Azure Resource Manager templates are environment agnostic. This means that all values that may differ between environments are parametrized. Azure Data Factory is smart enough to expose the majority of such values as parameters. For example, in the following template the connection properties to an Azure Machine Learning workspace are exposed as parameters:

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "factoryName": {  
            "value": "devops-ds-adf"  
        },  
        "AzureMLService_servicePrincipalKey": {  
            "value": ""  
        },  
        "AzureMLService_properties_typeProperties_subscriptionId": {  
            "value": "0fe1c235-5cfa-4152-17d7-5dff45a8d4ba"  
        },  
        "AzureMLService_properties_typeProperties_resourceGroupName": {  
            "value": "devops-ds-rg"  
        },  
        "AzureMLService_properties_typeProperties_servicePrincipalId": {  
            "value": "6e35e589-3b22-4edb-89d0-2ab7fc08d488"  
        },  
        "AzureMLService_properties_typeProperties_tenant": {  
            "value": "72f988bf-86f1-41af-912b-2d7cd611db47"  
        }  
    }  
}
```

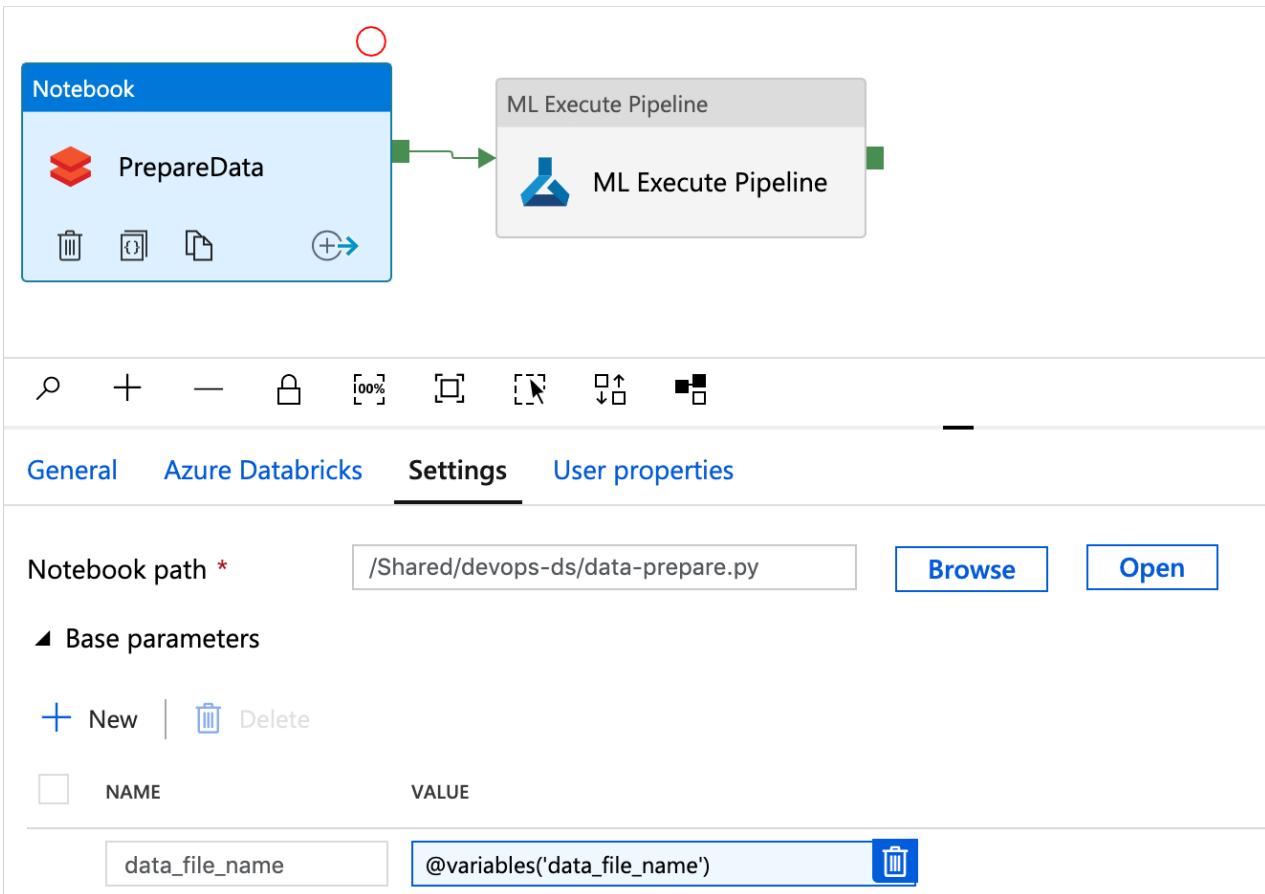
However, you may want to expose your custom properties that are not handled by the Azure Data Factory workspace by default. In the scenario of this article an Azure Data Factory pipeline invokes a Python notebook processing the data. The notebook accepts a parameter with the name of an input data file.

```
import pandas as pd  
import numpy as np  
  
data_file_name = getArguments("data_file_name")  
data = pd.read_csv(data_file_name)  
  
labels = np.array(data['target'])  
...
```

This name is different for *Dev*, *QA*, *UAT*, and *PROD* environments. In a complex pipeline with multiple activities, there can be several custom properties. It's good practice to collect all those values in one place and define them as pipeline *variables*:



The pipeline activities may refer to the pipeline variables while actually using them:



The Azure Data Factory workspace *doesn't* expose pipeline variables as Azure Resource Manager templates parameters by default. The workspace uses the [Default Parameterization Template](#) dictating what pipeline properties should be exposed as Azure Resource Manager template parameters. To add pipeline variables to the list, update the `"Microsoft.DataFactory/factories/pipelines"` section of the [Default Parameterization Template](#) with the following snippet and place the result json file in the root of the source folder:

```
"Microsoft.DataFactory/factories/pipelines": {  
    "properties": {  
        "variables": {  
            "*": {  
                "defaultValue": "="  
            }  
        }  
    }  
}
```

Doing so will force the Azure Data Factory workspace to add the variables to the parameters list when the *publish* button is clicked:

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "factoryName": {  
            "value": "devops-ds-adf"  
        },  
        ...  
        "data-ingestion-pipeline_properties_variables_data_file_nameDefaultValue": {  
            "value": "driver_prediction_train.csv"  
        }  
    }  
}
```

The values in the JSON file are default values configured in the pipeline definition. They're expected to be overridden with the target environment values when the Azure Resource Manager template is deployed.

Continuous delivery (CD)

The Continuous Delivery process takes the artifacts and deploys them to the first target environment. It makes sure that the solution works by running tests. If successful, it continues to the next environment.

The CD Azure Pipeline consists of multiple stages representing the environments. Each stage contains [deployments](#) and [jobs](#) that perform the following steps:

- Deploy a Python Notebook to Azure Databricks workspace
- Deploy an Azure Data Factory pipeline
- Run the pipeline
- Check the data ingestion result

The pipeline stages can be configured with [approvals](#) and [gates](#) that provide additional control on how the deployment process evolves through the chain of environments.

Deploy a Python Notebook

The following code snippet defines an Azure Pipeline [deployment](#) that copies a Python notebook to a Databricks cluster:

```

- stage: 'Deploy_to_QA'
  displayName: 'Deploy to QA'
  variables:
    - group: devops-ds-qa-vg
  jobs:
    - deployment: "Deploy_to_Databricks"
      displayName: 'Deploy to Databricks'
      timeoutInMinutes: 0
      environment: qa
      strategy:
        runOnce:
      deploy:
        steps:
          - task: UsePythonVersion@0
            inputs:
              versionSpec: '3.x'
              addToPath: true
              architecture: 'x64'
            displayName: 'Use Python3'

          - task: configuredatabricks@0
            inputs:
              url: '$(DATABRICKS_URL)'
              token: '$(DATABRICKS_TOKEN)'
            displayName: 'Configure Databricks CLI'

          - task: deploynotebooks@0
            inputs:
              notebooksFolderPath: '$(Pipeline.Workspace)/di-notebooks'
              workspaceFolder: '/Shared/devops-ds'
            displayName: 'Deploy (copy) data processing notebook to the Databricks cluster'

```

The artifacts produced by the CI are automatically copied to the deployment agent and are available in the `$(Pipeline.Workspace)` folder. In this case, the deployment task refers to the `di-notebooks` artifact containing the Python notebook. This [deployment](#) uses the [Databricks Azure DevOps extension](#) to copy the notebook files to the Databricks workspace.

The `Deploy_to_QA` stage contains a reference to the `devops-ds-qa-vg` variable group defined in the Azure DevOps project. The steps in this stage refer to the variables from this variable group (for example, `$(DATABRICKS_URL)` and `$(DATABRICKS_TOKEN)`). The idea is that the next stage (for example, `Deploy_to_UAT`) will operate with the same variable names defined in its own UAT-scoped variable group.

Deploy an Azure Data Factory pipeline

A deployable artifact for Azure Data Factory is an Azure Resource Manager template. It's going to be deployed with the [*Azure Resource Group Deployment*](#) task as it is demonstrated in the following snippet:

```

- deployment: "Deploy_to_ADF"
  displayName: 'Deploy to ADF'
  timeoutInMinutes: 0
  environment: qa
  strategy:
    runOnce:
      deploy:
        steps:
          - task: AzureResourceGroupDeployment@2
            displayName: 'Deploy ADF resources'
            inputs:
              azureSubscription: $(AZURE_RM_CONNECTION)
              resourceName: $(RESOURCE_GROUP)
              location: $(LOCATION)
              csmFile: '$(Pipeline.Workspace)/adf-pipelines/ARMTemplateForFactory.json'
              csmParametersFile: '$(Pipeline.Workspace)/adf-pipelines/ARMTemplateParametersForFactory.json'
              overrideParameters: -data-ingestion-pipeline_properties_variables_data_file_nameDefaultValue
              "$(DATA_FILE_NAME)"

```

The value of the data filename parameter comes from the `$(DATA_FILE_NAME)` variable defined in a QA stage variable group. Similarly, all parameters defined in *ARMTemplateForFactory.json* can be overridden. If they are not, then the default values are used.

Run the pipeline and check the data ingestion result

The next step is to make sure that the deployed solution is working. The following job definition runs an Azure Data Factory pipeline with a [PowerShell script](#) and executes a Python notebook on an Azure Databricks cluster. The notebook checks if the data has been ingested correctly and validates the result data file with `$(bin_FILE_NAME)` name.

```

- job: "Integration_test_job"
  displayName: "Integration test job"
  dependsOn: [Deploy_to_Databricks, Deploy_to_ADF]
  pool:
    vmImage: 'ubuntu-latest'
  timeoutInMinutes: 0
  steps:
    - task: AzurePowerShell@4
      displayName: 'Execute ADF Pipeline'
      inputs:
        azureSubscription: $(AZURE_RM_CONNECTION)
        ScriptPath: '$(Build.SourcesDirectory)/adf/utils/Invoke-ADFPipeline.ps1'
        ScriptArguments: '-ResourceGroupName $(RESOURCE_GROUP) -DataFactoryName $(DATA_FACTORY_NAME) -PipelineName $(PIPELINE_NAME)'
        azurePowerShellVersion: LatestVersion
    - task: UsePythonVersion@0
      inputs:
        versionSpec: '3.x'
        addToPath: true
        architecture: 'x64'
      displayName: 'Use Python3'

    - task: configuredatabricks@0
      inputs:
        url: '$(DATABRICKS_URL)'
        token: '$(DATABRICKS_TOKEN)'
      displayName: 'Configure Databricks CLI'

    - task: executenotebook@0
      inputs:
        notebookPath: '/Shared/devops-ds/test-data-ingestion'
        existingClusterId: '$(DATABRICKS_CLUSTER_ID)'
        executionParams: '{"bin_file_name": "$(bin_FILE_NAME)"}'
      displayName: 'Test data ingestion'

    - task: waitexecution@0
      displayName: 'Wait until the testing is done'

```

The final task in the job checks the result of the notebook execution. If it returns an error, it sets the status of pipeline execution to failed.

Putting pieces together

The complete CI/CD Azure Pipeline consists of the following stages:

- CI
- Deploy To QA
 - Deploy to Databricks + Deploy to ADF
 - Integration Test

It contains a number of ***Deploy*** stages equal to the number of target environments you have. Each ***Deploy*** stage contains two **deployments** that run in parallel and a **job** that runs after deployments to test the solution on the environment.

A sample implementation of the pipeline is assembled in the following ***yaml*** snippet:

```

variables:
- group: devops-ds-vg

stages:
- stage: 'CI'
  displayName: 'CI'
  ...

```



```

        displayName: 'Deploy (copy) data processing notebook to the Databricks cluster'
      - deployment: "Deploy_to_ADF"
        displayName: 'Deploy to ADF'
        timeoutInMinutes: 0
        environment: qa
        strategy:
          runOnce:
            deploy:
              steps:
                - task: AzureResourceGroupDeployment@2
                  displayName: 'Deploy ADF resources'
                  inputs:
                    azureSubscription: $(AZURE_RM_CONNECTION)
                    resourceGroupName: $(RESOURCE_GROUP)
                    location: $(LOCATION)
                    csmFile: '$(Pipeline.Workspace)/adf-pipelines/ARMTemplateForFactory.json'
                    csmParametersFile: '$(Pipeline.Workspace)/adf-pipelines/ARMTemplateParametersForFactory.json'
                    overrideParameters: -data-ingestion-pipeline_properties_variables_data_file_nameDefaultValue
$(DATA_FILE_NAME)"
                - job: "Integration_test_job"
                  displayName: "Integration test job"
                  dependsOn: [Deploy_to_Databricks, Deploy_to_ADF]
                  pool:
                    vmImage: 'ubuntu-latest'
                  timeoutInMinutes: 0
                  steps:
                    - task: AzurePowerShell@4
                      displayName: 'Execute ADF Pipeline'
                      inputs:
                        azureSubscription: $(AZURE_RM_CONNECTION)
                        ScriptPath: '$(Build.SourcesDirectory)/adf/utils/Invoke-ADFPipeline.ps1'
                        ScriptArguments: '-ResourceGroupName $(RESOURCE_GROUP) -DataFactoryName $(DATA_FACTORY_NAME) -PipelineName $(PIPELINE_NAME)'
                        azurePowerShellVersion: LatestVersion
                    - task: UsePythonVersion@0
                      inputs:
                        versionSpec: '3.x'
                        addToPath: true
                        architecture: 'x64'
                      displayName: 'Use Python3'

                    - task: configuredatabricks@0
                      inputs:
                        url: '$(DATABRICKS_URL)'
                        token: '$(DATABRICKS_TOKEN)'
                      displayName: 'Configure Databricks CLI'

                    - task: executenotebook@0
                      inputs:
                        notebookPath: '/Shared/devops-ds/test-data-ingestion'
                        existingClusterId: '$(DATABRICKS_CLUSTER_ID)'
                        executionParams: '{"bin_file_name": "$(bin_FILE_NAME)"}'
                      displayName: 'Test data ingestion'

                    - task: waitexecution@0
                      displayName: 'Wait until the testing is done'

```

Next steps

- [Source Control in Azure Data Factory](#)
- [Continuous integration and delivery in Azure Data Factory](#)
- [DevOps for Azure Databricks](#)

Import data into Azure Machine Learning designer

12/23/2020 • 4 minutes to read • [Edit Online](#)

In this article, you learn how to import your own data in the designer to create custom solutions. There are two ways you can import data into the designer:

- **Azure Machine Learning datasets** - Register [datasets](#) in Azure Machine Learning to enable advanced features that help you manage your data.
- **Import Data module** - Use the [Import Data](#) module to directly access data from online datasources.

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 Azure Machine Learning datasets

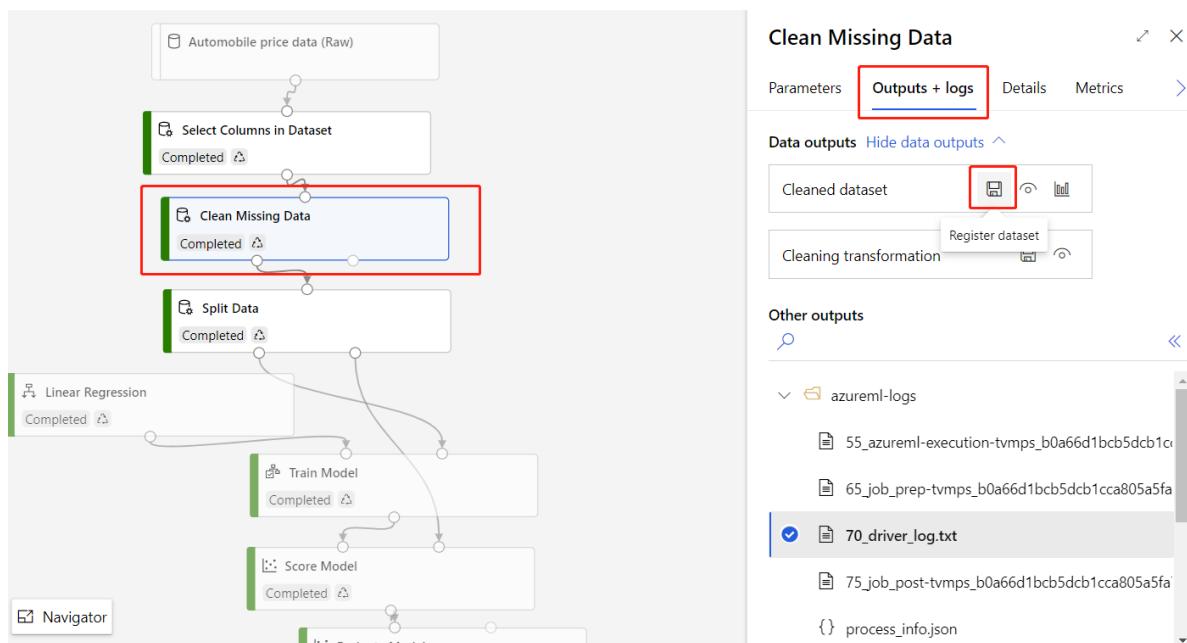
We recommend that you use [datasets](#) to import data into the designer. When you register a dataset, you can take full advantage of advanced data features like [versioning and tracking](#) and [data monitoring](#).

Register a dataset

You can register existing datasets [programmatically with the SDK](#) or [visually in Azure Machine Learning studio](#).

You can also register the output for any designer module as a dataset.

1. Select the module that outputs the data you want to register.
2. In the properties pane, select **Outputs + logs > Register dataset**.



If the module output data is in a tabular format, you must choose to register the output as a [file dataset](#) or [tabular dataset](#).

- **File dataset** registers the module's output folder as a file dataset. The output folder contains a data file and

meta files that the designer uses internally. Select this option if you want to continue to use the registered dataset in the designer.

- **Tabular dataset** registers only the module's the output data file as a tabular dataset. This format is easily consumed by other tools, for example in Automated Machine Learning or the Python SDK. Select this option if you plan to use the registered dataset outside of the designer.

Use a dataset

Your registered datasets can be found in the module palette, under **Datasets**. To use a dataset, drag and drop it onto the pipeline canvas. Then, connect the output port of the dataset to other modules in the canvas.

If you register a file dataset, the output port type of the dataset is **AnyDirectory**. If you register a Tabular dataset, the output port type of the dataset if **DataFrameDirectory**. Note that if you connect the output port of the dataset to other modules in the designer, the port type of datasets and modules need to be aligned.

99 assets in total

↻

▼ Datasets (5)

New Cleaned dataset
9/9/2020

MD-Sample_pipeline-Train_Model-Trained...
9/9/2020

Cleaned dataset
9/9/2020

NOTE
The designer supports [dataset versioning](#). Specify the dataset version in the property panel of the dataset module.

Limitations

- Currently you can only visualize tabular dataset in the designer. If you register a file dataset outside designer, you cannot visualize it in the designer canvas.
- Your dataset is stored in virtual network (VNet). If you want to visualize, you need to enable workspace managed identity of the datastore.

1. Go the the related datastore and click **Update Credentials**

workspaceblobstore (Default)

[Overview](#) [Browse \(preview\)](#)

↻ Refresh  Update credentials  Create dataset

2. Select **Yes** to enable workspace managed identity.

Update datastore credentials

X

Authentication type * ⓘ

Account key

*

Account key * ⓘ

Use workspace managed identity for data preview and profiling in Azure Machine Learning studio ⓘ

No

Yes

i

(i) Note: Azure Machine Learning service does not validate whether the underlying data source exi... ▾

Save

Cancel

Import data using the Import Data module

While we recommend that you use datasets to import data, you can also use the [Import Data](#) module. The Import Data module skips registering your dataset in Azure Machine Learning and imports data directly from a [datastore](#) or HTTP URL.

For detailed information on how to use the Import Data module, see the [Import Data reference page](#).

NOTE

If your dataset has too many columns, you may encounter the following error: "Validation failed due to size limitation". To avoid this, [register the dataset in the Datasets interface](#).

Supported sources

This section lists the data sources supported by the designer. Data comes into the designer from either a datastore or from [tabular dataset](#).

Datastore sources

For a list of supported datastore sources, see [Access data in Azure storage services](#).

Tabular dataset sources

The designer supports tabular datasets created from the following sources:

- Delimited files
- JSON files
- Parquet files
- SQL queries

Data types

The designer internally recognizes the following data types:

- String
- Integer
- Decimal
- Boolean
- Date

The designer uses an internal data type to pass data between modules. You can explicitly convert your data into data table format using the [Convert to Dataset](#) module. Any module that accepts formats other than the internal format will convert the data silently before passing it to the next module.

Data constraints

Modules in the designer are limited by the size of the compute target. For larger datasets, you should use a larger Azure Machine Learning compute resource. For more information on Azure Machine Learning compute, see [What are compute targets in Azure Machine Learning?](#)

Access data in a virtual network

If your workspace is in a virtual network, you must perform additional configuration steps to visualize data in the designer. For more information on how to use datastores and datasets in a virtual network, see [Use Azure Machine Learning studio in an Azure virtual network](#).

Next steps

Learn the designer fundamentals with this [Tutorial: Predict automobile price with the designer](#).

Connect to storage services on Azure

12/23/2020 • 11 minutes to read • [Edit Online](#)

In this article, learn how to **connect to storage services on Azure via Azure Machine Learning datastores**. Datastores securely connect to your Azure storage service without putting your authentication credentials and the integrity of your original data source at risk. They store connection information, like your subscription ID and token authorization in your [Key Vault](#) associated with the workspace, so you can securely access your storage without having to hard code them in your scripts. You can use the [Azure Machine Learning Python SDK](#) or the [Azure Machine Learning studio](#) to create and register datastores.

If you prefer to create and manage datastores using the Azure Machine Learning VS Code extension; visit the [VS Code resource management how-to guide](#) to learn more.

You can create datastores from [these Azure storage solutions](#). For unsupported storage solutions, and to save data egress cost during ML experiments, [move your data](#) to a supported Azure storage solution.

To understand where datastores fit in Azure Machine Learning's overall data access workflow, see the [Securely access data](#) article.

Prerequisites

You'll need:

- 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 storage account with a [supported storage type](#).
- The [Azure Machine Learning SDK for Python](#), or access to [Azure Machine Learning studio](#).
- An Azure Machine Learning workspace.

Either [create an Azure Machine Learning workspace](#) or use an existing one via the Python SDK.

Import the `Workspace` and `Datastore` class, and load your subscription information from the file `config.json` using the function `from_config()`. This looks for the JSON file in the current directory by default, but you can also specify a path parameter to point to the file using `from_config(path="your/file/path")`.

```
import azureml.core
from azureml.core import Workspace, Datastore

ws = Workspace.from_config()
```

When you create a workspace, an Azure blob container and an Azure file share are automatically registered as datastores to the workspace. They're named `workspaceblobstore` and `workspacefilestore`, respectively. The `workspaceblobstore` is used to store workspace artifacts and your machine learning experiment logs. It's also set as the **default datastore** and can't be deleted from the workspace. The `workspacefilestore` is used to store notebooks and R scripts authorized via [compute instance](#).

NOTE

Azure Machine Learning designer will create a datastore named `azureml_globaldatasets` automatically when you open a sample in the designer homepage. This datastore only contains sample datasets. Please **do not** use this datastore for any confidential data access.

Supported data storage service types

Datastores currently support storing connection information to the storage services listed in the following matrix.

STORAGE TYPE	AUTHENTICATION TYPE	AZURE MACHINE LEARNING STUDIO	AZURE MACHINE LEARNING PYTHON SDK	AZURE MACHINE LEARNING CLI	AZURE MACHINE LEARNING REST API	VS CODE
Azure Blob Storage	Account key SAS token	✓	✓	✓	✓	✓
Azure File Share	Account key SAS token	✓	✓	✓	✓	✓
Azure Data Lake Storage Gen 1	Service principal	✓	✓	✓	✓	
Azure Data Lake Storage Gen 2	Service principal	✓	✓	✓	✓	
Azure SQL Database	SQL authentication Service principal	✓	✓	✓	✓	
Azure PostgreSQL	SQL authentication	✓	✓	✓	✓	
Azure Database for MySQL	SQL authentication		✓*	✓*	✓*	
Databricks File System	No authentication		✓**	✓ **	✓**	

* MySQL is only supported for pipeline [DataTransferStep](#)

** Databricks is only supported for pipeline [DatabricksStep](#)

Storage guidance

We recommend creating a datastore for an [Azure Blob container](#). Both standard and premium storage are available for blobs. Although premium storage is more expensive, its faster throughput speeds might improve the speed of your training runs, particularly if you train against a large dataset. For information about the cost of storage accounts, see the [Azure pricing calculator](#).

Azure Data Lake Storage Gen2 is built on top of Azure Blob storage and designed for enterprise big data analytics. A fundamental part of Data Lake Storage Gen2 is the addition of a [hierarchical namespace](#) to Blob storage. The hierarchical namespace organizes objects/files into a hierarchy of directories for efficient data access.

Storage access and permissions

To ensure you securely connect to your Azure storage service, Azure Machine Learning requires that you have permission to access the corresponding data storage container. This access depends on the authentication credentials used to register the datastore.

Virtual network

If your data storage account is in a [virtual network](#), additional configuration steps are required to ensure Azure Machine Learning has access to your data. See [Use Azure Machine Learning studio in an Azure virtual network](#) to ensure the appropriate configuration steps are applied when you create and register your datastore.

Access validation

As part of the initial datastore creation and registration process, Azure Machine Learning automatically validates that the underlying storage service exists and the user provided principal (username, service principal, or SAS token) has access to the specified storage.

After datastore creation, this validation is only performed for methods that require access to the underlying storage container, **not** each time datastore objects are retrieved. For example, validation happens if you want to download files from your datastore; but if you just want to change your default datastore, then validation does not happen.

To authenticate your access to the underlying storage service, you can provide either your account key, shared access signatures (SAS) tokens, or service principal in the corresponding `register_azure_*()` method of the datastore type you want to create. The [storage type matrix](#) lists the supported authentication types that correspond to each datastore type.

You can find account key, SAS token, and service principal information on your [Azure portal](#).

- If you plan to use an account key or SAS token for authentication, select **Storage Accounts** on the left pane, and choose the storage account that you want to register.
 - The **Overview** page provides information such as the account name, container, and file share name.
 1. For account keys, go to **Access keys** on the **Settings** pane.
 2. For SAS tokens, go to **Shared access signatures** on the **Settings** pane.
- If you plan to use a service principal for authentication, go to your **App registrations** and select which app you want to use.
 - Its corresponding **Overview** page will contain required information like tenant ID and client ID.

IMPORTANT

- If you need to change your access keys for an Azure Storage account (account key or SAS token), be sure to sync the new credentials with your workspace and the datastores connected to it. Learn how to [sync your updated credentials](#).

Permissions

For Azure blob container and Azure Data Lake Gen 2 storage, make sure your authentication credentials have **Storage Blob Data Reader** access. Learn more about [Storage Blob Data Reader](#). An account SAS token defaults to no permissions.

- For data **read access**, your authentication credentials must have a minimum of list and read permissions

for containers and objects.

- For data **write access**, write and add permissions also are required.

Create and register datastores

When you register an Azure storage solution as a datastore, you automatically create and register that datastore to a specific workspace. Review the [storage access & permissions](#) section for guidance on virtual network scenarios, and where to find required authentication credentials.

Within this section are examples for how to create and register a datastore via the Python SDK for the following storage types. The parameters provided in these examples are the **required parameters** to create and register a datastore.

- [Azure blob container](#)
- [Azure file share](#)
- [Azure Data Lake Storage Generation 2](#)

To create datastores for other supported storage services, see the [reference documentation for the applicable register_azure_* methods](#).

If you prefer a low code experience, see [Connect to data with Azure Machine Learning studio](#).

IMPORTANT

If you unregister and re-register a datastore with the same name, and it fails, the Azure Key Vault for your workspace may not have soft-delete enabled. By default, soft-delete is enabled for the key vault instance created by your workspace, but it may not be enabled if you used an existing key vault or have a workspace created prior to October 2020. For information on how to enable soft-delete, see [Turn on Soft Delete for an existing key vault](#)."

NOTE

Datastore name should only consist of lowercase letters, digits and underscores.

Azure blob container

To register an Azure blob container as a datastore, use `register_azure_blob_container()`.

The following code creates and registers the `blob_datastore_name` datastore to the `ws` workspace. This datastore accesses the `my-container-name` blob container on the `my-account-name` storage account, by using the provided account access key. Review the [storage access & permissions](#) section for guidance on virtual network scenarios, and where to find required authentication credentials.

```
blob_datastore_name='azblobsdk' # Name of the datastore to workspace
container_name=os.getenv("BLOB_CONTAINER", "<my-container-name>") # Name of Azure blob container
account_name=os.getenv("BLOB_ACCOUNTNAME", "<my-account-name>") # Storage account name
account_key=os.getenv("BLOB_ACCOUNT_KEY", "<my-account-key>") # Storage account access key

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)
```

Azure file share

To register an Azure file share as a datastore, use `register_azure_file_share()`.

The following code creates and registers the `file_datastore_name` datastore to the `ws` workspace. This datastore accesses the `my-fileshare-name` file share on the `my-account-name` storage account, by using the provided account access key. Review the [storage access & permissions](#) section for guidance on virtual network scenarios, and where to find required authentication credentials.

```
file_datastore_name='azfilesharesdk' # Name of the datastore to workspace
file_share_name=os.getenv("FILE_SHARE_CONTAINER", "<my-fileshare-name>") # Name of Azure file share
account_name=os.getenv("FILE_SHARE_ACCOUNTNAME", "<my-account-name>") # Storage account name
account_key=os.getenv("FILE_SHARE_ACCOUNT_KEY", "<my-account-key>") # Storage account access key

file_datastore = Datastore.register_azure_file_share(workspace=ws,
                                                    datastore_name=file_datastore_name,
                                                    file_share_name=file_share_name,
                                                    account_name=account_name,
                                                    account_key=account_key)
```

Azure Data Lake Storage Generation 2

For an Azure Data Lake Storage Generation 2 (ADLS Gen 2) datastore, use `register_azure_data_lake_gen2()` to register a credential datastore connected to an Azure DataLake Gen 2 storage with [service principal permissions](#).

In order to utilize your service principal, you need to [register your application](#) and grant the service principal data access via either Azure role-based access control (Azure RBAC) or access control lists (ACL). Learn more about [access control set up for ADLS Gen 2](#).

The following code creates and registers the `adlsgen2_datastore_name` datastore to the `ws` workspace. This datastore accesses the file system `test` in the `account_name` storage account, by using the provided service principal credentials. Review the [storage access & permissions](#) section for guidance on virtual network scenarios, and where to find required authentication credentials.

```
adlsgen2_datastore_name = 'adlsgen2datastore'

subscription_id=os.getenv("ADL_SUBSCRIPTION", "<my_subscription_id>") # subscription id of ADLS account
resource_group=os.getenv("ADL_RESOURCE_GROUP", "<my_resource_group>") # resource group of ADLS account

account_name=os.getenv("ADLGEN2_ACCOUNTNAME", "<my_account_name>") # ADLS Gen2 account name
tenant_id=os.getenv("ADLGEN2_TENANT", "<my_tenant_id>") # tenant id of service principal
client_id=os.getenv("ADLGEN2_CLIENTID", "<my_client_id>") # client id of service principal
client_secret=os.getenv("ADLGEN2_CLIENT_SECRET", "<my_client_secret>") # the secret of service principal

adlsgen2_datastore = Datastore.register_azure_data_lake_gen2(workspace=ws,
                                                               datastore_name=adlsgen2_datastore_name,
                                                               account_name=account_name, # ADLS Gen2 account
                                                               name
                                                               filesystem='test', # ADLS Gen2 filesystem
                                                               tenant_id=tenant_id, # tenant id of service
                                                               principal
                                                               client_id=client_id, # client id of service
                                                               principal
                                                               client_secret=client_secret) # the secret of
                                                               service principal
```

Create datastores using Azure Resource Manager

There are a number of templates at https://github.com/Azure/azure-quickstart-templates/tree/master/101-machine-learning-datastore-create-* that can be used to create datastores.

For information on using these templates, see [Use an Azure Resource Manager template to create a workspace for Azure Machine Learning](#).

Use data in your datastores

After you create a datastore, [create an Azure Machine Learning dataset](#) to interact with your data. Datasets package your data into a lazily evaluated consumable object for machine learning tasks, like training. They also provide the ability to [download or mount](#) files of any format from Azure storage services like, Azure Blob storage and ADLS Gen 2. You can also use them to load tabular data into a pandas or Spark DataFrame.

Get datastores from your workspace

To get a specific datastore registered in the current workspace, use the `get()` static method on the `Datastore` class:

```
# Get a named datastore from the current workspace
datastore = Datastore.get(ws, datastore_name='your datastore name')
```

To get the list of datastores registered with a given workspace, you can use the `datastores` property on a workspace object:

```
# List all datastores registered in the current workspace
datastores = ws.datastores
for name, datastore in datastores.items():
    print(name, datastore.datastore_type)
```

To get the workspace's default datastore, use this line:

```
datastore = ws.get_default_datastore()
```

You can also change the default datastore with the following code. This ability is only supported via the SDK.

```
ws.set_default_datastore(new_default_datastore)
```

Access data during scoring

Azure Machine Learning provides several ways to use your models for scoring. Some of these methods don't provide access to datastores. Use the following table to understand which methods allow you to access datastores during scoring:

METHOD	DATASTORE ACCESS	DESCRIPTION
Batch prediction	✓	Make predictions on large quantities of data asynchronously.
Web service		Deploy models as a web service.
Azure IoT Edge module		Deploy models to IoT Edge devices.

For situations where the SDK doesn't provide access to datastores, you might be able to create custom code by using the relevant Azure SDK to access the data. For example, the [Azure Storage SDK for Python](#) is a client library that you can use to access data stored in blobs or files.

Move data to supported Azure storage solutions

Azure Machine Learning supports accessing data from Azure Blob storage, Azure Files, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure SQL Database, and Azure Database for PostgreSQL. If you're using unsupported storage, we recommend that you move your data to supported Azure storage solutions by using [Azure Data Factory and these steps](#). Moving data to supported storage can help you save data egress costs during machine learning experiments.

Azure Data Factory provides efficient and resilient data transfer with more than 80 prebuilt connectors at no additional cost. These connectors include Azure data services, on-premises data sources, Amazon S3 and Redshift, and Google BigQuery.

Next steps

- [Create an Azure machine learning dataset](#)
- [Train a model](#)
- [Deploy a model](#)

Create Azure Machine Learning datasets

12/23/2020 • 9 minutes to read • [Edit Online](#)

In this article, you learn how to create Azure Machine Learning datasets to access data for your local or remote experiments with the Azure Machine Learning Python SDK. To understand where datasets fit in Azure Machine Learning's overall data access workflow, see the [Securely access data](#) article.

By creating a dataset, 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. Also datasets are lazily evaluated, which aids in workflow performance speeds. You can create datasets from datastores, public URLs, and [Azure Open Datasets](#).

For a low-code experience, [Create Azure Machine Learning datasets with the Azure Machine Learning studio..](#)

With Azure Machine Learning datasets, you can:

- Keep a single copy of data in your storage, referenced by datasets.
- Seamlessly access data during model training without worrying about connection strings or data paths.[Learn more about how to train with datasets](#).
- Share data and collaborate with other users.

Prerequisites

To create and work with datasets, 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](#).
- The [Azure Machine Learning SDK for Python installed](#), which includes the azureml-datasets package.
 - Create an [Azure Machine Learning compute instance](#), which is a fully configured and managed development environment that includes integrated notebooks and the SDK already installed.

OR

- Work on your own Jupyter notebook and install the SDK yourself with [these instructions](#).

NOTE

Some dataset classes have dependencies on the [azureml-dataprep](#) package, which is only compatible with 64-bit Python. For Linux users, these classes are supported only on the following distributions: Red Hat Enterprise Linux (7, 8), Ubuntu (14.04, 16.04, 18.04), Fedora (27, 28), Debian (8, 9), and CentOS (7). If you are using unsupported distros, please follow [this guide](#) to install .NET Core 2.1 to proceed.

Compute size guidance

When creating a dataset, review your compute processing power and the size of your data in memory. The size of your data in storage is not the same as the size of data in a dataframe. For example, data in CSV files can expand up to 10x in a dataframe, so a 1 GB CSV file can become 10 GB in a dataframe.

If your data is compressed, it can expand further; 20 GB of relatively sparse data stored in compressed

parquet format can expand to ~800 GB in memory. Since Parquet files store data in a columnar format, if you only need half of the columns, then you only need to load ~400 GB in memory.

[Learn more about optimizing data processing in Azure Machine Learning.](#)

Dataset types

There are two dataset types, based on how users consume them in training; FileDatasets and TabularDatasets. Both types can be used in Azure Machine Learning training workflows involving, estimators, AutoML, hyperDrive and pipelines.

FileDataset

A [FileDataset](#) references single or multiple files in your datastores or public URLs. If your data is already cleansed, and ready to use in training experiments, you can [download or mount](#) the files to your compute as a FileDataset object.

We recommend FileDatasets for your machine learning workflows, since the source files can be in any format, which enables a wider range of machine learning scenarios, including deep learning.

Create a FileDataset with the [Python SDK](#) or the [Azure Machine Learning studio](#).

TabularDataset

A [TabularDataset](#) represents data in a tabular format by parsing the provided file or list of files. This provides you with the ability to materialize the data into a pandas or Spark DataFrame so you can work with familiar data preparation and training libraries without having to leave your notebook. You can create a [TabularDataset](#) object from .csv, .tsv, .parquet, jsonl files, and from [SQL query results](#).

With TabularDatasets, you can specify a time stamp from a column in the data or from wherever the path pattern data is stored to enable a time series trait. This specification allows for easy and efficient filtering by time. For an example, see [Tabular time series-related API demo with NOAA weather data](#).

Create a TabularDataset with [the Python SDK](#) or [Azure Machine Learning studio](#).

NOTE

AutoML workflows generated via the Azure Machine Learning studio currently only support TabularDatasets.

Access datasets in a virtual network

If your workspace is in a virtual network, you must configure the dataset to skip validation. For more information on how to use datastores and datasets in a virtual network, see [Secure a workspace and associated resources](#).

Create datasets

For the data to be accessible by Azure Machine Learning, datasets must be created from paths in [Azure datastores](#) or public web URLs.

To create datasets from an [Azure datastore](#) with the Python SDK:

1. Verify that you have [contributor](#) or [owner](#) access to the registered Azure datastore.
2. Create the dataset by referencing paths in the datastore. You can create a dataset from multiple paths in multiple datastores. There is no hard limit on the number of files or data size that you can create a dataset from.

NOTE

For each data path, a few requests will be sent to the storage service to check whether it points to a file or a folder. This overhead may lead to degraded performance or failure. A dataset referencing one folder with 1000 files inside is considered referencing one data path. We recommend creating dataset referencing less than 100 paths in datastores for optimal performance.

Create a FileDataset

Use the `from_files()` method on the `FileDatasetFactory` class to load files in any format and to create an unregistered FileDataset.

If your storage is behind a virtual network or firewall, set the parameter `validate=False` in your `from_files()` method. This bypasses the initial validation step, and ensures that you can create your dataset from these secure files. Learn more about how to [use datastores and datasets in a virtual network](#).

```
# create a FileDataset pointing to files in 'animals' folder and its subfolders recursively
datastore_paths = [(datastore, 'animals')]
animal_ds = Dataset.File.from_files(path=datastore_paths)

# create a FileDataset from image and label files behind public web urls
web_paths = ['https://azuredataprocstorage.blob.core.windows.net/mnist/train-images-idx3-ubyte.gz',
             'https://azuredataprocstorage.blob.core.windows.net/mnist/train-labels-idx1-ubyte.gz']
mnist_ds = Dataset.File.from_files(path=web_paths)
```

To reuse and share datasets across experiment in your workspace, [register your dataset](#).

TIP

Upload files from a local directory and create a FileDataset in a single method with the public preview method, `upload_directory()`. This method is an [experimental](#) preview feature, and may change at any time.

This method uploads data to your underlying storage, and as a result incur storage costs.

Create a TabularDataset

Use the `from_delimited_files()` method on the `TabularDatasetFactory` class to read files in .csv or .tsv format, and to create an unregistered TabularDataset. If you're reading from multiple files, results will be aggregated into one tabular representation.

If your storage is behind a virtual network or firewall, set the parameter `validate=False` in your `from_delimited_files()` method. This bypasses the initial validation step, and ensures that you can create your dataset from these secure files. Learn more about how to use [datastores and datasets in a virtual network](#).

The following code gets the existing workspace and the desired datastore by name. And then passes the datastore and file locations to the `path` parameter to create a new TabularDataset, `weather_ds`.

```

from azureml.core import Workspace, Datastore, Dataset

datastore_name = 'your datastore name'

# get existing workspace
workspace = Workspace.from_config()

# retrieve an existing datastore in the workspace by name
datastore = Datastore.get(workspace, datastore_name)

# create a TabularDataset from 3 file paths in datastore
datastore_paths = [(datastore, 'weather/2018/11.csv'),
                   (datastore, 'weather/2018/12.csv'),
                   (datastore, 'weather/2019/*.csv')]

weather_ds = Dataset.Tabular.from_delimited_files(path=datastore_paths)

```

By default, when you create a TabularDataset, column data types are inferred automatically. If the inferred types don't match your expectations, you can specify column types by using the following code. The parameter `infer_column_type` is only applicable for datasets created from delimited files. [Learn more about supported data types](#).

```

from azureml.core import Dataset
from azureml.data.dataset_factory import DataType

# create a TabularDataset from a delimited file behind a public web url and convert column "Survived" to boolean
web_path ='https://dprepdata.blob.core.windows.net/demo/Titanic.csv'
titanic_ds = Dataset.Tabular.from_delimited_files(path=web_path, set_column_types={'Survived': DataType.to_bool()})

# preview the first 3 rows of titanic_ds
titanic_ds.take(3).to_pandas_dataframe()

```

(INDEX)	PASSengerID	SURVived	PCLASS	NAMe	SEX	AGE	SIBSp	PARCH	TICKET	FARE	CABIN	EMBARKEd
0	1	False	3	Braund, Mr. Owen Harri	male	22.0	1	0	A/5 21171	7.2500		S
1	2	True	1	Cumings, Mrs. John Bradley (Florence Brigg	fema	38.0	1	0	PC 17599	71.2833	C85	C

(IND EX)	PASS ENGERID	SURVIVED	PCLASS	NAM E	SEX	AGE	SIBSP	PARC H	TICK ET	FARE	CABIN	EMBARKE D
2	3	True	3	Heik kine n, Miss. Laina	fema le	26.0	0	0	STO N/O 2. 3101 282	7.92 50		S

To reuse and share datasets across experiments in your workspace, [register your dataset](#).

Create a dataset from pandas dataframe

To create a TabularDataset from an in memory pandas dataframe, write the data to a local file, like a csv, and create your dataset from that file. The following code demonstrates this workflow.

```
# azureml-core of version 1.0.72 or higher is required
# azureml-dataprep[pandas] of version 1.1.34 or higher is required

from azureml.core import Workspace, Dataset
local_path = 'data/prepared.csv'
dataframe.to_csv(local_path)

# upload the local file to a datastore on the cloud

subscription_id = 'xxxxxxxxxxxxxxxxxxxxxx'
resource_group = 'xxxxxx'
workspace_name = 'xxxxxxxxxxxxxxxx'

workspace = Workspace(subscription_id, resource_group, workspace_name)

# get the datastore to upload prepared data
datastore = workspace.get_default_datastore()

# upload the local file from src_dir to the target_path in datastore
datastore.upload(src_dir='data', target_path='data')

# create a dataset referencing the cloud location
dataset = Dataset.Tabular.from_delimited_files(path = [(datastore, ('data/prepared.csv'))])
```

TIP

Create and register a TabularDataset from an in memory spark or pandas dataframe with a single method with public preview methods, `register_spark_dataframe()` and `register_pandas_dataframe()`. These register methods are experimental preview features, and may change at any time.

These methods upload data to your underlying storage, and as a result incur storage costs.

Register datasets

To complete the creation process, register your datasets with a workspace. Use the `register()` method to register datasets with your workspace in order to share them with others and reuse them across experiments in your workspace:

Create datasets using Azure Resource Manager

There are a number of templates at https://github.com/Azure/azure-quickstart-templates/tree/master/101-machine-learning-dataset-create-* that can be used to create datasets.

For information on using these templates, see [Use an Azure Resource Manager template to create a workspace for Azure Machine Learning](#).

Create datasets with Azure Open Datasets

[Azure Open Datasets](#) are curated public datasets that you can use to add scenario-specific features to machine learning solutions for more accurate models. Datasets include public-domain data for weather, census, holidays, public safety, and location that help you train machine learning models and enrich predictive solutions. Open Datasets are in the cloud on Microsoft Azure and are included in both the SDK and the studio.

Learn how to create [Azure Machine Learning Datasets from Azure Open Datasets](#).

Train with datasets

Use your datasets in your machine learning experiments for training ML models. [Learn more about how to train with datasets](#)

Version datasets

You can register a new dataset under the same name by creating a new version. A dataset version is a way to bookmark the state of your data so that you can apply a specific version of the dataset for experimentation or future reproduction. Learn more about [dataset versions](#).

```
# create a TabularDataset from Titanic training data
web_paths = ['https://dprepdata.blob.core.windows.net/demo/Titanic.csv',
             'https://dprepdata.blob.core.windows.net/demo/Titanic2.csv']
titanic_ds = Dataset.Tabular.from_delimited_files(path=web_paths)

# create a new version of titanic_ds
titanic_ds = titanic_ds.register(workspace = workspace,
                                  name = 'titanic_ds',
                                  description = 'new titanic training data',
                                  create_new_version = True)
```

Next steps

- Learn [how to train with datasets](#).
- Use automated machine learning to [train with TabularDatasets](#).
- For more dataset training examples, see the [sample notebooks](#).

Connect to data with the Azure Machine Learning studio

12/23/2020 • 8 minutes to read • [Edit Online](#)

In this article, learn how to access your data with the [Azure Machine Learning studio](#). Connect to your data in storage services on Azure with [Azure Machine Learning datastores](#), and then package that data for tasks in your ML workflows with [Azure Machine Learning datasets](#).

The following table defines and summarizes the benefits of datastores and datasets.

OBJECT	DESCRIPTION	BENEFITS
Datastores	Securely connect to your storage service on Azure, by storing your connection information, like your subscription ID and token authorization in your Key Vault associated with the workspace	<p>Because your information is securely stored, you</p> <p>Don't put authentication credentials or original data sources at risk.</p> <ul style="list-style-type: none">• No longer need to hard code them in your scripts.
Datasets	<p>By creating a dataset, you create a reference to the data source location, along with a copy of its metadata. With datasets you can,</p> <ul style="list-style-type: none">• Access data during model training.• Share data and collaborate with other users.• Leverage open-source libraries, like pandas, for data exploration.	<p>Because datasets are lazily evaluated, and the data remains in its existing location, you</p> <ul style="list-style-type: none">• Keep a single copy of data in your storage.• Incur no extra storage cost• Don't risk unintentionally changing your original data sources.• Improve ML workflow performance speeds.

To understand where datastores and datasets fit in Azure Machine Learning's overall data access workflow, see the [Securely access data](#) article.

For a code first experience, see the following articles to use the [Azure Machine Learning Python SDK](#) to:

- [Connect to Azure storage services with datastores](#).
- [Create Azure Machine Learning datasets](#).

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](#).
- Access to [Azure Machine Learning studio](#).
- An Azure Machine Learning workspace. [Create an Azure Machine Learning workspace](#).
 - When you create a workspace, an Azure blob container and an Azure file share are automatically registered as datastores to the workspace. They're named `workspaceblobstore` and `workspacefilestore`, respectively. If blob storage is sufficient for your needs, the `workspaceblobstore` is set as the default datastore, and already configured for use. Otherwise, you need a storage account on Azure with a [supported storage type](#).

Create datastores

You can create datastores from [these Azure storage solutions](#). For unsupported storage solutions, and to save data egress cost during ML experiments, you must [move your data](#) to a supported Azure storage solution. [Learn more about datastores](#).

Create a new datastore in a few steps with the Azure Machine Learning studio.

IMPORTANT

If your data storage account is in a virtual network, additional configuration steps are required to ensure the studio has access to your data. See [Network isolation & privacy](#) to ensure the appropriate configuration steps are applied.

1. Sign in to [Azure Machine Learning studio](#).
2. Select **Datastores** on the left pane under **Manage**.
3. Select **+ New datastore**.
4. Complete the form to create and register a new datastore. The form intelligently updates itself based on your selections for Azure storage type and authentication type. See the [storage access and permissions section](#) to understand where to find the authentication credentials you need to populate this form.

The following example demonstrates what the form looks like when you create an **Azure blob datastore**:

New datastore

Datastore name *

Datastore type *



Account selection method

 From Azure subscription Enter manually

Subscription ID *



Storage account *



Blob container *



Authentication type



Account key *



Create datasets

After you create a datastore, create a dataset to interact with your data. Datasets package your data into a lazily evaluated consumable object for machine learning tasks, like training. [Learn more about datasets](#).

There are two types of datasets, FileDataset and TabularDataset. [FileDatasets](#) create references to single or multiple files or public URLs. Whereas, [TabularDatasets](#) represent your data in a tabular format.

The following steps and animation show how to create a dataset in [Azure Machine Learning studio](#).

NOTE

Datasets created through Azure Machine Learning studio are automatically registered to the workspace.

The screenshot shows the Microsoft Azure Machine Learning studio interface. The left sidebar is titled 'Assets' and includes sections for 'Author' (Notebooks, Automated ML (preview), Designer (preview)), 'Assets' (Datasets, Experiments, Pipelines, Models, Endpoints), and 'Manage' (Compute, Datastores, Data Labeling). The main area is titled 'Welcome to the studio!' and features four cards: 'Create new' (Notebooks, Automated ML (preview), Designer (preview)), 'My recent resources' (Compute, Datastore, Dataset), and a table for 'Compute' resources.

Name	Type	Provisioning state	Created on
aml-test-compute	Machine Learning com...	Succeeded (0 nodes)	Sep 2, 2020 1:27 PM

To create a dataset in the studio:

1. Sign in to the [Azure Machine Learning studio](#).
2. Select **Datasets** in the **Assets** section of the left pane.
3. Select **Create Dataset** to choose the source of your dataset. This source can be local files, a datastore, public URLs, or [Azure Open Datasets](#).
4. Select **Tabular** or **File** for Dataset type.
5. Select **Next** to open the **Datastore and file selection** form. On this form you select where to keep your dataset after creation, as well as select what data files to use for your dataset.
 - a. Enable skip validation if your data is in a virtual network. Learn more about [virtual network isolation and privacy](#).
 - b. For Tabular datasets, you can specify a 'timeseries' trait to enable time related operations on your dataset. Learn how to [add the timeseries trait to your dataset](#).
6. Select **Next** to populate the **Settings and preview** and **Schema** forms; they are intelligently populated based on file type and you can further configure your dataset prior to creation on these forms.
7. Select **Next** to review the **Confirm details** form. Check your selections and create an optional data profile for your dataset. Learn more about [data profiling](#).
8. Select **Create** to complete your dataset creation.

Data profile and preview

After you create your dataset, verify you can view the profile and preview in the studio with the following steps.

1. Sign in to the [Azure Machine Learning studio](#)
2. Select **Datasets** in the **Assets** section of the left pane.
3. Select the name of the dataset you want to view.
4. Select the **Explore** tab.
5. Select the **Preview** or **Profile** tab.

You can get a vast variety of summary statistics across your data set to verify whether your data set is ML-ready. For non-numeric columns, they include only basic statistics like min, max, and error count. For numeric columns, you can also review their statistical moments and estimated quantiles.

Specifically, Azure Machine Learning dataset's data profile includes:

NOTE

Blank entries appear for features with irrelevant types.

STATISTIC	DESCRIPTION
Feature	Name of the column that is being summarized.
Profile	In-line visualization based on the type inferred. For example, strings, booleans, and dates will have value counts, while decimals (numerics) have approximated histograms. This allows you to gain a quick understanding of the distribution of the data.
Type distribution	In-line value count of types within a column. Nulls are their own type, so this visualization is useful for detecting odd or missing values.
Type	Inferred type of the column. Possible values include: strings, booleans, dates, and decimals.
Min	Minimum value of the column. Blank entries appear for features whose type does not have an inherent ordering (like, booleans).
Max	Maximum value of the column.

STATISTIC	DESCRIPTION
Count	Total number of missing and non-missing entries in the column.
Not missing count	Number of entries in the column that are not missing. Empty strings and errors are treated as values, so they will not contribute to the "not missing count."
Quantiles	Approximated values at each quantile to provide a sense of the distribution of the data.
Mean	Arithmetic mean or average of the column.
Standard deviation	Measure of the amount of dispersion or variation of this column's data.
Variance	Measure of how far spread out this column's data is from its average value.
Skewness	Measure of how different this column's data is from a normal distribution.
Kurtosis	Measure of how heavily tailed this column's data is compared to a normal distribution.

Storage access and permissions

To ensure you securely connect to your Azure storage service, Azure Machine Learning requires that you have permission to access the corresponding data storage. This access depends on the authentication credentials used to register the datastore.

Virtual network

If your data storage account is in a **virtual network**, additional configuration steps are required to ensure Azure Machine Learning has access to your data. See [Network isolation & privacy](#) to ensure the appropriate configuration steps are applied when you create and register your datastore.

Access validation

As part of the initial datastore creation and registration process, Azure Machine Learning automatically validates that the underlying storage service exists and the user provided principal (username, service principal, or SAS token) has access to the specified storage.

After datastore creation, this validation is only performed for methods that require access to the underlying storage container, **not** each time datastore objects are retrieved. For example, validation happens if you want to download files from your datastore; but if you just want to change your default datastore, then validation does not happen.

To authenticate your access to the underlying storage service, you can provide either your account key, shared access signatures (SAS) tokens, or service principal according to the datastore type you want to create. The [storage type matrix](#) lists the supported authentication types that correspond to each datastore type.

You can find account key, SAS token, and service principal information on your [Azure portal](#).

- If you plan to use an account key or SAS token for authentication, select **Storage Accounts** on the left pane, and choose the storage account that you want to register.

- The [Overview](#) page provides information such as the account name, container, and file share name.
 1. For account keys, go to [Access keys](#) on the [Settings](#) pane.
 2. For SAS tokens, go to [Shared access signatures](#) on the [Settings](#) pane.
- If you plan to use a [service principal](#) for authentication, go to your [App registrations](#) and select which app you want to use.
 - Its corresponding [Overview](#) page will contain required information like tenant ID and client ID.

IMPORTANT

- If you need to change your access keys for an Azure Storage account (account key or SAS token), be sure to sync the new credentials with your workspace and the datastores connected to it. Learn how to [sync your updated credentials](#).
- If you unregister and re-register a datastore with the same name, and it fails, the Azure Key Vault for your workspace may not have soft-delete enabled. By default, soft-delete is enabled for the key vault instance created by your workspace, but it may not be enabled if you used an existing key vault or have a workspace created prior to October 2020. For information on how to enable soft-delete, see [Turn on Soft Delete for an existing key vault](#)."

Permissions

For Azure blob container and Azure Data Lake Gen 2 storage, make sure your authentication credentials have **Storage Blob Data Reader** access. Learn more about [Storage Blob Data Reader](#). An account SAS token defaults to no permissions.

- For data **read access**, your authentication credentials must have a minimum of list and read permissions for containers and objects.
- For data **write access**, write and add permissions also are required.

Train with datasets

Use your datasets in your machine learning experiments for training ML models. [Learn more about how to train with datasets](#)

Next steps

- [A step-by-step example of training with TabularDatasets and automated machine learning](#).
- [Train a model](#).
- For more dataset training examples, see the [sample notebooks](#).

Train with datasets in Azure Machine Learning

12/23/2020 • 8 minutes to read • [Edit Online](#)

In this article, you learn how to work with [Azure Machine Learning datasets](#) in your training experiments. You can use datasets in your local or remote compute target without worrying about connection strings or data paths.

Azure Machine Learning datasets provide a seamless integration with Azure Machine Learning training functionality like [ScriptRunConfig](#), [HyperDrive](#) and [Azure Machine Learning pipelines](#).

Prerequisites

To create and train with datasets, you need:

- 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](#).
- The [Azure Machine Learning SDK for Python installed](#) (>= 1.13.0), which includes the `azureml-datasets` package.

NOTE

Some Dataset classes have dependencies on the `azureml-dataprep` package. For Linux users, these classes are supported only on the following distributions: Red Hat Enterprise Linux, Ubuntu, Fedora, and CentOS.

Use datasets directly in training scripts

If you have structured data not yet registered as a dataset, create a `TabularDataset` and use it directly in your training script for your local or remote experiment.

In this example, you create an unregistered `TabularDataset` and specify it as a script argument in the `ScriptRunConfig` object for training. If you want to reuse this `TabularDataset` with other experiments in your workspace, see [how to register datasets to your workspace](#).

Create a `TabularDataset`

The following code creates an unregistered `TabularDataset` from a web url.

```
from azureml.core.dataset import Dataset  
  
web_path ='https://dprepdata.blob.core.windows.net/demo/Titanic.csv'  
titanic_ds = Dataset.Tabular.from_delimited_files(path=web_path)
```

`TabularDataset` objects provide the ability to load the data in your `TabularDataset` into a `pandas` or `Spark DataFrame` so that you can work with familiar data preparation and training libraries without having to leave your notebook.

Access dataset in training script

The following code configures a script argument `--input-data` that you will specify when you configure your training run (see next section). When the tabular dataset is passed in as the argument value, Azure ML will resolve that to ID of the dataset, which you can then use to access the dataset in your training script (without

having to hardcode the name or ID of the dataset in your script). It then uses the `to_pandas_dataframe()` method to load that dataset into a pandas dataframe for further data exploration and preparation prior to training.

NOTE

If your original data source contains NaN, empty strings or blank values, when you use `to_pandas_dataframe()`, then those values are replaced as a *Null* value.

If you need to load the prepared data into a new dataset from an in-memory pandas dataframe, write the data to a local file, like a parquet, and create a new dataset from that file. You can also create datasets from local files or paths in datastores. Learn more about [how to create datasets](#).

```
%%writefile $script_folder/train_titanic.py

import argparse
from azureml.core import Dataset, Run

parser = argparse.ArgumentParser()
parser.add_argument("--input-data", type=str)
args = parser.parse_args()

run = Run.get_context()
ws = run.experiment.workspace

# get the input dataset by ID
dataset = Dataset.get_by_id(ws, id=args.input_data)

# load the TabularDataset to pandas DataFrame
df = dataset.to_pandas_dataframe()
```

Configure the training run

A `ScriptRunConfig` object is used to configure and submit the training run.

This code creates a `ScriptRunConfig` object, `src`, that specifies

- A script directory for your scripts. All the files in this directory are uploaded into the cluster nodes for execution.
- The training script, *train_titanic.py*.
- The input dataset for training, `titanic_ds`, as a script argument. Azure ML will resolve this to corresponding ID of the dataset when it is passed to your script.
- The compute target for the run.
- The environment for the run.

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory=script_folder,
                      script='train_titanic.py',
                      # pass dataset as an input with friendly name 'titanic'
                      arguments=['--input-data', titanic_ds.as_named_input('titanic')],
                      compute_target=compute_target,
                      environment=myenv)

# Submit the run configuration for your training run
run = experiment.submit(src)
run.wait_for_completion(show_output=True)
```

Mount files to remote compute targets

If you have unstructured data, create a [FileDataset](#) and either mount or download your data files to make them available to your remote compute target for training. Learn about when to use [mount vs. download](#) for your remote training experiments.

The following example creates a FileDataset and mounts the dataset to the compute target by passing it as an argument to the training script.

NOTE

If you are using a custom Docker base image, you will need to install fuse via `apt-get install -y fuse` as a dependency for dataset mount to work. Learn how to [build a custom build image](#).

Create a FileDataset

The following example creates an unregistered FileDataset from web urls. Learn more about [how to create datasets](#) from other sources.

```
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'
]
mnist_ds = Dataset.File.from_files(path = web_paths)
```

Configure the training run

We recommend passing the dataset as an argument when mounting via the `arguments` parameter of the `ScriptRunConfig` constructor. By doing so, you will get the data path (mounting point) in your training script via arguments. This way, you will be able use the same training script for local debugging and remote training on any cloud platform.

The following example creates a ScriptRunConfig that passes in the FileDataset via `arguments`. After you submit the run, data files referred by the dataset `mnist_ds` will be mounted to the compute target.

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory=script_folder,
                      script='train_mnist.py',
                      # the dataset will be mounted on the remote compute and the mounted path passed as an
                      # argument to the script
                      arguments=['--data-folder', mnist_ds.as_mount(), '--regularization', 0.5],
                      compute_target=compute_target,
                      environment=myenv)

# Submit the run configuration for your training run
run = experiment.submit(src)
run.wait_for_completion(show_output=True)
```

Retrieve the data in your training script

The following code shows how to retrieve the data in your script.

```

%%writefile $script_folder/train_mnist.py

import argparse
import os
import numpy as np
import glob

from utils import load_data

# retrieve the 2 arguments configured through `arguments` in the ScriptRunConfig
parser = argparse.ArgumentParser()
parser.add_argument('--data-folder', type=str, dest='data_folder', help='data folder mounting point')
parser.add_argument('--regularization', type=float, dest='reg', default=0.01, help='regularization rate')
args = parser.parse_args()

data_folder = args.data_folder
print('Data folder:', data_folder)

# get the file paths on the compute
X_train_path = glob.glob(os.path.join(data_folder, '**/train-images-idx3-ubyte.gz'), recursive=True)[0]
X_test_path = glob.glob(os.path.join(data_folder, '**/t10k-images-idx3-ubyte.gz'), recursive=True)[0]
y_train_path = glob.glob(os.path.join(data_folder, '**/train-labels-idx1-ubyte.gz'), recursive=True)[0]
y_test = glob.glob(os.path.join(data_folder, '**/t10k-labels-idx1-ubyte.gz'), recursive=True)[0]

# load train and test set into numpy arrays
X_train = load_data(X_train_path, False) / 255.0
X_test = load_data(X_test_path, False) / 255.0
y_train = load_data(y_train_path, True).reshape(-1)
y_test = load_data(y_test, True).reshape(-1)

```

Mount vs download

Mounting or downloading files of any format are supported for datasets created from Azure Blob storage, Azure Files, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure SQL Database, and Azure Database for PostgreSQL.

When you **mount** a dataset, you attach the files referenced by the dataset to a directory (mount point) and make it available on the compute target. Mounting is supported for Linux-based computes, including Azure Machine Learning Compute, virtual machines, and HDInsight.

When you **download** a dataset, all the files referenced by the dataset will be downloaded to the compute target. Downloading is supported for all compute types.

If your script processes all files referenced by the dataset, and your compute disk can fit your full dataset, downloading is recommended to avoid the overhead of streaming data from storage services. If your data size exceeds the compute disk size, downloading is not possible. For this scenario, we recommend mounting since only the data files used by your script are loaded at the time of processing.

The following code mounts `dataset` to the temp directory at `mounted_path`

```

import tempfile
mounted_path = tempfile.mkdtemp()

# mount dataset onto the mounted_path of a Linux-based compute
mount_context = dataset.mount(mounted_path)

mount_context.start()

import os
print(os.listdir(mounted_path))
print (mounted_path)

```

Directly access datasets in your script

Registered datasets are accessible both locally and remotely on compute clusters like the Azure Machine Learning compute. To access your registered dataset across experiments, use the following code to access your workspace and registered dataset by name. By default, the `get_by_name()` method on the `Dataset` class returns the latest version of the dataset that's registered with the workspace.

```
%%writefile $script_folder/train.py

from azureml.core import Dataset, Run

run = Run.get_context()
workspace = run.experiment.workspace

dataset_name = 'titanic_ds'

# Get a dataset by name
titanic_ds = Dataset.get_by_name(workspace=workspace, name=dataset_name)

# Load a TabularDataset into pandas DataFrame
df = titanic_ds.to_pandas_dataframe()
```

Accessing source code during training

Azure Blob storage has higher throughput speeds than an Azure file share and will scale to large numbers of jobs started in parallel. For this reason, we recommend configuring your runs to use Blob storage for transferring source code files.

The following code example specifies in the run configuration which blob datastore to use for source code transfers.

```
# workspaceblobstore is the default blob storage
src.run_config.source_directory_data_store = "workspaceblobstore"
```

Notebook examples

- The [dataset notebooks](#) demonstrate and expand upon concepts in this article.
- See how to [parametrize datasets in your ML pipelines](#).

Troubleshooting

- **Dataset initialization failed: Waiting for mount point to be ready has timed out:**
 - If you don't have any outbound [network security group](#) rules and are using `azureml-sdk>=1.12.0`, update `azureml-dataset-runtime` and its dependencies to be the latest for the specific minor version, or if you are using it in a run, recreate your environment so it can have the latest patch with the fix.
 - If you are using `azureml-sdk<1.12.0`, upgrade to the latest version.
 - If you have outbound NSG rules, make sure there is an outbound rule that allows all traffic for the service tag `AzureResourceMonitor`.

Overloaded AzureFile storage

If you receive an error

`Unable to upload project files to working directory in AzureFile because the storage is overloaded`, apply following workarounds.

If you are using file share for other workloads, such as data transfer, the recommendation is to use blobs so that

file share is free to be used for submitting runs. You may also split the workload between two different workspaces.

Passing data as input

- **TypeError: FileNotFoundError: No such file or directory:** This error occurs if the file path you provide isn't where the file is located. You need to make sure the way you refer to the file is consistent with where you mounted your dataset on your compute target. To ensure a deterministic state, we recommend using the abstract path when mounting a dataset to a compute target. For example, in the following code we mount the dataset under the root of the filesystem of the compute target, `/tmp`.

```
# Note the leading / in '/tmp/dataset'  
script_params = {  
    '--data-folder': dset.as_named_input('dogscats_train').as_mount('/tmp/dataset'),  
}
```

If you don't include the leading forward slash, `'/'`, you'll need to prefix the working directory e.g. `/mnt/batch/.../tmp/dataset` on the compute target to indicate where you want the dataset to be mounted.

Next steps

- [Auto train machine learning models](#) with TabularDatasets.
- [Train image classification models](#) with FileDatasets.
- [Train with datasets using pipelines](#).

Detect data drift (preview) on datasets

12/23/2020 • 14 minutes to read • [Edit Online](#)

IMPORTANT

Data drift detection for datasets 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 monitor data drift and set alerts when drift is high.

With Azure Machine Learning dataset monitors (preview), you can:

- **Analyze drift in your data** to understand how it changes over time.
- **Monitor model data** for differences between training and serving datasets. Start by [collecting model data from deployed models](#).
- **Monitor new data** for differences between any baseline and target dataset.
- **Profile features in data** to track how statistical properties change over time.
- **Set up alerts on data drift** for early warnings to potential issues.

An [Azure Machine learning dataset](#) is used to create the monitor. The dataset must include a timestamp column.

You can view data drift metrics with the Python SDK or in Azure Machine Learning studio. Other metrics and insights are available through the [Azure Application Insights](#) resource associated with the Azure Machine Learning workspace.

Prerequisites

To create and work with dataset monitors, you need:

- 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](#).
- The [Azure Machine Learning SDK for Python installed](#), which includes the azureml-datasets package.
- Structured (tabular) data with a timestamp specified in the file path, file name, or column in the data.

What is data drift?

Data drift is one of the top reasons model accuracy degrades over time. For machine learning models, data drift is the change in model input data that leads to model performance degradation. Monitoring data drift helps detect these model performance issues.

Causes of data drift include:

- Upstream process changes, such as a sensor being replaced that changes the units of measurement from inches to centimeters.
- Data quality issues, such as a broken sensor always reading 0.
- Natural drift in the data, such as mean temperature changing with the seasons.
- Change in relation between features, or covariate shift.

Azure Machine Learning simplifies drift detection by computing a single metric abstracting the complexity of

datasets being compared. These datasets may have hundreds of features and tens of thousands of rows. Once drift is detected, you drill down into which features are causing the drift. You then inspect feature level metrics to debug and isolate the root cause for the drift.

This top down approach makes it easy to monitor data instead of traditional rules-based techniques. Rules-based techniques such as allowed data range or allowed unique values can be time consuming and error prone.

In Azure Machine Learning, you use dataset monitors to detect and alert for data drift.

Dataset monitors

With a dataset monitor you can:

- Detect and alert to data drift on new data in a dataset.
- Analyze historical data for drift.
- Profile new data over time.

The data drift algorithm provides an overall measure of change in data and indication of which features are responsible for further investigation. Dataset monitors produce a number of other metrics by profiling new data in the `timeseries` dataset.

Custom alerting can be set up on all metrics generated by the monitor through [Azure Application Insights](#). Dataset monitors can be used to quickly catch data issues and reduce the time to debug the issue by identifying likely causes.

Conceptually, there are three primary scenarios for setting up dataset monitors in Azure Machine Learning.

SCENARIO	DESCRIPTION
Monitor a model's serving data for drift from the training data	Results from this scenario can be interpreted as monitoring a proxy for the model's accuracy, since model accuracy degrades when the serving data drifts from the training data.
Monitor a time series dataset for drift from a previous time period.	This scenario is more general, and can be used to monitor datasets involved upstream or downstream of model building. The target dataset must have a timestamp column. The baseline dataset can be any tabular dataset that has features in common with the target dataset.
Perform analysis on past data.	This scenario can be used to understand historical data and inform decisions in settings for dataset monitors.

Dataset monitors depend on the following Azure services.

AZURE SERVICE	DESCRIPTION
<i>Dataset</i>	Drift uses Machine Learning datasets to retrieve training data and compare data for model training. Generating profile of data is used to generate some of the reported metrics such as min, max, distinct values, distinct values count.
<i>Azureml pipeline and compute</i>	The drift calculation job is hosted in azureml pipeline. The job is triggered on demand or by schedule to run on a compute configured at drift monitor creation time.
<i>Application insights</i>	Drift emits metrics to Application Insights belonging to the machine learning workspace.

AZURE SERVICE	DESCRIPTION
<i>Azure blob storage</i>	Drift emits metrics in json format to Azure blob storage.

How dataset monitors data

Use Machine Learning datasets to monitor for data drift. Specify a baseline dataset - usually the training dataset for a model. A target dataset - usually model input data - is compared over time to your baseline dataset. This comparison means that your target dataset must have a timestamp column specified.

Create target dataset

The target dataset needs the `timeseries` trait set on it by specifying the timestamp column either from a column in the data or a virtual column derived from the path pattern of the files. Create the dataset with a timestamp through the [Python SDK](#) or [Azure Machine Learning studio](#). A column representing a "timestamp" must be specified to add `timeseries` trait to the dataset. If your data is partitioned into folder structure with time info, such as '{yyyy/MM/dd}', create a virtual column through the path pattern setting and set it as the "partition timestamp" to improve the importance of time series functionality.

Python SDK

The `Dataset` class `with_timestamp_columns()` method defines the time stamp column for the dataset.

```
from azureml.core import Workspace, Dataset, Datastore

# get workspace object
ws = Workspace.from_config()

# get datastore object
dstore = Datastore.get(ws, 'your datastore name')

# specify datastore paths
dstore_paths = [(dstore, 'weather/*/*/*/*/data.parquet')]

# specify partition format
partition_format = 'weather/{state}/{date:yyyy/MM/dd}/data.parquet'

# create the Tabular dataset with 'state' and 'date' as virtual columns
dset = Dataset.Tabular.from_parquet_files(path=dstore_paths, partition_format=partition_format)

# assign the timestamp attribute to a real or virtual column in the dataset
dset = dset.with_timestamp_columns('date')

# register the dataset as the target dataset
dset = dset.register(ws, 'target')
```

For a full example of using the `timeseries` trait of datasets, see the [example notebook](#) or the [datasets SDK documentation](#).

Azure Machine Learning studio

If you create your dataset using Azure Machine Learning studio, ensure the path to your data contains timestamp information, include all subfolders with data, and set the partition format.

In the following example, all data under the subfolder *Noaa/sdFlorida/2019* is taken, and the partition format specifies the timestamp's year, month, and day.

Create dataset from datastore

- Basic info
- Settings and preview
- Schema
- Confirm details

Basic info

Name *	Dataset name	Dataset version
Dataset type *	Tabular	
Datastore *	workspaceblobstore (AzureBlob)	Refresh
Path *	<input type="button" value="Browse"/> <input type="text" value="NoaaIsdFlorida/2019/**"/> <small>To include files in subfolders, append '/**' after the folder name like so: '[Folder]/**'.</small>	
Description	<input type="text" value="Dataset description"/>	
▼ Advanced settings		
Partition format <input type="text" value="/{timestampyyy/MM/dd}/data.parquet"/>		
<input type="checkbox"/> Ignore unmatched file paths		

[Back](#) [Next](#) [Cancel](#)

In the **Schema** settings, specify the timestamp column from a virtual or real column in the specified dataset:

Create dataset from local files

- Basic info
- Datastore and file selection
- Settings and preview
- Schema
- Confirm details

Schema

Include	Column name	Properties	Type
<input checked="" type="checkbox"/>	datetime	Timestamp	Date
<input checked="" type="checkbox"/>	latitude	Not applicable to selected type	Decimal
<input checked="" type="checkbox"/>	longitude	Not applicable to selected type	Decimal
<input checked="" type="checkbox"/>	elevation	Not applicable to selected type	Integer
<input checked="" type="checkbox"/>	windAngle	Not applicable to selected type	Integer
<input checked="" type="checkbox"/>	windSpeed	Not applicable to selected type	Decimal
<input checked="" type="checkbox"/>	temperature	Not applicable to selected type	Decimal
<input checked="" type="checkbox"/>	seaLvlPressure	Not applicable to selected type	Decimal

[Back](#) [Next](#) [Cancel](#)

If your data is partitioned by date, as is the case here, you can also specify the `partition_timestamp`. This allows more efficient processing of dates.

The screenshot shows the 'Schema' step of the 'Create dataset from local files' wizard. On the left, a vertical navigation bar lists steps: 'Basic info' (checked), 'Datastore and file selection' (checked), 'Settings and preview' (checked), 'Schema' (checked, highlighted in blue), and 'Confirm details' (unchecked). The main area is titled 'Schema' and contains a table with columns: 'Include' (checkboxes), 'Column name', 'Properties' (dropdowns), and 'Type' (dropdowns). The table rows are: stationName (String), countryOrRegion (String), p_k (String), year (Integer), day (Integer), version (Integer), __index_level_0__ (Integer), and partitionDate (Date, with a dropdown menu showing 'Partition timestamp'). At the bottom are 'Back', 'Next' (highlighted in blue), and 'Cancel' buttons.

Create dataset monitors

Create dataset monitors to detect and alert to data drift on a new dataset. Use either the [Python SDK](#) or [Azure Machine Learning studio](#).

Python SDK

See the [Python SDK reference documentation on data drift](#) for full details.

The following example shows how to create a dataset monitor using the Python SDK

```

from azureml.core import Workspace, Dataset
from azureml.datadrift import DataDriftDetector
from datetime import datetime

# get the workspace object
ws = Workspace.from_config()

# get the target dataset
dset = Dataset.get_by_name(ws, 'target')

# set the baseline dataset
baseline = target.time_before(datetime(2019, 2, 1))

# set up feature list
features = ['latitude', 'longitude', 'elevation', 'windAngle', 'windSpeed', 'temperature', 'snowDepth',
'stationName', 'countryOrRegion']

# set up data drift detector
monitor = DataDriftDetector.create_from_datasets(ws, 'drift-monitor', baseline, target,
compute_target='cpu-cluster',
frequency='Week',
feature_list=None,
drift_threshold=.6,
latency=24)

# get data drift detector by name
monitor = DataDriftDetector.get_by_name(ws, 'drift-monitor')

# update data drift detector
monitor = monitor.update(feature_list=features)

# run a backfill for January through May
backfill1 = monitor.backfill(datetime(2019, 1, 1), datetime(2019, 5, 1))

# run a backfill for May through today
backfill1 = monitor.backfill(datetime(2019, 5, 1), datetime.today())

# disable the pipeline schedule for the data drift detector
monitor = monitor.disable_schedule()

# enable the pipeline schedule for the data drift detector
monitor = monitor.enable_schedule()

```

For a full example of setting up a `timeseries` dataset and data drift detector, see our [example notebook](#).

Azure Machine Learning studio

1. Navigate to the [studio's homepage](#).

2. Select the **Datasets** tab on the left.

3. Select **Dataset monitors**.

Datasets				
Registered datasets	Dataset monitors			
+ Create monitor ⟳ Refresh		🔍 Search to filter items...		
Name	Baseline dataset	Target dataset	State	Created time ↓
weatherDriftBackfillWeekly2	weatherBaselineJan2019	weatherAll2019	Disabled	10/2/2019, 1:48:23 PM
tempDriftMonitor	weatherBaselineJan2019	weatherAll2019	Disabled	10/2/2019, 1:07:00 PM
weatherDriftBackfillWeekly	weatherBaselineJan2019	weatherAll2019	Disabled	10/2/2019, 12:27:36 PM
weatherDriftBackfill	weatherBaselineJan2019	weatherAll2019	Disabled	10/2/2019, 12:24:58 PM

◀ Prev
Next ▶

4. Click on the **+Create monitor** button and continue through the wizard by clicking **Next**.

The screenshot shows the 'Create new data drift monitor' wizard in the Microsoft Azure Machine Learning interface. The left sidebar has icons for various datasets and models. The main panel shows three steps: 'Select target dataset', 'Select baseline dataset', and 'Monitor settings'. The first step is currently selected. A tooltip on the right side of the screen provides a note about avoiding sensitive data in the target dataset. Below the steps is a dropdown menu labeled 'Target dataset *' with the placeholder 'Choose a target dataset'. At the bottom of the panel are 'Back', 'Next', and 'Cancel' buttons.

- **Select target dataset.** The target dataset is a tabular dataset with timestamp column specified which will be analyzed for data drift. The target dataset must have features in common with the baseline dataset, and should be a **timeseries** dataset, which new data is appended to. Historical data in the target dataset can be analyzed, or new data can be monitored.
- **Select baseline dataset.** Select the tabular dataset to be used as the baseline for comparison of the target dataset over time. The baseline dataset must have features in common with the target dataset. Select a time range to use a slice of the target dataset, or specify a separate dataset to use as the baseline.
- **Monitor settings.** These settings are for the scheduled dataset monitor pipeline, which will be created.

SETTING	DESCRIPTION	TIPS	MUTABLE
Name	Name of the dataset monitor.		No
Features	List of features that will be analyzed for data drift over time.	Set to a model's output feature(s) to measure concept drift. Don't include features that naturally drift over time (month, year, index, etc.). You can backfill and existing data drift monitor after adjusting the list of features.	Yes
Compute target	Azure Machine Learning compute target to run the dataset monitor jobs.		Yes

Setting	Description	Tips	Mutable
Enable	Enable or disable the schedule on the dataset monitor pipeline	Disable the schedule to analyze historical data with the backfill setting. It can be enabled after the dataset monitor is created.	Yes
Frequency	The frequency that will be used to schedule the pipeline job and analyze historical data if running a backfill. Options include daily, weekly, or monthly.	Each run compares data in the target dataset according to the frequency: <ul style="list-style-type: none">• Daily: Compare most recent complete day in target dataset with baseline• Weekly: Compare most recent complete week (Monday - Sunday) in target dataset with baseline• Monthly: Compare most recent complete month in target dataset with baseline	No
Latency	Time, in hours, it takes for data to arrive in the dataset. For instance, if it takes three days for data to arrive in the SQL DB the dataset encapsulates, set the latency to 72.	Cannot be changed after the dataset monitor is created	No
Email addresses	Email addresses for alerting based on breach of the data drift percentage threshold.	Emails are sent through Azure Monitor.	Yes
Threshold	Data drift percentage threshold for email alerting.	Further alerts and events can be set on many other metrics in the workspace's associated Application Insights resource.	Yes

After finishing the wizard, the resulting dataset monitor will appear in the list. Select it to go to that monitor's details page.

Understand data drift results

This section shows you the results of monitoring a dataset, found in the **Datasets / Dataset monitors** page in Azure studio. You can update the settings as well as analyze existing data for a specific time period on this page.

Start with the top-level insights into the magnitude of data drift and a highlight of features to be further investigated.

Summary

Summary of the drift metrics captured with the latest run on 2020-06-11

Drift magnitude ⓘ Top drifting features ⓘ**85%**

temperature	68%
countryOrRegion	13%
wban	9%

Threshold ⓘ**20%****Drift runs**

Last run	1 days 2 hrs ago ⓘ
Next run	5 days 22 hrs
Drift frequency	Week

Monitor timeseries

this chart shows the date range for which baseline, target and drift data is available.

Baseline dataset**target****Target dataset****baseline-201...****Drift Monitor****weather-monit...**

19-08-04 19-08-18 19-09-01 19-09-15 19-09-29

METRIC	DESCRIPTION
Data drift magnitude	A percentage of drift between the baseline and target dataset over time. Ranging from 0 to 100, 0 indicates identical datasets and 100 indicates the Azure Machine Learning data drift model can completely tell the two datasets apart. Noise in the precise percentage measured is expected due to machine learning techniques being used to generate this magnitude.
Top drifting features	Shows the features from the dataset that have drifted the most and are therefore contributing the most to the Drift Magnitude metric. Due to covariate shift, the underlying distribution of a feature does not necessarily need to change to have relatively high feature importance.
Threshold	Data Drift magnitude beyond the set threshold will trigger alerts. This can be configured in the monitor settings.

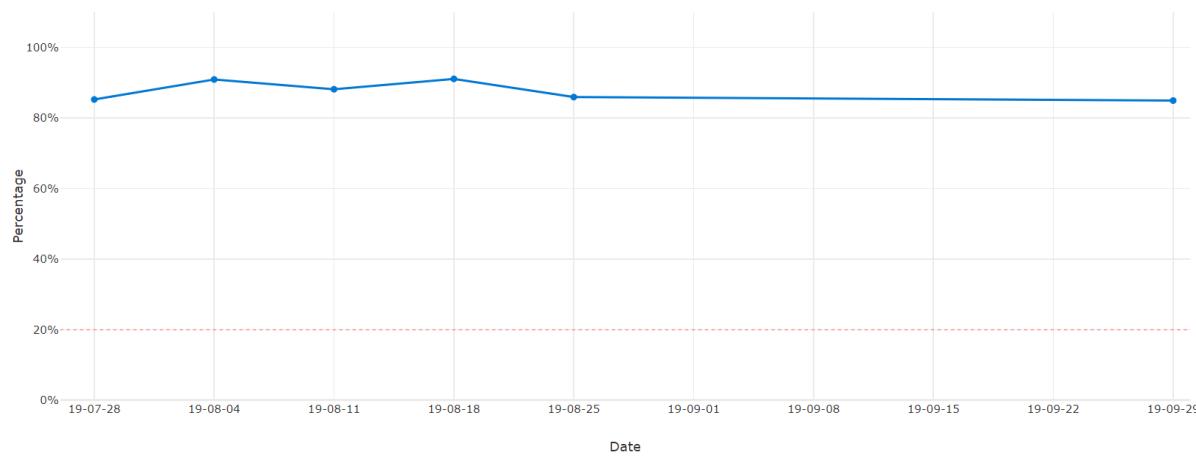
Drift magnitude trend

See how the dataset differs from the target dataset in the specified time period. The closer to 100%, the more the two datasets differ.

Start date: End date:

Drift magnitude trend

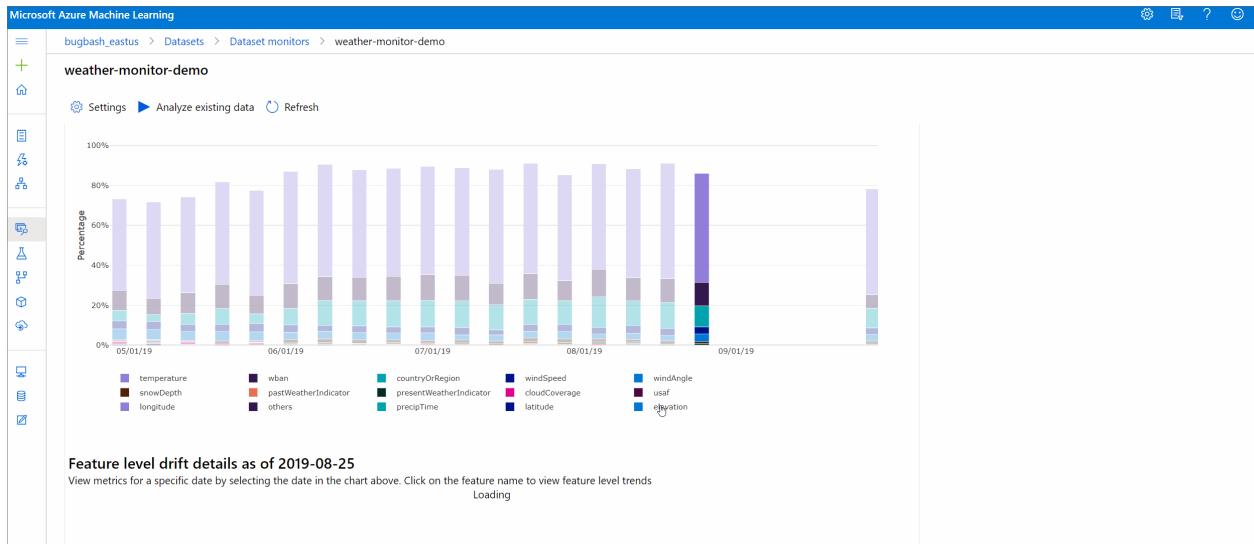
Drift magnitude metric measures the difference between target dataset for the time interval and baseline dataset. 0% implies that the target data is identical to the baseline data and 100% implies the target data is completely different from the baseline data.

**Drift magnitude by features**

This section contains feature-level insights into the change in the selected feature's distribution, as well as other statistics, over time.

The target dataset is also profiled over time. The statistical distance between the baseline distribution of each feature is compared with the target dataset's over time. Conceptually, this is similar to the data drift magnitude. However this statistical distance is for an individual feature rather than all features. Min, max, and mean are also available.

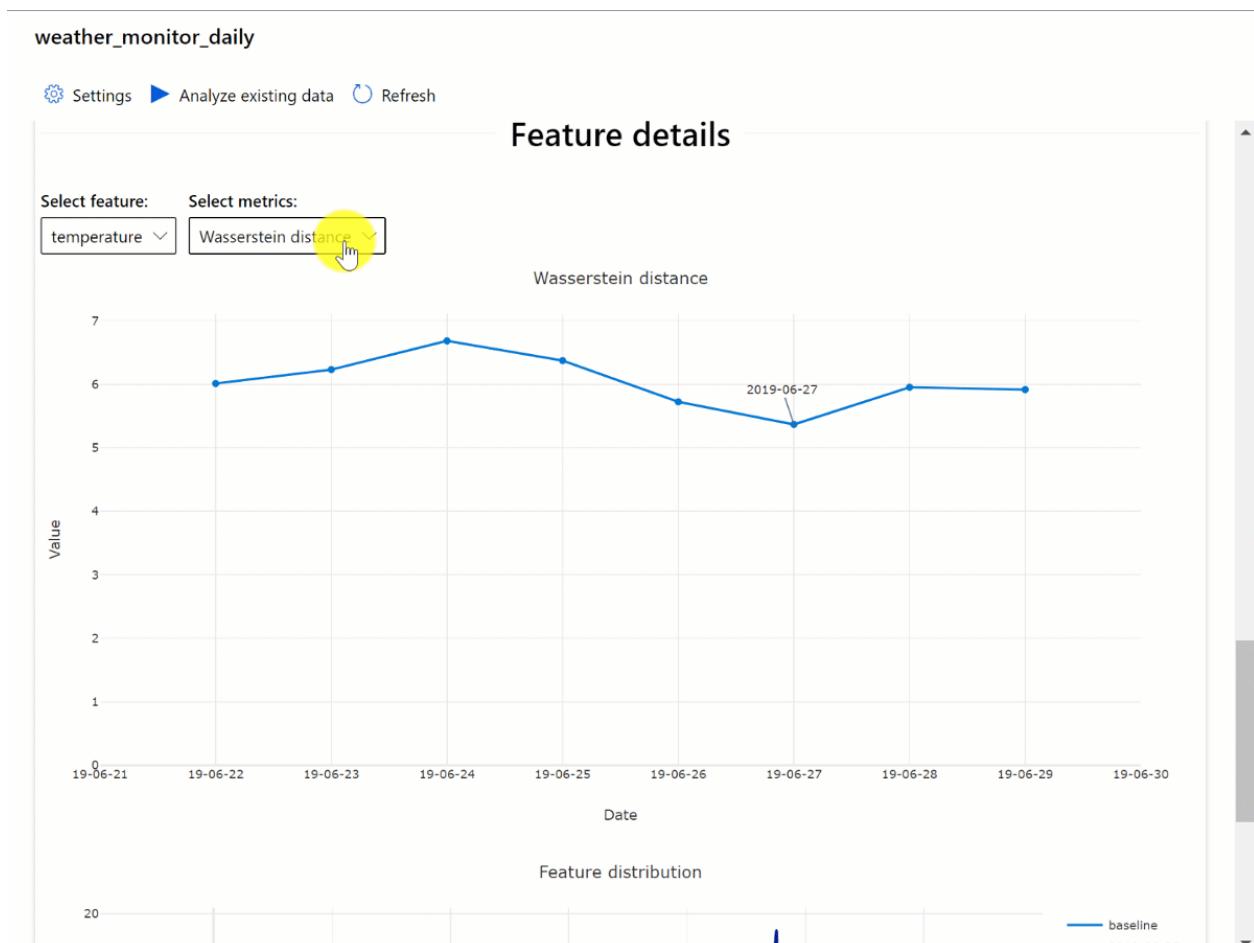
In the Azure Machine Learning studio, click on a bar in the graph to see the the feature level details for that date. By default, you see the baseline dataset's distribution and the most recent run's distribution of the same feature.



These metrics can also be retrieved in the Python SDK through the `get_metrics()` method on a `DataDriftDetector` object.

Feature details

Finally, scroll down to view details for each individual feature. Use the dropdowns above the chart to select the feature, and additionally select the metric you want to view.



Metrics in the chart depend on the type of feature.

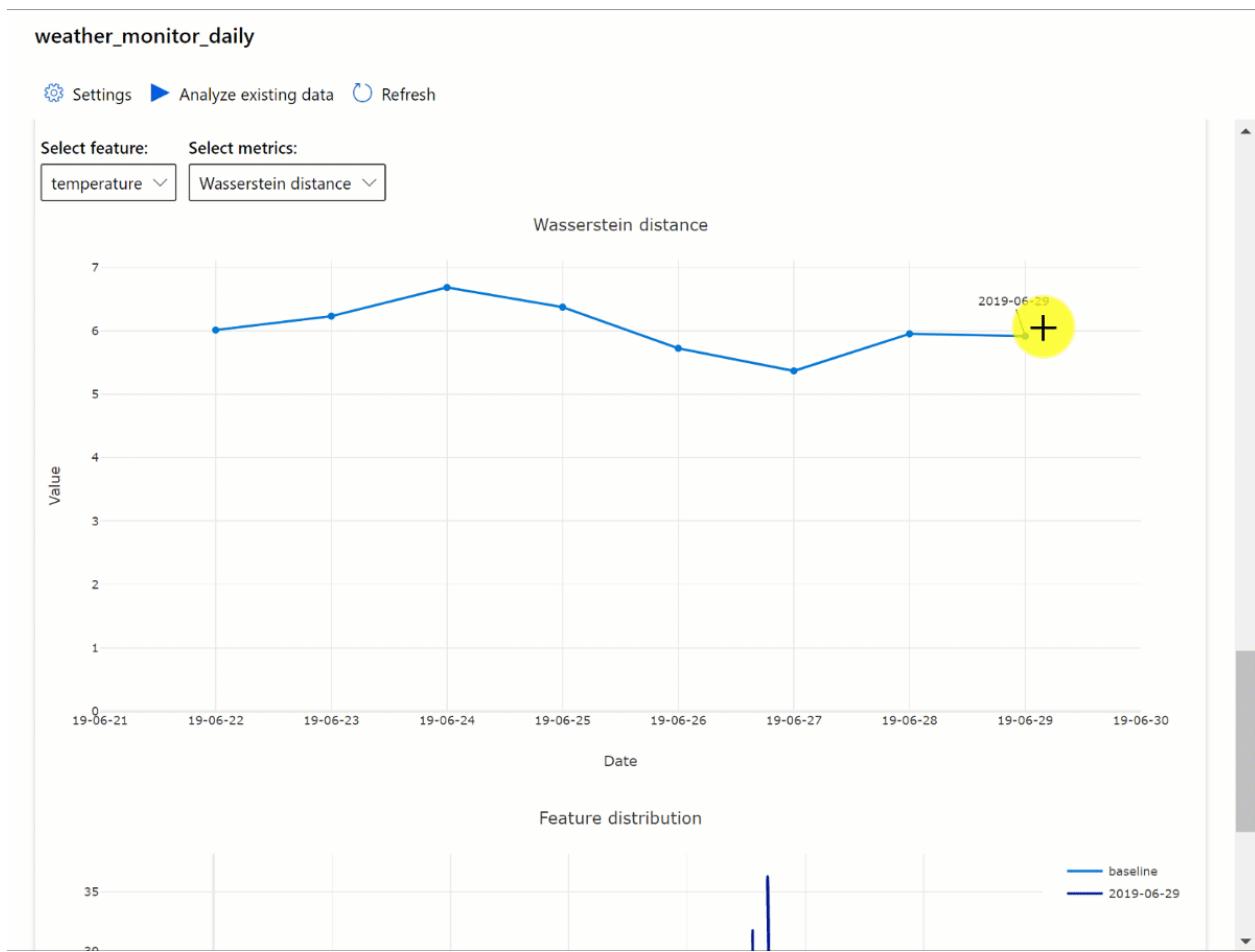
- Numeric features

METRIC	DESCRIPTION
Wasserstein distance	Minimum amount of work to transform baseline distribution into the target distribution.
Mean value	Average value of the feature.
Min value	Minimum value of the feature.
Max value	Maximum value of the feature.

- Categorical features

METRIC	DESCRIPTION
Euclidian distance	Computed for categorical columns. Euclidean distance is computed on two vectors, generated from empirical distribution of the same categorical column from two datasets. 0 indicates there is no difference in the empirical distributions. The more it deviates from 0, the more this column has drifted. Trends can be observed from a time series plot of this metric and can be helpful in uncovering a drifting feature.
Unique values	Number of unique values (cardinality) of the feature.

On this chart, select a single date to compare the feature distribution between the target and this date for the displayed feature. For numeric features, this shows two probability distributions. If the feature is numeric, a bar chart is shown.



Metrics, alerts, and events

Metrics can be queried in the [Azure Application Insights](#) resource associated with your machine learning workspace. You have access to all features of Application Insights including set up for custom alert rules and action groups to trigger an action such as, an Email/SMS/Push/Voice or Azure Function. Refer to the complete Application Insights documentation for details.

To get started, navigate to the [Azure portal](#) and select your workspace's **Overview** page. The associated Application Insights resource is on the far right:

Select Logs (Analytics) under Monitoring on the left pane:

The screenshot shows the Azure Application Insights portal interface. At the top, there's a header with the application name "ncusazureml" and "Application Insights". Below the header is a search bar labeled "Search (Ctrl+/)". The left sidebar contains several sections and their sub-items:

- Overview**
- Activity log**
- Access control (IAM)**
- Tags**
- Diagnose and solve problems**
- Investigate**
 - Application map**
 - Smart Detection**
 - Live Metrics Stream**
 - Search**
 - Availability**
 - Failures**
 - Performance**
 - Servers**
 - Browser**
 - Troubleshooting guides (pre...)**
- Monitoring**
 - Alerts**
 - Metrics**
 - Logs (Analytics)** (This item is highlighted with a red box.)
 - Workbooks**
- Usage**
 - Users**

The dataset monitor metrics are stored as `customMetrics`. You can write and run a query after setting up a dataset monitor to view them:

The screenshot shows the Azure Log Analytics workspace. In the top navigation bar, there are links for Help, Settings, Sample queries, Query explorer, Save, Copy, Export, New alert rule, and Pin to dashboard. Below the navigation, there's a search bar with 'New Query 1' and a '+' button. The schema dropdown is set to 'customMetrics'. The time range is 'Last 7 days'. The main area displays a table of results with columns: timestamp [UTC], name, value, valueCount, valueSum, valueMin, valueMax, customDimensions, and operation_Syn. The results show various metrics like energy_distance, datadrift_contribution, max, mean, min, and standard deviation. The table has 1,000 records and a display time of (UTC+00:00). The bottom of the table shows pagination with 'Page 1 of 20' and '50 items per page'.

After identifying metrics to set up alert rules, create a new alert rule:

The screenshot shows the 'Create rule' blade in the Azure portal. The 'RESOURCE' section shows 'ncusazureml' selected. The 'HIERARCHY' section shows the resource path 'Feedback-FeatureSynthesizer > copetersrg2'. The 'CONDITION' section has a placeholder 'Whenever the Custom log search is <logic undefined>'. The 'Monthly cost in USD (Estimated)' section shows a total of '\$ 1.50'. The 'ACTIONS' section has an 'Action group name' field and 'Select action group' and 'Create action group' buttons. A note says 'Azure Alerts are currently limited to either 2 metric, 1 log, or 1 activity log signal per alert rule. To alert on more signals, please create additional alert rules.' The 'Customize Actions' section includes checkboxes for 'Email subject' and 'Include custom Json payload for webhook'.

You can use an existing action group, or create a new one to define the action to be taken when the set conditions are met:

Add action group

Action group name * ⓘ	<input type="text"/>			
Short name * ⓘ	<input type="text"/>			
Subscription * ⓘ	Feedback-FeatureSynthesizer			
Resource group * ⓘ	Default-ActivityLogAlerts			
Actions				
Action Name *	Action Type *	Status	Configure	Actions
<input type="text" value="Unique name for the action"/>	<input type="button" value="Select an action type"/> <ul style="list-style-type: none"> Automation Runbook Azure Function Email Azure Resource Manager Role Email/SMS/Push/Voice ITSM LogicApp Secure Webhook Webhook 			
Privacy Statement Pricing <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> i Have a consistent format in email details. Learn more </div>				
<input type="button" value="OK"/>				

Troubleshooting

Limitations and known issues for data drift monitors:

- The time range when analyzing historical data is limited to 31 intervals of the monitor's frequency setting.
- Limitation of 200 features, unless a feature list is not specified (all features used).
- Compute size must be large enough to handle the data.
- Ensure your dataset has data within the start and end date for a given monitor run.
- Dataset monitors will only work on datasets that contain 50 rows or more.
- Columns, or features, in the dataset are classified as categorical or numeric based on the conditions in the following table. If the feature does not meet these conditions - for instance, a column of type string with > 100 unique values - the feature is dropped from our data drift algorithm, but is still profiled.

FEATURE TYPE	DATA TYPE	CONDITION	LIMITATIONS
--------------	-----------	-----------	-------------

FEATURE TYPE	DATA TYPE	CONDITION	LIMITATIONS
Categorical	string, bool, int, float	The number of unique values in the feature is less than 100 and less than 5% of the number of rows.	Null is treated as its own category.
Numerical	int, float	The values in the feature are of a numerical data type and do not meet the condition for a categorical feature.	Feature dropped if >15% of values are null.

- When you have [created a data drift monitor](#) but cannot see data on the **Dataset monitors** page in Azure Machine Learning studio, try the following.
 - Check if you have selected the right date range at the top of the page.
 - On the **Dataset Monitors** tab, select the experiment link to check run status. This link is on the far right of the table.
 - If run completed successfully, check driver logs to see how many metrics has been generated or if there's any warning messages. Find driver logs in the **Output + logs** tab after you click on an experiment.
- If the SDK `backfill()` function does not generate the expected output, it may be due to an authentication issue. When you create the compute to pass into this function, do not use `Run.get_context().experiment.workspace.compute_targets`. Instead, use [ServicePrincipalAuthentication](#) such as the following to create the compute that you pass into that `backfill()` function:

```
auth = ServicePrincipalAuthentication(
    tenant_id=tenant_id,
    service_principal_id=app_id,
    service_principal_password=client_secret
)
ws = Workspace.get("xxx", auth=auth, subscription_id="xxx", resource_group="xxx")
compute = ws.compute_targets.get("xxx")
```

- From the Model Data Collector, it can take up to (but usually less than) 10 minutes for data to arrive in your blob storage account. In a script or Notebook, wait 10 minutes to ensure cells below will run.

```
import time
time.sleep(600)
```

Next steps

- Head to the [Azure Machine Learning studio](#) or the [Python notebook](#) to set up a dataset monitor.
- See how to set up data drift on [models deployed to Azure Kubernetes Service](#).
- Set up dataset drift monitors with [event grid](#).
- See these common [troubleshooting tips](#) if you're having problems.

Version and track datasets in experiments

12/23/2020 • 5 minutes to read • [Edit Online](#)

In this article, you'll learn how to version and track Azure Machine Learning datasets for reproducibility. Dataset versioning is a way to bookmark the state of your data so that you can apply a specific version of the dataset for future experiments.

Typical versioning scenarios:

- When new data is available for retraining
- When you're applying different data preparation or feature engineering approaches

Prerequisites

For this tutorial, you need:

- [Azure Machine Learning SDK for Python installed](#). This SDK includes the `azureml-datasets` package.
- An [Azure Machine Learning workspace](#). Retrieve an existing one by running the following code, or [create a new workspace](#).

```
import azureml.core
from azureml.core import Workspace

ws = Workspace.from_config()
```

- An [Azure Machine Learning dataset](#).

Register and retrieve dataset versions

By registering a dataset, you can version, reuse, and share it across experiments and with colleagues. You can register multiple datasets under the same name and retrieve a specific version by name and version number.

Register a dataset version

The following code registers a new version of the `titanic_ds` dataset by setting the `create_new_version` parameter to `True`. If there's no existing `titanic_ds` dataset registered with the workspace, the code creates a new dataset with the name `titanic_ds` and sets its version to 1.

```
titanic_ds = titanic_ds.register(workspace = workspace,
                                 name = 'titanic_ds',
                                 description = 'titanic training data',
                                 create_new_version = True)
```

Retrieve a dataset by name

By default, the `get_by_name()` method on the `Dataset` class returns the latest version of the dataset registered with the workspace.

The following code gets version 1 of the `titanic_ds` dataset.

```

from azureml.core import Dataset
# Get a dataset by name and version number
titanic_ds = Dataset.get_by_name(workspace = workspace,
                                 name = 'titanic_ds',
                                 version = 1)

```

Versioning best practice

When you create a dataset version, you're *not* creating an extra copy of data with the workspace. Because datasets are references to the data in your storage service, you have a single source of truth, managed by your storage service.

IMPORTANT

If the data referenced by your dataset is overwritten or deleted, calling a specific version of the dataset does *not* revert the change.

When you load data from a dataset, the current data content referenced by the dataset is always loaded. If you want to make sure that each dataset version is reproducible, we recommend that you not modify data content referenced by the dataset version. When new data comes in, save new data files into a separate data folder and then create a new dataset version to include data from that new folder.

The following image and sample code show the recommended way to structure your data folders and to create dataset versions that reference those folders:

NAME	ACCESS TIER	ACCESS TIER LAST MODIFIED	LAST MODIFIED	BLOB TYPE	CONTENT TYPE	SIZE
week 27				Folder		
week 28				Folder		
week 29				Folder		

```

from azureml.core import Dataset

# get the default datastore of the workspace
datastore = workspace.get_default_datastore()

# create & register weather_ds version 1 pointing to all files in the folder of week 27
datastore_path1 = [(datastore, 'Weather/week 27')]
dataset1 = Dataset.File.from_files(path=datastore_path1)
dataset1.register(workspace = workspace,
                  name = 'weather_ds',
                  description = 'weather data in week 27',
                  create_new_version = True)

# create & register weather_ds version 2 pointing to all files in the folder of week 27 and 28
datastore_path2 = [(datastore, 'Weather/week 27'), (datastore, 'Weather/week 28')]
dataset2 = Dataset.File.from_files(path = datastore_path2)
dataset2.register(workspace = workspace,
                  name = 'weather_ds',
                  description = 'weather data in week 27, 28',
                  create_new_version = True)

```

Version an ML pipeline output dataset

You can use a dataset as the input and output of each [ML pipeline](#) step. When you rerun pipelines, the output of each pipeline step is registered as a new dataset version.

ML pipelines populate the output of each step into a new folder every time the pipeline reruns. This behavior allows the versioned output datasets to be reproducible. Learn more about [datasets in pipelines](#).

```
from azureml.core import Dataset
from azureml.pipeline.steps import PythonScriptStep
from azureml.pipeline.core import Pipeline, PipelineData
from azureml.core. runconfig import CondaDependencies, RunConfiguration

# get input dataset
input_ds = Dataset.get_by_name(workspace, 'weather_ds')

# register pipeline output as dataset
output_ds = PipelineData('prepared_weather_ds', datastore=datastore).as_dataset()
output_ds = output_ds.register(name='prepared_weather_ds', create_new_version=True)

conda = CondaDependencies.create(
    pip_packages=['azureml-defaults', 'azureml-dataprep[fuse,pandas]'],
    pin_sdk_version=False)

run_config = RunConfiguration()
run_config.environment.docker.enabled = True
run_config.environment.python.conda_dependencies = conda

# configure pipeline step to use dataset as the input and output
prep_step = PythonScriptStep(script_name="prepare.py",
                             inputs=[input_ds.as_named_input('weather_ds')],
                             outputs=[output_ds],
                             runconfig=run_config,
                             compute_target=compute_target,
                             source_directory=project_folder)
```

Track data in your experiments

Azure Machine Learning tracks your data throughout your experiment as input and output datasets.

The following are scenarios where your data is tracked as an **input dataset**.

- As a `DatasetConsumptionConfig` object through either the `inputs` or `arguments` parameter of your `ScriptRunConfig` object when submitting the experiment run.
- When methods like, `get_by_name()` or `get_by_id()` are called in your script. For this scenario, the name assigned to the dataset when you registered it to the workspace is the name displayed.

The following are scenarios where your data is tracked as an **output dataset**.

- Pass an `OutputFileDatasetConfig` object through either the `outputs` or `arguments` parameter when submitting an experiment run. `OutputFileDatasetConfig` objects can also be used to persist data between pipeline steps. See [Move data between ML pipeline steps](#).

TIP

`OutputFileDatasetConfig` is a public preview class containing [experimental](#) preview features that may change at any time.

- Register a dataset in your script. For this scenario, the name assigned to the dataset when you registered it

to the workspace is the name displayed. In the following example, `training_ds` is the name that would be displayed.

```
training_ds = unregistered_ds.register(workspace = workspace,
                                         name = 'training_ds',
                                         description = 'training data'
                                         )
```

- Submit child run with an unregistered dataset in script. This results in an anonymous saved dataset.

Trace datasets in experiment runs

For each Machine Learning experiment, you can easily trace the datasets used as input with the experiment `Run` object.

The following code uses the `get_details()` method to track which input datasets were used with the experiment run:

```
# get input datasets
inputs = run.get_details()['inputDatasets']
input_dataset = inputs[0]['dataset']

# list the files referenced by input_dataset
input_dataset.to_path()
```

You can also find the `input_datasets` from experiments by using the .

The following image shows where to find the input dataset of an experiment on Azure Machine Learning studio. For this example, go to your **Experiments** pane and open the **Properties** tab for a specific run of your experiment, `keras-mnist`.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a sidebar with navigation links: New, Home, Author, Notebooks, Automated ML, Designer, Assets, Datasets, Experiments (which is selected), Pipelines, Models, Endpoints, Manage, Compute, and Environments. The main area shows a breadcrumb path: mayworkspace > Experiments > keras-mnist > keras-mnist_1570739212_92bc7af. Below the path is the title "Run 24". There are "Refresh" and "Switch to old experience" buttons. The "Properties" tab is selected. The properties table has two columns: "Properties" and "Metrics". The "Properties" column contains the following rows:

- Status: Completed
- Created: Oct 10, 2019 1:31 PM
- Duration: 1m 55.69s
- Compute target: local
- Run ID: keras-mnist_1570739212_92bc7af
- Run number: 24
- Script name: keras_mnist.py

The "Metrics" column contains the following rows:

- Accuracy: Vector
- Final test accuracy: 0.9794999957084656
- Final test loss: 0.17366168976166854
- Loss: Vector
- Loss v.s. Accuracy: Image

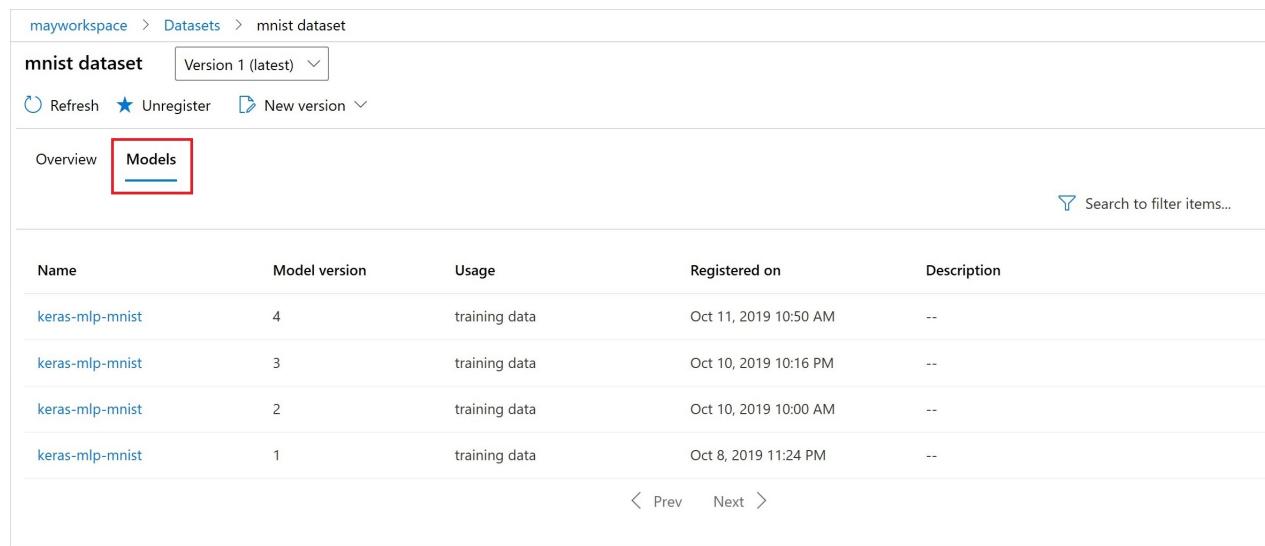
A red box highlights the "Input datasets" row in the "Properties" section, which lists "keras-mnist".

Use the following code to register models with datasets:

```
model = run.register_model(model_name='keras-mlp-mnist',
                           model_path=model_path,
                           datasets =[('training data',train_dataset)])
```

After registration, you can see the list of models registered with the dataset by using Python or go to the [studio](#).

The following view is from the **Datasets** pane under **Assets**. Select the dataset and then select the **Models** tab for a list of the models that are registered with the dataset.



The screenshot shows the Datasets pane with the following interface elements:

- Path: mayworkspace > Datasets > mnist dataset
- Dataset Selection: mnist dataset, Version 1 (latest)
- Actions: Refresh, Unregister, New version
- Tab Selection: Overview (disabled) and Models (selected, highlighted with a red border)
- Search: Search to filter items...
- Table: A list of registered models with columns: Name, Model version, Usage, Registered on, and Description. The data is as follows:

Name	Model version	Usage	Registered on	Description
keras-mlp-mnist	4	training data	Oct 11, 2019 10:50 AM	--
keras-mlp-mnist	3	training data	Oct 10, 2019 10:16 PM	--
keras-mlp-mnist	2	training data	Oct 10, 2019 10:00 AM	--
keras-mlp-mnist	1	training data	Oct 8, 2019 11:24 PM	--

Pagination: < Prev Next >

Next steps

- [Train with datasets](#)
- [More sample dataset notebooks](#)

Use differential privacy in Azure Machine Learning (preview)

12/23/2020 • 5 minutes to read • [Edit Online](#)

Learn how to apply differential privacy best practices to Azure Machine Learning models by using the SmartNoise Python packages.

Differential privacy is the gold-standard definition of privacy. Systems that adhere to this definition of privacy provide strong assurances against a wide range of data reconstruction and reidentification attacks, including attacks by adversaries who possess auxiliary information. Learn more about how [differential privacy works](#).

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](#)

Install SmartNoise packages

Standalone installation

The libraries are designed to work from distributed Spark clusters, and can be installed just like any other package.

The instructions below assume that your `python` and `pip` commands are mapped to `python3` and `pip3`.

Use pip to install the [SmartNoise Python packages](#).

```
pip install opendp-smartnoise
```

To verify that the packages are installed, launch a python prompt and type:

```
import opendp.smartnoise.core
import opendp.smartnoise.sql
```

If the imports succeed, the libraries are installed and ready to use.

Docker image

You can also use SmartNoise packages with Docker.

Pull the `opendp/smartnoise` image to use the libraries inside a Docker container that includes Spark, Jupyter, and sample code.

```
docker pull opendp/smartnoise:privacy
```

Once you've pulled the image, launch the Jupyter server:

```
docker run --rm -p 8989:8989 --name smartnoise-run opendp/smartnoise:privacy
```

This starts a Jupyter server at port `8989` on your `localhost`, with password `pass@word99`. Assuming you used the command line above to start the container with name `smartnoise-privacy`, you can open a bash terminal in the

Jupyter server by running:

```
docker exec -it smartnoise-run bash
```

The Docker instance clears all state on shutdown, so you will lose any notebooks you create in the running instance. To remedy this, you can bind mount a local folder to the container when you launch it:

```
docker run --rm -p 8989:8989 --name smartnoise-run --mount type=bind,source=/Users/your_name/my-notebooks,target=/home/privacy/my-notebooks opendp/smartnoise:privacy
```

Any notebooks you create under the *my-notebooks* folder will be stored in your local filesystem.

Perform data analysis

To prepare a differentially private release, you need to choose a data source, a statistic, and some privacy parameters, indicating the level of privacy protection.

This sample references the California Public Use Microdata (PUMS), representing anonymized records of citizen demographics:

```
import os
import sys
import numpy as np
import opendp.smartnoise.core as sn

data_path = os.path.join('.', 'data', 'PUMS_california_demographics_1000', 'data.csv')
var_names = ["age", "sex", "educ", "race", "income", "married", "pid"]
```

In this example, we compute the mean and the variance of the age. We use a total `epsilon` of 1.0 (epsilon is our privacy parameter, spreading our privacy budget across the two quantities we want to compute. Learn more about [privacy metrics](#).

```
with sn.Analysis() as analysis:
    # load data
    data = sn.Dataset(path = data_path, column_names = var_names)

    # get mean of age
    age_mean = sn.dp_mean(data = sn.cast(data['age'], type="FLOAT"),
                           privacy_usage = {'epsilon': .65},
                           data_lower = 0.,
                           data_upper = 100.,
                           data_n = 1000
                           )
    # get variance of age
    age_var = sn.dp_variance(data = sn.cast(data['age'], type="FLOAT"),
                             privacy_usage = {'epsilon': .35},
                             data_lower = 0.,
                             data_upper = 100.,
                             data_n = 1000
                             )
    analysis.release()

    print("DP mean of age: {}".format(age_mean.value))
    print("DP variance of age: {}".format(age_var.value))
    print("Privacy usage: {}".format(analysis.privacy_usage))
```

The results look something like those below:

```

DP mean of age: 44.55598845931517
DP variance of age: 231.79044646429134
Privacy usage: approximate {
    epsilon: 1.0
}

```

There are some important things to note about this example. First, the `Analysis` object represents a data processing graph. In this example, the mean and variance are computed from the same source node. However, you can include more complex expressions that combine inputs with outputs in arbitrary ways.

The analysis graph includes `data_upper` and `data_lower` metadata, specifying the lower and upper bounds for ages. These values are used to precisely calibrate the noise to ensure differential privacy. These values are also used in some handling of outliers or missing values.

Finally, the analysis graph keeps track of the total privacy budget spent.

You can use the library to compose more complex analysis graphs, with several mechanisms, statistics, and utility functions:

STATISTICS	MECHANISMS	UTILITIES
Count	Gaussian	Cast
Histogram	Geometric	Clamping
Mean	Laplace	Digitize
Quantiles		Filter
Sum		Imputation
Variance/Covariance		Transform

See the [data analysis notebook](#) for more details.

Approximate utility of differentially private releases

Because differential privacy operates by calibrating noise, the utility of releases may vary depending on the privacy risk. Generally, the noise needed to protect each individual becomes negligible as sample sizes grow large, but overwhelm the result for releases that target a single individual. Analysts can review the accuracy information for a release to determine how useful the release is:

```

with sn.Analysis() as analysis:
    # load data
    data = sn.Dataset(path = data_path, column_names = var_names)

    # get mean of age
    age_mean = sn.dp_mean(data = sn.cast(data['age'], type="FLOAT"),
                          privacy_usage = {'epsilon': .65},
                          data_lower = 0.,
                          data_upper = 100.,
                          data_n = 1000
                         )
analysis.release()

print("Age accuracy is: {}".format(age_mean.get_accuracy(0.05)))

```

The result of that operation should look similar to that below:

```
Age accuracy is: 0.2995732273553991
```

This example computes the mean as above, and uses the `get_accuracy` function to request accuracy at `alpha` of 0.05. An `alpha` of 0.05 represents a 95% interval, in that released value will fall within the reported accuracy bounds about 95% of the time. In this example, the reported accuracy is 0.3, which means the released value will be within an interval of width 0.6, about 95% of the time. It is not correct to think of this value as an error bar, since the released value will fall outside the reported accuracy range at the rate specified by `alpha`, and values outside the range may be outside in either direction.

Analysts may query `get_accuracy` for different values of `alpha` to get narrower or wider confidence intervals, without incurring additional privacy cost.

Generate a histogram

The built-in `dp_histogram` function creates differentially private histograms over any of the following data types:

- A continuous variable, where the set of numbers has to be divided into bins
- A boolean or dichotomous variable, that can only take on two values
- A categorical variable, where there are distinct categories enumerated as strings

Here is an example of an `Analysis` specifying bins for a continuous variable histogram:

```
income_edges = list(range(0, 100000, 10000))

with sn.Analysis() as analysis:
    data = sn.Dataset(path = data_path, column_names = var_names)

    income_histogram = sn.dp_histogram(
        sn.cast(data['income'], type='int', lower=0, upper=100),
        edges = income_edges,
        upper = 1000,
        null_value = 150,
        privacy_usage = {'epsilon': 0.5}
    )
```

Because the individuals are disjointly partitioned among histogram bins, the privacy cost is incurred only once per histogram, even if the histogram includes many bins.

For more on histograms, see the [histograms notebook](#).

Generate a covariance matrix

SmartNoise offers three different functionalities with its `dp_covariance` function:

- Covariance between two vectors
- Covariance matrix of a matrix
- Cross-covariance matrix of a pair of matrices

Here is an example of computing a scalar covariance:

```
with sn.Analysis() as analysis:  
    wn_data = sn.Dataset(path = data_path, column_names = var_names)  
  
    age_income_cov_scalar = sn.dp_covariance(  
        left = sn.cast(wn_data['age']),  
        type = "FLOAT"),  
        right = sn.cast(wn_data['income']),  
        type = "FLOAT"),  
        privacy_usage = {'epsilon': 1.0},  
        left_lower = 0.,  
        left_upper = 100.,  
        left_n = 1000,  
        right_lower = 0.,  
        right_upper = 500_000.,  
        right_n = 1000)
```

For more information, see the [covariance notebook](#)

Next Steps

- Explore [SmartNoise sample notebooks](#).

Configure and submit training runs

12/23/2020 • 8 minutes to read • [Edit Online](#)

In this article, you learn how to configure and submit Azure Machine Learning runs to train your models. Snippets of code explain the key parts of configuration and submission of a training script. Then use one of the [example notebooks](#) to find the full end-to-end working examples.

When training, it is common to start on your local computer, and then later scale out to a cloud-based cluster. With Azure Machine Learning, you can run your script on various compute targets without having to change your training script.

All you need to do is define the environment for each compute target within a [script run configuration](#). Then, when you want to run your training experiment on a different compute target, specify the run configuration for that compute.

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 for Python](#) (>= 1.13.0)
- An [Azure Machine Learning workspace](#), `ws`
- A compute target, `my_compute_target`. [Create a compute target](#)

What's a script run configuration?

A [ScriptRunConfig](#) is used to configure the information necessary for submitting a training run as part of an experiment.

You submit your training experiment with a ScriptRunConfig object. This object includes the:

- **source_directory**: The source directory that contains your training script
- **script**: The training script to run
- **compute_target**: The compute target to run on
- **environment**: The environment to use when running the script
- and some additional configurable options (see the [reference documentation](#) for more information)

Train your model

The code pattern to submit a training run is the same for all types of compute targets:

1. Create an experiment to run
2. Create an environment where the script will run
3. Create a ScriptRunConfig, which specifies the compute target and environment
4. Submit the run
5. Wait for the run to complete

Or you can:

- Submit a HyperDrive run for [hyperparameter tuning](#).
- Submit an experiment via the [VS Code extension](#).

Create an experiment

Create an experiment in your workspace.

```
from azureml.core import Experiment

experiment_name = 'my_experiment'
experiment = Experiment(workspace=ws, name=experiment_name)
```

Select a compute target

Select the compute target where your training script will run on. If no compute target is specified in the `ScriptRunConfig`, or if `compute_target='local'`, Azure ML will execute your script locally.

The example code in this article assumes that you have already created a compute target `my_compute_target` from the "Prerequisites" section.

Create an environment

Azure Machine Learning [environments](#) are an encapsulation of the environment where your machine learning training happens. They specify the Python packages, Docker image, environment variables, and software settings around your training and scoring scripts. They also specify runtimes (Python, Spark, or Docker).

You can either define your own environment, or use an Azure ML curated environment. [Curated environments](#) are predefined environments that are available in your workspace by default. These environments are backed by cached Docker images which reduces the run preparation cost. See [Azure Machine Learning Curated Environments](#) for the full list of available curated environments.

For a remote compute target, you can use one of these popular curated environments to start with:

```
from azureml.core import Workspace, Environment

ws = Workspace.from_config()
myenv = Environment.get(workspace=ws, name="AzureML-Minimal")
```

For more information and details about environments, see [Create & use software environments in Azure Machine Learning](#).

Local compute target

If your compute target is your **local machine**, you are responsible for ensuring that all the necessary packages are available in the Python environment where the script runs. Use

`python.user_managed_dependencies` to use your current Python environment (or the Python on the path you specify).

```
from azureml.core import Environment

myenv = Environment("user-managed-env")
myenv.python.user_managed_dependencies = True

# You can choose a specific Python environment by pointing to a Python path
# myenv.python.interpreter_path = '/home/johndoe/miniconda3/envs/myenv/bin/python'
```

Create the script run configuration

Now that you have a compute target (`my_compute_target`) and environment (`myenv`), create a script run configuration that runs your training script (`train.py`) located in your `project_folder` directory:

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory=project_folder,
                      script='train.py',
                      compute_target=my_compute_target,
                      environment=myenv)

# Set compute target
# Skip this if you are running on your local computer
script_run_config.run_config.target = my_compute_target
```

If you do not specify an environment, a default environment will be created for you.

If you have command-line arguments you want to pass to your training script, you can specify them via the `arguments` parameter of the `ScriptRunConfig` constructor, e.g.

```
arguments=['--arg1', arg1_val, '--arg2', arg2_val].
```

If you want to override the default maximum time allowed for the run, you can do so via the `max_run_duration_seconds` parameter. The system will attempt to automatically cancel the run if it takes longer than this value.

Specify a distributed job configuration

If you want to run a distributed training job, provide the distributed job-specific config to the `distributed_job_config` parameter. Supported config types include [Mpiconfiguration](#), [TensorflowConfiguration](#), and [PyTorchConfiguration](#).

For more information and examples on running distributed Horovod, TensorFlow and PyTorch jobs, see:

- [Train TensorFlow models](#)
- [Train PyTorch models](#)

Submit the experiment

```
run = experiment.submit(config=src)
run.wait_for_completion(show_output=True)
```

IMPORTANT

When you submit the training run, a snapshot of the directory that contains your training scripts is created and sent to the compute target. It is also stored as part of the experiment in your workspace. If you change files and submit the run again, only the changed files will be uploaded.

To prevent unnecessary files from being included in the snapshot, make an ignore file (`.gitignore` or `.amlignore`) 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 `.gitignore`. The `.amlignore` file uses the same syntax. *If both files exist, the `.amlignore` file takes precedence.*

For more information about snapshots, see [Snapshots](#).

IMPORTANT

Special Folders Two folders, *outputs* and *logs*, receive special treatment by Azure Machine Learning. During training, when you write files to folders named *outputs* and *logs* that are relative to the root directory (`./outputs` and `./logs`, respectively), the files will automatically upload to your run history so that you have access to them once your run is finished.

To create artifacts during training (such as model files, checkpoints, data files, or plotted images) write these to the `./outputs` folder.

Similarly, you can write any logs from your training run to the `./logs` folder. To utilize Azure Machine Learning's [TensorBoard integration](#) make sure you write your TensorBoard logs to this folder. While your run is in progress, you will be able to launch TensorBoard and stream these logs. Later, you will also be able to restore the logs from any of your previous runs.

For example, to download a file written to the *outputs* folder to your local machine after your remote training run:

```
run.download_file(name='outputs/my_output_file', output_file_path='my_destination_path')
```

Git tracking and integration

When you start a training run where the source directory is a local Git repository, information about the repository is stored in the run history. For more information, see [Git integration for Azure Machine Learning](#).

Notebook examples

See these notebooks for examples of configuring runs for various training scenarios:

- [Training on various compute targets](#)
- [Training with ML frameworks](#)
- [tutorials/img-classification-part1-training.ipynb](#)

Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service](#).

Troubleshooting

- **ModuleErrors (No module named)**: If you are running into ModuleErrors while submitting experiments in Azure ML, the training script is expecting a package to be installed but it isn't added. Once you provide the package name, Azure ML installs the package in the environment used for your training run.

If you are using Estimators to submit experiments, you can specify a package name via `pip_packages` or `conda_packages` parameter in the estimator based on from which source you want to install the package. You can also specify a yml file with all your dependencies using `conda_dependencies_file` or list all your pip requirements in a txt file using `pip_requirements_file` parameter. If you have your own Azure ML Environment object that you want to override the default image used by the estimator, you can specify that environment via the `environment` parameter of the estimator constructor.

Azure ML maintained docker images and their contents can be seen in [AzureML Containers](#).

Framework-specific dependencies are listed in the respective framework documentation:

- [Chainer](#)
- [PyTorch](#)
- [TensorFlow](#)
- [SKLearn](#)

NOTE

If you think a particular package is common enough to be added in Azure ML maintained images and environments please raise a GitHub issue in [AzureML Containers](#).

- **NameError (Name not defined), AttributeError (Object has no attribute)**: This exception should come from your training scripts. You can look at the log files from Azure portal to get more information about the specific name not defined or attribute error. From the SDK, you can use `run.get_details()` to look at the error message. This will also list all the log files generated for your run. Please make sure to take a look at your training script and fix the error before resubmitting your run.
- **Run or experiment deletion**: Experiments can be archived by using the [Experiment.archive](#) method, or from the Experiment tab view in Azure Machine Learning studio client via the "Archive experiment" button. This action hides the experiment from list queries and views, but does not delete it.
Permanent deletion of individual experiments or runs is not currently supported. For more information on deleting Workspace assets, see [Export or delete your Machine Learning service workspace data](#).
- **Metric Document is too large**: Azure Machine Learning has internal limits on the size of metric objects that can be logged at once from a training run. If you encounter a "Metric Document is too large" error when logging a list-valued metric, try splitting the list into smaller chunks, for example:

```
run.log_list("my metric name", my_metric[:N])
run.log_list("my metric name", my_metric[N:])
```

Internally, Azure ML concatenates the blocks with the same metric name into a contiguous list.

Next steps

- [Tutorial: Train a model](#) uses a managed compute target to train a model.
- See how to train models with specific ML frameworks, such as [Scikit-learn](#), [TensorFlow](#), and [PyTorch](#).
- Learn how to [efficiently tune hyperparameters](#) to build better models.
- Once you have a trained model, learn [how and where to deploy models](#).
- View the [ScriptRunConfig class](#) SDK reference.
- [Use Azure Machine Learning with Azure Virtual Networks](#)

Tune hyperparameters for your model with Azure Machine Learning

12/23/2020 • 12 minutes to read • [Edit Online](#)

Automate efficient hyperparameter tuning by using Azure Machine Learning [HyperDrive package](#). Learn how to complete the steps required to tune hyperparameters with the [Azure Machine Learning SDK](#):

1. Define the parameter search space
2. Specify a primary metric to optimize
3. Specify early termination policy for low-performing runs
4. Allocate resources
5. Launch an experiment with the defined configuration
6. Visualize the training runs
7. Select the best configuration for your model

What are hyperparameters?

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 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

```
{  
    "batch_size": choice(16, 32, 64, 128)  
    "number_of_hidden_layers": choice(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(low, high, q)` - Returns a value like $\text{round}(\text{uniform}(\text{low}, \text{high}) / q) * q$
- `qloguniform(low, high, q)` - Returns a value like $\text{round}(\exp(\text{uniform}(\text{low}, \text{high})) / q) * q$
- `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(low, high)` - Returns a value uniformly distributed between low and high
- `loguniform(low, high)` - Returns a value drawn according to $\exp(\text{uniform}(\text{low}, \text{high}))$ 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:

```
{
    "learning_rate": normal(10, 3),
    "keep_probability": uniform(0.05, 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.

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 runs. 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.

```
from azureml.train.hyperdrive import RandomParameterSampling
from azureml.train.hyperdrive import normal, uniform, choice
param_sampling = RandomParameterSampling( {
    "learning_rate": normal(10, 3),
    "keep_probability": uniform(0.05, 0.1),
    "batch_size": choice(16, 32, 64, 128)
})
```

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 runs.

Performs 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 azureml.train.hyperdrive import GridParameterSampling
from azureml.train.hyperdrive import choice
param_sampling = GridParameterSampling( {
    "num_hidden_layers": choice(1, 2, 3),
    "batch_size": choice(16, 32)
})
```

Bayesian sampling

[Bayesian sampling](#) is based on the Bayesian optimization algorithm. It picks samples based on how previous samples performed, 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 runs greater than or equal to 20 times the number of hyperparameters being tuned.

The number of concurrent runs has an impact on the effectiveness of the tuning process. A smaller number of concurrent runs may lead to better sampling convergence, since the smaller degree of parallelism increases the number of runs that benefit from previously completed runs.

Bayesian sampling only supports `choice`, `uniform`, and `quniform` distributions over the search space.

```
from azureml.train.hyperdrive import BayesianParameterSampling
from azureml.train.hyperdrive import uniform, choice
param_sampling = BayesianParameterSampling( {
    "learning_rate": uniform(0.05, 0.1),
    "batch_size": choice(16, 32, 64, 128)
})
```

Specify primary metric

Specify the [primary metric](#) you want hyperparameter tuning to optimize. Each training run is evaluated for the primary metric. The early termination policy uses the primary metric to identify low-performance runs.

Specify the following attributes for your primary metric:

- `primary_metric_name` : The name of the primary metric needs to exactly match the name of the metric logged by the training script
- `primary_metric_goal` : It can be either `PrimaryMetricGoal.MAXIMIZE` or `PrimaryMetricGoal.MINIMIZE` and determines whether the primary metric will be maximized or minimized when evaluating the runs.

```
primary_metric_name="accuracy",
primary_metric_goal=PrimaryMetricGoal.MAXIMIZE
```

This sample maximizes "accuracy".

Log metrics for hyperparameter tuning

The training script for your model **must** log the primary metric during model training so that HyperDrive can access it for hyperparameter tuning.

Log the primary metric in your training script with the following sample snippet:

```
from azureml.core.run import Run
run_logger = Run.get_context()
run_logger.log("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 in model training runs, see [Enable logging in Azure ML training runs](#).

Specify early termination policy

Automatically terminate poorly performing runs 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 1 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 runs 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`.

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 terminates runs where the primary metric is not within the specified slack factor/slack amount compared to the best performing run.

NOTE

Bayesian sampling does not support early termination. When using Bayesian sampling, set `early_termination_policy = None`.

Specify the following configuration parameters:

- `slack_factor` or `slack_amount` : the slack allowed with respect to the best performing training run. `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 run 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 runs whose best metric at interval 10 is less than $0.8/(1 + \text{slack_factor})$ 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 azureml.train.hyperdrive import BanditPolicy
early_termination_policy = BanditPolicy(slack_factor = 0.1, evaluation_interval=1, delay_evaluation=5)
```

In this example, the early termination policy is applied at every interval when metrics are reported, starting at evaluation interval 5. Any run whose best metric is less than $(1/(1+0.1))$ or 91% of the best performing run will be terminated.

Median stopping policy

[Median stopping](#) is an early termination policy based on running averages of primary metrics reported by the runs. This policy computes running averages across all training runs and terminates runs with primary metric values worse than the median of 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 azureml.train.hyperdrive import MedianStoppingPolicy
early_termination_policy = MedianStoppingPolicy(evaluation_interval=1, delay_evaluation=5)
```

In this example, the early termination policy is applied at every interval starting at evaluation interval 5. A run is terminated at interval 5 if its best primary metric is worse than the median of the running averages over intervals 1:5 across all training runs.

Truncation selection policy

[Truncation selection](#) cancels a percentage of lowest performing runs at each evaluation interval. Runs are compared using the primary metric.

This policy takes the following configuration parameters:

- `truncation_percentage` : the percentage of lowest performing runs 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

```
from azureml.train.hyperdrive import TruncationSelectionPolicy
early_termination_policy = TruncationSelectionPolicy(evaluation_interval=1, truncation_percentage=20,
delay_evaluation=5)
```

In this example, the early termination policy is applied at every interval starting at evaluation interval 5. A run terminates at interval 5 if its performance at interval 5 is in the lowest 20% of performance of all runs at interval 5.

No termination policy (default)

If no policy is specified, the hyperparameter tuning service will let all training runs execute to completion.

```
policy=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.

Allocate resources

Control your resource budget by specifying the maximum number of training runs.

- `max_total_runs` : Maximum number of training runs. Must be an integer between 1 and 1000.
- `max_duration_minutes` : (optional) Maximum duration, in minutes, of the hyperparameter tuning experiment. Runs after this duration are canceled.

NOTE

If both `max_total_runs` and `max_duration_minutes` are specified, the hyperparameter tuning experiment terminates when the first of these two thresholds is reached.

Additionally, specify the maximum number of training runs to run concurrently during your hyperparameter tuning search.

- `max_concurrent_runs` : (optional) 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.

```
max_total_runs=20,  
max_concurrent_runs=4
```

This code configures the hyperparameter tuning experiment to use a maximum of 20 total runs, running four configurations at a time.

Configure experiment

To [configure your hyperparameter tuning](#) experiment, provide the following:

- The defined hyperparameter search space
- Your early termination policy
- The primary metric
- Resource allocation settings
- ScriptRunConfig `src`

The ScriptRunConfig is the training script that will run with the sampled hyperparameters. It defines the resources per job (single or multi-node), and the compute target to use.

NOTE

The compute target specified in `src` must have enough resources to satisfy your concurrency level. For more information on ScriptRunConfig, see [Configure training runs](#).

Configure your hyperparameter tuning experiment:

```
from azureml.train.hyperdrive import HyperDriveConfig
hd_config = HyperDriveConfig(run_config=src,
                             hyperparameter_sampling=param_sampling,
                             policy=early_termination_policy,
                             primary_metric_name="accuracy",
                             primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                             max_total_runs=100,
                             max_concurrent_runs=4)
```

Submit experiment

After you define your hyperparameter tuning configuration, [submit the experiment](#):

```
from azureml.core.experiment import Experiment
experiment = Experiment(workspace, experiment_name)
hyperdrive_run = experiment.submit(hd_config)
```

Warm start your hyperparameter tuning experiment (optional)

Finding the best hyperparameter values for your model can be an iterative process. You can reuse knowledge from the five previous runs to accelerate hyperparameter tuning.

Warm starting is handled differently depending on the sampling method:

- **Bayesian sampling:** Trials from the previous run are used as prior knowledge to pick new samples, and to improve the primary metric.
- **Random sampling or grid sampling:** Early termination uses knowledge from previous runs to determine poorly performing runs.

Specify the list of parent runs you want to warm start from.

```
from azureml.train.hyperdrive import HyperDriveRun

warmstart_parent_1 = HyperDriveRun(experiment, "warmstart_parent_run_ID_1")
warmstart_parent_2 = HyperDriveRun(experiment, "warmstart_parent_run_ID_2")
warmstart_parents_to_resume_from = [warmstart_parent_1, warmstart_parent_2]
```

If a hyperparameter tuning experiment is canceled, you can resume training runs from the last checkpoint. However, your training script must handle checkpoint logic.

The training run must use the same hyperparameter configuration and mounted the outputs folders. The training script must accept the `resume-from` argument, which contains the checkpoint or model files from which to resume the training run. You can resume individual training runs using the following snippet:

```
from azureml.core.run import Run

resume_child_run_1 = Run(experiment, "resume_child_run_ID_1")
resume_child_run_2 = Run(experiment, "resume_child_run_ID_2")
child_runs_to_resume = [resume_child_run_1, resume_child_run_2]
```

You can configure your hyperparameter tuning experiment to warm start from a previous experiment or resume individual training runs using the optional parameters `resume_from` and `resume_child_runs` in the config:

```

from azureml.train.hyperdrive import HyperDriveConfig

hd_config = HyperDriveConfig(run_config=src,
                             hyperparameter_sampling=param_sampling,
                             policy=early_termination_policy,
                             resume_from=warmstart_parents_to_resume_from,
                             resume_child_runs=child_runs_to_resume,
                             primary_metric_name="accuracy",
                             primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                             max_total_runs=100,
                             max_concurrent_runs=4)

```

Visualize experiment

Use the [Notebook widget](#) to visualize the progress of your training runs. The following snippet visualizes all your hyperparameter tuning runs in one place in a Jupyter notebook:

```

from azureml.widgets import RunDetails
RunDetails(hyperdrive_run).show()

```

This code displays a table with details about the training runs for each of the hyperparameter configurations.

Run Properties		Output Logs	
Status	Completed	previous status = 'RUNNING')] [2018-09-11T21:57:36.389083][CONTROLLER][INFO]Experiment was 'ExperimentStatus.RUNNING', is 'ExperimentStatus.FINISHED'. Run is completed.	
Start Time	9/4/2018 2:55:54 PM		
Duration	7 days, 0:01:41		
Run Id	cifar_1536098154351		
Max concurrent runs	4		
Max total runs	100		

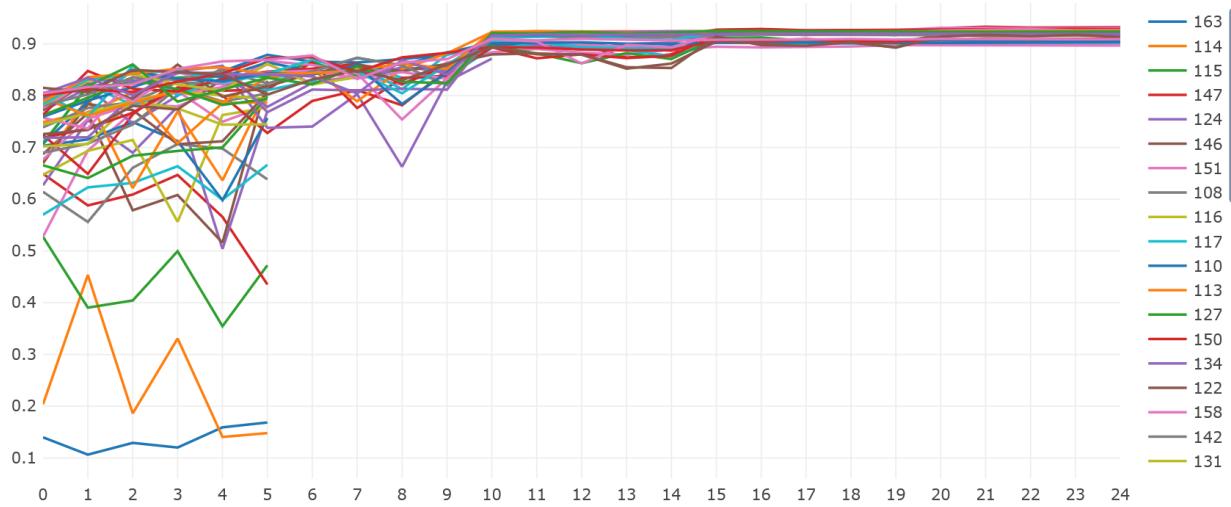
Failed (6)	Completed (36)					Canceled (58)
Run	Best Metric*	Status	Started	Duration	Run Id	
103	0.909600019454956	Completed	Sep 4, 2018 2:56 PM	5:58:16	swatig-cifar_1536098154351_0	^ v
105	0.9283999800682068	Completed	Sep 4, 2018 2:56 PM	6:04:28	swatig-cifar_1536098154351_3	^ v
104	0.9247000217437744	Completed	Sep 4, 2018 2:56 PM	3:32:35	swatig-cifar_1536098154351_2	^ v
106	0.9251000285148621	Completed	Sep 4, 2018 2:56 PM	6:01:17	swatig-cifar_1536098154351_1	^ v
107	0.9108999967575073	Completed	Sep 4, 2018 6:29 PM	4:18:35	swatig-cifar_1536098154351_4	^ v

Pages: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) [12](#) [13](#) [14](#) [15](#) ... [Next](#) [Last](#)

* The best metric field is obtained from the min/max of primary metric achieved by a run

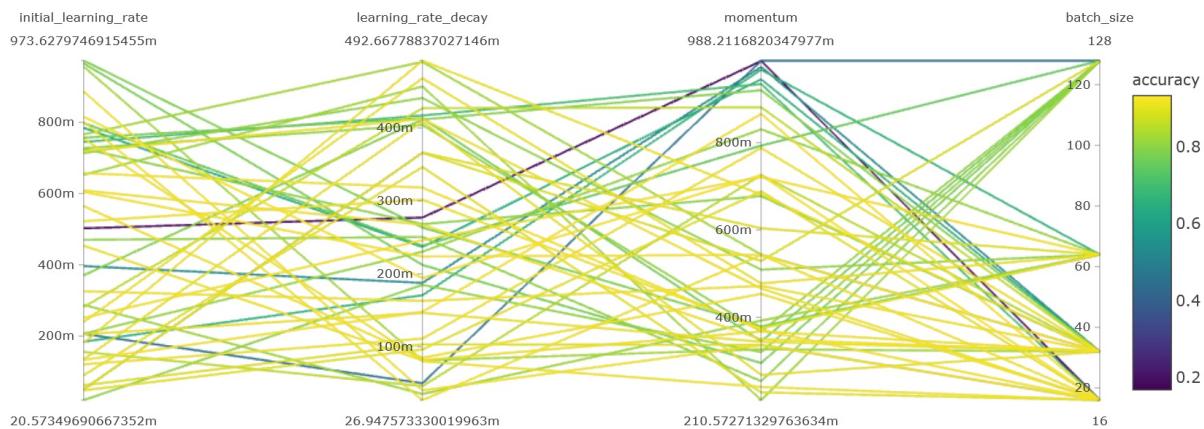
You can also visualize the performance of each of the runs as training progresses.

HyperDrive Run Primary Metric : accuracy



You can visually identify the correlation between performance and values of individual hyperparameters by using a Parallel Coordinates Plot.

Parallel Coordinates Chart ▾



You can also visualize all of your hyperparameter tuning runs in the Azure web portal. For more information on how to view an experiment in the portal, see [how to track experiments](#).

Find the best model

Once all of the hyperparameter tuning runs have completed, identify the best performing configuration and hyperparameter values:

```
best_run = hyperdrive_run.get_best_run_by_primary_metric()
best_run_metrics = best_run.get_metrics()
parameter_values = best_run.get_details()['runDefinition']['Arguments']

print('Best Run Id: ', best_run.id)
print('\n Accuracy:', best_run_metrics['accuracy'])
print('\n learning rate:', parameter_values[3])
print('\n keep probability:', parameter_values[5])
print('\n batch size:', parameter_values[7])
```

Sample notebook

Refer to train-hyperparameter-* notebooks in this folder:

- [how-to-use-azureml/ml-frameworks](#)

Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service.](#)

Next steps

- [Track an experiment](#)
- [Deploy a trained model](#)

Track experiment runs and deploy ML models with MLflow and Azure Machine Learning (preview)

12/23/2020 • 9 minutes to read • [Edit Online](#)

In this article, learn how to enable MLflow's tracking URI and logging API, collectively known as [MLflow Tracking](#), to connect Azure Machine Learning as the backend of your MLflow experiments.

Supported capabilities include:

- Track and log experiment metrics and artifacts in your [Azure Machine Learning workspace](#). If you already use MLflow Tracking for your experiments, the workspace provides a centralized, secure, and scalable location to store training metrics and models.
- Submit training jobs with MLflow Projects with Azure Machine Learning backend support (preview). You can submit jobs locally with Azure Machine Learning tracking or migrate your runs to the cloud like via an [Azure Machine Learning Compute](#).
- Track and manage models in MLflow and Azure Machine Learning model registry.
- Deploy your MLflow experiments as an Azure Machine Learning web service. By deploying as a web service, you can apply the Azure Machine Learning monitoring and data drift detection functionalities to your production models.

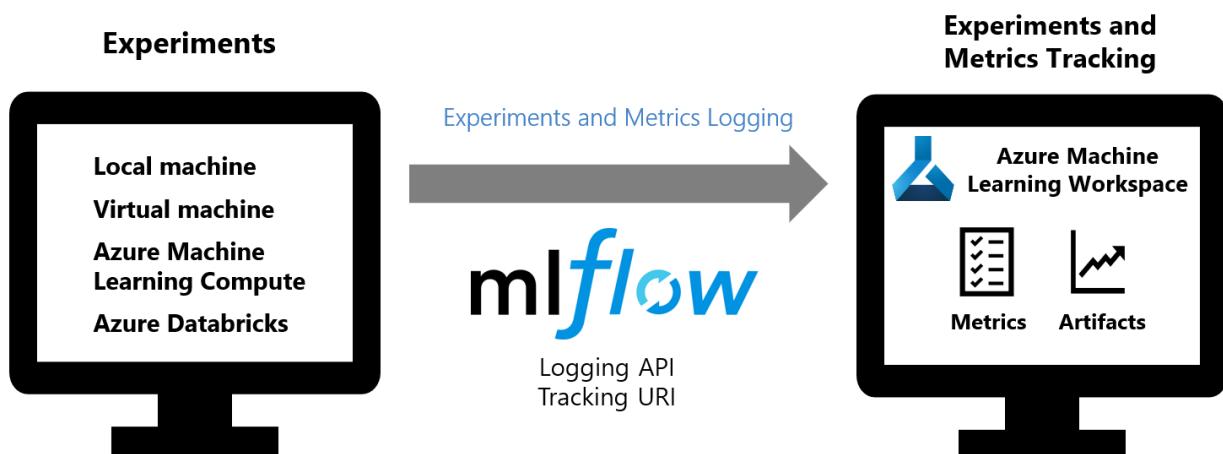
[MLflow](#) is an open-source library for managing the life cycle of your machine learning experiments. MLFlow Tracking is a component of MLflow that logs and tracks your training run metrics and model artifacts, no matter your experiment's environment--locally on your computer, on a remote compute target, a virtual machine, or an [Azure Databricks cluster](#).

NOTE

As an open source library, MLflow changes frequently. As such, the functionality made available via the Azure Machine Learning and MLflow integration should be considered as a preview, and not fully supported by Microsoft.

The following diagram illustrates that with MLflow Tracking, you track an experiment's run metrics and store model artifacts in your Azure Machine Learning workspace.

MLflow with Azure Machine Learning Experimentation



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](#).

Compare MLflow and Azure Machine Learning clients

The following table summarizes the different clients that can use Azure Machine Learning, and their respective function capabilities.

MLflow Tracking offers metric logging and artifact storage functionalities that are only otherwise available via the [Azure Machine Learning Python SDK](#).

CAPABILITY	MLFLOW TRACKING & DEPLOYMENT	AZURE MACHINE LEARNING PYTHON SDK	AZURE MACHINE LEARNING CLI	AZURE MACHINE LEARNING STUDIO
Manage workspace		✓	✓	✓
Use data stores		✓	✓	
Log metrics	✓	✓		
Upload artifacts	✓	✓		
View metrics	✓	✓	✓	✓
Manage compute		✓	✓	✓
Deploy models	✓	✓	✓	✓
Monitor model performance		✓		
Detect data drift		✓		✓

Prerequisites

- Install the `azureml-mlflow` package.
 - This package automatically brings in `azureml-core` of the [The Azure Machine Learning Python SDK](#), which provides the connectivity for MLflow to access your workspace.
- [Create an Azure Machine Learning Workspace](#).

Track local runs

MLflow Tracking with Azure Machine Learning lets you store the logged metrics and artifacts from your local runs into your Azure Machine Learning workspace.

Import the `mlflow` and `Workspace` classes to access MLflow's tracking URI and configure your workspace.

In the following code, the `get_mlflow_tracking_uri()` method assigns a unique tracking URI address to the workspace, `ws`, and `set_tracking_uri()` points the MLflow tracking URI to that address.

```
import mlflow
from azureml.core import Workspace

ws = Workspace.from_config()

mlflow.set_tracking_uri(ws.get_mlflow_tracking_uri())
```

NOTE

The tracking URI is valid up to an hour or less. If you restart your script after some idle time, use the `get_mlflow_tracking_uri` API to get a new URI.

Set the MLflow experiment name with `set_experiment()` and start your training run with `start_run()`. Then use `log_metric()` to activate the MLflow logging API and begin logging your training run metrics.

```
experiment_name = 'experiment_with_mlflow'
mlflow.set_experiment(experiment_name)

with mlflow.start_run():
    mlflow.log_metric('alpha', 0.03)
```

Track remote runs

Remote runs let you train your models on more powerful computes, such as GPU enabled virtual machines, or Machine Learning Compute clusters. See [Use compute targets for model training](#) to learn about different compute options.

MLflow Tracking with Azure Machine Learning lets you store the logged metrics and artifacts from your remote runs into your Azure Machine Learning workspace. Any run with MLflow Tracking code in it will have metrics logged automatically to the workspace.

The following example conda environment includes `mlflow` and `azureml-mlflow` as pip packages.

```
name: sklearn-example
dependencies:
  - python=3.6.2
  - scikit-learn
  - matplotlib
  - numpy
  - pip:
    - azureml-mlflow
    - numpy
```

In your script, configure your compute and training run environment with the [Environment](#) class. Then, construct [ScriptRunConfig](#) with your remote compute as the compute target.

```
import mlflow

with mlflow.start_run():
    mlflow.log_metric('example', 1.23)
```

With this compute and training run configuration, use the `Experiment.submit()` method to submit a run. This method automatically sets the MLflow tracking URI and directs the logging from MLflow to your Workspace.

```
run = exp.submit(src)
```

Train with MLflow Projects

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 enables you to track and manage your training runs in your workspace.

This example shows how to submit MLflow projects locally with Azure Machine Learning tracking.

Install the `azureml-mlflow` package to use MLflow Tracking with Azure Machine Learning on your experiments locally. Your experiments can run via a Jupyter Notebook or code editor.

```
pip install azureml-mlflow
```

Import the `mlflow` and `Workspace` classes to access MLflow's tracking URI and configure your workspace.

```
import mlflow
from azureml.core import Workspace

ws = Workspace.from_config()

mlflow.set_tracking_uri(ws.get_mlflow_tracking_uri())
```

Set the MLflow experiment name with `set_experiment()` and start your training run with `start_run()`. Then, use `log_metric()` to activate the MLflow logging API and begin logging your training run metrics.

```
experiment_name = 'experiment-with-mlflow-projects'
mlflow.set_experiment(experiment_name)
```

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,
                                      )
```

View metrics and artifacts in your workspace

The metrics and artifacts from MLflow logging are kept 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.

```
run.get_metrics()
```

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 tagged with the registered model for traceability. Users can submit training runs, 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 run, use the following steps:

1. Once the run is complete call the `register_model()` method.

```
# the model folder produced from the run is registered. This includes the MLmodel file, model.pkl and
# the conda.yaml.
run.register_model(model_name = 'my-model', model_path = 'model')
```

2. 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

e2edemos > Models > my-model:1

my-model:1

Details Artifacts Endpoints Explanations (preview) Fairness (preview) Datasets

⟳ Refresh ➡ Deploy

Attributes

Version	1
ID	my-model:1
Date registered	9/2/2020, 2:15:28 PM
Description	--
Framework	Custom
Framework version	0.10
Experiment name	my-test-run
Run ID	example_15990
Created by	Contoso

Tags

model_uri: runs:/example_15990/model	python_version: 3.6.10
mlflow_run_id: example_15990	

Properties

No data

3. Select the **Artifacts** tab to see all the model files that align with the MLflow model schema (conda.yaml, MLmodel, model.pkl).

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 run.

my_model:1

Details

Artifacts

Endpoints

Explanations (preview)

Fairness (preview)

Datasets

⟳ Refresh ▷ Deploy

⌄ 📁 model
📄 conda.yaml
✔ 📁 MLmodel
🗄 model.pkl

```

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dypickle
11    sklearn_version: 0.23.2
12  run_id: example_15990
13  utc_time_created: '2020-09-02 16:54:26.734153'
14

```

Deploy and register MLflow models

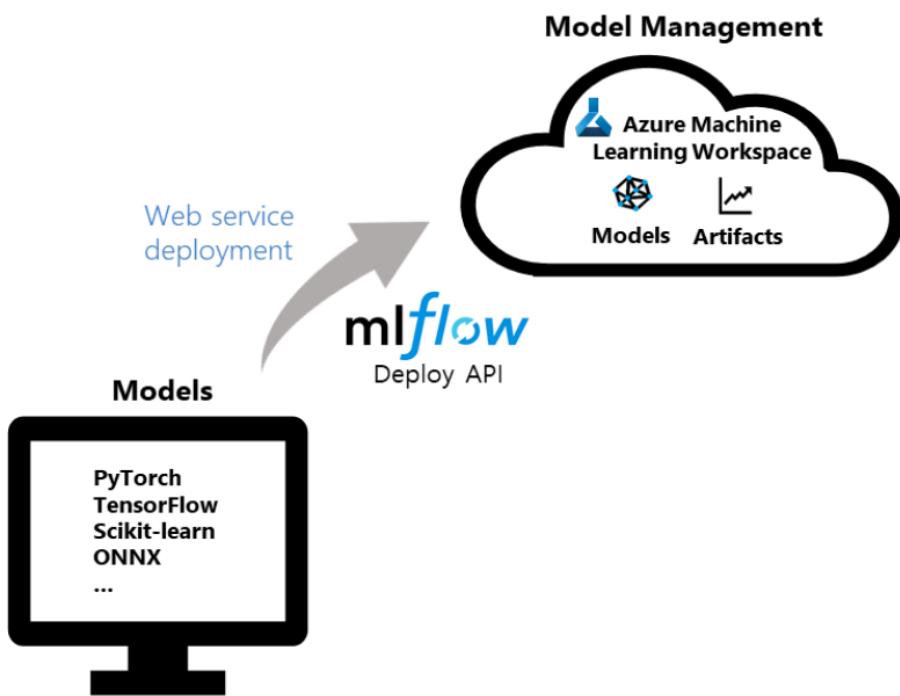
Deploying your MLflow experiments as an Azure Machine Learning web service allows you to leverage and apply the Azure Machine Learning model management and data drift detection capabilities to your production models.

To do so, you need to

1. Register your model.
2. Determine which deployment configuration you want to use for your scenario.
 - a. [Azure Container Instance \(ACI\)](#) is a suitable choice for a quick dev-test deployment.
 - b. [Azure Kubernetes Service \(AKS\)](#) is suitable for scalable production deployments.

The following diagram demonstrates that with the MLflow deploy API you can deploy your existing MLflow models as an Azure Machine Learning web service, despite their frameworks--PyTorch, Tensorflow, scikit-learn, ONNX, etc., and manage your production models in your workspace.

MLflow with Azure Machine Learning Deployment



Deploy to ACI

Set up your deployment configuration with the [deploy_configuration\(\)](#) method. You can also add tags and descriptions to help keep track of your web service.

```
from azureml.core.webservice import AciWebservice, Webservice

# Set the model path to the model folder created by your run
model_path = "model"

# Configure
aci_config = AciWebservice.deploy_configuration(cpu_cores=1,
                                                memory_gb=1,
                                                tags={'method' : 'sklearn'},
                                                description='Diabetes model',
                                                location='eastus2')
```

Then, register and deploy the model in one step with the Azure Machine Learning SDK [deploy](#) method.

```
(webservice,model) = mlflow.azureml.deploy( model_uri='runs:/{}{}'.format(run.id, model_path),
    workspace=ws,
    model_name='sklearn-model',
    service_name='diabetes-model-1',
    deployment_config=aci_config,
    tags=None, mlflow_home=None, synchronous=True)

webservice.wait_for_deployment(show_output=True)
```

Deploy to AKS

To deploy to AKS, first create an AKS cluster. Create an AKS cluster using the [ComputeTarget.create\(\)](#) method. It may take 20-25 minutes to create a new cluster.

```
from azureml.core.compute import AksCompute, ComputeTarget

# Use the default configuration (can also provide parameters to customize)
prov_config = AksCompute.provisioning_configuration()

aks_name = 'aks-mlflow'

# Create the cluster
aks_target = ComputeTarget.create(workspace=ws,
                                   name=aks_name,
                                   provisioning_configuration=prov_config)

aks_target.wait_for_completion(show_output = True)

print(aks_target.provisioning_state)
print(aks_target.provisioning_errors)
```

Set up your deployment configuration with the [deploy_configuration\(\)](#) method. You can also add tags and descriptions to help keep track of your web service.

```
from azureml.core.webservice import Webservice, AksWebservice

# Set the web service configuration (using default here with app insights)
aks_config = AksWebservice.deploy_configuration(enable_app_insights=True, compute_target_name='aks-mlflow')
```

Then, register and deploy the model by using the Azure Machine Learning SDK [deploy](#) method.

```
# Webservice creation using single command
from azureml.core.webservice import AksWebservice, Webservice

# set the model path
model_path = "model"

(webservice, model) = mlflow.azureml.deploy( model_uri='runs:/{}{}'.format(run.id, model_path),
    workspace=ws,
    model_name='sklearn-model',
    service_name='my-aks',
    deployment_config=aks_config,
    tags=None, mlflow_home=None, synchronous=True)

webservice.wait_for_deployment()
```

The service deployment can take several minutes.

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.

The screenshot shows the Microsoft Azure portal's 'Resource groups' blade. On the left sidebar, under 'FAVORITES', 'Resource groups' is selected and highlighted with a red box. The main area displays a table with one item: 'aml-get-started-rg'. A context menu is open over this item, with the 'Delete resource group' option highlighted by a red box. Other options visible in the menu include 'Pin to dashboard' 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 [MLflow with Azure ML notebooks](#) demonstrate and expand upon concepts presented in this article.

NOTE

A community-driven repository of examples using mlflow can be found at <https://github.com/Azure/azureml-examples>.

Next steps

- [Manage your models](#).
- Monitor your production models for [data drift](#).
- [Track Azure Databricks runs with MLflow](#).

Track Azure Databricks ML experiments with MLflow and Azure Machine Learning (preview)

12/23/2020 • 6 minutes to read • [Edit Online](#)

In this article, learn how to enable MLflow's tracking URI and logging API, collectively known as [MLflow Tracking](#), to connect your Azure Databricks (ADB) experiments, MLflow, and Azure Machine Learning.

[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 [Azure Databricks and MLflow](#).

See [Track experiment runs and create endpoints with MLflow and Azure Machine Learning](#) for additional MLflow and Azure Machine Learning functionality integrations.

NOTE

As an open source library, MLflow changes frequently. As such, the functionality made available via the Azure Machine Learning and MLflow integration should be considered as a preview, and not fully supported by Microsoft.

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.
 - This package automatically brings in `azureml-core` of the [The Azure Machine Learning Python SDK](#), which provides the connectivity for MLflow to access your workspace.
- An [Azure Databricks workspace and cluster](#).
- [Create an Azure Machine Learning Workspace](#).

Track Azure Databricks runs

MLflow Tracking with Azure Machine Learning lets you store the logged metrics and artifacts from your Azure Databricks runs into both your:

- Azure Databricks workspace.
- Azure Machine Learning workspace

After you create your Azure Databricks workspace and cluster,

1. Install the `azureml-mlflow` library from PyPi, to ensure that your cluster has access to the necessary functions and classes.
2. Set up your experiment notebook.
3. Connect your Azure Databricks workspace and Azure Machine Learning workspace.

Additional detail for these steps are in the following sections so you can successfully run your MLflow experiments with Azure Databricks.

Install libraries

To install libraries on your cluster, navigate to the **Libraries** tab and select **Install New**

The screenshot shows the 'Clusters / mlflow-demo' interface. At the top, there is a cluster icon and the name 'mlflow-demo'. Below it are four tabs: 'Configuration' (blue), 'Notebooks (0)', 'Libraries' (red box), and 'Event Log'. Underneath these tabs are two buttons: 'Uninstall' and 'Install New' (red box). The 'Install New' button has a magnifying glass icon and the text '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.

The screenshot shows the 'Install Library' dialog box. At the top left is the title 'Install Library' and a close button 'x'. Below it is a 'Library Source' section with tabs: 'Upload', 'DBFS', 'PyPI' (blue), 'Maven', 'CRAN', and 'Workspace'. The next section is 'Repository' with a note 'Optional'. The final section is 'Package' with the value 'azureml-mlflow'. At the bottom right are 'Cancel' and 'Install' buttons ('Install' is blue).

Set up your notebook

Once your ADB cluster is set up,

1. Select **Workspaces** on the left navigation pane.
2. Expand the workspaces drop down menu and select **Import**
3. Drag and drop, or browse to find, your experiment notebook to import your ADB workspace.
4. Select **Import**. Your experiment notebook opens automatically.
5. Under the notebook title on the top left, select the cluster want to attach to your experiment notebook.

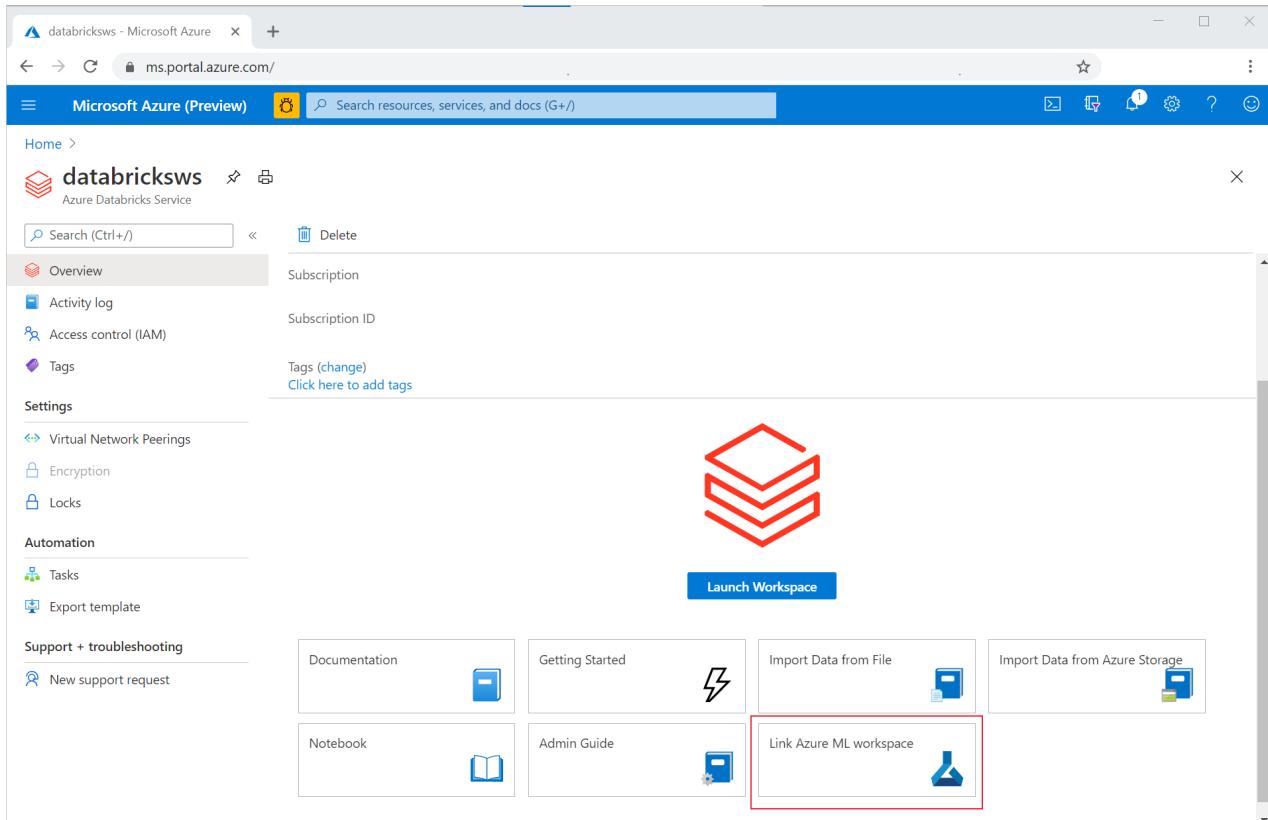
Connect your Azure Databricks and Azure Machine Learning

workspaces

Linking your ADB workspace to your Azure Machine Learning workspace enables you to track your experiment data in the Azure Machine Learning workspace.

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.



MLflow Tracking in your workspaces

After you instantiate your 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.

All your experiments land in the managed Azure Machine Learning tracking service.

The following code should be in your experiment notebook to get your linked Azure Machine Learning workspace.

This code,

- Gets the details of your Azure subscription to instantiate your Azure Machine Learning workspace.
- Assumes you have an existing resource group and Azure Machine Learning workspace, otherwise you can [create them](#).
- Sets the experiment name. The `user_name` here is consistent with the `user_name` associated with the Azure Databricks workspace.

```

import mlflow
import mlflow.azureml
import azureml.mlflow
import azureml.core

from azureml.core import Workspace

subscription_id = 'subscription_id'

# Azure Machine Learning resource group NOT the managed resource group
resource_group = 'resource_group_name'

#Azure Machine Learning workspace name, NOT Azure Databricks workspace
workspace_name = 'workspace_name'

# Instantiate Azure Machine Learning workspace
ws = Workspace.get(name=workspace_name,
                    subscription_id=subscription_id,
                    resource_group=resource_group)

#Set MLflow experiment.
experimentName = "/Users/{user_name}/{experiment_folder}/{experiment_name}"
mlflow.set_experiment(experimentName)

```

Set MLflow Tracking to only track in your 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.

Include the following code in your script:

```

uri = ws.get_mlflow_tracking_uri()
mlflow.set_tracking_uri(uri)

```

In your training script, import `mlflow` to use the MLflow logging APIs, and start logging your run metrics. The following example, logs the epoch loss metric.

```

import mlflow
mlflow.log_metric('epoch_loss', loss.item())

```

Register models with MLflow

After your model is trained, you can log and register your models to the backend 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.](#)

The backend tracking server is the Azure Databricks workspace by default; unless you chose to [set MLflow Tracking to only track in your Azure Machine Learning workspace](#), then the backend tracking server is the Azure Machine Learning workspace.

- **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.

```
mlflow.spark.log_model(model, artifact_path = "model",
                       registered_model_name = 'model_name')

mlflow.sklearn.log_model(model, artifact_path = "model",
                        registered_model_name = 'model_name')
```

Create endpoints for MLflow models

When you are ready to create an endpoint for your ML models. You can deploy as,

- An Azure Machine Learning Request-Response web service for interactive scoring. This deployment allows you to leverage and apply the Azure Machine Learning model management, and data drift detection capabilities to your production models.
- MLFlow model objects, which can be used in streaming or batch pipelines as Python functions or Pandas UDFs in Azure Databricks workspace.

Deploy models to Azure Machine Learning endpoints

You can leverage the [mlflow.azureml.deploy](#) API to deploy a model to your Azure Machine Learning workspace. If you only registered the model to the Azure Databricks workspace, as described in the [register models with MLflow](#) section, specify the `model_name` parameter to register the model into Azure Machine Learning workspace.

Azure Databricks runs can be deployed to the following endpoints,

- [Azure Container Instance](#)
- [Azure Kubernetes Service](#)

Deploy models to ADB endpoints for batch scoring

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 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.

The screenshot shows the Microsoft Azure Resource groups page. On the left sidebar, under 'FAVORITES', the 'Resource groups' item is highlighted with a red box. In the main content area, a single resource group named 'aml-get-started-rg' is listed. To the right of the list, there is a context menu with options: 'Pin to dashboard' (with a star icon), 'Delete resource group' (with a trash bin icon), and three vertical dots (...). The 'Delete resource group' option is also 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**.

Example notebooks

The [MLflow with Azure Machine Learning notebooks](#) demonstrate and expand upon concepts presented in this article.

Next steps

- [Manage your models](#).
- [Track experiment runs and create endpoints with MLflow and Azure Machine Learning](#).
- Learn more about [Azure Databricks](#) and [MLflow](#).

Train scikit-learn models at scale with Azure Machine Learning

12/23/2020 • 5 minutes to read • [Edit Online](#)

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

Run this code on either of these environments:

- Azure Machine Learning compute instance - no downloads or installation necessary
 - Complete the [Tutorial: Setup environment and workspace](#) to create a dedicated notebook server pre-loaded with the SDK and the sample repository.
 - In the samples training folder on the notebook server, 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.
- Your own Jupyter Notebook server
 - [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 [here](#). 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. For more info, see [here](#). 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,'train-iris').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

12/23/2020 • 10 minutes to read • [Edit Online](#)

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 [Tutorial: Setup environment and workspace](#) 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](#) ($\geq 1.15.0$).
 - [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.

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 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)
```

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.

```
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 [here](#).

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 will have to modify the environment to include the missing dependencies. Note that if the environment is modified, you will 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, e.g.:

```
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 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.

```
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/azurermi3.1.2-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='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.

Horovod

[Horovod](#) is an open-source, all reduce framework for distributed training developed by Uber. It offers an easy path to writing distributed TensorFlow code for training.

Your training code will have to be instrumented with Horovod for distributed training. For more information using Horovod with TensorFlow, refer to Horovod documentation:

For more information on using Horovod with TensorFlow, refer to Horovod documentation:

- [Horovod with TensorFlow](#)
- [Horovod with TensorFlow's Keras API](#)

Additionally, make sure your training environment includes the **horovod** package. If you are using a TensorFlow curated environment, horovod is already included as one of the dependencies. If you are using your own environment, make sure the horovod dependency is included, for example:

```
channels:
- conda-forge
dependencies:
- python=3.6.2
- pip:
  - azureml-defaults
  - tensorflow-gpu==2.2.0
  - horovod==0.19.5
```

In order to execute a distributed job using MPI/Horovod on Azure ML, you must specify an [MpiConfiguration](#) to the `distributed_job_config` parameter of the `ScriptRunConfig` constructor. The below code will configure a 2-node distributed job running one process per node. If you would also like to run multiple processes per node (i.e. if your cluster SKU has multiple GPUs), additionally specify the `process_count_per_node` parameter in `MpiConfiguration` (the default is `1`).

```
from azureml.core import ScriptRunConfig
from azureml.core.runconfig import MpiConfiguration

src = ScriptRunConfig(source_directory=project_folder,
                      script='tf_horovod_word2vec.py',
                      arguments=['--input_data', dataset.as_mount()],
                      compute_target=compute_target,
                      environment=tf_env,
                      distributed_job_config=MpiConfiguration(node_count=2))
```

For a full tutorial on running distributed TensorFlow with Horovod on Azure ML, see [Distributed TensorFlow with Horovod](#).

tf.distribute

If you are using [native distributed TensorFlow](#) in your training code, e.g. TensorFlow 2.x's `tf.distribute.Strategy` API, you can also launch the distributed job via Azure ML.

To do so, specify a `TensorflowConfiguration` to the `distributed_job_config` parameter of the `ScriptRunConfig` constructor. If you are using `tf.distribute.experimental.MultiWorkerMirroredStrategy`, specify the `worker_count` in the `TensorflowConfiguration` corresponding to the number of nodes for your training job.

```
import os
from azureml.core import ScriptRunConfig
from azureml.core.runconfig import TensorflowConfiguration

distr_config = TensorflowConfiguration(worker_count=2, parameter_server_count=0)

model_path = os.path.join("./outputs", "keras-model")

src = ScriptRunConfig(source_directory=source_dir,
                      script='train.py',
                      arguments=['--epochs', 30, '--model-dir', model_path],
                      compute_target=compute_target,
                      environment=tf_env,
                      distributed_job_config=distr_config)
```

In TensorFlow, the `TF_CONFIG` environment variable is required for training on multiple machines. 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 via `os.environ['TF_CONFIG']`.

Example structure of `TF_CONFIG` set on a chief worker node:

```
TF_CONFIG='{
  "cluster": {
    "worker": ["host0:2222", "host1:2222"]
  },
  "task": {"type": "worker", "index": 0},
  "environment": "cloud"
}'
```

If your training script uses the parameter server strategy for distributed training, i.e. for legacy TensorFlow 1.x, you will also need to specify the number of parameter servers to use in the job, e.g.

```
distr_config = TensorflowConfiguration(worker_count=2, parameter_server_count=1).
```

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 simply 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

12/23/2020 • 7 minutes to read • [Edit Online](#)

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 [Tutorial: Setup environment and workspace](#) 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.

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.

```
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='keras-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 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](#)
- [Deploy a trained model](#)
- [Reference architecture for distributed deep learning training in Azure](#)

Train PyTorch models at scale with Azure Machine Learning

12/23/2020 • 10 minutes to read • [Edit Online](#)

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. This short cuts the training process by requiring less data, time, and compute resources than training from scratch.

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 [Tutorial: Setup environment and workspace](#) 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](#) ($\geq 1.15.0$).
 - [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.

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 will 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](#).

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.

```
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 PyTorch corresponding to different versions of PyTorch. For more info, see [here](#).

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 will have to modify the environment to include the missing dependencies. Note that if the environment is modified, you will 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, e.g.:

```
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:  
- 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 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.

```
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.

```
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='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.

Horovod

[Horovod](#) is an open-source, all reduce framework for distributed training developed by Uber. It offers an easy path to writing distributed PyTorch code for training.

Your training code will have to be instrumented with Horovod for distributed training. For more information using Horovod with PyTorch, see the [Horovod documentation](#).

Additionally, make sure your training environment includes the `horovod` package. If you are using a PyTorch curated environment, horovod is already included as one of the dependencies. If you are using your own environment, make sure the horovod dependency is included, for example:

```
channels:
- conda-forge
dependencies:
- python=3.6.2
- pip:
  - azureml-defaults
  - torch==1.6.0
  - torchvision==0.7.0
  - horovod==0.19.5
```

In order to execute a distributed job using MPI/Horovod on Azure ML, you must specify an [MpiConfiguration](#) to the `distributed_job_config` parameter of the `ScriptRunConfig` constructor. The below code will configure a 2-node distributed job running one process per node. If you would also like to run multiple processes per node (i.e. if your cluster SKU has multiple GPUs), additionally specify the `process_count_per_node` parameter in `MpiConfiguration` (the default is `1`).

```
from azureml.core import ScriptRunConfig
from azureml.core.runconfig import MpiConfiguration

src = ScriptRunConfig(source_directory=project_folder,
                      script='pytorch_horovod_mnist.py',
                      compute_target=compute_target,
                      environment=pytorch_env,
                      distributed_job_config=MpiConfiguration(node_count=2))
```

For a full tutorial on running distributed PyTorch with Horovod on Azure ML, see [Distributed PyTorch with Horovod](#).

DistributedDataParallel

If you are using PyTorch's built-in [DistributedDataParallel](#) module that is built using the `torch.distributed` package in your training code, you can also launch the distributed job via Azure ML.

In order to run a distributed PyTorch job with `DistributedDataParallel`, specify a [PyTorchConfiguration](#) to the `distributed_job_config` parameter of the `ScriptRunConfig` constructor. To use the NCCL backend for `torch.distributed`, specify `communication_backend='Nccl'` in the `PyTorchConfiguration`. The below code will configure a 2-node distributed job. The NCCL backend is the recommended backend for PyTorch distributed GPU training.

For distributed PyTorch jobs configured via `PyTorchConfiguration`, Azure ML will set the following environment variables on the nodes of the compute target:

- `AZ_BATCHAI_PYTORCH_INIT_METHOD` : URL for the shared file system initialization of the process group
- `AZ_BATCHAI_TASK_INDEX` : global rank of the worker process

You can specify these environment variables to the corresponding arguments of the training script via the `arguments` parameter of `ScriptRunConfig`.

```

from azureml.core import ScriptRunConfig
from azureml.core.runconfig import PyTorchConfiguration

args = ['--dist-backend', 'nccl',
        '--dist-url', '$AZ_BATCHAI_PYTORCH_INIT_METHOD',
        '--rank', '$AZ_BATCHAI_TASK_INDEX',
        '--world-size', 2]

src = ScriptRunConfig(source_directory=project_folder,
                      script='pytorch_mnist.py',
                      arguments=args,
                      compute_target=compute_target,
                      environment=pytorch_env,
                      distributed_job_config=PyTorchConfiguration(communication_backend='Nccl', node_count=2))

```

If you would instead like to use the Gloo backend for distributed training, specify `communication_backend='Gloo'` instead. The Gloo backend is recommended for distributed CPU training.

For a full tutorial on running distributed PyTorch on Azure ML, see [Distributed PyTorch with DistributedDataParallel](#).

Troubleshooting

- **Horovod has been shut down:** In most cases, if you encounter "AbortedError: Horovod has been shut down", there was an underlying exception in one of the processes that caused Horovod to shut down. Each rank in the MPI job gets its own dedicated log file in Azure ML. These logs are named `70_driver_logs`. In case of distributed training, the log names are suffixed with `_rank` to make it easier to differentiate the logs. To find the exact error that caused Horovod to shut down, go through all the log files and look for `Traceback` at the end of the driver_log files. One of these files will give you the actual underlying exception.

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. See the [tutorial](#) for an example.

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](#)
- [Deploy a trained model](#)
- [Reference architecture for distributed deep learning training in Azure](#)

Train a model by using a custom Docker image

12/23/2020 • 4 minutes to read • [Edit Online](#)

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 [Set up environment and workspace](#) tutorial to create a dedicated notebook server preloaded with the SDK and the sample repository.
 - In the Azure Machine Learning [examples repository](#), find a completed notebook by going to the notebooks > fastai > train-pets-resnet34.ipynb directory.
- 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 and enable Docker.

```
from azureml.core import Environment  
  
fastai_env = Environment("fastai2")  
fastai_env.docker.enabled = True
```

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/base:intelmpi2018.3-ubuntu16.04
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"
```

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())

```

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,'fastai-custom-image').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.

Migrating from Estimators to ScriptRunConfig

12/23/2020 • 3 minutes to read • [Edit Online](#)

Up until now, there have been multiple methods for configuring a training job in Azure Machine Learning via the SDK, including Estimators, ScriptRunConfig, and the lower-level RunConfiguration. To address this ambiguity and inconsistency, we are simplifying the job configuration process in Azure ML. You should now use ScriptRunConfig as the recommended option for configuring training jobs.

Estimators are deprecated with the 1.19.0 release of the Python SDK. You should also generally avoid explicitly instantiating a RunConfiguration object yourself, and instead configure your job using the ScriptRunConfig class.

This article covers common considerations when migrating from Estimators to ScriptRunConfig.

IMPORTANT

To migrate to ScriptRunConfig from Estimators, make sure you are using >= 1.15.0 of the Python SDK.

ScriptRunConfig documentation and samples

Azure Machine Learning documentation and samples have been updated to use [ScriptRunConfig](#) for job configuration and submission.

For information on using ScriptRunConfig, refer to the following documentation:

- [Configure and submit training runs](#)
- [Configuring PyTorch training runs](#)
- [Configuring TensorFlow training runs](#)
- [Configuring scikit-learn training runs](#)

In addition, refer to the following samples & tutorials:

- [Azure/MachineLearningNotebooks](#)
- [Azure/azureml-examples](#)

Defining the training environment

While the various framework estimators have preconfigured environments that are backed by Docker images, the Dockerfiles for these images are private. Therefore you do not have a lot of transparency into what these environments contain. In addition, the estimators take in environment-related configurations as individual parameters (such as `pip_packages`, `custom_docker_image`) on their respective constructors.

When using ScriptRunConfig, all environment-related configurations are encapsulated in the `Environment` object that gets passed into the `environment` parameter of the ScriptRunConfig constructor. To configure a training job, provide an environment that has all the dependencies required for your training script. If no environment is provided, Azure ML will use one of the Azure ML base images, specifically the one defined by `azureml.core.environment.DEFAULT_CPU_IMAGE`, as the default environment. There are a couple of ways to provide an environment:

- [Use a curated environment](#) - curated environments are predefined environments available in your workspace by default. There is a corresponding curated environment for each of the preconfigured framework/version Docker images that backed each framework estimator.

- [Define your own custom environment](#)

Here is an example of using the curated PyTorch 1.6 environment for training:

```
from azureml.core import Workspace, ScriptRunConfig, Environment

curated_env_name = 'AzureML-PyTorch-1.6-GPU'
pytorch_env = Environment.get(workspace=ws, name=curated_env_name)

compute_target = ws.compute_targets['my-cluster']
src = ScriptRunConfig(source_directory='.',
                      script='train.py',
                      compute_target=compute_target,
                      environment=pytorch_env)
```

If you want to specify **environment variables** that will get set on the process where the training script is executed, use the Environment object:

```
myenv.environment_variables = {"MESSAGE": "Hello from Azure Machine Learning"}
```

For information on configuring and managing Azure ML environments, see:

- [How to use environments](#)
- [Curated environments](#)
- [Train with a custom Docker image](#)

Using data for training

Datasets

If you are using an Azure ML dataset for training, pass the dataset as an argument to your script using the `arguments` parameter. By doing so, you will get the data path (mounting point or download path) in your training script via arguments.

The following example configures a training job where the FileDataset, `mnist_ds`, will get mounted on the remote compute.

```
src = ScriptRunConfig(source_directory='.',
                      script='train.py',
                      arguments=['--data-folder', mnist_ds.as_mount()], # or mnist_ds.as_download() to
download
                      compute_target=compute_target,
                      environment=pytorch_env)
```

DataReference (old)

While we recommend using Azure ML Datasets over the old DataReference way, if you are still using DataReferences for any reason, you must configure your job as follows:

```
# if you want to pass a DataReference object, such as the below:  
datastore = ws.get_default_datastore()  
data_ref = datastore.path('./foo').as_mount()  
  
src = ScriptRunConfig(source_directory='.',  
                      script='train.py',  
                      arguments=['--data-folder', str(data_ref)], # cast the DataReference object to str  
                      compute_target=compute_target,  
                      environment=pytorch_env)  
src.run_config.data_references = {data_ref.data_reference_name: data_ref.to_config()} # set a dict of the  
DataReference(s) you want to the `data_references` attribute of the ScriptRunConfig's underlying  
RunConfiguration object.
```

For more information on using data for training, see:

- [Train with datasets in Azure ML](#)

Distributed training

If you need to configure a distributed job for training, do so by specifying the `distributed_job_config` parameter in the `ScriptRunConfig` constructor. Pass in an [Mpiconfiguration](#), [PyTorchConfiguration](#), or [TensorflowConfiguration](#) for distributed jobs of the respective types.

The following example configures a PyTorch training job to use distributed training with MPI/Horovod:

```
from azureml.core.runconfig import Mpiconfiguration  
  
src = ScriptRunConfig(source_directory='.',  
                      script='train.py',  
                      compute_target=compute_target,  
                      environment=pytorch_env,  
                      distributed_job_config=Mpiconfiguration(node_count=2, process_count_per_node=2))
```

For more information, see:

- [Distributed training with PyTorch](#)
- [Distributed training with TensorFlow](#)

Miscellaneous

If you need to access the underlying `RunConfiguration` object for a `ScriptRunConfig` for any reason, you can do so as follows:

```
src.run_config
```

Next steps

- [Configure and submit training runs](#)

Start, monitor, and cancel training runs in Python

12/23/2020 • 10 minutes to read • [Edit Online](#)

The [Azure Machine Learning SDK for Python](#), [Machine Learning CLI](#), and [Azure Machine Learning studio](#) provide various methods to monitor, organize, and manage your runs for training and experimentation.

This article shows examples of the following tasks:

- Monitor run performance.
- Cancel or fail runs.
- Create child runs.
- Tag and find runs.

TIP

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 as web services or IoT Edge modules, see [Collect model data](#) and [Monitor with Application Insights](#).

Prerequisites

You'll need the following items:

- 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](#).
- The Azure Machine Learning SDK for Python (version 1.0.21 or later). To install or update to the latest version of the SDK, see [Install or update the SDK](#).

To check your version of the Azure Machine Learning SDK, use the following code:

```
print(azureml.core.VERSION)
```

- The [Azure CLI](#) and [CLI extension for Azure Machine Learning](#).

Monitor run performance

- Start a run and its logging process
 - [Python](#)
 - [Azure CLI](#)
 - [Studio](#)
1. Set up your experiment by importing the [Workspace](#), [Experiment](#), [Run](#), and [ScriptRunConfig](#) classes from the [azureml.core](#) package.

```
import azureml.core
from azureml.core import Workspace, Experiment, Run
from azureml.core import ScriptRunConfig

ws = Workspace.from_config()
exp = Experiment(workspace=ws, name="explore-runs")
```

2. Start a run and its logging process with the `start_logging()` method.

```
notebook_run = exp.start_logging()
notebook_run.log(name="message", value="Hello from run!")
```

- Monitor the status of a run

- Python
- Azure CLI
- Studio
- Get the status of a run with the `get_status()` method.

```
print(notebook_run.get_status())
```

- To get the run ID, execution time, and additional details about the run, use the `get_details()` method.

```
print(notebook_run.get_details())
```

- When your run finishes successfully, use the `complete()` method to mark it as completed.

```
notebook_run.complete()
print(notebook_run.get_status())
```

- If you use Python's `with...as` design pattern, the run will automatically mark itself as completed when the run is out of scope. You don't need to manually mark the run as completed.

```
with exp.start_logging() as notebook_run:
    notebook_run.log(name="message", value="Hello from run!")
    print(notebook_run.get_status())

print(notebook_run.get_status())
```

Cancel or fail runs

If you notice a mistake or if your run is taking too long to finish, you can cancel the run.

- Python
- Azure CLI
- Studio

To cancel a run using the SDK, use the `cancel()` method:

```
src = ScriptRunConfig(source_directory='.', script='hello_with_delay.py')
local_run = exp.submit(src)
print(local_run.get_status())

local_run.cancel()
print(local_run.get_status())
```

If your run finishes, but it contains an error (for example, the incorrect training script was used), you can use the `fail()` method to mark it as failed.

```
local_run = exp.submit(src)
local_run.fail()
print(local_run.get_status())
```

Create child runs

Create child runs to group together related runs, such as for different hyperparameter-tuning iterations.

NOTE

Child runs can only be created using the SDK.

This code example uses the `hello_with_children.py` script to create a batch of five child runs from within a submitted run by using the `child_run()` method:

```
!more hello_with_children.py
src = ScriptRunConfig(source_directory='.', script='hello_with_children.py')

local_run = exp.submit(src)
local_run.wait_for_completion(show_output=True)
print(local_run.get_status())

with exp.start_logging() as parent_run:
    for c,count in enumerate(range(5)):
        with parent_run.child_run() as child:
            child.log(name="Hello from child run", value=c)
```

NOTE

As they move out of scope, child runs are automatically marked as completed.

To create many child runs efficiently, use the `create_children()` method. Because each creation results in a network call, creating a batch of runs is more efficient than creating them one by one.

Submit child runs

Child runs can also be submitted from a parent run. This allows you to create hierarchies of parent and child runs. You cannot create a parentless child run: even if the parent run does nothing but launch child runs, it's still necessary to create the hierarchy. The status of all runs are independent: a parent can be in the "Completed" successful state even if one or more child runs were canceled or failed.

You may wish your child runs to use a different run configuration than the parent run. For instance, you might use a less-powerful, CPU-based configuration for the parent, while using GPU-based configurations for your children. Another common desire is to pass each child different arguments and data. To customize a child run, create a `ScriptRunConfig` object for the child run. The below code does the following:

- Retrieve a compute resource named "gpu-cluster" from the workspace `ws`
- Iterates over different argument values to be passed to the children `ScriptRunConfig` objects
- Creates and submits a new child run, using the custom compute resource and argument
- Blocks until all of the child runs complete

```
# parent.py
# This script controls the launching of child scripts
from azureml.core import Run, ScriptRunConfig

compute_target = ws.compute_targets["gpu-cluster"]

run = Run.get_context()

child_args = ['Apple', 'Banana', 'Orange']
for arg in child_args:
    run.log('Status', f'Launching {arg}')
    child_config = ScriptRunConfig(source_directory=".", script='child.py', arguments=['--fruit', arg],
compute_target=compute_target)
    # Starts the run asynchronously
    run.submit_child(child_config)

# Experiment will "complete" successfully at this point.
# Instead of returning immediately, block until child runs complete

for child in run.get_children():
    child.wait_for_completion()
```

To create many child runs with identical configurations, arguments, and inputs efficiently, use the `create_children()` method. Because each creation results in a network call, creating a batch of runs is more efficient than creating them one by one.

Within a child run, you can view the parent run ID:

```
## In child run script
child_run = Run.get_context()
child_run.parent.id
```

Query child runs

To query the child runs of a specific parent, use the `get_children()` method. The `recursive = True` argument allows you to query a nested tree of children and grandchildren.

```
print(parent_run.get_children())
```

Log to parent or root run

You can use the `Run.parent` field to access the run that launched the current child run. A common use-case for this is when you wish to consolidate log results in a single place. Note that child runs execute asynchronously and there is no guarantee of ordering or synchronization beyond the ability of the parent to wait for its child runs to complete.

```

# in child (or even grandchild) run

def root_run(self : Run) -> Run :
    if self.parent is None :
        return self
    return root_run(self.parent)

current_child_run = Run.get_context()
root_run(current_child_run).log("MyMetric", f"Data from child run {current_child_run.id}")

```

Tag and find runs

In Azure Machine Learning, you can use properties and tags to help organize and query your runs for important information.

- Add properties and tags

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

To add searchable metadata to your runs, use the [add_properties\(\)](#) method. For example, the following code adds the "author" property to the run:

```

local_run.add_properties({"author": "azureml-user"})
print(local_run.get_properties())

```

Properties are immutable, so they create a permanent record for auditing purposes. The following code example results in an error, because we already added "azureml-user" as the "author" property value in the preceding code:

```

try:
    local_run.add_properties({"author": "different-user"})
except Exception as e:
    print(e)

```

Unlike properties, tags are mutable. To add searchable and meaningful information for consumers of your experiment, use the [tag\(\)](#) method.

```

local_run.tag("quality", "great run")
print(local_run.get_tags())

local_run.tag("quality", "fantastic run")
print(local_run.get_tags())

```

You can also add simple string tags. When these tags appear in the tag dictionary as keys, they have a value of `None`.

```

local_run.tag("worth another look")
print(local_run.get_tags())

```

- Query properties and tags

You can query runs within an experiment to return a list of runs that match specific properties and tags.

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

```
list(exp.get_runs(properties={"author":"azureml-user"},tags={"quality":"fantastic run"}))
list(exp.get_runs(properties={"author":"azureml-user"},tags="worth another look"))
```

Example notebooks

The following notebooks demonstrate the concepts in this article:

- To learn more about the logging APIs, see the [logging API notebook](#).
- For more information about managing runs with the Azure Machine Learning SDK, see the [manage runs notebook](#).

Next steps

- To learn how to log metrics for your experiments, see [Log metrics during training runs](#).
- To learn how to monitor resources and logs from Azure Machine Learning, see [Monitoring Azure Machine Learning](#).

Enable logging in ML training runs

12/23/2020 • 2 minutes to read • [Edit Online](#)

The Azure Machine Learning Python SDK lets you log real-time information using both the default Python logging package and SDK-specific functionality. You can log locally and send logs to your workspace in the portal.

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:

- Interactive training sessions
- Submitting training jobs using `ScriptRunConfig`
- Python native `logging` settings
- Logging from additional sources

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 runs, or completed model deployments, see [Monitoring Azure Machine Learning](#).

Data types

You can log multiple data types including scalar values, lists, tables, images, directories, and more. For more information, and Python code examples for different data types, see the [Run class reference page](#).

Interactive logging session

Interactive logging sessions are typically used in notebook environments. The method `Experiment.start_logging()` starts an interactive logging session. Any metrics logged during the session are added to the run record in the experiment. The method `run.complete()` ends the sessions and marks the run as completed.

ScriptRun logs

In this section, you learn how to add logging code inside of runs created when configured with `ScriptRunConfig`. You can use the `ScriptRunConfig` class to encapsulate scripts and environments for repeatable runs. You can also use this option to show a visual Jupyter Notebooks widget for monitoring.

This example performs a parameter sweep over alpha values and captures the results using the `run.log()` method.

1. Create a training script that includes the logging logic, `train.py`.

```
!code-python
```

2. Submit the `train.py` script to run in a user-managed environment. The entire script folder is submitted for training.

```
!notebook-python !notebook-python
```

The `show_output` parameter turns on verbose logging, which lets you see details from the training

process as well as information about any remote resources or compute targets. Use the following code to turn on verbose logging when you submit the experiment.

```
run = exp.submit(src, show_output=True)
```

You can also use the same parameter in the `wait_for_completion` function on the resulting run.

```
run.wait_for_completion(show_output=True)
```

Native Python logging

Some logs in the SDK may contain an error that instructs you to set the logging level to DEBUG. To set the logging level, add the following code to your script.

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

Additional logging sources

Azure Machine Learning can also log information from other sources during training, such as automated machine learning runs, or Docker containers that run the jobs. These logs aren't documented, but if you encounter problems and contact Microsoft support, they may be able to use these logs during troubleshooting.

For information on logging metrics in Azure Machine Learning designer, see [How to log metrics in the designer](#)

Example notebooks

The following notebooks demonstrate concepts in this article:

- [how-to-use-azureml/training/train-on-local](#)
- [how-to-use-azureml/track-and-monitor-experiments/logging-api](#)

[!INCLUDE aml-clone-in-azure-notebook](#)

Next steps

See these articles to learn more on how to use Azure Machine Learning:

- Learn how to [log metrics in Azure Machine Learning designer](#).
- See an example of how to register the best model and deploy it in the tutorial, [Train an image classification model with Azure Machine Learning](#).

Monitor and view ML run logs and metrics

12/23/2020 • 6 minutes to read • [Edit Online](#)

Learn how to monitor Azure Machine Learning runs and view their logs.

When you run an experiment, logs and metrics are streamed for you. In addition, you can add your own. To learn how, see [Enable logging in Azure ML training runs](#).

The logs can help you diagnose errors and warnings for your run. Performance metrics like parameters and model accuracy help you track and monitor your runs.

In this article, you learn how to view logs using the following methods:

- Monitor runs in the studio
- Monitor runs using the Jupyter Notebook widget
- Monitor automated machine learning runs
- View output logs upon completion
- View output logs in the studio

For general information on how to manage your experiments, see [Start, monitor, and cancel training runs](#).

Monitor runs using the Jupyter notebook widget

When you use the `ScriptRunConfig` method to submit runs, you can watch the progress of the run using the [Jupyter widget](#). Like the run submission, the widget is asynchronous and provides live updates every 10-15 seconds until the job completes.

View the Jupyter widget while waiting for the run to complete.

```
from azureml.widgets import RunDetails  
RunDetails(run).show()
```

```

1 from azureml.train.widgets import RunDetails
2 RunDetails(run).show()

```

Run Properties	
Status	Running
Start Time	9/15/2018 7:15:37 PM
Duration	0:00:20
Run Id	train-on-local_1537053337_839d0780
Arguments	N/A

Output Logs

```

Uploading experiment status to history service.
Adding run profile attachment azureml-logs/80_driver_log.txt

alpha is 0.00, and mse is 3424.32
alpha is 0.05, and mse is 3408.92
alpha is 0.10, and mse is 3372.65
alpha is 0.15, and mse is 3345.15
alpha is 0.20, and mse is 3325.29
alpha is 0.25, and mse is 3311.56
alpha is 0.30, and mse is 3302.67

```

The figure consists of two side-by-side line charts. The left chart, titled 'alpha', plots a variable against time, showing a linear increase from 0 to approximately 0.35. The right chart, titled 'mse', plots Mean Squared Error (mse) against time, showing a non-linear decrease from approximately 3424 to 3302.

Time	alpha
0	0.00
1	0.05
2	0.10
3	0.15
4	0.20
5	0.25
6	0.30
7	0.35

Time	mse
0	3424.32
1	3408.92
2	3372.65
3	3345.15
4	3325.29
5	3311.56
6	3302.67

[Click here to see the run in Azure portal](#)

You can also get a link to the same display in your workspace.

```
print(run.get_portal_url())
```

Monitor automated machine learning runs

For automated machine learning runs, to access the charts from a previous run, replace `<<experiment_name>>` with the appropriate experiment name:

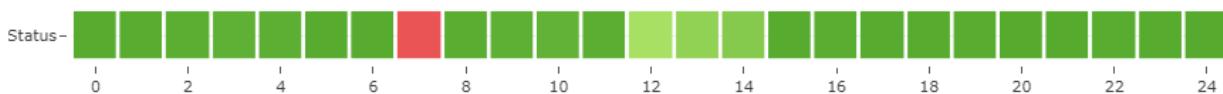
```

from azureml.widgets import RunDetails
from azureml.core.run import Run

experiment = Experiment (workspace, <<experiment_name>>)
run_id = 'autoML_my_runID' #replace with run_ID
run = Run(experiment, run_id)
RunDetails(run).show()

```

AutoML_ed181129-3876-452a-82b0-39f33a8290b7:
Status: **Completed**

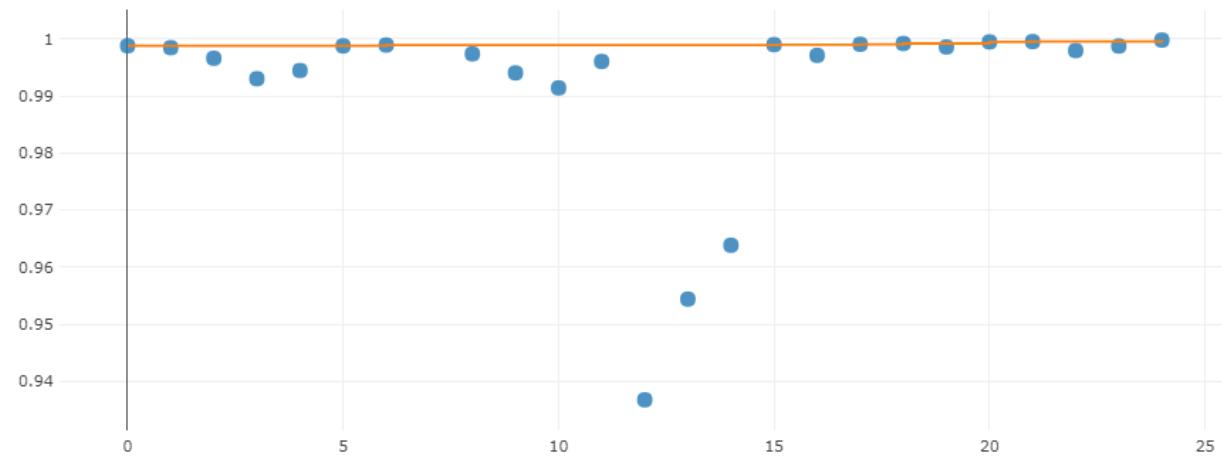


Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started
0	StandardScalerWrapper, KNN	0.99883057	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
1	StandardScalerWrapper, KNN	0.99849239	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
2	MaxAbsScaler, LightGBM	0.99664006	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
3	StandardScalerWrapper, LightGBM	0.9930566	0.99883057	Completed	0:00:00	Nov 26, 2018 1:31 PM
4	StandardScalerWrapper, LogisticRegression	0.9944965	0.99883057	Completed	0:00:00	Nov 26, 2018 1:31 PM

Pages: 1 2 3 4 5 Next Last 5 per page

AUC_weighted

Run with metric : AUC_weighted



[Click here to see the run in Azure portal](#)

Show output upon completion

When you use `ScriptRunConfig`, you can use `run.wait_for_completion(show_output = True)` to show when the model training is complete. The `show_output` flag gives you verbose output. For more information, see the `ScriptRunConfig` section of [How to enable logging](#).

Query run metrics

You can view the metrics of a trained model using `run.get_metrics()`. For example, you could use this with the example above to determine the best model by looking for the model with the lowest mean square error (`mse`) value.

View run records in the studio

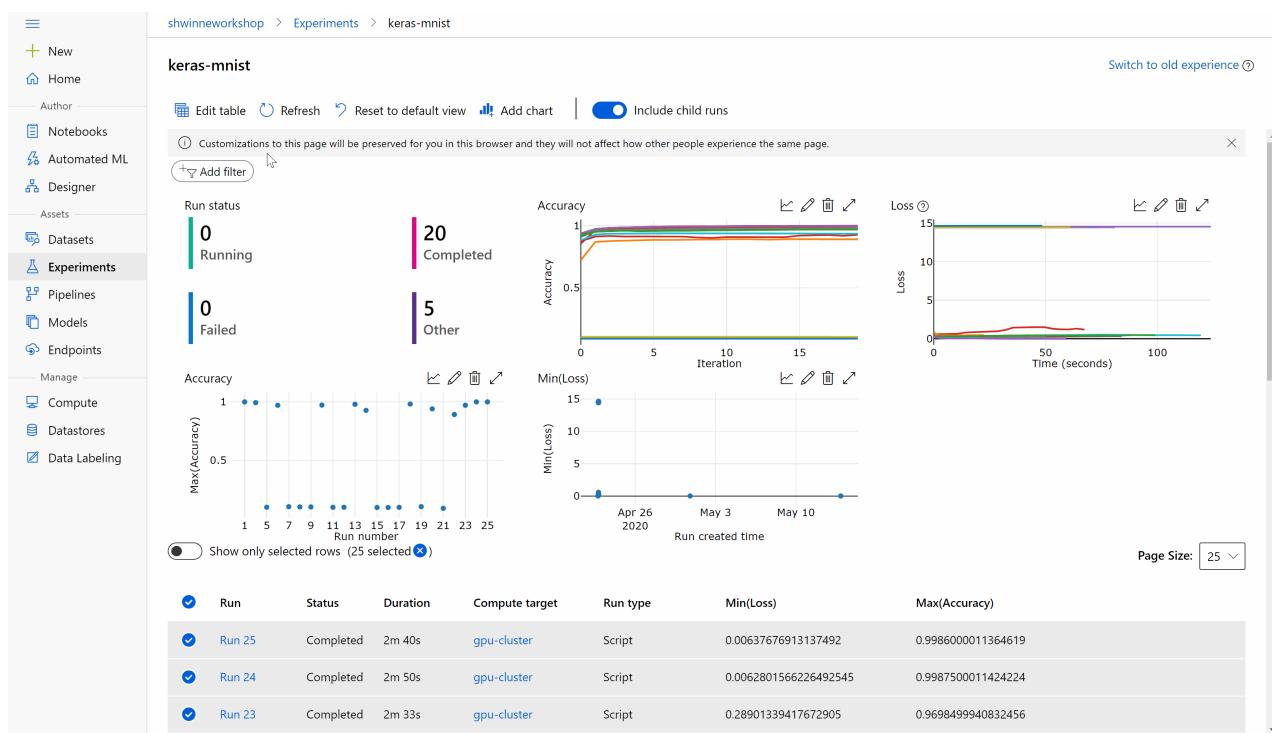
You can browse completed run records, including logged metrics, in the [Azure Machine Learning studio](#).

Navigate to the **Experiments** tab. To view all your runs in your Workspace across Experiments, select the **All runs** tab. You can drill down on runs for specific Experiments by applying the Experiment filter in the top menu bar.

For the individual Experiment view, select the **All experiments** tab. On the experiment run dashboard, you can see tracked metrics and logs for each run.

You can also edit the run list table to select multiple runs and display either the last, minimum, or maximum logged

value for your runs. Customize your charts to compare the logged metrics values and aggregates across multiple runs.



Format charts

Use the following methods in the logging APIs to influence the metrics visualizations.

LOGGED VALUE	EXAMPLE CODE	FORMAT IN PORTAL
Log an array of numeric values	<pre>run.log_list(name='Fibonacci', value=[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89])</pre>	single-variable line chart
Log a single numeric value with the same metric name repeatedly used (like from within a for loop)	<pre>for i in tqdm(range(-10, 10)): run.log(name='Sigmoid', value=1 / (1 + np.exp(-i))) angle = i / 2.0</pre>	Single-variable line chart
Log a row with 2 numerical columns repeatedly	<pre>run.log_row(name='Cosine Wave', angle=angle, cos=np.cos(angle)) sines['angle'].append(angle) sines['sine'].append(np.sin(angle))</pre>	Two-variable line chart
Log table with 2 numerical columns	<pre>run.log_table(name='Sine Wave', value=sines)</pre>	Two-variable line chart

View log files for a run

Log files are an essential resource for debugging the Azure ML workloads. Drill down to a specific run to view its logs and outputs:

1. Navigate to the **Experiments** tab.
2. Select the runID for a specific run.
3. Select **Outputs and logs** at the top of the page.

The screenshot shows the Microsoft Azure Machine Learning interface. On the left, there's a sidebar with various options like New, Home, Notebooks, Automated ML, Designer, Assets, Datasets, Pipelines, Models, Endpoints, Compute, Datastores, and Data Labeling. The 'Experiments' option is highlighted with a red box. The main area shows a 'Run 1' experiment under 'sklearn-mnist > Run 1'. At the top, there are buttons for Refresh, Resubmit, Cancel, Delete, and a toggle for 'Enable log streaming'. Below that is a navigation bar with Details, Metrics, Images, Child runs, Outputs + logs (which is selected and highlighted with a red box), Snapshot, Explanations (preview), and Fairness (preview). A search bar is above the file list. The file list shows several log files: 20_image_build_log.txt, 55_azureml-execution-tvmps_5990bd7506e8d00ab..., 65_job_prep-tvmps_5990bd7506e8d00abd88a939c..., 70_driver_log.txt (selected and highlighted with a blue border), 75_job_post-tvmps_5990bd7506e8d00abd88a939c..., process_info.json, and process_status.json. Under logs, there are folders for azureml and outputs. The right pane shows the contents of the selected log file, 70_driver_log.txt, with lines numbered 42 to 67.

```

42 Current directory: /mnt/batch/tasks/shared/LS_root/
43 Preparing to call script [ train.py ] with arguments
44 After variable expansion, calling script [ train.py
45
46 Data folder: /tmp/tmpbjg82tun
47 (6000, 784)
48 (6000,)
49 (10000, 784)
50 (10000,)
51 Train a logistic regression model with regularization
52 Predict the test set
53 Accuracy is 0.9193
54 Starting the daemon thread to refresh tokens in background
55
56
57 [2020-11-19T19:33:44.040330] The experiment completed
58 Cleaning up all outstanding Run operations, waiting
59 2 items cleaning up...
60 Cleanup took 5.483366250991821 seconds
61 Enter __exit__ of DatasetContextManager
62 Unmounting /tmp/tmpbjg82tun.
63 Finishing unmounting /tmp/tmpbjg82tun.
64 Exit __exit__ of DatasetContextManager
65 [2020-11-19T19:33:50.354718] Finished context manager
66 2020/11/19 19:33:56 Attempt 1 of http call to http://
67 2020/11/19 19:33:56 Process Exiting with Code: 0

```

The tables below show the contents of the log files in the folders you'll see in this section.

NOTE

Information the user should notice even if skimming. You will not necessarily see every file for every run. For example, the 20_image_build_log*.txt only appears when a new image is built (e.g. when you change your environment).

FILE	DESCRIPTION
20_image_build_log.txt	Docker image building log for the training environment, optional, one per run. Only applicable when updating your Environment. Otherwise AML will reuse cached image. If successful, contains image registry details for the corresponding image.
55_azureml-execution-<node_id>.txt	stdout/stderr log of host tool, one per node. Pulls image to compute target. Note, this log only appears once you have secured compute resources.
65_job_prep-<node_id>.txt	stdout/stderr log of job preparation script, one per node. Download your code to compute target and datastores (if requested).
70_driver_log(_x).txt	stdout/stderr log from AML control script and customer training script, one per process. This is the standard output from your script. This is where your code's logs (e.g. print statements) show up. In the majority of cases you will monitor the logs here.

FILE	DESCRIPTION
70_mpi_log.txt	MPI framework log, optional, one per run. Only for MPI run.
75_job_post-<node_id>.txt	stdout/stderr log of job release script, one per node. Send logs, release the compute resources back to Azure.
process_info.json	show which process is running on which node.
process_status.json	show process status, i.e. if a process is not started, running or completed.

logs > azureml **folder**

FILE	DESCRIPTION
110_azureml.log	
job_prep_azureml.log	system log for job preparation
job_release_azureml.log	system log for job release

logs > azureml > sidecar > node_id **folder**

When sidecar is enabled, job prep and job release scripts will be run within sidecar container. There is one folder for each node.

FILE	DESCRIPTION
start_cmds.txt	Log of process that starts when Sidecar Container starts
prep_cmd.txt	Log for ContextManagers entered when <code>job_prep.py</code> is run (some of this will be streamed to <code>azureml-logs/65-job_prep</code>)
release_cmd.txt	Log for ContextManagers exited when <code>job_release.py</code> is run

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 is one additional logs folder for overall execution, stderr, and stdout logs.

Azure Machine Learning logs information from a variety of sources during training, such as AutoML or the Docker container that runs the training job. Many of these logs are not documented. If you encounter problems and contact Microsoft support, they may be able to use these logs during troubleshooting.

Monitor a compute cluster

To monitor runs for a specific compute target from your browser, use the following steps:

1. In the [Azure Machine Learning studio](#), select your workspace, and then select **Compute** from the left side of the page.
2. Select **Training Clusters** to display a list of compute targets used for training. Then select the cluster.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a sidebar with various options like 'New', 'Home', 'Notebooks', etc. The 'Compute' option is selected and highlighted with a red box. In the main area, under 'Compute', the 'Training Clusters' tab is selected (also highlighted with a red box). A table lists a single entry: 'Name: training-cluster', 'Type: Machine Learning Com...', 'Provisioning state: Succeeded (2 nodes)', and 'Created on: December 4, 2019 2:29 PM'. Below the table are navigation links: '< Prev' and 'Next >'.

3. Select **Runs**. The list of runs that use this cluster is displayed. To view details for a specific run, use the link in the **Run** column. To view details for the experiment, use the link in the **Experiment** column.

The screenshot shows the 'Compute Details' page for the 'training-cluster'. The 'Runs' tab is selected (highlighted with a red box). A table lists two runs: one with 'Status: Running', 'Submitted: December 5, 2019 8:52 AM', 'Start time: December 5, 2019 8:57 AM', 'Created by: Larry Franks', 'Run: 5', 'Experiment: estimator-test', and 'Tags: _aml_system_ComputeTarget...'; and another with 'Status: Running', 'Submitted: December 5, 2019 8:52 AM', 'Start time: December 5, 2019 8:56 AM', 'Created by: Larry Franks', 'Run: 2', 'Experiment: estimator-test', and 'Tags: _aml_system_ComputeTarget...'. There is a search bar at the top right labeled 'Search to filter items...' and navigation links '< Prev' and 'Next >' at the bottom.

TIP

Since training compute targets are a shared resource, they can have multiple runs queued or active at a given time.

A run can contain child runs, so one training job can result in multiple entries.

Once a run completes, it is no longer displayed on this page. To view information on completed runs, visit the **Experiments** section of the studio and select the experiment and run. For more information, see the section [View metrics for completed runs](#).

Next steps

Try these next steps to learn how to use Azure Machine Learning:

- Learn how to [track experiments and enable logs in the Azure Machine Learning designer](#).
- See an example of how to register the best model and deploy it in the tutorial, [Train an image classification model with Azure Machine Learning](#).

Visualize experiment runs and metrics with TensorBoard and Azure Machine Learning

12/23/2020 • 7 minutes to read • [Edit Online](#)

In this article, you learn how to view your experiment runs and metrics in TensorBoard using the [tensorboard package](#) in the main Azure Machine Learning SDK. Once you've inspected your experiment runs, 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 run 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 run 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 runs, or completed model deployments, see [Monitoring Azure Machine Learning](#).

Prerequisites

- To launch TensorBoard and view your experiment run 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 [Tutorial: Setup environment and workspace](#) 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 run 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 run 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 run 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 run 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 run 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 run 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 run

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 run history loaded in the `run`, and then launches TensorBoard with the `start()` method.

The [TensorBoard constructor](#) takes an array of runs, so be sure and pass it in as a single-element array.

```

from azureml.tensorboard import Tensorboard

tb = Tensorboard([run])

# 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 run history APIs, and exports the experiment run history into logs consumable by TensorBoard for visualization.

Set up experiment

The following code sets up a new experiment and names the run directory `root_run`.

```

from azureml.core import Workspace, Experiment
import azureml.core

# set experiment name and run 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 runs to TensorBoard

With the SDK's `export_to_tensorboard()` method, we can export the run 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 run history and logs from `root_run` and then mark that run 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 run 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 run 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 runs, 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 run 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](#).

Use the interpretability package to explain ML models & predictions in Python (preview)

12/23/2020 • 15 minutes to read • [Edit Online](#)

In this how-to guide, you learn to use the interpretability package of the Azure Machine Learning Python SDK to perform the following tasks:

- Explain the entire model behavior or individual predictions on your personal machine locally.
- Enable interpretability techniques for engineered features.
- Explain the behavior for the entire model and individual predictions in Azure.
- Use a visualization dashboard to interact with your model explanations.
- Deploy a scoring explainer alongside your model to observe explanations during inferencing.

For more information on the supported interpretability techniques and machine learning models, see [Model interpretability in Azure Machine Learning](#) and [sample notebooks](#).

Generate feature importance value on your personal machine

The following example shows how to use the interpretability package on your personal machine without contacting Azure services.

1. Install the `azureml-interpret` package.

```
pip install azureml-interpret
```

2. Train a sample model in a local Jupyter Notebook.

```
# load breast cancer dataset, a well-known small dataset that comes with scikit-learn
from sklearn.datasets import load_breast_cancer
from sklearn import svm
from sklearn.model_selection import train_test_split
breast_cancer_data = load_breast_cancer()
classes = breast_cancer_data.target_names.tolist()

# split data into train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(breast_cancer_data.data,
                                                    breast_cancer_data.target,
                                                    test_size=0.2,
                                                    random_state=0)
clf = svm.SVC(gamma=0.001, C=100., probability=True)
model = clf.fit(x_train, y_train)
```

3. Call the explainer locally.

- To initialize an explainer object, pass your model and some training data to the explainer's constructor.
- To make your explanations and visualizations more informative, you can choose to pass in feature names and output class names if doing classification.

The following code blocks show how to instantiate an explainer object with `TabularExplainer`,

`MimicExplainer`, and `PFIExplainer` locally.

- `TabularExplainer` calls one of the three SHAP explainers underneath (`TreeExplainer`, `DeepExplainer`, or `KernelExplainer`).
- `TabularExplainer` automatically selects the most appropriate one for your use case, but you can call each of its three underlying explainers directly.

```
from interpret.ext.blackbox import TabularExplainer

# "features" and "classes" fields are optional
explainer = TabularExplainer(model,
                             x_train,
                             features=breast_cancer_data.feature_names,
                             classes=classes)
```

or

```
from interpret.ext.blackbox import MimicExplainer

# you can use one of the following four interpretable models as a global surrogate to the black box
model

from interpret.ext.glassbox import LGBMExplainableModel
from interpret.ext.glassbox import LinearExplainableModel
from interpret.ext.glassbox import SGDExplainableModel
from interpret.ext.glassbox import DecisionTreeExplainableModel

# "features" and "classes" fields are optional
# augment_data is optional and if true, oversamples the initialization examples to improve surrogate
model accuracy to fit original model. Useful for high-dimensional data where the number of rows is
less than the number of columns.
# max_num_of_augmentations is optional and defines max number of times we can increase the input data
size.
# LGBMExplainableModel can be replaced with LinearExplainableModel, SGDExplainableModel, or
DecisionTreeExplainableModel
explainer = MimicExplainer(model,
                           x_train,
                           LGBMExplainableModel,
                           augment_data=True,
                           max_num_of_augmentations=10,
                           features=breast_cancer_data.feature_names,
                           classes=classes)
```

or

```
from interpret.ext.blackbox import PFIExplainer

# "features" and "classes" fields are optional
explainer = PFIExplainer(model,
                         features=breast_cancer_data.feature_names,
                         classes=classes)
```

Explain the entire model behavior (global explanation)

Refer to the following example to help you get the aggregate (global) feature importance values.

```
# you can use the training data or the test data here, but test data would allow you to use Explanation
Exploration
global_explanation = explainer.explain_global(x_test)

# if you used the PFIExplainer in the previous step, use the next line of code instead
# global_explanation = explainer.explain_global(x_train, true_labels=y_train)

# sorted feature importance values and feature names
sorted_global_importance_values = global_explanation.get_ranked_global_values()
sorted_global_importance_names = global_explanation.get_ranked_global_names()
dict(zip(sorted_global_importance_names, sorted_global_importance_values))

# alternatively, you can print out a dictionary that holds the top K feature names and values
global_explanation.get_feature_importance_dict()
```

Explain an individual prediction (local explanation)

Get the individual feature importance values of different datapoints by calling explanations for an individual instance or a group of instances.

NOTE

PFIExplainer does not support local explanations.

```
# get explanation for the first data point in the test set
local_explanation = explainer.explain_local(x_test[0:5])

# sorted feature importance values and feature names
sorted_local_importance_names = local_explanation.get_ranked_local_names()
sorted_local_importance_values = local_explanation.get_ranked_local_values()
```

Raw feature transformations

You can opt to get explanations in terms of raw, untransformed features rather than engineered features. For this option, you pass your feature transformation pipeline to the explainer in train_explain.py. Otherwise, the explainer provides explanations in terms of engineered features.

The format of supported transformations is the same as described in [sklearn-pandas](#). In general, any transformations are supported as long as they operate on a single column so that it's clear they're one-to-many.

Get an explanation for raw features by using a `sklearn.compose.ColumnTransformer` or with a list of fitted transformer tuples. The following example uses `sklearn.compose.ColumnTransformer`.

In case you want to run the example with the list of fitted transformer tuples, use the following code:

Generate feature importance values via remote runs

The following example shows how you can use the `ExplanationClient` class to enable model interpretability for remote runs. It is conceptually similar to the local process, except you:

- Use the `ExplanationClient` in the remote run to upload the interpretability context.
- Download the context later in a local environment.

1. Install the `azureml-interpret` package.

```
pip install azureml-interpret
```

2. Create a training script in a local Jupyter Notebook. For example, `train_explain.py`.

```
from azureml.interpret import ExplanationClient
from azureml.core.run import Run
from interpret.ext.blackbox import TabularExplainer

run = Run.get_context()
client = ExplanationClient.from_run(run)

# write code to get and split your data into train and test sets here
# write code to train your model here

# explain predictions on your local machine
# "features" and "classes" fields are optional
explainer = TabularExplainer(model,
                             x_train,
                             features=feature_names,
                             classes=classes)

# explain overall model predictions (global explanation)
global_explanation = explainer.explain_global(x_test)

# uploading global model explanation data for storage or visualization in webUX
# the explanation can then be downloaded on any compute
# multiple explanations can be uploaded
client.upload_model_explanation(global_explanation, comment='global explanation: all features')
# or you can only upload the explanation object with the top k feature info
#client.upload_model_explanation(global_explanation, top_k=2, comment='global explanation: Only top 2 features')
```

3. Set up an Azure Machine Learning Compute as your compute target and submit your training run. See [Create and manage Azure Machine Learning compute clusters](#) for instructions. You might also find the [example notebooks](#) helpful.

4. Download the explanation in your local Jupyter Notebook.

```
from azureml.interpret import ExplanationClient

client = ExplanationClient.from_run(run)

# get model explanation data
explanation = client.download_model_explanation()
# or only get the top k (e.g., 4) most important features with their importance values
explanation = client.download_model_explanation(top_k=4)

global_importance_values = explanation.get_ranked_global_values()
global_importance_names = explanation.get_ranked_global_names()
print('global importance values: {}'.format(global_importance_values))
print('global importance names: {}'.format(global_importance_names))
```

Visualizations

After you download the explanations in your local Jupyter Notebook, you can use the visualization dashboard to understand and interpret your model. To load the visualization dashboard widget in your Jupyter Notebook, use the following code:

```
from interpret_community.widget import ExplanationDashboard

ExplanationDashboard(global_explanation, model, datasetX=x_test)
```

The visualization supports explanations on both engineered and raw features. Raw explanations are based on the features from the original dataset and engineered explanations are based on the features from the dataset with feature engineering applied.

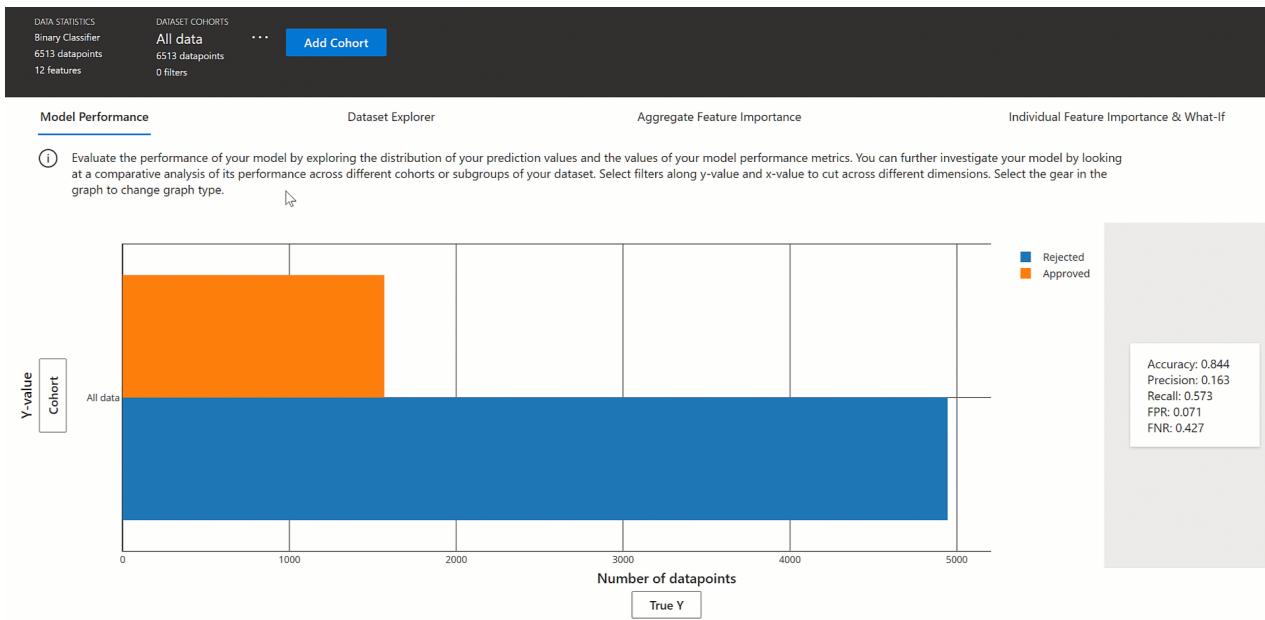
When attempting to interpret a model with respect to the original dataset it is recommended to use raw explanations as each feature importance will correspond to a column from the original dataset. One scenario where engineered explanations might be useful is when examining the impact of individual categories from a categorical feature. If a one-hot encoding is applied to a categorical feature, then the resulting engineered explanations will include a different importance value per category, one per one-hot engineered feature. This can be useful when narrowing down which part of the dataset is most informative to the model.

NOTE

Engineered and raw explanations are computed sequentially. First an engineered explanation is created based on the model and featurization pipeline. Then the raw explanation is created based on that engineered explanation by aggregating the importance of engineered features that came from the same raw feature.

Create, edit and view dataset cohorts

The top ribbon shows the overall statistics on your model and data. You can slice and dice your data into dataset cohorts, or subgroups, to investigate or compare your model's performance and explanations across these defined subgroups. By comparing your dataset statistics and explanations across those subgroups, you can get a sense of why possible errors are happening in one group versus another.

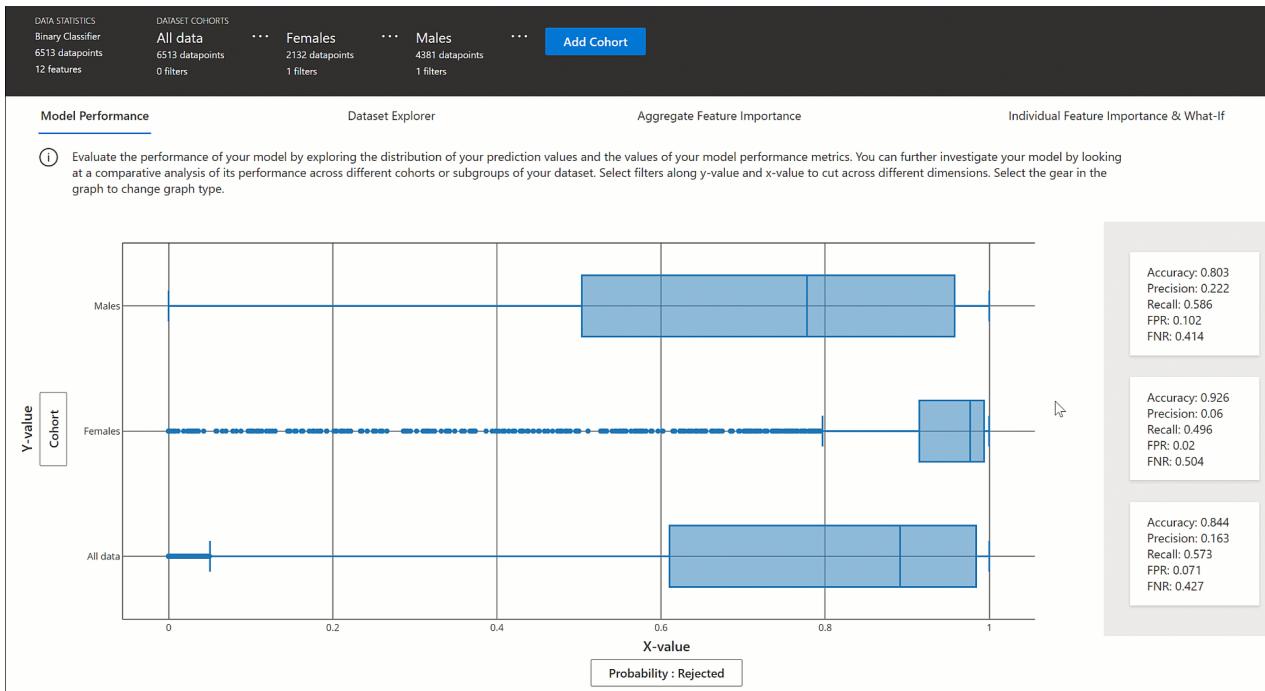


Understand entire model behavior (global explanation)

The first three tabs of the explanation dashboard provide an overall analysis of the trained model along with its predictions and explanations.

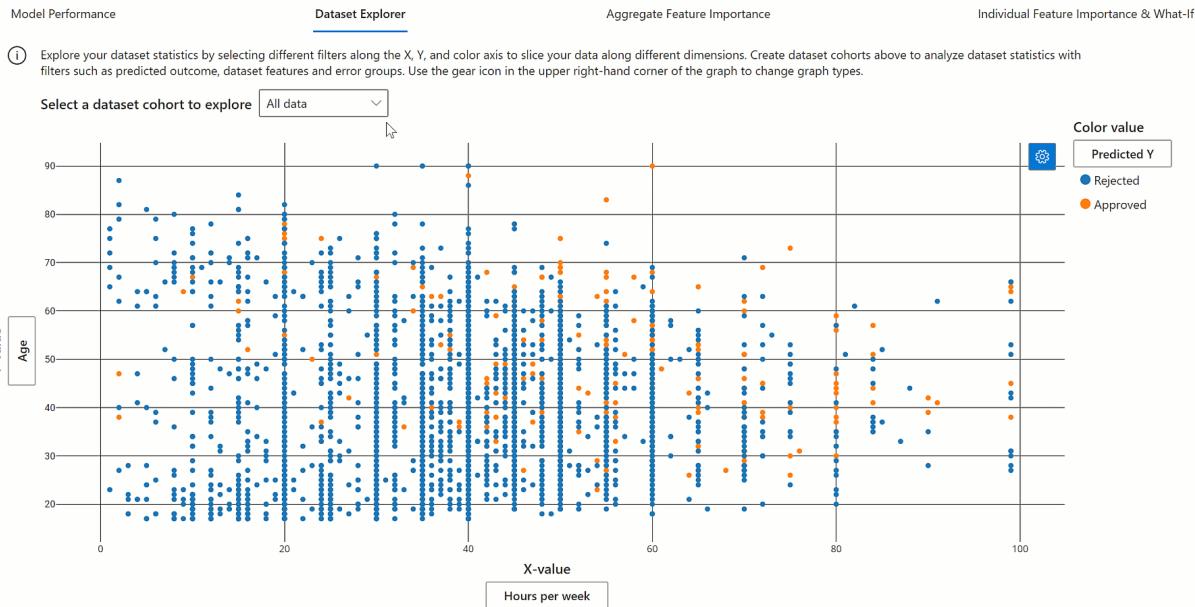
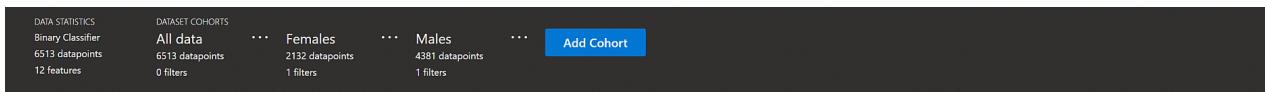
Model performance

Evaluate the performance of your model by exploring the distribution of your prediction values and the values of your model performance metrics. You can further investigate your model by looking at a comparative analysis of its performance across different cohorts or subgroups of your dataset. Select filters along y-value and x-value to cut across different dimensions. View metrics such as accuracy, precision, recall, false positive rate (FPR) and false negative rate (FNR).



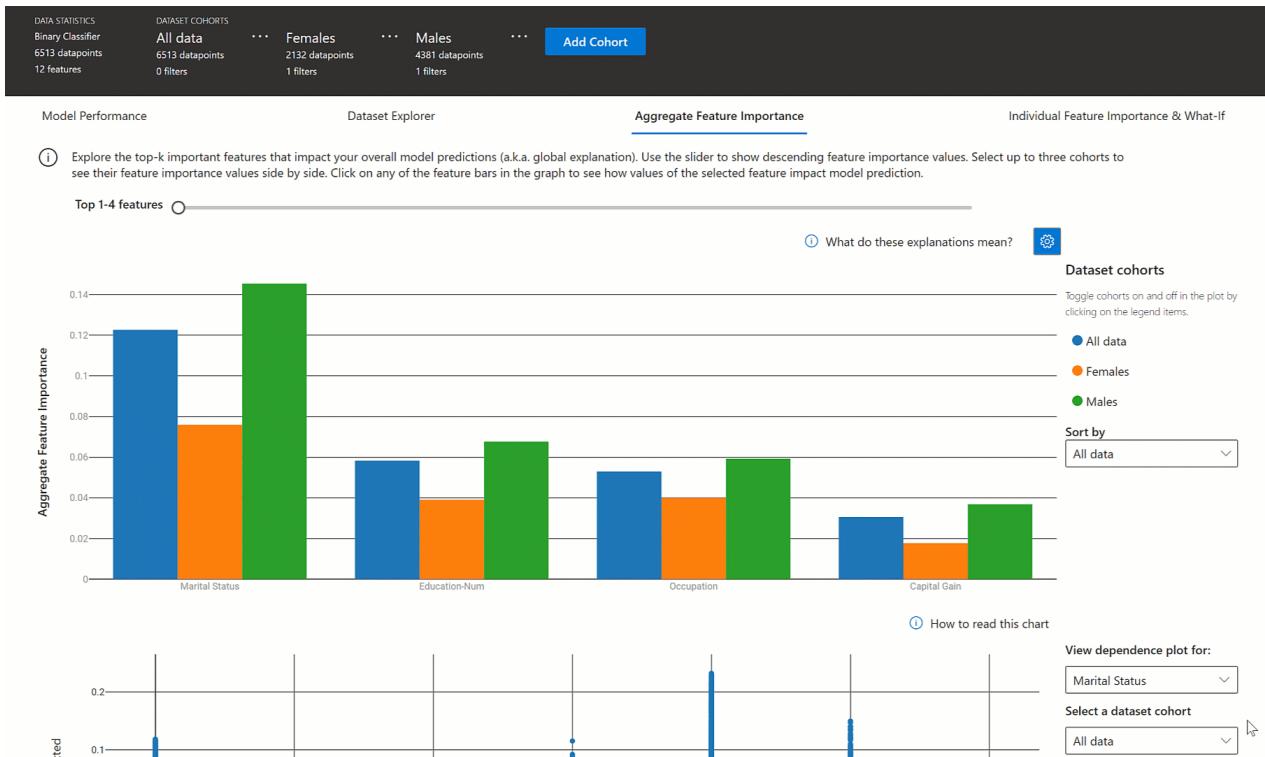
Dataset explorer

Explore your dataset statistics by selecting different filters along the X, Y, and color axes to slice your data along different dimensions. Create dataset cohorts above to analyze dataset statistics with filters such as predicted outcome, dataset features and error groups. Use the gear icon in the upper right-hand corner of the graph to change graph types.



Aggregate feature importance

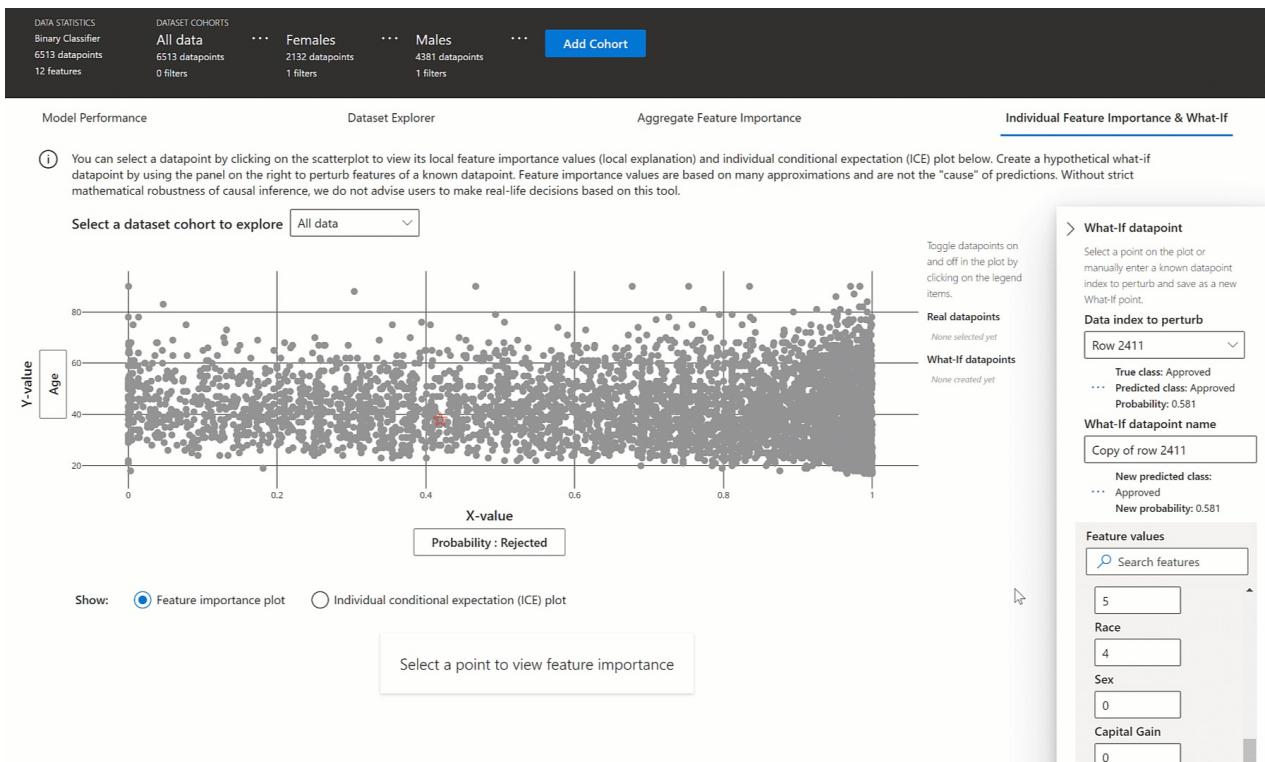
Explore the top-k important features that impact your overall model predictions (also known as global explanation). Use the slider to show descending feature importance values. Select up to three cohorts to see their feature importance values side by side. Click on any of the feature bars in the graph to see how values of the selected feature impact model prediction in the dependence plot below.



Understand individual predictions (local explanation)

The fourth tab of the explanation tab lets you drill into an individual datapoint and their individual feature importances. You can load the individual feature importance plot for any data point by clicking on any of the individual data points in the main scatter plot or selecting a specific datapoint in the panel wizard on the right.

PLOT	DESCRIPTION
Individual feature importance	Shows the top-k important features for an individual prediction. Helps illustrate the local behavior of the underlying model on a specific data point.
What-If analysis	Allows changes to feature values of the selected real data point and observe resulting changes to prediction value by generating a hypothetical datapoint with the new feature values.
Individual Conditional Expectation (ICE)	Allows feature value changes from a minimum value to a maximum value. Helps illustrate how the data point's prediction changes when a feature changes.



NOTE

These are explanations based on many approximations and are not the "cause" of predictions. Without strict mathematical robustness of causal inference, we do not advise users to make real-life decisions based on the feature perturbations of the What-If tool. This tool is primarily for understanding your model and debugging.

Visualization in Azure Machine Learning studio

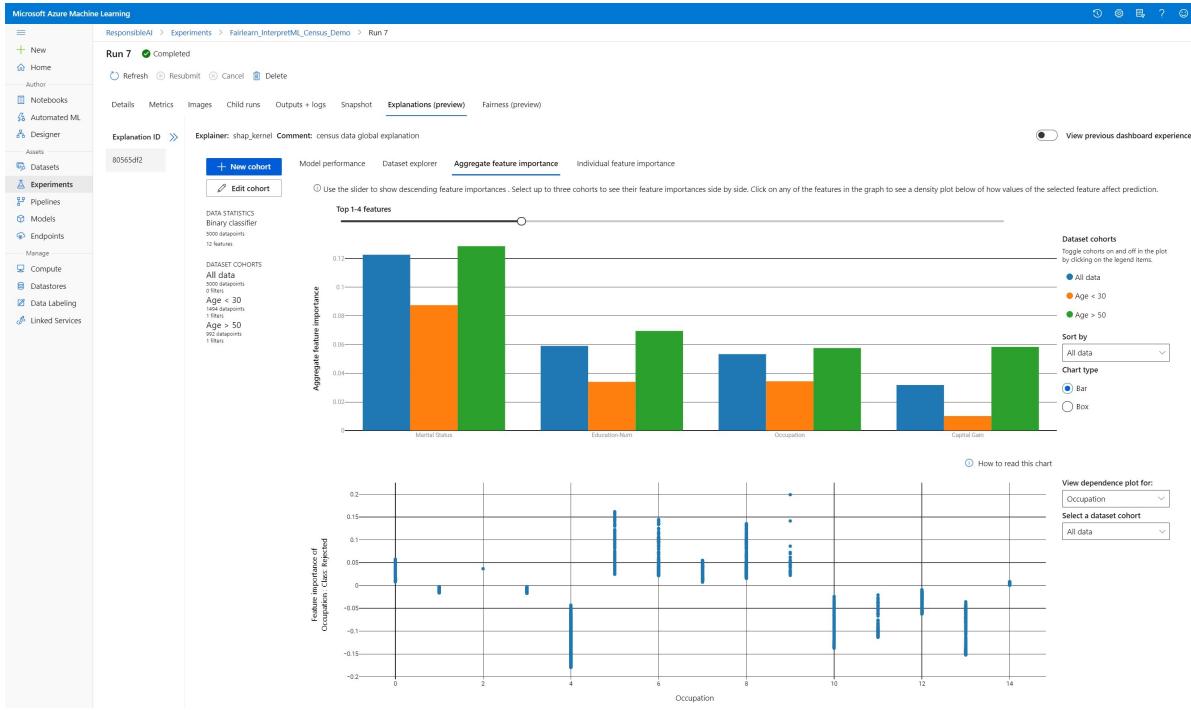
If you complete the [remote interpretability](#) steps (uploading generated explanation to Azure Machine Learning Run History), you can view the visualization dashboard in [Azure Machine Learning studio](#). This dashboard is a simpler version of the visualization dashboard explained above. What-If datapoint generation and ICE plots are disabled as there is no active compute in Azure Machine Learning studio that can perform their real time computations.

If the dataset, global, and local explanations are available, data populates all of the tabs. If only a global explanation is available, the Individual feature importance tab will be disabled.

Follow one of these paths to access the visualization dashboard in Azure Machine Learning studio:

- Experiments pane (Preview)

1. Select **Experiments** in the left pane to see a list of experiments that you've run on Azure Machine Learning.
2. Select a particular experiment to view all the runs in that experiment.
3. Select a run, and then the **Explanations** tab to the explanation visualization dashboard.



- **Models** pane

1. If you registered your original model by following the steps in [Deploy models with Azure Machine Learning](#), you can select **Models** in the left pane to view it.
2. Select a model, and then the **Explanations** tab to view the explanation visualization dashboard.

Interpretability at inference time

You can deploy the explainer along with the original model and use it at inference time to provide the individual feature importance values (local explanation) for any new datapoint. We also offer lighter-weight scoring explainers to improve interpretability performance at inference time, which is currently supported only in Azure Machine Learning SDK. The process of deploying a lighter-weight scoring explainer is similar to deploying a model and includes the following steps:

1. Create an explanation object. For example, you can use `TabularExplainer`:

```
from interpret.ext.blackbox import TabularExplainer

explainer = TabularExplainer(model,
                             initialization_examples=x_train,
                             features=dataset_feature_names,
                             classes=dataset_classes,
                             transformations=transformations)
```

2. Create a scoring explainer with the explanation object.

```
from azureml.interpret.scoring.scoring_explainer import KernelScoringExplainer, save

# create a lightweight explainer at scoring time
scoring_explainer = KernelScoringExplainer(explainer)

# pickle scoring explainer
# pickle scoring explainer locally
OUTPUT_DIR = 'my_directory'
save(scoring_explainer, directory=OUTPUT_DIR, exist_ok=True)
```

3. Configure and register an image that uses the scoring explainer model.

```
# register explainer model using the path from ScoringExplainer.save - could be done on remote compute
# scoring_explainer.pkl is the filename on disk, while my_scoring_explainer.pkl will be the filename in
# cloud storage
run.upload_file('my_scoring_explainer.pkl', os.path.join(OUTPUT_DIR, 'scoring_explainer.pkl'))

scoring_explainer_model = run.register_model(model_name='my_scoring_explainer',
                                              model_path='my_scoring_explainer.pkl')
print(scoring_explainer_model.name, scoring_explainer_model.id, scoring_explainer_model.version, sep =
'\t')
```

4. As an optional step, you can retrieve the scoring explainer from cloud and test the explanations.

```
from azureml.interpret.scoring.scoring_explainer import load

# retrieve the scoring explainer model from cloud"
scoring_explainer_model = Model(ws, 'my_scoring_explainer')
scoring_explainer_model_path = scoring_explainer_model.download(target_dir=os.getcwd(), exist_ok=True)

# load scoring explainer from disk
scoring_explainer = load(scoring_explainer_model_path)

# test scoring explainer locally
preds = scoring_explainer.explain(x_test)
print(preds)
```

5. Deploy the image to a compute target, by following these steps:

- If needed, register your original prediction model by following the steps in [Deploy models with Azure Machine Learning](#).
- Create a scoring file.

```

%%writefile score.py
import json
import numpy as np
import pandas as pd
import os
import pickle
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression
from azureml.core.model import Model

def init():

    global original_model
    global scoring_model

    # retrieve the path to the model file using the model name
    # assume original model is named original_prediction_model
    original_model_path = Model.get_model_path('original_prediction_model')
    scoring_explainer_path = Model.get_model_path('my_scoring_explainer')

    original_model = joblib.load(original_model_path)
    scoring_explainer = joblib.load(scoring_explainer_path)

def run(raw_data):
    # get predictions and explanations for each data point
    data = pd.read_json(raw_data)
    # make prediction
    predictions = original_model.predict(data)
    # retrieve model explanations
    local_importance_values = scoring_explainer.explain(data)
    # you can return any data type as long as it is JSON-serializable
    return {'predictions': predictions.tolist(), 'local_importance_values':
local_importance_values}

```

c. Define the deployment configuration.

This configuration depends on the requirements of your model. The following example defines a configuration that uses one CPU core and one GB of memory.

```

from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores=1,
                                                memory_gb=1,
                                                tags={"data": "NAME_OF_THE_DATASET",
                                                      "method" : "local_explanation"},
                                                description='Get local explanations for
NAME_OF_THE_PROBLEM')

```

d. Create a file with environment dependencies.

```

from azureml.core.conda_dependencies import CondaDependencies

# WARNING: to install this, g++ needs to be available on the Docker image and is not by default
# (look at the next cell)

azureml_pip_packages = ['azureml-defaults', 'azureml-contrib-interpret', 'azureml-core',
'azureml-telemetry', 'azureml-interpret']

# specify CondaDependencies obj
myenv = CondaDependencies.create(conda_packages=['scikit-learn', 'pandas'],
                                pip_packages=['sklearn-pandas'] + azureml_pip_packages,
                                pin_sdk_version=False)

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())

with open("myenv.yml","r") as f:
    print(f.read())

```

- e. Create a custom dockerfile with g++ installed.

```

%%writefile dockerfile
RUN apt-get update && apt-get install -y g++

```

- f. Deploy the created image.

This process takes approximately five minutes.

```

from azureml.core.webservice import Webservice
from azureml.core.image import ContainerImage

# use the custom scoring, docker, and conda files we created above
image_config = ContainerImage.image_configuration(execution_script="score.py",
                                                    docker_file="dockerfile",
                                                    runtime="python",
                                                    conda_file="myenv.yml")

# use configs and models generated above
service = Webservice.deploy_from_model(workspace=ws,
                                         name='model-scoring-service',
                                         deployment_config=aciconfig,
                                         models=[scoring_explainer_model, original_model],
                                         image_config=image_config)

service.wait_for_deployment(show_output=True)

```

6. Test the deployment.

```

import requests

# create data to test service with
examples = x_list[:4]
input_data = examples.to_json()

headers = {'Content-Type':'application/json'}

# send request to service
resp = requests.post(service.scoring_uri, input_data, headers=headers)

print("POST to url", service.scoring_uri)
# can convert back to Python objects from json string if desired
print("prediction:", resp.text)

```

7. Clean up.

To delete a deployed web service, use `service.delete()`.

Troubleshooting

- **Sparse data not supported:** The model explanation dashboard breaks/slows down substantially with a large number of features, therefore we currently do not support sparse data format. Additionally, general memory issues will arise with large datasets and large number of features.
- **Forecasting models not supported with model explanations:** Interpretability, best model explanation, is not available for AutoML forecasting experiments that recommend the following algorithms as the best model: TCNForecaster, AutoArima, Prophet, ExponentialSmoothing, Average, Naive, Seasonal Average, and Seasonal Naive. AutoML Forecasting has regression models which support explanations. However, in the explanation dashboard, the "Individual feature importance" tab is just not supported for forecasting because of complexity in their data pipelines.
- **Local explanation for data index:** The explanation dashboard does not support relating local importance values to a row identifier from the original validation dataset if that dataset is greater than 5000 datapoints as the dashboard randomly downsamples the data. However, the dashboard shows raw dataset feature values for each datapoint passed into the dashboard under the Individual feature importance tab. Users can map local importances back to the original dataset through matching the raw dataset feature values. If the validation dataset size is less than 5000 samples, the `index` feature in AzureML studio will correspond to the index in the validation dataset.
- **What-if/ICE plots not supported in studio:** What-If and Individual Conditional Expectation (ICE) plots are not supported in Azure Machine Learning studio under the Explanations tab since the uploaded explanation needs an active compute to recalculate predictions and probabilities of perturbed features. It is currently supported in Jupyter notebooks when run as a widget using the SDK.

Next steps

[Learn more about model interpretability](#)

[Check out Azure Machine Learning Interpretability sample notebooks](#)

Interpretability: model explanations in automated machine learning (preview)

12/23/2020 • 4 minutes to read • [Edit Online](#)

In this article, you learn how to get explanations for automated machine learning (AutoML) in Azure Machine Learning. AutoML 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 AutoML experiments. For more information on how to use the Azure Machine Learning SDK, complete this [regression model tutorial](#) or see how to [configure AutoML experiments](#).

Interpretability during training for the best model

Retrieve the explanation from the `best_run`, which includes explanations for both raw and engineered features.

WARNING

Interpretability, best model explanation, is not available for Auto ML forecasting experiments that recommend the following algorithms as the best model:

- TCNForecaster
- AutoArima
- ExponentialSmoothing
- Prophet
- Average
- Naive
- Seasonal Average
- Seasonal Naive

Download engineered feature importance from artifact store

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())
```

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 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 AutoML 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` AutoML 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 MimicExplainer 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 use `ExplanationDashboard` to view the dashboard visualization of the feature importance values of the generated engineered features by AutoML featurizers.

```
engineered_explanations = explainer.explain(['local', 'global'],
eval_dataset=automl_explainer_setup_obj.X_test_transform)
print(engineered_explanations.get_feature_importance_dict())
```

Interpretability during inference

In this section, you learn how to operationalize an AutoML 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())
```

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.

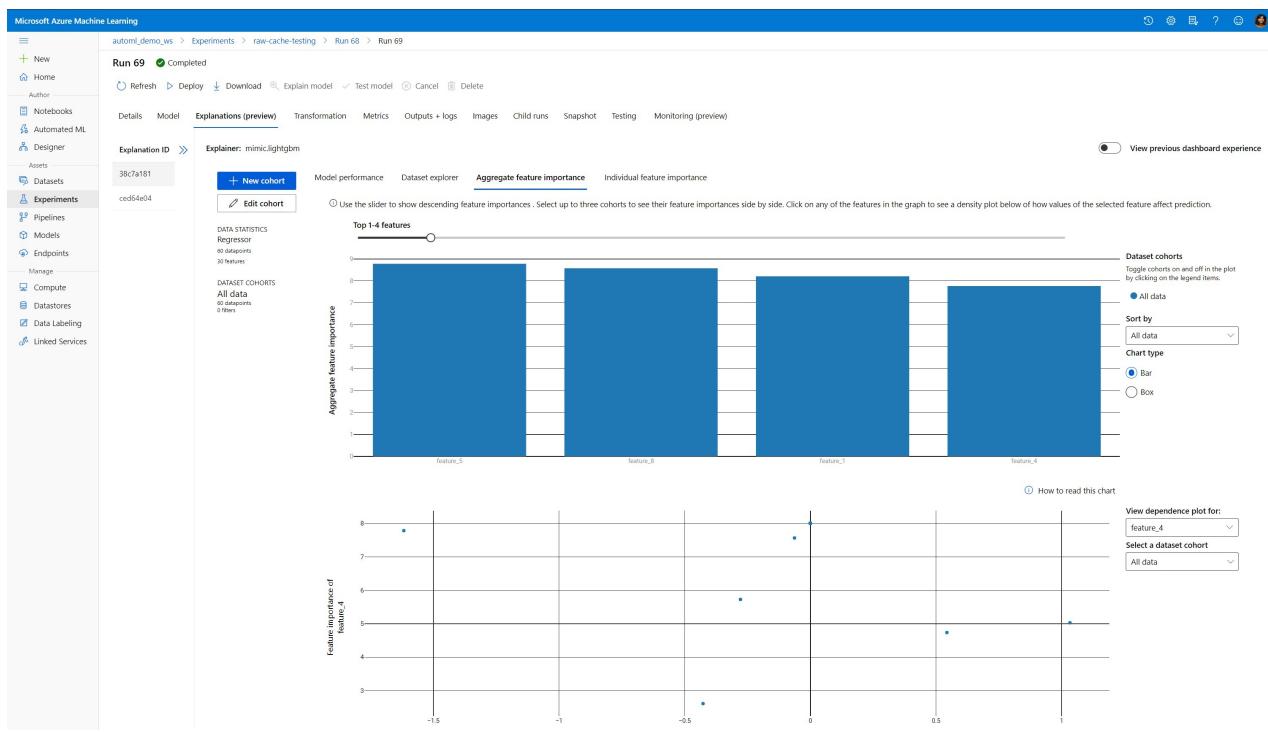
```

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'])

```

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 explanation visualization 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 of the Azure Machine Learning SDK other than automated machine learning, see the [concept article on interpretability](#).

Use Azure Machine Learning with the Fairlearn open-source package to assess the fairness of ML models (preview)

12/23/2020 • 7 minutes to read • [Edit Online](#)

In this how-to guide, you will learn to use the [Fairlearn](#) open-source Python package with Azure Machine Learning to perform the following tasks:

- Assess the fairness of your model predictions. To learn more about fairness in machine learning, see the [fairness in machine learning article](#).
- Upload, list and download fairness assessment insights to/from Azure Machine Learning studio.
- See a fairness assessment dashboard in Azure Machine Learning studio to interact with your model(s)' fairness insights.

NOTE

Fairness assessment is not a purely technical exercise. **This package can help you assess the fairness of a machine learning model, but only you can configure and make decisions as to how the model performs.** While this package helps to identify quantitative metrics to assess fairness, developers of machine learning models must also perform a qualitative analysis to evaluate the fairness of their own models.

Azure Machine Learning Fairness SDK

The Azure Machine Learning Fairness SDK, `azureml-contrib-fairness`, integrates the open-source Python package, [Fairlearn](#), within Azure Machine Learning. To learn more about Fairlearn's integration within Azure Machine Learning, check out these [sample notebooks](#). For more information on Fairlearn, see the [example guide](#) and [sample notebooks](#).

Use the following commands to install the `azureml-contrib-fairness` and `fairlearn` packages:

```
pip install azureml-contrib-fairness
pip install fairlearn==0.4.6
```

Later versions of Fairlearn should also work in the following example code.

Upload fairness insights for a single model

The following example shows how to use the fairness package. We will upload model fairness insights into Azure Machine Learning and see the fairness assessment dashboard in Azure Machine Learning studio.

1. Train a sample model in a Jupyter notebook.

For the dataset, we use the well-known adult census dataset, which we fetch from OpenML. We pretend we have a loan decision problem with the label indicating whether an individual repaid a previous loan. We will train a model to predict if previously unseen individuals will repay a loan. Such a model might be used in making loan decisions.

```
import copy
import numpy as np
```

```

import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.datasets import fetch_openml
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_selector as selector
from sklearn.pipeline import Pipeline

from fairlearn.widget import FairlearnDashboard

# Load the census dataset
data = fetch_openml(data_id=1590, as_frame=True)
X_raw = data.data
y = (data.target == ">50K") * 1

# (Optional) Separate the "sex" and "race" sensitive features out and drop them from the main data
# prior to training your model
X_raw = data.data
y = (data.target == ">50K") * 1
A = X_raw[["race", "sex"]]
X = X_raw.drop(labels=['sex', 'race'], axis=1)

# Split the data in "train" and "test" sets
(X_train, X_test, y_train, y_test, A_train, A_test) = train_test_split(
    X_raw, y, A, test_size=0.3, random_state=12345, stratify=y
)

# Ensure indices are aligned between X, y and A,
# after all the slicing and splitting of DataFrames
# and Series
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
A_train = A_train.reset_index(drop=True)
A_test = A_test.reset_index(drop=True)

# Define a processing pipeline. This happens after the split to avoid data leakage
numeric_transformer = Pipeline(
    steps=[
        ("impute", SimpleImputer()),
        ("scaler", StandardScaler()),
    ]
)
categorical_transformer = Pipeline(
    [
        ("impute", SimpleImputer(strategy="most_frequent")),
        ("ohe", OneHotEncoder(handle_unknown="ignore")),
    ]
)
preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, selector(dtype_exclude="category")),
        ("cat", categorical_transformer, selector(dtype_include="category")),
    ]
)

# Put an estimator onto the end of the pipeline
lr_predictor = Pipeline(
    steps=[
        ("preprocessor", copy.deepcopy(preprocessor)),
        (
            "classifier",
            LogisticRegression(solver="liblinear", fit_intercept=True),
        ),
    ],
)

```

```

        )

# Train the model on the test data
lr_predictor.fit(X_train, y_train)

# (Optional) View this model in Fairlearn's fairness dashboard, and see the disparities which appear:
from fairlearn.widget import FairlearnDashboard
FairlearnDashboard(sensitive_features=A_test,
                   sensitive_feature_names=['Race', 'Sex'],
                   y_true=y_test,
                   y_pred={"lr_model": lr_predictor.predict(X_test)})

```

2. Log into Azure Machine Learning and register your model.

The fairness dashboard can integrate with registered or unregistered models. Register your model in Azure Machine Learning with the following steps:

```

from azureml.core import Workspace, Experiment, Model
import joblib
import os

ws = Workspace.from_config()
ws.get_details()

os.makedirs('models', exist_ok=True)

# Function to register models into Azure Machine Learning
def register_model(name, model):
    print("Registering ", name)
    model_path = "models/{0}.pkl".format(name)
    joblib.dump(value=model, filename=model_path)
    registered_model = Model.register(model_path=model_path,
                                       model_name=name,
                                       workspace=ws)
    print("Registered ", registered_model.id)
    return registered_model.id

# Call the register_model function
lr_reg_id = register_model("fairness_logistic_regression", lr_predictor)

```

3. Precompute fairness metrics.

Create a dashboard dictionary using Fairlearn's `metrics` package. The `_create_group_metric_set` method has arguments similar to the Dashboard constructor, except that the sensitive features are passed as a dictionary (to ensure that names are available). We must also specify the type of prediction (binary classification in this case) when calling this method.

```

# Create a dictionary of model(s) you want to assess for fairness
sf = { 'Race': A_test.race, 'Sex': A_test.sex}
ys_pred = { lr_reg_id:lr_predictor.predict(X_test) }
from fairlearn.metrics._group_metric_set import _create_group_metric_set

dash_dict = _create_group_metric_set(y_true=y_test,
                                      predictions=ys_pred,
                                      sensitive_features=sf,
                                      prediction_type='binary_classification')

```

4. Upload the precomputed fairness metrics.

Now, import `azureml.contrib.fairness` package to perform the upload:

```
from azureml.contrib.fairness import upload_dashboard_dictionary, download_dashboard_by_upload_id
```

Create an Experiment, then a Run, and upload the dashboard to it:

```
exp = Experiment(ws, "Test_Fairness_Census_Demo")
print(exp)

run = exp.start_logging()

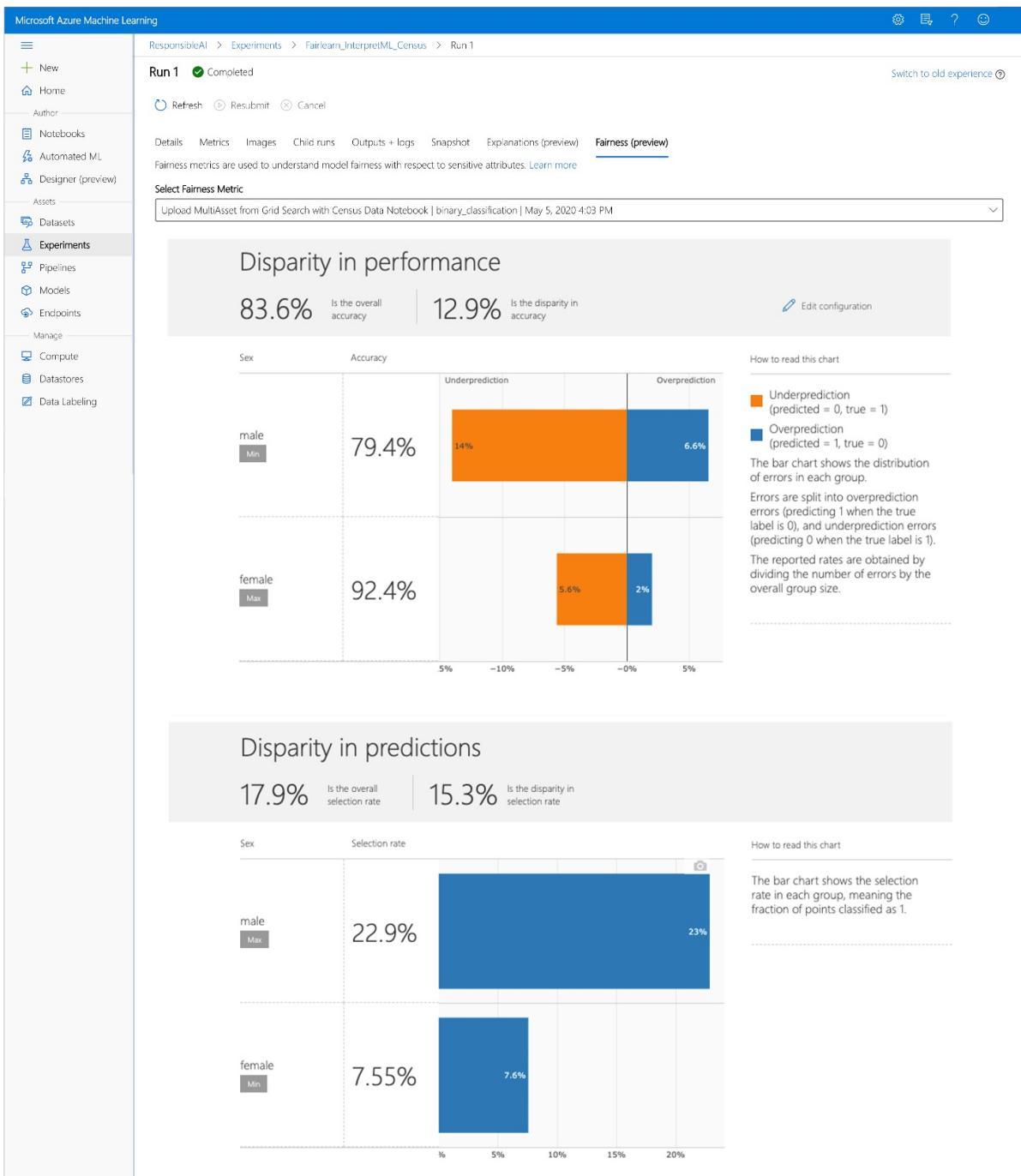
# Upload the dashboard to Azure Machine Learning
try:
    dashboard_title = "Fairness insights of Logistic Regression Classifier"
    # Set validate_model_ids parameter of upload_dashboard_dictionary to False if you have not
    registered your model(s)
    upload_id = upload_dashboard_dictionary(run,
                                              dash_dict,
                                              dashboard_name=dashboard_title)
    print("\nUploaded to id: {0}\n".format(upload_id))

    # To test the dashboard, you can download it back and ensure it contains the right information
    downloaded_dict = download_dashboard_by_upload_id(run, upload_id)
finally:
    run.complete()
```

5. Check the fairness dashboard from Azure Machine Learning studio

If you complete the previous steps (uploading generated fairness insights to Azure Machine Learning), you can view the fairness dashboard in [Azure Machine Learning studio](#). This dashboard is the same visualization dashboard provided in Fairlearn, enabling you to analyze the disparities among your sensitive feature's subgroups (e.g., male vs. female). Follow one of these paths to access the visualization dashboard in Azure Machine Learning studio:

- **Experiments pane (Preview)**
 - a. Select **Experiments** in the left pane to see a list of experiments that you've run on Azure Machine Learning.
 - b. Select a particular experiment to view all the runs in that experiment.
 - c. Select a run, and then the **Fairness** tab to the explanation visualization dashboard.



• Models pane

- If you registered your original model by following the previous steps, you can select **Models** in the left pane to view it.
- Select a model, and then the **Fairness** tab to view the explanation visualization dashboard.

To learn more about the visualization dashboard and what it contains, check out Fairlearn's [user guide](#).

Upload fairness insights for multiple models

To compare multiple models and see how their fairness assessments differ, you can pass more than one model to the visualization dashboard and compare their performance-fairness trade-offs.

1. Train your models:

We now create a second classifier, based on a Support Vector Machine estimator, and upload a fairness dashboard dictionary using Fairlearn's `metrics` package. We assume that the previously trained model is still available.

```

# Put an SVM predictor onto the preprocessing pipeline
from sklearn import svm
svm_predictor = Pipeline(
    steps=[
        ("preprocessor", copy.deepcopy(preprocessor)),
        (
            "classifier",
            svm.SVC(),
        ),
    ],
)

# Train your second classification model
svm_predictor.fit(X_train, y_train)

```

2. Register your models

Next register both models within Azure Machine Learning. For convenience, store the results in a dictionary, which maps the `id` of the registered model (a string in `name:version` format) to the predictor itself:

```

model_dict = {}

lr_reg_id = register_model("fairness_logistic_regression", lr_predictor)
model_dict[lr_reg_id] = lr_predictor

svm_reg_id = register_model("fairness_svm", svm_predictor)
model_dict[svm_reg_id] = svm_predictor

```

3. Load the Fairlearn dashboard locally

Before uploading the fairness insights into Azure Machine Learning, you can examine these predictions in a locally invoked Fairlearn dashboard.

```

# Generate models' predictions and load the fairness dashboard locally
ys_pred = {}
for n, p in model_dict.items():
    ys_pred[n] = p.predict(X_test)

from fairlearn.widget import FairlearnDashboard

FairlearnDashboard(sensitive_features=A_test,
                   sensitive_feature_names=['Race', 'Sex'],
                   y_true=y_test.tolist(),
                   y_pred=ys_pred)

```

4. Precompute fairness metrics.

Create a dashboard dictionary using Fairlearn's `metrics` package.

```

sf = { 'Race': A_test.race, 'Sex': A_test.sex }

from fairlearn.metrics._group_metric_set import _create_group_metric_set

dash_dict = _create_group_metric_set(y_true=Y_test,
                                      predictions=ys_pred,
                                      sensitive_features=sf,
                                      prediction_type='binary_classification')

```

5. Upload the precomputed fairness metrics.

Now, import `azureml.contrib.fairness` package to perform the upload:

```
from azureml.contrib.fairness import upload_dashboard_dictionary, download_dashboard_by_upload_id
```

Create an Experiment, then a Run, and upload the dashboard to it:

```
exp = Experiment(ws, "Compare_Two_Models_Fairness_Census_Demo")
print(exp)

run = exp.start_logging()

# Upload the dashboard to Azure Machine Learning
try:
    dashboard_title = "Fairness Assessment of Logistic Regression and SVM Classifiers"
    # Set validate_model_ids parameter of upload_dashboard_dictionary to False if you have not
    registered your model(s)
    upload_id = upload_dashboard_dictionary(run,
                                              dash_dict,
                                              dashboard_name=dashboard_title)
    print("\nUploaded to id: {}".format(upload_id))

    # To test the dashboard, you can download it back and ensure it contains the right information
    downloaded_dict = download_dashboard_by_upload_id(run, upload_id)
finally:
    run.complete()
```

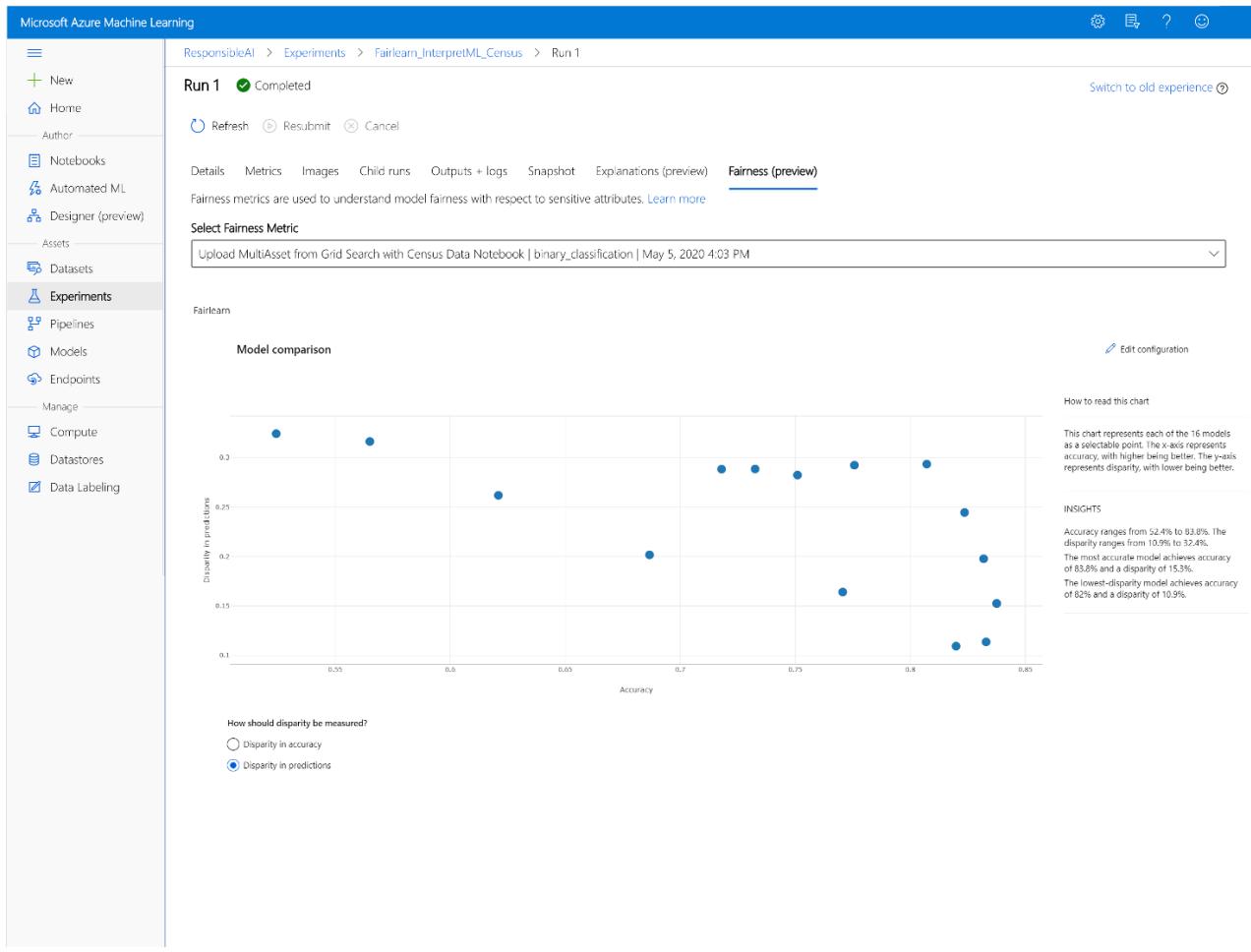
Similar to the previous section, you can follow one of the paths described above (via **Experiments** or **Models**) in Azure Machine Learning studio to access the visualization dashboard and compare the two models in terms of fairness and performance.

Upload unmitigated and mitigated fairness insights

You can use Fairlearn's [mitigation algorithms](#), compare their generated mitigated model(s) to the original unmitigated model, and navigate the performance/fairness trade-offs among compared models.

To see an example that demonstrates the use of the [Grid Search](#) mitigation algorithm (which creates a collection of mitigated models with different fairness and performance trade offs) check out this [sample notebook](#).

Uploading multiple models' fairness insights in a single Run allows for comparison of models with respect to fairness and performance. You can click on any of the models displayed in the model comparison chart to see the detailed fairness insights of the particular model.



Next steps

[Learn more about model fairness](#)

[Check out Azure Machine Learning Fairness sample notebooks](#)

Use authentication credential secrets in Azure Machine Learning training runs

12/23/2020 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to use secrets in training runs securely. 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 run context. Coding such values into training scripts in cleartext is insecure as it would expose the secret.

Instead, your Azure Machine Learning workspace has an associated resource called a [Azure Key Vault](#). Use this Key Vault to pass secrets to remote runs securely through a set of APIs in the Azure Machine Learning Python SDK.

The standard flow for using secrets is:

1. On local computer, log in to Azure and connect to your workspace.
2. On local computer, set a secret in Workspace Key Vault.
3. Submit a remote run.
4. Within the remote run, get the secret from Key Vault and use it.

Set secrets

In the Azure Machine Learning, the [KeyVault](#) class contains methods for setting secrets. In your local Python session, first obtain a reference to your workspace Key Vault, and then use the [set_secret\(\)](#) method to set a secret by name and value. The [set_secret](#) method updates the secret value if the name already exists.

```
from azureml.core import Workspace
from azureml.core import KeyVault
import os

ws = Workspace.from_config()
my_secret = os.environ.get("MY_SECRET")
keyvault = ws.get_default_keyvault()
keyvault.set_secret(name="mysecret", value = my_secret)
```

Do not put the secret value in your Python code as it is insecure to store it in file as cleartext. Instead, obtain the secret value from an environment variable, for example Azure DevOps build secret, or from interactive user input.

You can list secret names using the [list_secrets\(\)](#) method and there is also a batch version, [set_secrets\(\)](#) that allows you to set multiple secrets at a time.

Get secrets

In your local code, you can use the [get_secret\(\)](#) method to get the secret value by name.

For runs submitted the [Experiment.submit](#), use the [get_secret\(\)](#) method with the [Run](#) class. Because a submitted run is aware of its workspace, this method shortcuits the Workspace instantiation and returns the secret value directly.

```
# Code in submitted run
from azureml.core import Experiment, Run

run = Run.get_context()
secret_value = run.get_secret(name="mysecret")
```

Be careful not to expose the secret value by writing or printing it out.

There is also a batch version, [get_secrets\(\)](#) for accessing multiple secrets at once.

Next steps

- [View example notebook](#)
- [Learn about enterprise security with Azure Machine Learning](#)

Reinforcement learning (preview) with Azure Machine Learning

12/23/2020 • 11 minutes to read • [Edit Online](#)

NOTE

Azure Machine Learning Reinforcement Learning is currently a preview feature. Only Ray and RLLib frameworks are supported at this time.

In this article, you learn how to train a reinforcement learning (RL) agent to play the video game Pong. You use the open-source Python library [Ray RLLib](#) with Azure Machine Learning to manage the complexity of distributed RL.

In this article you learn how to:

- Set up an experiment
- Define head and worker nodes
- Create an RL estimator
- Submit an experiment to start a run
- View results

This article is based on the [RLLib Pong example](#) that can be found in the Azure Machine Learning notebook [GitHub repository](#).

Prerequisites

Run this code in either of these environments. We recommend you try Azure Machine Learning compute instance for the fastest start-up experience. You can quickly clone and run the reinforcement sample notebooks on an Azure Machine Learning compute instance.

- Azure Machine Learning compute instance
 - Learn how to clone sample notebooks in [Tutorial: Setup environment and workspace](#).
 - Clone the **how-to-use-azureml** folder instead of **tutorials**
 - Run the virtual network setup notebook located at
`/how-to-use-azureml/reinforcement-learning/setup/devenv_setup.ipynb` to open network ports used for distributed reinforcement learning.
 - Run the sample notebook
`/how-to-use-azureml/reinforcement-learning/atari-on-distributed-compute/pong_rllib.ipynb`
- Your own Jupyter Notebook server
 - Install the [Azure Machine Learning SDK](#).
 - Install the [Azure Machine Learning RL SDK](#):
`pip install --upgrade azureml-contrib-reinforcementlearning`
 - Create a [workspace configuration file](#).
 - Run the virtual network to open network ports used for distributed reinforcement learning.

How to train a Pong-playing agent

Reinforcement learning (RL) is an approach to machine learning that learns by doing. While other machine

learning techniques learn by passively taking input data and finding patterns within it, RL uses **training agents** to actively make decisions and learn from their outcomes.

Your training agents learn to play Pong in a **simulated environment**. Training agents make a decision every frame of the game to move the paddle up, down, or stay in place. It looks at the state of the game (an RGB image of the screen) to make a decision.

RL uses **rewards** to tell the agent if its decisions are successful. In this example, the agent gets a positive reward when it scores a point and a negative reward when a point is scored against it. Over many iterations, the training agent learns to choose the action, based on its current state, that optimizes for the sum of expected future rewards. It's common to use **deep neural networks** (DNN) to perform this optimization in RL.

Training ends when the agent reaches an average reward score of 18 in a training epoch. This means that the agent has beaten its opponent by an average of at least 18 points in matches up to 21.

The process of iterating through simulation and retraining a DNN is computationally expensive, and requires a lot of data. One way to improve performance of RL jobs is by **parallelizing work** so that multiple training agents can act and learn simultaneously. However, managing a distributed RL environment can be a complex undertaking.

Azure Machine Learning provides the framework to manage these complexities to scale out your RL workloads.

Set up the environment

Set up the local RL environment by:

1. Loading the required Python packages
2. Initializing your workspace
3. Creating an experiment
4. Specifying a configured virtual network.

Import libraries

Import the necessary Python packages to run the rest of this example.

```
# Azure ML Core imports
import azureml.core
from azureml.core import Workspace
from azureml.core import Experiment
from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTargetException
from azureml.core.runconfig import EnvironmentDefinition
from azureml.widgets import RunDetails
from azureml.tensorboard import Tensorboard

# Azure ML Reinforcement Learning imports
from azureml.contrib.train.rl import ReinforcementLearningEstimator, Ray
from azureml.contrib.train.rl import WorkerConfiguration
```

Initialize a workspace

Initialize a [workspace](#) object from the `config.json` file created in the [prerequisites section](#). If you are executing this code in an Azure Machine Learning Compute Instance, the configuration file has already been created for you.

```
ws = Workspace.from_config()
```

Create a reinforcement learning experiment

Create an [experiment](#) to track your reinforcement learning run. In Azure Machine Learning, experiments are logical collections of related trials to organize run logs, history, outputs, and more.

```
experiment_name='rllib-pong-multi-node'

exp = Experiment(workspace=ws, name=experiment_name)
```

Specify a virtual network

For RL jobs that use multiple compute targets, you must specify a virtual network with open ports that allow worker nodes and head nodes to communicate with each other.

The virtual network can be in any resource group, but it should be in the same region as your workspace. For more information on setting up your virtual network, see the workspace setup notebook in the prerequisites section. Here, you specify the name of the virtual network in your resource group.

```
vnet = 'your_vnet'
```

Define head and worker compute targets

This example uses separate compute targets for the Ray head and workers nodes. These settings let you scale your compute resources up and down depending on your workload. Set the number of nodes, and the size of each node, based on your needs.

Head computing target

You can use a GPU-equipped head cluster to improve deep learning performance. The head node trains the neural network that the agent uses to make decisions. The head node also collects data points from the worker nodes to train the neural network.

The head compute uses a single [STANDARD_NC6](#) virtual machine (VM). It has 6 virtual CPUs to distribute work across.

```

from azureml.core.compute import AmlCompute, ComputeTarget

# choose a name for the Ray head cluster
head_compute_name = 'head-gpu'
head_compute_min_nodes = 0
head_compute_max_nodes = 2

# This example uses GPU VM. For using CPU VM, set SKU to STANDARD_D2_V2
head_vm_size = 'STANDARD_NC6'

if head_compute_name in ws.compute_targets:
    head_compute_target = ws.compute_targets[head_compute_name]
    if head_compute_target and type(head_compute_target) is AmlCompute:
        print(f'found head compute target. just use it {head_compute_name}')
else:
    print('creating a new head compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(vm_size = head_vm_size,
                                                               min_nodes = head_compute_min_nodes,
                                                               max_nodes = head_compute_max_nodes,
                                                               vnet_resourcegroup_name = ws.resource_group,
                                                               vnet_name = vnet_name,
                                                               subnet_name = 'default')

# create the cluster
head_compute_target = ComputeTarget.create(ws, head_compute_name, provisioning_config)

# can poll for a minimum number of nodes and for a specific timeout.
# if no min node count is provided it will use the scale settings for the cluster
head_compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)

# For a more detailed view of current AmlCompute status, use get_status()
print(head_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](#).

Worker computing cluster

This example uses four [STANDARD_D2_V2 VMs](#) for the worker compute target. Each worker node has 2 available CPUs for a total of 8 available CPUs.

GPUs aren't necessary for the worker nodes since they aren't performing deep learning. The workers run the game simulations and collect data.

```

# choose a name for your Ray worker cluster
worker_compute_name = 'worker-cpu'
worker_compute_min_nodes = 0
worker_compute_max_nodes = 4

# This example uses CPU VM. For using GPU VM, set SKU to STANDARD_NC6
worker_vm_size = 'STANDARD_D2_V2'

# Create the compute target if it hasn't been created already
if worker_compute_name in ws.compute_targets:
    worker_compute_target = ws.compute_targets[worker_compute_name]
    if worker_compute_target and type(worker_compute_target) is AmlCompute:
        print(f'found worker compute target. just use it {worker_compute_name}')
else:
    print('creating a new worker compute target...')
provisioning_config = AmlCompute.provisioning_configuration(vm_size = worker_vm_size,
                                                               min_nodes = worker_compute_min_nodes,
                                                               max_nodes = worker_compute_max_nodes,
                                                               vnet_resourcegroup_name = ws.resource_group,
                                                               vnet_name = vnet_name,
                                                               subnet_name = 'default')

# create the cluster
worker_compute_target = ComputeTarget.create(ws, worker_compute_name, provisioning_config)

# can poll for a minimum number of nodes and for a specific timeout.
# if no min node count is provided it will use the scale settings for the cluster
worker_compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)

# For a more detailed view of current AmlCompute status, use get_status()
print(worker_compute_target.get_status().serialize())

```

Create a reinforcement learning estimator

Use the [ReinforcementLearningEstimator](#) to submit a training job to Azure Machine Learning.

Azure Machine Learning uses estimator classes to encapsulate run configuration information. This lets you specify how to configure a script execution.

Define a worker configuration

The WorkerConfiguration object tells Azure Machine Learning how to initialize the worker cluster that runs the entry script.

```

# Pip packages we will use for both head and worker
pip_packages=["ray[rllib]==0.8.3"] # Latest version of Ray has fixes for issues related to object transfers

# Specify the Ray worker configuration
worker_conf = WorkerConfiguration(
    # Azure ML compute cluster to run Ray workers
    compute_target=worker_compute_target,
    # Number of worker nodes
    node_count=4,
    # GPU
    use_gpu=False,
    # PIP packages to use
    pip_packages=pip_packages
)

```

Define script parameters

The entry script `pong_rllib.py` accepts a list of parameters that defines how to execute the training job. Passing these parameters through the estimator as a layer of encapsulation makes it easy to change script parameters and run configurations independently of each other.

Specifying the correct `num_workers` makes the most out of your parallelization efforts. Set the number of workers to the same as the number of available CPUs. For this example, you can use the following calculation:

The head node is a [Standard_NC6](#) with 6 vCPUs. The worker cluster is 4 [Standard_D2_V2 VMs](#) with 2 CPUs each, for a total of 8 CPUs. However, you must subtract 1 CPU from the worker count since 1 must be dedicated to the head node role.

6 CPUs + 8 CPUs - 1 head CPU = 13 simultaneous workers. Azure Machine Learning uses head and worker clusters to distinguish compute resources. However, Ray does not distinguish between head and workers, and all CPUs are available as worker threads.

```
training_algorithm = "IMPALA"
rl_environment = "PongNoFrameskip-v4"

# Training script parameters
script_params = {

    # Training algorithm, IMPALA in this case
    "--run": training_algorithm,

    # Environment, Pong in this case
    "--env": rl_environment,

    # Add additional single quotes at the both ends of string values as we have spaces in the
    # string parameters, outermost quotes are not passed to scripts as they are not actually part of string
    # Number of GPUs
    # Number of ray workers
    "--config": '\'{\"num_gpus\": 1, \"num_workers\": 13}\'',

    # Target episode reward mean to stop the training
    # Total training time in seconds
    "--stop": '\'{\"episode_reward_mean\": 18, \"time_total_s\": 3600}\'',

}
```

Define the reinforcement learning estimator

Use the parameter list and the worker configuration object to construct the estimator.

```

# RL estimator
rl_estimator = ReinforcementLearningEstimator(
    # Location of source files
    source_directory='files',
    # Python script file
    entry_script="pong_rllib.py",
    # Parameters to pass to the script file
    # Defined above.
    script_params=script_params,
    # The Azure ML compute target set up for Ray head nodes
    compute_target=head_compute_target,
    # Pip packages
    pip_packages=pip_packages,
    # GPU usage
    use_gpu=True,
    # RL framework. Currently must be Ray.
    rl_framework=Ray(),
    # Ray worker configuration defined above.
    worker_configuration=worker_conf,
    # How long to wait for whole cluster to start
    cluster_coordination_timeout_seconds=3600,
    # Maximum time for the whole Ray job to run
    # This will cut off the run after an hour
    max_run_duration_seconds=3600,
    # Allow the docker container Ray runs in to make full use
    # of the shared memory available from the host OS.
    shm_size=24*1024*1024*1024
)

```

Entry script

The [entry script](#) `pong_rllib.py` trains a neural network using the [OpenAI Gym environment](#) `PongNoFrameSkip-v4`. OpenAI Gyms are standardized interfaces to test reinforcement learning algorithms on classic Atari games.

This example uses a training algorithm known as [IMPALA](#) (Importance Weighted Actor-Learner Architecture). IMPALA parallelizes each individual learning actor to scale across many compute nodes without sacrificing speed or stability.

[Ray Tune](#) orchestrates the IMPALA worker tasks.

```

import ray
import ray.tune as tune
from ray.rllib import train

import os
import sys

from azureml.core import Run
from utils import callbacks

DEFAULT_RAY_ADDRESS = 'localhost:6379'

if __name__ == "__main__":

    # Parse arguments
    train_parser = train.create_parser()

    args = train_parser.parse_args()
    print("Algorithm config:", args.config)

    if args.ray_address is None:
        args.ray_address = DEFAULT_RAY_ADDRESS

    ray.init(address=args.ray_address)

    tune.run(run_or_experiment=args.run,
             config={
                 "env": args.env,
                 "num_gpus": args.config["num_gpus"],
                 "num_workers": args.config["num_workers"],
                 "callbacks": {"on_train_result": callbacks.on_train_result},
                 "sample_batch_size": 50,
                 "train_batch_size": 1000,
                 "num_sgd_iter": 2,
                 "num_data_loader_buffers": 2,
                 "model": {
                     "dim": 42
                 },
                 "stop": args.stop,
                 "local_dir": './logs')

```

Logging callback function

The entry script uses a utility function to define a [custom RLib callback function](#) to log metrics to your Azure Machine Learning workspace. Learn how to view these metrics in the [Monitor and view results](#) section.

```

'''RLlib callbacks module:
Common callback methods to be passed to RLlib trainer.
...
from azureml.core import Run

def on_train_result(info):
    '''Callback on train result to record metrics returned by trainer.
    ...
    run = Run.get_context()
    run.log(
        name='episode_reward_mean',
        value=info["result"]["episode_reward_mean"])
    run.log(
        name='episodes_total',
        value=info["result"]["episodes_total"])

```

Submit a run

Run handles the run history of in-progress or complete jobs.

```
run = exp.submit(config=rl_estimator)
```

NOTE

The run may take up to 30 to 45 minutes to complete.

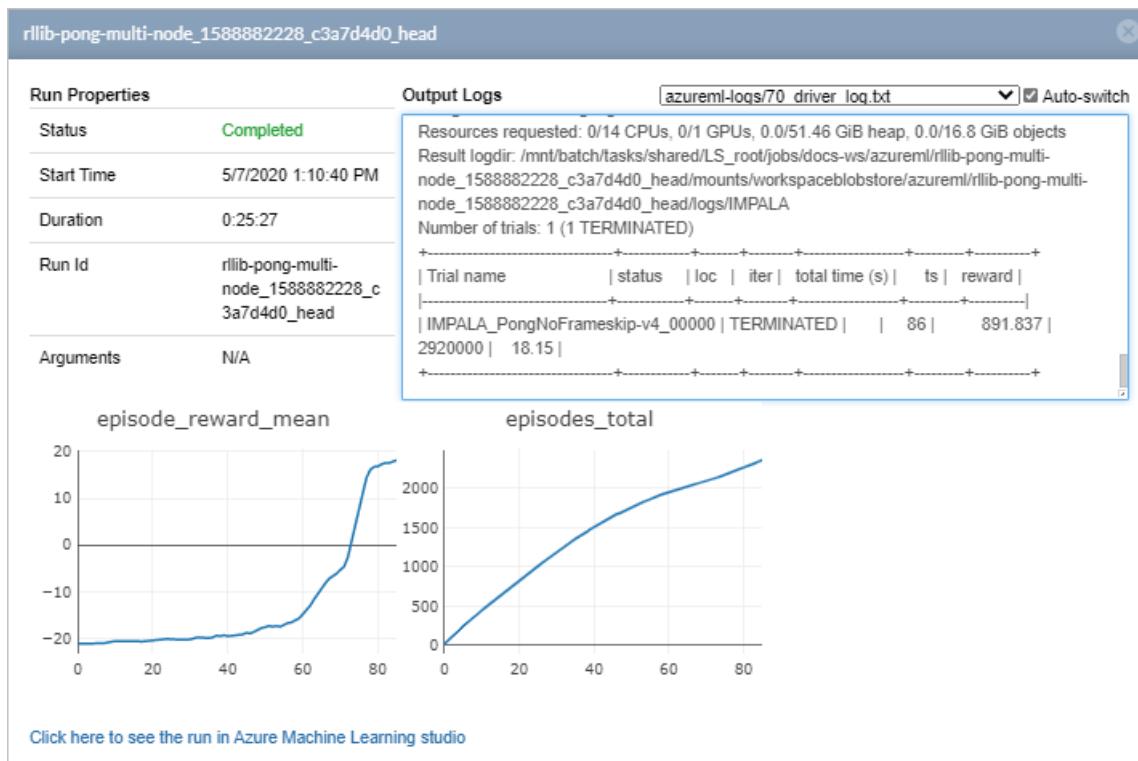
Monitor and view results

Use the Azure Machine Learning Jupyter widget to see the status of your runs in real time. The widget shows two child runs: one for head and one for workers.

```
from azureml.widgets import RunDetails  
  
RunDetails(run).show()  
run.wait_for_completion()
```

1. Wait for the widget to load.
2. Select the head run in the list of runs.

Select [Click here to see the run in Azure Machine Learning studio](#) for additional run information in the studio. You can access this information while the run is in progress or after it completes.



The **episode_reward_mean** plot shows the mean number of points scored per training epoch. You can see that the training agent initially performed poorly, losing its matches without scoring a single point (shown by a reward_mean of -21). Within 100 iterations, the training agent learned to beat the computer opponent by an average of 18 points.

If you browse logs of the child run, you can see the evaluation results recorded in driver_log.txt file. You may need to wait several minutes before these metrics become available on the Run page.

In short work, you have learned to configure multiple compute resources to train a reinforcement learning agent to play Pong very well against a computer opponent.

Next steps

In this article, you learned how to train a reinforcement learning agent using an IMPALA learning agent. To see additional examples, go to the [Azure Machine Learning Reinforcement Learning GitHub repository](#).

Configure automated ML experiments in Python

12/23/2020 • 19 minutes to read • [Edit Online](#)

In this guide, learn how to define various configuration settings of your automated machine learning experiments with the [Azure Machine Learning SDK](#). Automated machine learning picks an algorithm and hyperparameters for you and generates a model ready for deployment. There are several options that you can use to configure automated machine learning experiments.

To view examples of automated machine learning experiments, see [Tutorial: Train a classification model with automated machine learning](#) or [Train models with automated machine learning in the cloud](#).

Configuration options available in automated machine learning:

- Select your experiment type: Classification, Regression, or Time Series Forecasting
- Data source, formats, and fetch data
- Choose your compute target: local or remote
- Automated machine learning experiment settings
- Run an automated machine learning experiment
- Explore model metrics
- Register and deploy model

If you prefer a no code experience, you can also [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 an Azure Machine Learning workspace](#).
- The Azure Machine Learning Python SDK installed. To install the SDK you can either,
 - Create a compute instance, which automatically installs the SDK and is preconfigured for ML workflows. See [Create and manage an Azure Machine Learning compute instance](#) for more information.
 - [Install the `automl` package yourself](#), which includes the [default installation](#) of the SDK.

Select your experiment type

Before you begin your experiment, you should determine the kind of machine learning problem you are solving. Automated machine learning supports task types of `classification`, `regression`, and `forecasting`. Learn more about [task types](#).

The following code uses the `task` parameter in the `AutoMLConfig` constructor to specify the experiment type as `classification`.

```
from azureml.train.automl import AutoMLConfig

# task can be one of classification, regression, forecasting
automl_config = AutoMLConfig(task = "classification")
```

Data source and format

Automated machine learning supports data that resides on your local desktop or in the cloud such as Azure Blob Storage. The data can be read into a **Pandas DataFrame** or an **Azure Machine Learning TabularDataset**. [Learn more about datasets.](#)

Requirements for training data:

- Data must be in tabular form.
- The value to predict, target column, must be in the data.

For **remote experiments**, training data must be accessible from the remote compute. AutoML only accepts [Azure Machine Learning TabularDatasets](#) when working on a remote compute.

Azure Machine Learning datasets expose functionality to:

- Easily transfer data from static files or URL sources into your workspace.
- Make your data available to training scripts when running on cloud compute resources. See [How to train with datasets](#) for an example of using the `Dataset` class to mount data to your remote compute target.

The following code creates a `TabularDataset` from a web url. See [Create a TabularDatasets](#) for code examples on how to create datasets from other sources like local files and datastores.

```
from azureml.core.dataset import Dataset
data = "https://automlsamplenotebookdata.blob.core.windows.net/automl-sample-notebook-data/creditcard.csv"
dataset = Dataset.Tabular.from_delimited_files(data)
```

For **local compute experiments**, we recommend pandas dataframes for faster processing times.

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("your-local-file.csv")
train_data, test_data = train_test_split(df, test_size=0.1, random_state=42)
label = "label-col-name"
```

Training, validation, and test data

You can specify separate **training** and **validation** sets directly in the `AutoMLConfig` constructor. Learn more about [how to configure data splits and cross validation](#) for your AutoML experiments.

If you do not explicitly specify a `validation_data` or `n_cross_validation` parameter, AutoML 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.

TRAINING DATA SIZE	VALIDATION TECHNIQUE
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.

At this time, you need to provide your own **test data** for model evaluation. For a code example of bringing your own test data for model evaluation see the **Test** section of [this Jupyter notebook](#).

Compute to run experiment

Next determine where the model will be trained. An automated machine learning training experiment can run on the following compute options. Learn the [pros and cons of local and remote compute options](#).

- Your **local** machine such as a local desktop or laptop – Generally when you have a small dataset and you are still in the exploration stage. See [this notebook](#) for a local compute example.
- A **remote** machine in the cloud – [Azure Machine Learning Managed Compute](#) is a managed service that enables the ability to train machine learning models on clusters of Azure virtual machines.
See [this notebook](#) for a remote example using Azure Machine Learning Managed Compute.
- An **Azure Databricks cluster** in your Azure subscription. You can find more details in [Set up an Azure Databricks cluster for automated ML](#). See this [GitHub site](#) for examples of notebooks with Azure Databricks.

Configure your experiment settings

There are several options that you can use to configure your automated machine learning experiment. These parameters are set by instantiating an `AutoMLConfig` object. See the [AutoMLConfig class](#) for a full list of parameters.

Some examples include:

1. Classification experiment using AUC weighted as the primary metric with experiment timeout minutes set to 30 minutes and 2 cross-validation folds.

```
automl_classifier=AutoMLConfig(task='classification',
                                primary_metric='AUC_weighted',
                                experiment_timeout_minutes=30,
                                blocked_models=['XGBoostClassifier'],
                                training_data=train_data,
                                label_column_name=label,
                                n_cross_validations=2)
```

2. The following example is a regression experiment set to end after 60 minutes with five validation cross folds.

```

automl_regressor = AutoMLConfig(task='regression',
                                 experiment_timeout_minutes=60,
                                 allowed_models=['KNN'],
                                 primary_metric='r2_score',
                                 training_data=train_data,
                                 label_column_name=label,
                                 n_cross_validations=5)

```

3. Forecasting tasks require extra setup, see the [Autotrain a time-series forecast model](#) article for more details.

```

time_series_settings = {
    'time_column_name': time_column_name,
    'time_series_id_column_names': time_series_id_column_names,
    'drop_column_names': ['logQuantity'],
    'forecast_horizon': n_test_periods
}

automl_config = AutoMLConfig(task = 'forecasting',
                             debug_log='automl_oj_sales_errors.log',
                             primary_metric='normalized_root_mean_squared_error',
                             experiment_timeout_minutes=20,
                             training_data=train_data,
                             label_column_name=label,
                             n_cross_validations=5,
                             path=project_folder,
                             verbosity=logging.INFO,
                             **time_series_settings)

```

Supported models

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 three different `task` parameter values determine the list of algorithms, or models, to apply. Use the `allowed_models` or `blocked_models` parameters to further modify iterations with the available models to include or exclude.

The following table summarizes the supported models by task type.

NOTE

If you plan to export your auto ML created models to an [ONNX model](#), only those algorithms indicated with an * are able to be converted to the ONNX format. Learn more about [converting models to ONNX](#).

Also note, ONNX only supports classification and regression tasks at this time.

CLASSIFICATION	REGRESSION	TIME SERIES FORECASTING
Logistic Regression*	Elastic Net*	Elastic Net
Light GBM*	Light GBM*	Light GBM
Gradient Boosting*	Gradient Boosting*	Gradient Boosting
Decision Tree*	Decision Tree*	Decision Tree

CLASSIFICATION	REGRESSION	TIME SERIES FORECASTING
K Nearest Neighbors*	K Nearest Neighbors*	K Nearest Neighbors
Linear SVC*	LARS Lasso*	LARS Lasso
Support Vector Classification (SVC)*	Stochastic Gradient Descent (SGD)*	Stochastic Gradient Descent (SGD)
Random Forest*	Random Forest*	Random Forest
Extremely Randomized Trees*	Extremely Randomized Trees*	Extremely Randomized Trees
Xgboost*	Xgboost*	Xgboost
Averaged Perceptron Classifier	Online Gradient Descent Regressor	Auto-ARIMA
Naive Bayes*	Fast Linear Regressor	Prophet
Stochastic Gradient Descent (SGD)*		ForecastTCN
Linear SVM Classifier*		

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, and the following table shows valid primary metrics for each task type.

Learn about the specific definitions of these metrics in [Understand automated machine learning results](#).

CLASSIFICATION	REGRESSION	TIME SERIES FORECASTING
accuracy	spearman_correlation	spearman_correlation
AUC_weighted	normalized_root_mean_squared_error	normalized_root_mean_squared_error
average_precision_score_weighted	r2_score	r2_score
norm_macro_recall	normalized_mean_absolute_error	normalized_mean_absolute_error
precision_score_weighted		

Data featurization

In every automated machine learning experiment, your data is automatically scaled and normalized to help *certain* algorithms that are sensitive to features that are on different scales. This scaling and normalization is referred to as featurization. See [Featurization in AutoML](#) for more detail and code examples.

When configuring your experiments in your `AutoMLConfig` object, you can enable/disable the setting `featurization`. The following table shows the accepted settings for featurization in the `AutoMLConfig` object.

FEATURIZATION CONFIGURATION	DESCRIPTION

FEATURIZATION CONFIGURATION	DESCRIPTION
"featurization": "auto"	Indicates that as part of preprocessing, data guardrails and featurization steps are performed automatically. Default setting.
"featurization": "off"	Indicates featurization step shouldn't be done automatically.
"featurization": 'FeaturizationConfig'	Indicates customized featurization step should be used. Learn how to customize 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.

Ensemble configuration

Ensemble models are enabled by default, and appear as the final run iterations in an AutoML run. Currently **VotingEnsemble** and **StackEnsemble** are supported.

Voting implements soft-voting, which uses weighted averages. The stacking implementation uses a two layer implementation, where the first layer has the same models as the voting ensemble, and the second layer model is used to find the optimal combination of the models from the first layer.

If you are using ONNX models, or have model-explainability enabled, stacking is disabled and only voting is utilized.

Ensemble training can be disabled by using the `enable_voting_ensemble` and `enable_stack_ensemble` boolean parameters.

```
automl_classifier = AutoMLConfig(
    task='classification',
    primary_metric='AUC_weighted',
    experiment_timeout_minutes=30,
    training_data=data_train,
    label_column_name=label,
    n_cross_validations=5,
    enable_voting_ensemble=False,
    enable_stack_ensemble=False
)
```

To alter the default ensemble behavior, there are multiple default arguments that can be provided as `kwargs` in an `AutoMLConfig` object.

IMPORTANT

The following parameters aren't explicit parameters of the `AutoMLConfig` class.

- `ensemble_download_models_timeout_sec`: During **VotingEnsemble** and **StackEnsemble** model generation, multiple fitted models from the previous child runs are downloaded. If you encounter this error: `AutoMLEnsembleException: Could not find any models for running ensembling`, then you may need to provide more time for the models to be downloaded. The default value is 300 seconds for downloading these models in parallel and there is no maximum timeout limit. Configure this parameter with a higher

value than 300 secs, if more time is needed.

NOTE

If the timeout is reached and there are models downloaded, then the ensembling proceeds with as many models it has downloaded. It's not required that all the models need to be downloaded to finish within that timeout.

The following parameters only apply to `StackEnsemble` models:

- `stack_meta_learner_type`: the meta-learner is a model trained on the output of the individual heterogeneous models. Default meta-learners are `LogisticRegression` for classification tasks (or `LogisticRegressionCV` if cross-validation is enabled) and `ElasticNet` for regression/forecasting tasks (or `ElasticNetCV` if cross-validation is enabled). This parameter can be one of the following strings: `LogisticRegression`, `LogisticRegressionCV`, `LightGBMClassifier`, `ElasticNet`, `ElasticNetCV`, `LightGBMRegressor`, or `LinearRegression`.
- `stack_meta_learner_train_percentage`: specifies the proportion of the training set (when choosing train and validation type of training) to be reserved for training the meta-learner. Default value is `0.2`.
- `stack_meta_learner_kwargs`: optional parameters to pass to the initializer of the meta-learner. These parameters and parameter types mirror the parameters and parameter types from the corresponding model constructor, and are forwarded to the model constructor.

The following code shows an example of specifying custom ensemble behavior in an `AutoMLConfig` object.

```
ensemble_settings = {
    "ensemble_download_models_timeout_sec": 600
    "stack_meta_learner_type": "LogisticRegressionCV",
    "stack_meta_learner_train_percentage": 0.3,
    "stack_meta_learner_kwargs": {
        "refit": True,
        "fit_intercept": False,
        "class_weight": "balanced",
        "multi_class": "auto",
        "n_jobs": -1
    }
}

automl_classifier = AutoMLConfig(
    task='classification',
    primary_metric='AUC_weighted',
    experiment_timeout_minutes=30,
    training_data=train_data,
    label_column_name=label,
    n_cross_validations=5,
    **ensemble_settings
)
```

Exit criteria

There are a few options you can define in your `AutoMLConfig` to end your experiment.

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.

CRITERIA	DESCRIPTION
After a length of time	<p>Use <code>experiment_timeout_minutes</code> in your settings to define how long, in minutes, your experiment should continue to run.</p> <p>To help avoid experiment time out failures, there is a minimum of 15 minutes, or 60 minutes if your row by column size exceeds 10 million.</p>
A score has been reached	<p>Use <code>experiment_exit_score</code> completes the experiment after a specified primary metric score has been reached.</p>

Run experiment

For automated ML, you create an `Experiment` object, which is a named object in a `Workspace` used to run experiments.

```
from azureml.core.experiment import Experiment

ws = Workspace.from_config()

# Choose a name for the experiment and specify the project folder.
experiment_name = 'automl-classification'
project_folder = './sample_projects/automl-classification'

experiment = Experiment(ws, experiment_name)
```

Submit the experiment to run and generate a model. Pass the `AutoMLConfig` to the `submit` method to generate the model.

```
run = experiment.submit(automl_config, show_output=True)
```

NOTE

Dependencies are first installed on a new machine. It may take up to 10 minutes before output is shown. Setting `show_output` to `True` results in output being shown on the console.

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 your `AutoMLConfig` object. If it is not configured, then by default only one concurrent child run/iteration is allowed per experiment.

Explore models and metrics

You can view your training results in a widget or inline if you are in a notebook. See [Track and evaluate models](#) for more details.

See [Evaluate automated machine learning experiment results](#) for definitions and examples of the performance charts and metrics provided for each run.

To get a featurization summary and understand what features were added to a particular model, see [Featurization transparency](#).

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.

Register and deploy models

For details on how to download or register a model for deployment to a web service, see [how and where to deploy a model](#).

Model interpretability

Model interpretability allows you to understand why your models made predictions, and the underlying feature importance values. The SDK includes various packages for enabling model interpretability features, both at training and inference time, for local and deployed models.

See the [how-to](#) for code samples on how to enable interpretability features specifically within automated machine learning experiments.

For general information on how model explanations and feature importance can be enabled in other areas of the SDK outside of automated machine learning, see the [concept](#) article on interpretability.

NOTE

The ForecastTCN model is not currently supported by the Explanation Client. This model will not return an explanation dashboard if it is returned as the best model, and does not support on-demand explanation runs.

Troubleshooting

- Recent upgrade of `AutoML` dependencies to newer versions will be breaking compatibility: As of version 1.13.0 of the SDK, models won't be loaded in older SDKs due to incompatibility between the older versions we pinned in our previous packages, and the newer versions we pin now. You will see error such as:

- Module not found: Ex. `No module named 'sklearn.decomposition._truncated_svd'`,
- Import errors: Ex. `ImportError: cannot import name 'RollingOriginValidator'`,
- Attribute errors: Ex. `AttributeError: 'SimpleImputer' object has no attribute 'add_indicator'`

To work around this issue, take either of the following two steps depending 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

`sckit-learn==0.22.1`. If there is a version mismatch, upgrade scikit-learn and/or pandas to correct version as shown below:

```
pip install --upgrade pandas==0.25.1
pip install --upgrade scikit-learn==0.22.1
```

- o If your `AutoML` SDK training version is less than or equal to 1.12.0, you need `pandas == 0.23.4` and `sckit-learn==0.20.3`. If there is a version mismatch, downgrade scikit-learn and/or pandas to correct version as shown below:

```
pip install --upgrade pandas==0.23.4
pip install --upgrade scikit-learn==0.20.3
```

- **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
- **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.
 - **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'])
```

- **Experiment Charts:** Binary classification charts (precision-recall, ROC, gain curve etc.) shown in automated ML experiment iterations are not rendering correctly in user interface since 4/12. Chart plots are currently showing inverse results, where better performing models are shown with lower results. A resolution is under investigation.
- **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))
```

- **automl_setup fails:**

- On Windows, run automl_setup from an Anaconda Prompt. Use this link to [install Miniconda](#).
- Ensure that conda 64-bit is installed, rather than 32-bit by running the `conda info` command. The `platform` should be `win-64` for Windows or `osx-64` for Mac.
- Ensure that conda 4.4.10 or later is installed. You can check the version with the command `conda -v`. If you have a previous version installed, you can update it by using the command: `conda update conda`.
- Linux - `gcc: error trying to exec 'cc1plus'`
 - If the `gcc: error trying to exec 'cc1plus': execvp: No such file or directory` error is encountered, install build essentials using the command `sudo apt-get install build-essential`.
 - 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_linus.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 VM, 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.

- `import AutoMLConfig` fails: There were package changes in the automated machine learning version 1.0.76, which require the previous version to be uninstalled before updating to the new version. If the `ImportError: cannot import name AutoMLConfig` is encountered 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.

- **workspace.from_config fails:** If the calls `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.

- **Sample notebook fails:** 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. So, if you switch to a new conda environment, you will have 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`.
 - Ensure the notebook is for the SDK version that you are using. You can 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 by clicking the `Branch` button, selecting the `Tags` tab and then selecting the version.
- `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 may 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.

Next steps

- Learn more about [how and where to deploy a model](#).
- 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 how to train multiple models with AutoML in the [Many Models Solution Accelerator](#).

Create, review, and deploy automated machine learning models with Azure Machine Learning

12/23/2020 • 9 minutes to read • [Edit Online](#)

In this article, you learn how to create, explore, and deploy automated machine learning models without a single line of code in Azure Machine Learning studio.

Automated machine learning 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 automated machine learning](#).

For an end to end example, try the [tutorial for creating a classification model with Azure Machine Learning's automated ML interface](#).

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 an Azure Machine Learning workspace](#).

Get started

1. Sign in to Azure Machine Learning at <https://ml.azure.com>.
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. On the left, there's a sidebar with various navigation options like 'New', 'Home', 'Notebooks', 'Automated ML' (which is highlighted with a red box), 'Designer', 'Assets', 'Datasets', 'Experiments', 'Pipelines', 'Models', and 'Endpoints'. Below that is a 'Manage' section with 'Compute', 'Datastores', and 'Data Labeling'. 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 '+ New Automated ML run' (also highlighted with a red box) and a message 'No recent Automated ML runs to display.' with a link to 'Create your first run'. There's also a 'Learn more on creating Automated ML runs' link. At the bottom of this section is a 'Documentation' block 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.

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 machine learning experiments, including those created with the SDK.

Create and run experiment

1. Select **+ New automated ML run** and populate the form.
2. Select a dataset from your storage container, or create a new dataset. Datasets can be created from local files, web urls, datastores, or Azure open datasets. Learn more about [dataset 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 run** form, enter a unique experiment name.
5. Select a target column; this is the column that you would like to do predictions on.
6. Select a compute for the data profiling and training job. A list of your existing computes is available in the dropdown. To create a new compute, follow the instructions in step 7.
7. 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.

FIELD	DESCRIPTION
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 AML 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**.

8. 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,

- a. Enable deep learning.

b. Select *time column*: This column contains the time data to be used.

c. 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 will become.

[Learn more about forecasting and forecast horizon](#).

9. (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 models for each task type .

ADDITIONAL CONFIGURATIONS	DESCRIPTION
Exit criterion	<p>When any of these criteria are met, the training job is stopped.</p> <p><i>Training job time (hours)</i>: How long to allow the training job to run.</p> <p><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.</p>
Validation	<p>Select one of the cross validation options to use in the training job.</p> <p>Learn more about cross validation.</p> <p>Forecasting only supports k-fold cross validation.</p>
Concurrency	<p><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 runs on clusters.</p>

10. (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) (i)



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](#)

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

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 **Run Detail** screen opens to the **Details** tab. This screen shows you a summary of the experiment run including a status bar at the top next to the run 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.

automl_ws > Automated ML (preview) > new-experiment > Run 1

Run 1 Running

↻ Refresh ✖ Cancel

Details Data guardrails Models Outputs + Logs Child runs Snapshot

Properties

Status
Running

Created
Jul 7, 2020 4:37 PM

Compute target
[automl-compute](#)

Run ID
AutoML_badd2cb1-c415-44c2-b0a8-1e6ee55fd8b8

Run number
1

Script name
--

Created by
Nina Baccam

Input datasets
Input name: input_data, [5f9ae693-c4cc-4f14-97ae-442ec7297840](#)
ID: [442ec7297840](#)

[Output datasets](#)

Run summary

Task type
Classification View all run settings

Primary metric
AUC weighted

Run status
Running

Experiment name
new-experiment

View training run details

Drill down on any of the completed models to see training run details, like a model summary on the **Model** tab or performance metric charts on the **Metrics** tab. [Learn more about charts](#).

Run 5 ✓ Completed

↻ Refresh ▷ Deploy ⬇ Download 🔍 Explain model ✖ Cancel

Details Model Explanations (preview) Metrics Outputs + logs Images Child runs Snapshot

Model summary

Algorithm name
MaxAbsScaler, LightGBM

AUC weighted
0.93702 ☰ View all other metrics

Sampling
100% ⓘ

Registered models
No registration yet

Deploy status
No deployment yet

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.

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.
 - a. After the experiment is complete, navigate to the parent run page by selecting **Run 1** at the top of the screen.
 - b. Select the model listed in the **Best model summary** section.
 - c. Select **Deploy** on the top left of the window.
 - Option 2: To deploy a specific model iteration from this experiment.
 - a. Select the desired model from the **Models** tab
 - b. 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: <i>Azure Kubernetes Service (AKS)</i> or <i>Azure Container Instance (ACI)</i> .
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. 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

12/23/2020 • 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. 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](#).

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	Runtime 7.1 (scala 2.21, spark 3.0.0) - Not ML
Python version	always	3
Worker Type (determines max # of concurrent iterations)	Automated ML only	Memory optimized VM preferred

SETTING	APPLIES TO	VALUE
Workers	always	2 or higher
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:

- a. In the left pane, select **Clusters**.
- b. In the table, select your cluster name.
- c. On the **Libraries** tab, select **Restart**.

A successful install looks like the following:

The screenshot shows the Azure Databricks cluster configuration interface. On the left, there's a sidebar with icons for Home, Workspace, and Recents. The main area shows a cluster named 'adb-aml-demo' with a green lightning bolt icon. Below the cluster name are buttons for Edit, Clone, Restart, Terminate, and Delete. A navigation bar includes Configuration, Notebooks (0), Libraries (1), Event Log, Spark UI, Driver Logs, Apps, and Spark Cluster UI - Master. Under the Libraries section, there are 'Uninstall' and 'Install New' buttons. A table lists the installed library: 'Name' (azureml-sdk[databricks]), 'Type' (PyPI), and 'Status' (Installed).

Add the Azure ML SDK with AutoML to Databricks

If the cluster was created with Databricks Runtime 7.1 or above (*not* ML), run the following command in the first cell of your notebook to install the AML SDK.

```
%pip install --upgrade --force-reinstall -r https://aka.ms/automl_linux_requirements.txt
```

For Databricks Runtime 7.0 and lower, install the Azure Machine Learning SDK using the [init script](#).

AutoML config settings

In AutoML config, when using Azure Databricks add the following parameters:

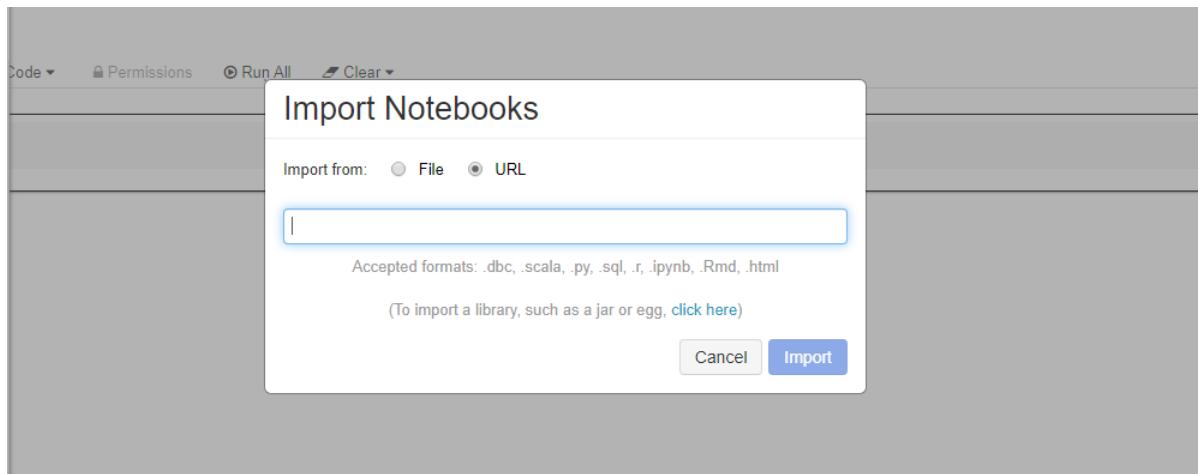
- `max_concurrent_iterations` is based on number of worker nodes in your cluster.
- `spark_context=sc` is based on the default spark context.

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:

The screenshot shows the Azure Workspace interface. On the left, there's a sidebar with icons for Home, Workspace, and Recents. The main area shows a 'Users' list with an item 'username@email..'. A context menu is open over this user entry, showing options: Create, Clone, Import (which is highlighted in blue), Export, and Permissions.



- Learn how to [create a pipeline with Databricks as the training compute](#).

Troubleshooting

- **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 a model](#) on Azure Machine Learning with the MNIST dataset.
- See the [Azure Machine Learning SDK for Python reference](#).

Train models with automated machine learning in the cloud

12/23/2020 • 5 minutes to read • [Edit Online](#)

In Azure Machine Learning, you train your model on different types of compute resources that you manage. The compute target could be a local computer or a resource in the cloud.

You can easily scale up or scale out your machine learning experiment by adding additional compute targets, such as Azure Machine Learning Compute (AmlCompute). AmlCompute is a managed-compute infrastructure that allows you to easily create a single or multi-node compute.

In this article, you learn how to build a model using automated ML with AmlCompute.

How does remote differ from local?

More features are available when you use a remote compute target. For more details, see [Local and remote compute targets](#).

The tutorial "[Train a classification model with automated machine learning](#)" teaches you how to use a local computer to train a model with automated ML. The workflow when training locally also applies to remote targets as well. To train remotely, you first create a remote compute target such as AmlCompute. Then you configure the remote resource and submit your code there.

This article shows the extra steps needed to run an automated ML experiment on a remote AmlCompute target. The workspace object, `ws`, from the tutorial is used throughout the code here.

```
ws = Workspace.from_config()
```

Create resource

Create the `AmlCompute` target in your workspace (`ws`) if it doesn't already exist.

Time estimate: Creation of the AmlCompute target takes approximately 5 minutes.

```

from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
import os

# choose a name for your cluster
compute_name = os.environ.get("AML_COMPUTE_CLUSTER_NAME", "cpu-cluster")
compute_min_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MAX_NODES", 4)

# This example uses CPU VM. For using GPU VM, set SKU to STANDARD_NC6
vm_size = os.environ.get("AML_COMPUTE_CLUSTER_SKU", "STANDARD_D2_V2")

if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('found compute target. just use it. ' + compute_name)
else:
    print('creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(vm_size =
                                                               min_nodes = compute_min_nodes,
                                                               max_nodes = compute_max_nodes)

    # create the cluster
    compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

    # can poll for a minimum number of nodes and for a specific timeout.
    # if no min node count is provided it will use the scale settings for the cluster
    compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)

    # For a more detailed view of current AmlCompute status, use get_status()
    print(compute_target.get_status().serialize())

```

You can now use the `compute_target` object as the remote compute target.

Cluster name restrictions include:

- Must be shorter than 64 characters.
- Cannot include any of the following characters: \ ~ ! @ # \$ % ^ & * () = + _ [] { } \\ | ; : ' " , < > / ? `

Access data using TabularDataset function

Defined training_data as `TabularDataset` and the label, which are passed to Automated ML in the `AutoMLConfig`.

The `TabularDataset` method `from_delimited_files`, by default, sets the `infer_column_types` to true, which will infer the columns type automatically.

If you do wish to manually set the column types, you can set the `set_column_types` argument to manually set the type of each column. In the following code sample, the data comes from the sklearn package.

```

from sklearn import datasets
from azureml.core.dataset import Dataset
from scipy import sparse
import numpy as np
import pandas as pd
import os

# Create a project_folder if it doesn't exist
if not os.path.isdir('data'):
    os.mkdir('data')

if not os.path.exists('project_folder'):
    os.makedirs('project_folder')

X = pd.DataFrame(data_train.data[100:,:])
y = pd.DataFrame(data_train.target[100:])

# merge X and y
label = "digit"
X[label] = y

training_data = X

training_data.to_csv('data/digits.csv')
ds = ws.get_default_datastore()
ds.upload(src_dir='./data', target_path='digitsdata', overwrite=True, show_progress=True)

training_data = Dataset.Tabular.from_delimited_files(path=ds.path('digitsdata/digits.csv'))

```

Configure experiment

Specify the settings for `AutoMLConfig`. (See a [full list of parameters](#) and their possible values.)

```

from azureml.train.automl import AutoMLConfig
import time
import logging

automl_settings = {
    "name": "AutoML_Demo_Experiment_{0}".format(time.time()),
    "experiment_timeout_minutes" : 20,
    "enable_early_stopping" : True,
    "iteration_timeout_minutes": 10,
    "n_cross_validations": 5,
    "primary_metric": 'AUC_weighted',
    "max_concurrent_iterations": 10,
}

automl_config = AutoMLConfig(task='classification',
                             debug_log='automl_errors.log',
                             path=project_folder,
                             compute_target=compute_target,
                             training_data=training_data,
                             label_column_name=label,
                             **automl_settings,
)

```

Submit training experiment

Now submit the configuration to automatically select the algorithm, hyper parameters, and train the model.

```
from azureml.core.experiment import Experiment
experiment = Experiment(ws, 'automl_remote')
remote_run = experiment.submit(automl_config, show_output=True)
```

You will see output similar to the following example:

```
Running on remote compute: mydsvmParent Run ID: AutoML_015ffe76-c331-406d-9bfd-0fd42d8ab7f6
*****
ITERATION: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
*****
```

ITERATION	PIPELINE	DURATION	METRIC	BEST
2	Standardize SGD classifier	0:02:36	0.954	0.954
7	Normalizer DT	0:02:22	0.161	0.954
0	Scale MaxAbs 1 extra trees	0:02:45	0.936	0.954
4	Robust Scaler SGD classifier	0:02:24	0.867	0.954
1	Normalizer kNN	0:02:44	0.984	0.984
9	Normalizer extra trees	0:03:15	0.834	0.984
5	Robust Scaler DT	0:02:18	0.736	0.984
8	Standardize kNN	0:02:05	0.981	0.984
6	Standardize SVM	0:02:18	0.984	0.984
10	Scale MaxAbs 1 DT	0:02:18	0.077	0.984
11	Standardize SGD classifier	0:02:24	0.863	0.984
3	Standardize gradient boosting	0:03:03	0.971	0.984
12	Robust Scaler logistic regression	0:02:32	0.955	0.984
14	Scale MaxAbs 1 SVM	0:02:15	0.989	0.989
13	Scale MaxAbs 1 gradient boosting	0:02:15	0.971	0.989
15	Robust Scaler kNN	0:02:28	0.904	0.989
17	Standardize kNN	0:02:22	0.974	0.989
16	Scale 0/1 gradient boosting	0:02:18	0.968	0.989
18	Scale 0/1 extra trees	0:02:18	0.828	0.989
19	Robust Scaler kNN	0:02:32	0.983	0.989

Explore results

You can use the same [Jupyter widget](#) as shown in [the training tutorial](#) to see a graph and table of results.

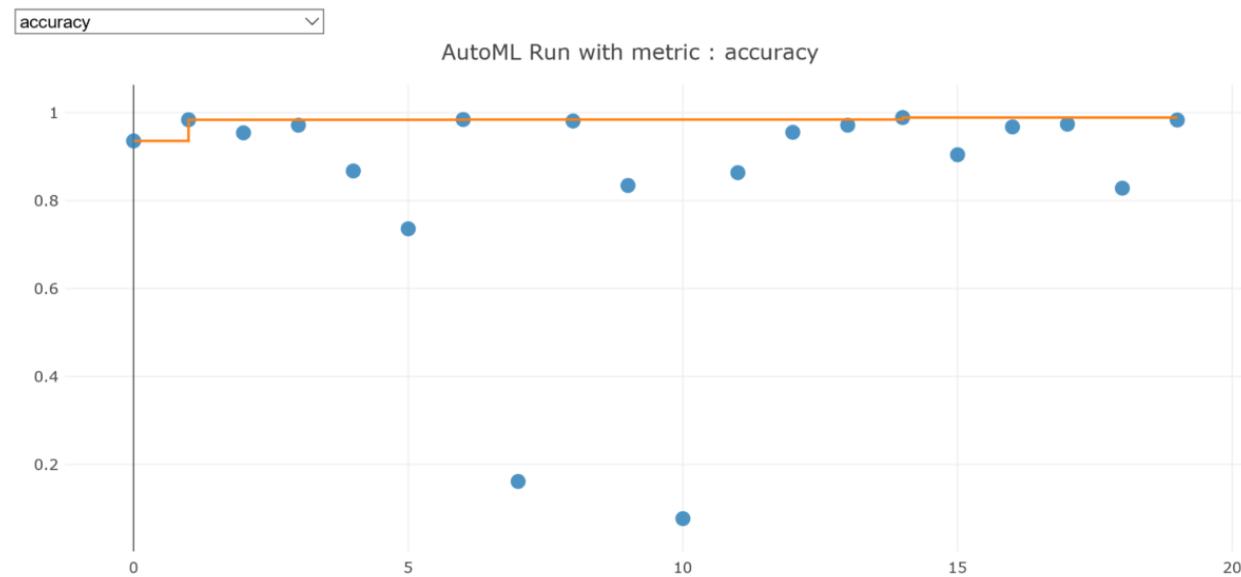
```
from azureml.widgets import RunDetails
RunDetails(remote_run).show()
```

Here is a static image of the widget. In the notebook, you can click on any line in the table to see run properties and output logs for that run. You can also use the dropdown above the graph to view a graph of each available metric for each iteration.



Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started
0	Scale MaxAbs 1, extra trees	0.93564621	0.93564621	Completed	0:02:45	Sep 5, 2018 9:58 PM
1	Normalizer, kNN	0.98376915	0.98376915	Completed	0:02:44	Sep 5, 2018 9:58 PM
2	Standardize, SGD classifier	0.95410701	0.98376915	Completed	0:02:36	Sep 5, 2018 9:58 PM
3	Standardize, gradient boosting	0.97145517	0.98376915	Completed	0:03:03	Sep 5, 2018 9:58 PM
4	Robust Scaler, SGD classifier	0.86735521	0.98376915	Completed	0:02:24	Sep 5, 2018 9:58 PM

Pages: 1 2 3 4 Next Last 1 of 4



[Click here to see the run in Azure portal](#)

The widget displays a URL you can use to see and explore the individual run details.

If you aren't in a Jupyter notebook, you can display the URL from the run itself:

```
remote_run.get_portal_url()
```

The same information is available in your workspace. To learn more about these results, see [Evaluate automated machine learning results](#).

Example

The following [notebook](#) demonstrates concepts in this article.

Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service](#).

Next steps

- Learn [how to configure settings for automatic training](#).
- See the [how-to](#) on enabling model interpretability features in automated ML experiments.

Auto-train a time-series forecast model

12/23/2020 • 14 minutes to read • [Edit Online](#)

In this article, you learn how to configure and train a time-series forecasting regression model using automated machine learning, AutoML, 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 machine learning 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 an Azure Machine Learning workspace](#).
- 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.

Preparing data

The most important difference between a forecasting regression task type and regression task type within AutoML is including a feature in your 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.

Consider the following snapshot of a file `sample.csv`. This data set is of daily sales data for a company that has two different stores, A, and B.

Additionally, there are features for

- `week_of_year` : allows the model to detect weekly seasonality.
- `day_datetime` : represents a clean time series with daily frequency.
- `sales_quantity` : the target column for running predictions.

```
day_datetime,store,sales_quantity,week_of_year  
9/3/2018,A,2000,36  
9/3/2018,B,600,36  
9/4/2018,A,2300,36  
9/4/2018,B,550,36  
9/5/2018,A,2100,36  
9/5/2018,B,650,36  
9/6/2018,A,2400,36  
9/6/2018,B,700,36  
9/7/2018,A,2450,36  
9/7/2018,B,650,36
```

Read the data into a Pandas dataframe, then use the `to_datetime` function to ensure the time series is a `datetime` type.

```
import pandas as pd  
data = pd.read_csv("sample.csv")  
data["day_datetime"] = pd.to_datetime(data["day_datetime"])
```

In this case, the data is already sorted ascending by the time field `day_datetime`. However, when setting up an experiment, ensure the desired time column is sorted in ascending order to build a valid time series.

The following code,

- Assumes the data contains 1,000 records, and makes a deterministic split in the data to create training and test data sets.
- Identifies the label column as `sales_quantity`.
- Separates the label field from `test_data` to form the `test_target` set.

```
train_data = data.iloc[:950]  
test_data = data.iloc[-50:]  
  
label = "sales_quantity"  
  
test_labels = test_data.pop(label).values
```

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.

Training and validation data

You can specify separate train and validation sets 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. Pass the training and validation data together, and set the number of cross validation folds with the `n_cross_validations` parameter in your `AutoMLConfig`. 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

You can also bring your own validation data, learn more in [Configure data splits and cross-validation in AutoML](#).

```
automl_config = AutoMLConfig(task='forecasting',
                               n_cross_validations=3,
                               ...
                               **time_series_settings)
```

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. The following table summarizes this subset of models.

TIP

Traditional regression models are also tested as part of the recommendation system for forecasting experiments. See the [supported model table](#) for the full list of models.

MODELS	DESCRIPTION	BENEFITS
Prophet (Preview)	Prophet works best with time series that have strong seasonal effects and several seasons of historical data. To leverage this model, install it locally using <code>pip install fbprophet</code> .	Accurate & fast, robust to outliers, missing data, and dramatic changes in your time series.
Auto-ARIMA (Preview)	Auto-Regressive Integrated Moving Average (ARIMA) performs best, when the data is stationary. This means that its statistical properties like the mean and variance are constant over the entire set. For example, if you flip a coin, then the probability of you getting heads is 50%, regardless if you flip today, tomorrow, or next year.	Great for univariate series, since the past values are used to predict the future values.
ForecastTCN (Preview)	ForecastTCN is a neural network model designed to tackle the most demanding forecasting tasks, capturing nonlinear local and global trends in your data as well as relationships between time series.	Capable of leveraging complex trends in your data and readily scales to the largest of datasets.

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. For forecasting tasks, there are additional parameters that must be

set that affect the experiment.

The following table summarizes these additional parameters. See the [ForecastingParameter class reference documentation](#) for syntax design patterns.

PARAMETER NAME	DESCRIPTION	REQUIRED
<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.	✓
<code>enable_dnn</code>	Enable Forecasting Models	
<code>time_series_id_column_names</code>	The column name(s) used to uniquely identify the time series in data that has multiple rows with the same timestamp. If time series identifiers are not defined, the data set is assumed to be one time-series. To learn more about single time-series, see the energy_demand_notebook .	
<code>freq</code>	The time series dataset frequency. This parameter represents the period with which events are expected to occur, such as daily, weekly, yearly, etc. The frequency must be a pandas offset alias .	
<code>target_lags</code>	Number of rows to lag the target values based on the frequency of the data. The lag is represented as a list or single integer. Lag should be used when the relationship between the independent variables and dependent variable doesn't match up or correlate by default.	
<code>feature_lags</code>	The features to lag will be automatically decided by automated ML when <code>target_lags</code> are set and <code>feature_lags</code> is set to <code>auto</code> . Enabling feature lags may help to improve accuracy. Feature lags are disabled by default.	

PARAMETER NAME	DESCRIPTION	REQUIRED
<code>target_rolling_window_size</code>	<p><i>n</i> historical periods to use to generate forecasted values, <= training set size. If omitted, <i>n</i> is the full training set size. Specify this parameter when you only want to consider a certain amount of history when training the model. Learn more about target rolling window aggregation.</p>	
<code>short_series_handling_config</code>	<p>Enables short time series handling to avoid failing during training due to insufficient data. Short series handling is set to <code>auto</code> by default. Learn more about short series handling.</p>	

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.
- Defines the `time_series_id_column_names` parameter to `"store"`. This ensures that **two separate time-series groups** are created for the data; one for store A and B.
- Sets the `forecast_horizon` to 50 in order to predict for the entire test set.
- Sets a forecast window to 10 periods with `target_rolling_window_size`
- Specifies a single lag on the target values for two periods ahead with the `target_lags` parameter.
- Sets `target_lags` to the recommended "auto" setting, which will automatically detect this value for you.

```
from azureml.core.forecasting_parameters import ForecastingParameters

forecasting_parameters = ForecastingParameters(time_column_name='day_datetime',
                                                forecast_horizon=50,
                                                time_series_id_column_names=["store"],
                                                freq='W',
                                                target_lags='auto',
                                                target_rolling_window_size=10)
```

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=5,
                             enable_ensembling=False,
                             verbosity=logging.INFO,
                             **forecasting_parameters)
```

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

To get a summary of what features are created as result of these steps, see [Featurization transparency](#)

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](#).

```
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.

Enable deep learning

NOTE

DNN support for forecasting in Automated Machine Learning is in [preview](#) and not supported for local runs.

You can also leverage 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)
```

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 how-to](#).

View the [Beverage Production Forecasting notebook](#) for a detailed code example leveraging DNNs.

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 additional 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.

View a Python code example leveraging 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. For the exact calculation see the [short_series_handling_configuration reference documentation](#).

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 [here](#). 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"><i>If all series are short</i>, pad the data.<i>If not all series are short</i>, 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 NaNsNumeric columns with 0Boolean/logic columns with FalseThe 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.
<code>None</code>	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, "forecasting_example")
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 the test data set.

The `forecast()` function allows specifications of when predictions should start, unlike the `predict()`, which is typically used for classification and regression tasks.

In the following example, you first replace all values in `y_pred` with `NaN`. The forecast origin will be 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()` function to forecast values up until a specified date.

```
label_query = test_labels.copy().astype(np.float)
label_query.fill(np.nan)
label_fcst, data_trans = fitted_pipeline.forecast(
    test_data, label_query, forecast_destination=pd.Timestamp(2019, 1, 8))
```

Calculate root mean squared error (RMSE) between the `actual_labels` actual values, and the forecasted values in `predict_labels`.

```
from sklearn.metrics import mean_squared_error
from math import sqrt

rmse = sqrt(mean_squared_error(actual_labels, predict_labels))
rmse
```

Now that 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_data` 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.predict(test_data)` to predict future values.

NOTE

Values cannot be predicted for number of periods greater than the `forecast_horizon`. The model must be re-trained with a larger horizon to predict future values beyond the current horizon.

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](#)
- [DNN](#)

Next steps

- Learn more about [how and where to deploy a model](#).
- Learn about [Interpretability: model explanations in automated machine learning \(preview\)](#).
- Learn how to train multiple models with AutoML in the [Many Models Solution Accelerator](#).
- Follow the [tutorial](#) for an end to end example for creating experiments with automated machine learning.

Configure data splits and cross-validation in automated machine learning

12/23/2020 • 4 minutes to read • [Edit Online](#)

In this article, you learn the different options for configuring training/validation data splits and cross-validation for your automated machine learning, AutoML, experiments.

In Azure Machine Learning, when you use AutoML 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.

AutoML 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](#).

NOTE

The studio currently supports training/validation data splits and cross-validation options, but it does not support specifying individual data files for your validation set.

Prerequisites

For this article you need,

- An Azure Machine Learning workspace. To create the workspace, see [Create an Azure Machine Learning workspace](#).
- 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 cross-validation and train/validation data splits as ML concepts. For a high-level explanation,
 - [About Train, Validation and Test Sets in Machine Learning](#)
 - [Understanding Cross Validation](#)

Default data splits and cross-validation

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_validation` or `validation_data` are **not** included.

```

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_validation` parameter, AutoML applies default techniques depending on the number of rows in the single dataset `training_data` provided:

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 and validation 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.

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 AutoML 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).

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'
                            )
```

Set the number of cross-validations

To perform 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.

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 5 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'
                            )
```

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.

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.

Next steps

- [Prevent imbalanced data and overfitting.](#)
- [Tutorial: Use automated machine learning to predict taxi fares - Split data section.](#)
- How to [Auto-train a time-series forecast model](#).

Data featurization in automated machine learning

12/23/2020 • 13 minutes to read • [Edit Online](#)

Learn about the data featurization settings in Azure Machine Learning and how to customize those features for [automated ML experiments](#).

Feature engineering and featurization

Feature engineering is the process of using domain knowledge of the data to create features that help machine learning (ML) 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 machine learning, or *autoML*, experiments.

Prerequisites

This article assumes that you already know how to configure an AutoML experiment. 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.

FEATURIZATION CONFIGURATION	DESCRIPTION
<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. 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.
Generate more features*	For DateTime features: Year, Month, Day, Day of week, Day of year, Quarter, Week of the year, Hour, Minute, Second. <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) For Text features: Term frequency based on unigrams, bigrams, and trigrams. Learn more about how this is done with BERT .
Transform and encode*	Transform numeric features that have few unique values into categorical features. One-hot encoding is used for low-cardinality categorical features. One-hot-hash encoding is used for high-cardinality categorical features.
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.

FEATURIZATION STEPS	DESCRIPTION
Target encodings	For categorical features, this step maps each category with an averaged target value for regression problems, and to the class probability for each class for classification problems. Frequency-based weighting and k-fold cross-validation are applied to reduce overfitting of the mapping and noise caused by sparse data categories.
Text target encoding	For text input, a stacked linear model with bag-of-words is used to generate the probability of each class.
Weight of Evidence (WoE)	Calculates WoE as a measure of correlation of categorical columns to the target column. WoE is calculated as the log of the ratio of in-class vs. out-of-class probabilities. This step produces one numeric feature column per class and removes the need to explicitly impute missing values and outlier treatment.
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.

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.
High cardinality feature handling	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	
	Done	Imbalanced classes were detected in your inputs. To fix model bias, fix the balancing problem. Learn more about imbalanced data .
		Imbalanced classes were detected in your inputs and the sweeping logic has determined to apply balancing.

GUARDRAIL	STATUS	CONDITION FOR TRIGGER
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.
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. Learn more about data preparation for time-series forecasting .

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 forecasting 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.

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] + ' - ')
        else:
            pprint(step[1].get_params())
            print()

print_model(model)
```

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 AutoML. 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.

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.

AutoML 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](#).

Use automated ML in an Azure Machine Learning pipeline in Python

12/23/2020 • 17 minutes to read • [Edit Online](#)

Azure Machine Learning's automated ML capability helps you discover high-performing models without you reimplementing every possible approach. Combined with Azure Machine Learning pipelines, you can create deployable workflows that can quickly discover the algorithm that works best for your data. This article will show you how to efficiently join a data preparation step to an automated ML step. Automated ML can quickly discover the algorithm that works best for your data, while putting you on the road to MLOps and model lifecycle operationalization with pipelines.

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 an Azure Machine Learning workspace](#).
- Familiarity with Azure's [automated machine learning](#) and [machine learning pipelines](#) facilities and SDK.

Review automated ML's central classes

Automated ML in a pipeline is represented by an `AutoMLStep` object. The `AutoMLStep` class is a subclass of `PipelineStep`. A graph of `PipelineStep` objects defines a `Pipeline`.

There are several subclasses of `PipelineStep`. In addition to the `AutoMLStep`, this article will show a `PythonScriptStep` for data preparation and another for registering the model.

The preferred way to initially move data *into* an ML pipeline is with `Dataset` objects. To move data *between* steps, the preferred way is with `PipelineData` objects. To be used with `AutoMLStep`, the `PipelineData` object must be transformed into a `PipelineOutputTabularDataset` object. For more information, see [Input and output data from ML pipelines](#).

TIP

An improved experience for passing temporary data between pipeline steps is available in the public preview classes, `OutputFileDatasetConfig` and `OutputTabularDatasetConfig`. These classes are [experimental](#) preview features, and may change at any time.

The `AutoMLStep` is configured via an `AutoMLConfig` object. `AutoMLConfig` is a flexible class, as discussed in [Configure automated ML experiments in Python](#).

A `Pipeline` runs in an `Experiment`. The pipeline `Run` has, for each step, a child `StepRun`. The outputs of the automated ML `StepRun` are the training metrics and highest-performing model.

To make things concrete, this article creates a simple pipeline for a classification task. The task is predicting Titanic survival, but we won't be discussing the data or task except in passing.

Get started

Retrieve initial dataset

Often, an ML workflow starts with pre-existing baseline data. This is a good scenario for a registered dataset. Datasets are visible across the workspace, support versioning, and can be interactively explored. There are many ways to create and populate a dataset, as discussed in [Create Azure Machine Learning datasets](#). Since we'll be using the Python SDK to create our pipeline, use the SDK to download baseline data and register it with the name 'titanic_ds'.

```
from azureml.core import Workspace, Dataset

ws = Workspace.from_config()
if not 'titanic_ds' in ws.datasets.keys() :
    # create a TabularDataset from Titanic training data
    web_paths = ['https://dprepdata.blob.core.windows.net/demo/Titanic.csv',
                 'https://dprepdata.blob.core.windows.net/demo/Titanic2.csv']
    titanic_ds = Dataset.Tabular.from_delimited_files(path=web_paths)

    titanic_ds.register(workspace = ws,
                        name = 'titanic_ds',
                        description = 'Titanic baseline data',
                        create_new_version = True)

titanic_ds = Dataset.get_by_name(ws, 'titanic_ds')
```

The code first logs in to the Azure Machine Learning workspace defined in `config.json` (for an explanation, see [Create a workspace configuration file](#)). If there isn't already a dataset named `'titanic_ds'` registered, then it creates one. The code downloads CSV data from the Web, uses them to instantiate a `TabularDataset` and then registers the dataset with the workspace. Finally, the function `Dataset.get_by_name()` assigns the `Dataset` to `titanic_ds`.

Configure your storage and compute target

Additional resources that the pipeline will need are storage and, generally, Azure Machine Learning compute resources.

```
from azureml.core import Datastore
from azureml.core.compute import AmlCompute, ComputeTarget

datastore = ws.get_default_datastore()

compute_name = 'cpu-cluster'
if not compute_name in ws.compute_targets :
    print('creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                               min_nodes=0,
                                                               max_nodes=1)
    compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

    compute_target.wait_for_completion(
        show_output=True, min_node_count=None, timeout_in_minutes=20)

    # Show the result
    print(compute_target.get_status().serialize())

compute_target = ws.compute_targets[compute_name]
```

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](#).

The intermediate data between the data preparation and the automated ML step can be stored in the workspace's default datastore, so we don't need to do more than call `get_default_datastore()` on the `Workspace` object.

After that, the code checks if the AML compute target `'cpu-cluster'` already exists. If not, we specify that we want a small CPU-based compute target. If you plan to use automated ML's deep learning features (for instance, text featurization with DNN support) you should choose a compute with strong GPU support, as described in [GPU optimized virtual machine sizes](#).

The code blocks until the target is provisioned and then prints some details of the just-created compute target. Finally, the named compute target is retrieved from the workspace and assigned to `compute_target`.

Configure the training run

The next step is making sure that the remote training run has all the dependencies that are required by the training steps. Dependencies and the runtime context are set by creating and configuring a `RunConfiguration` object.

```
from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core import Environment

aml_run_config = RunConfiguration()
# Use just-specified compute target ("cpu-cluster")
aml_run_config.target = compute_target

USE_CURATED_ENV = True
if USE_CURATED_ENV :
    curated_environment = Environment.get(workspace=ws, name="AzureML-Tutorial")
    aml_run_config.environment = curated_environment
else:
    aml_run_config.environment.python.user_managed_dependencies = False

    # Add some packages relied on by data prep step
    aml_run_config.environment.python.conda_dependencies = CondaDependencies.create(
        conda_packages=['pandas', 'scikit-learn'],
        pip_packages=['azureml-sdk[automl]', 'azureml-dataprep[fuse,pandas]'],
        pin_sdk_version=False)
```

The code above shows two options for handling dependencies. As presented, with `USE_CURATED_ENV = True`, the configuration is based on a curated environment. Curated environments are "prebaked" with common inter-dependent libraries and can be significantly faster to bring online. Curated environments have prebuilt Docker images in the [Microsoft Container Registry](#). The path taken if you change `USE_CURATED_ENV` to `False` shows the pattern for explicitly setting your dependencies. In that scenario, a new custom Docker image will be created and registered in an Azure Container Registry within your resource group (see [Introduction to private Docker container registries in Azure](#)). Building and registering this image can take quite a few minutes.

Prepare data for automated machine learning

Write the data preparation code

The baseline Titanic dataset consists of mixed numerical and text data, with some values missing. To prepare it for automated machine learning, the data preparation pipeline step will:

- Fill missing data with either random data or a category corresponding to "Unknown"
- Transform categorical data to integers
- Drop columns that we don't intend to use
- Split the data into training and testing sets
- Write the transformed data to either
 - `PipelineData` output paths

```

%%writefile dataprep.py
from azureml.core import Run

import pandas as pd
import numpy as np
import pyarrow as pa
import pyarrow.parquet as pq
import argparse

RANDOM_SEED=42

def prepare_age(df):
    # Fill in missing Age values from distribution of present Age values
    mean = df["Age"].mean()
    std = df["Age"].std()
    is_null = df["Age"].isnull().sum()
    # compute enough (== is_null().sum()) random numbers between the mean, std
    rand_age = np.random.randint(mean - std, mean + std, size = is_null)
    # fill NaN values in Age column with random values generated
    age_slice = df["Age"].copy()
    age_slice[np.isnan(age_slice)] = rand_age
    df["Age"] = age_slice
    df["Age"] = df["Age"].astype(int)

    # Quantize age into 5 classes
    df['Age_Group'] = pd.qcut(df['Age'],5, labels=False)
    df.drop(['Age'], axis=1, inplace=True)
    return df

def prepare_fare(df):
    df['Fare'].fillna(0, inplace=True)
    df['Fare_Group'] = pd.qcut(df['Fare'],5,labels=False)
    df.drop(['Fare'], axis=1, inplace=True)
    return df

def prepare_genders(df):
    genders = {"male": 0, "female": 1, "unknown": 2}
    df['Sex'] = df['Sex'].map(genders)
    df['Sex'].fillna(2, inplace=True)
    df['Sex'] = df['Sex'].astype(int)
    return df

def prepare_embarked(df):
    df['Embarked'].replace('', 'U', inplace=True)
    df['Embarked'].fillna('U', inplace=True)
    ports = {"S": 0, "C": 1, "Q": 2, "U": 3}
    df['Embarked'] = df['Embarked'].map(ports)
    return df

parser = argparse.ArgumentParser()
parser.add_argument('--output_path', dest='output_path', required=True)
args = parser.parse_args()

titanic_ds = Run.get_context().input_datasets['titanic_ds']
df = titanic_ds.to_pandas_dataframe().drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
df = prepare_embarked(prepare_genders(prepare_fare(prepare_age(df)))))

os.makedirs(os.path.dirname(args.output_path), exist_ok=True)
pq.write_table(pa.Table.from_pandas(df), args.output_path)

print(f"Wrote test to {args.output_path} and train to {args.output_path}")

```

The above code snippet is a complete, but minimal, example of data preparation for the Titanic data. The snippet starts with a Jupyter "magic command" to output the code to a file. If you aren't using a Jupyter notebook, remove that line and create the file manually.

The various `prepare_` functions in the above snippet modify the relevant column in the input dataset. These functions work on the data once it has been changed into a Pandas `DataFrame` object. In each case, missing data is either filled with representative random data or categorical data indicating "Unknown." Text-based categorical data is mapped to integers. No-longer-needed columns are overwritten or dropped.

After the code defines the data preparation functions, the code parses the input argument, which is the path to which we want to write our data. (These values will be determined by `PipelineData` objects that will be discussed in the next step.) The code retrieves the registered `'titanic_cs'` `Dataset`, converts it to a Pandas `DataFrame`, and calls the various data preparation functions.

Since the `output_path` is fully qualified, the function `os.makedirs()` is used to prepare the directory structure. At this point, you could use `DataFrame.to_csv()` to write the output data, but Parquet files are more efficient. This efficiency would probably be irrelevant with such a small dataset, but using the `PyArrow` package's `from_pandas()` and `write_table()` functions are only a few more keystrokes than `to_csv()`.

Parquet files are natively supported by the automated ML step discussed below, so no special processing is required to consume them.

Write the data preparation pipeline step (`PythonScriptStep`)

The data preparation code described above must be associated with a `PythonScriptStep` object to be used with a pipeline. The path to which the Parquet data-preparation output is written is generated by a `PipelineData` object. The resources prepared earlier, such as the `ComputeTarget`, the `RunConfig`, and the `'titanic_ds'` `Dataset` are used to complete the specification.

PipelineData users

```
from azureml.pipeline.core import PipelineData

from azureml.pipeline.steps import PythonScriptStep
prepped_data_path = PipelineData("titanic_train", datastore).as_dataset()

dataprep_step = PythonScriptStep(
    name="dataprep",
    script_name="dataprep.py",
    compute_target=compute_target,
    runconfig=aml_run_config,
    arguments=[("--output_path", prepped_data_path),
               inputs=[titanic_ds.as_named_input("titanic_ds")],
               outputs=[prepped_data_path],
               allow_reuse=True
    )
)
```

The `prepped_data_path` object is of type `PipelineOutputFileDataset`. Notice that it's specified in both the `arguments` and `outputs` arguments. If you review the previous step, you'll see that within the data preparation code, the value of the argument `--output_path` is the file path to which the Parquet file was written.

TIP

An improved experience for passing intermediate data between pipeline steps is available with the public preview class, `OutputFileDatasetConfig`. For a code example using the `OutputFileDatasetConfig` class, see how to [build a two step ML pipeline](#).

Train with AutoMLStep

Configuring an automated ML pipeline step is done with the `AutoMLConfig` class. This flexible class is described in [Configure automated ML experiments in Python](#). Data input and output are the only aspects of configuration that

require special attention in an ML pipeline. Input and output for `AutoMLConfig` in pipelines is discussed in detail below. Beyond data, an advantage of ML pipelines is the ability to use different compute targets for different steps. You might choose to use a more powerful `ComputeTarget` only for the automated ML process. Doing so is as straightforward as assigning a more powerful `RunConfiguration` to the `AutoMLConfig` object's `run_configuration` parameter.

Send data to `AutoMLStep`

In an ML pipeline, the input data must be a `Dataset` object. The highest-performing way is to provide the input data in the form of `PipelineOutputTabularDataset` objects. You create an object of that type with the `parse_parquet_files()` or `parse_delimited_files()` on a `PipelineOutputFileDataset`, such as the `prepped_data_path` object.

```
# type(prepped_data_path) == PipelineOutputFileDataset
# type(prepped_data) == PipelineOutputTabularDataset
prepped_data = prepped_data_path.parse_parquet_files(file_extension=None)
```

The snippet above creates a high-performing `PipelineOutputTabularDataset` from the `PipelineOutputFileDataset` output of the data preparation step.

TIP

The public preview class, `OutputFileDatasetConfig`, contains the `read_delimited_files()` method that converts an `OutputFileDatasetConfig` into an `OutputTabularDatasetConfig` for consumption in AutoML runs.

Another option is to use `Dataset` objects registered in the workspace:

```
prepped_data = Dataset.get_by_name(ws, 'Data_prepared')
```

Comparing the two techniques:

TECHNIQUE	BENEFITS AND DRAWBACKS
<code>PipelineOutputTabularDataset</code>	Higher performance Natural route from <code>PipelineData</code> Data isn't persisted after pipeline run
	Notebook showing <code>PipelineOutputTabularDataset</code> technique
Registered <code>Dataset</code>	Lower performance Can be generated in many ways Data persists and is visible throughout workspace
	Notebook showing registered <code>Dataset</code> technique

Specify automated ML outputs

The outputs of the `AutoMLStep` are the final metric scores of the higher-performing model and that model itself. To

use these outputs in further pipeline steps, prepare `PipelineData` objects to receive them.

```
from azureml.pipeline.core import TrainingOutput

metrics_data = PipelineData(name='metrics_data',
                            datastore=datastore,
                            pipeline_output_name='metrics_output',
                            training_output=TrainingOutput(type='Metrics'))

model_data = PipelineData(name='best_model_data',
                          datastore=datastore,
                          pipeline_output_name='model_output',
                          training_output=TrainingOutput(type='Model'))
```

The snippet above creates the two `PipelineData` objects for the metrics and model output. Each is named, assigned to the default datastore retrieved earlier, and associated with the particular `type` of `TrainingOutput` from the `AutoMLStep`. Because we assign `pipeline_output_name` on these `PipelineData` objects, their values will be available not just from the individual pipeline step, but from the pipeline as a whole, as will be discussed below in the section "Examine pipeline results."

Configure and create the automated ML pipeline step

Once the inputs and outputs are defined, it's time to create the `AutoMLConfig` and `AutoMLStep`. The details of the configuration will depend on your task, as described in [Configure automated ML experiments in Python](#). For the Titanic survival classification task, the following snippet demonstrates a simple configuration.

```
from azureml.train.automl import AutoMLConfig
from azureml.pipeline.steps import AutoMLStep

# Change iterations to a reasonable number (50) to get better accuracy
automl_settings = {
    "iteration_timeout_minutes" : 10,
    "iterations" : 2,
    "experiment_timeout_hours" : 0.25,
    "primary_metric" : 'AUC_weighted'
}

automl_config = AutoMLConfig(task = 'classification',
                             path = '.',
                             debug_log = 'automated_ml_errors.log',
                             compute_target = compute_target,
                             run_configuration = aml_run_config,
                             featurization = 'auto',
                             training_data = prepped_data,
                             label_column_name = 'Survived',
                             **automl_settings)

train_step = AutoMLStep(name='AutoML_Classification',
                       automl_config=automl_config,
                       outputs=[metrics_data,model_data],
                       enable_default_model_output=False,
                       enable_default_metrics_output=False,
                       allow_reuse=True)
```

The snippet shows an idiom commonly used with `AutoMLConfig`. Arguments that are more fluid (hyperparameter-ish) are specified in a separate dictionary while the values less likely to change are specified directly in the `AutoMLConfig` constructor. In this case, the `automl_settings` specify a brief run: the run will stop after only 2 iterations or 15 minutes, whichever comes first.

The `automl_settings` dictionary is passed to the `AutoMLConfig` constructor as kwargs. The other parameters aren't

complex:

- `task` is set to `classification` for this example. Other valid values are `regression` and `forecasting`
- `path` and `debug_log` describe the path to the project and a local file to which debug information will be written
- `compute_target` is the previously defined `compute_target` that, in this example, is an inexpensive CPU-based machine. If you're using AutoML's Deep Learning facilities, you would want to change the compute target to be GPU-based
- `featurization` is set to `auto`. More details can be found in the [Data Featurization](#) section of the automated ML configuration document
- `label_column_name` indicates which column we are interested in predicting
- `training_data` is set to the `PipelineOutputTabularDataset` objects made from the outputs of the data preparation step

The `AutoMLStep` itself takes the `AutoMLConfig` and has, as outputs, the `PipelineData` objects created to hold the metrics and model data.

IMPORTANT

You must set `enable_default_model_output` and `enable_default_metrics_output` to `True` only if you are using `AutoMLStepRun`.

In this example, the automated ML process will perform cross-validations on the `training_data`. You can control the number of cross-validations with the `n_cross_validations` argument. If you've already split your training data as part of your data preparation steps, you can set `validation_data` to its own `Dataset`.

You might occasionally see the use `x` for data features and `y` for data labels. This technique is deprecated and you should use `training_data` for input.

Register the model generated by automated ML

The last step in a simple ML pipeline is registering the created model. By adding the model to the workspace's model registry, it will be available in the portal and can be versioned. To register the model, write another `PythonScriptStep` that takes the `model_data` output of the `AutoMLStep`.

Write the code to register the model

A model is registered in a `Workspace`. You're probably familiar with using `Workspace.from_config()` to log on to your workspace on your local machine, but there's another way to get the workspace from within a running ML pipeline. The `Run.get_context()` retrieves the active `Run`. This `run` object provides access to many important objects, including the `Workspace` used here.

```

%%writefile register_model.py
from azureml.core.model import Model, Dataset
from azureml.core.run import Run, _OfflineRun
from azureml.core import Workspace
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--model_name", required=True)
parser.add_argument("--model_path", required=True)
args = parser.parse_args()

print(f"model_name : {args.model_name}")
print(f"model_path: {args.model_path}")

run = Run.get_context()
ws = Workspace.from_config() if type(run) == _OfflineRun else run.experiment.workspace

model = Model.register(workspace=ws,
                      model_path=args.model_path,
                      model_name=args.model_name)

print("Registered version {} of model {}".format(model.version, model.name))

```

Write the PythonScriptStep code

The model-registering `PythonScriptStep` uses a `PipelineParameter` for one of its arguments. Pipeline parameters are arguments to pipelines that can be easily set at run-submission time. Once declared, they're passed as normal arguments.

```

from azureml.pipeline.core.graph import PipelineParameter

# The model name with which to register the trained model in the workspace.
model_name = PipelineParameter("model_name", default_value="TitanicSurvivalInitial")

register_step = PythonScriptStep(script_name="register_model.py",
                                 name="register_model",
                                 allow_reuse=False,
                                 arguments=[ "--model_name", model_name, "--model_path", model_data],
                                 inputs=[model_data],
                                 compute_target=compute_target,
                                 runconfig=aml_run_config)

```

Create and run your automated ML pipeline

Creating and running a pipeline that contains an `AutoMLStep` is no different than a normal pipeline.

```

from azureml.pipeline.core import Pipeline
from azureml.core import Experiment

pipeline = Pipeline(ws, [dataprep_step, train_step, register_step])

experiment = Experiment(workspace=ws, name='titanic_automl')

run = experiment.submit(pipeline, show_output=True)
run.wait_for_completion()

```

The code above combines the data preparation, automated ML, and model-registering steps into a `Pipeline` object. It then creates an `Experiment` object. The `Experiment` constructor will retrieve the named experiment if it exists or create it if necessary. It submits the `Pipeline` to the `Experiment`, creating a `Run` object that will

asynchronously run the pipeline. The `wait_for_completion()` function blocks until the run completes.

Examine pipeline results

Once the `run` completes, you can retrieve `PipelineData` objects that have been assigned a `pipeline_output_name`. You can download the results and load them for further processing.

```
metrics_output_port = run.get_pipeline_output('metrics_output')
model_output_port = run.get_pipeline_output('model_output')

metrics_output_port.download('.', show_progress=True)
model_output_port.download('.', show_progress=True)
```

Downloaded files are written to the subdirectory `azureml/{run.id}/`. The metrics file is JSON-formatted and can be converted into a Pandas dataframe for examination.

For local processing, you may need to install relevant packages, such as Pandas, Pickle, the AzureML SDK, and so forth. For this example, it's likely that the best model found by automated ML will depend on XGBoost.

```
!pip install xgboost==0.90
```

```
import pandas as pd
import json

metrics_filename = metrics_output._path_on_datastore
# metrics_filename = path to downloaded file
with open(metrics_filename) as f:
    metrics_output_result = f.read()

deserialized_metrics_output = json.loads(metrics_output_result)
df = pd.DataFrame(deserialized_metrics_output)
df
```

The code snippet above shows the metrics file being loaded from its location on the Azure datastore. You can also load it from the downloaded file, as shown in the comment. Once you've deserialized it and converted it to a Pandas DataFrame, you can see detailed metrics for each of the iterations of the automated ML step.

The model file can be deserialized into a `Model` object that you can use for inferencing, further metrics analysis, and so forth.

```
import pickle

model_filename = model_output._path_on_datastore
# model_filename = path to downloaded file

with open(model_filename, "rb") as f:
    best_model = pickle.load(f)

# ... inferencing code not shown ...
```

For more information on loading and working with existing models, see [Use an existing model with Azure Machine Learning](#).

Download the results of an automated ML run

If you've been following along with the article, you'll have an instantiated `run` object. But you can also retrieve completed `Run` objects from the `Workspace` by way of an `Experiment` object.

The workspace contains a complete record of all your experiments and runs. You can either use the portal to find

and download the outputs of experiments or use code. To access the records from a historic run, use Azure Machine Learning to find the ID of the run in which you are interested. With that ID, you can choose the specific `run` by way of the `Workspace` and `Experiment`.

```
# Retrieved from Azure Machine Learning web UI
run_id = 'aaaaaaaa-bbbb-cccc-dddd-0123456789AB'
experiment = ws.experiments['titanic_automl']
run = next(run for run in ex.get_runs() if run.id == run_id)
```

You would have to change the strings in the above code to the specifics of your historical run. The snippet above assumes that you've assigned `ws` to the relevant `Workspace` with the normal `from_config()`. The experiment of interest is directly retrieved and then the code finds the `Run` of interest by matching the `run.id` value.

Once you have a `Run` object, you can download the metrics and model.

```
automl_run = next(r for r in run.get_children() if r.name == 'AutoML_Classification')
outputs = automl_run.get_outputs()
metrics = outputs['default_metrics_AutoML_Classification']
model = outputs['default_model_AutoML_Classification']

metrics.get_port_data_reference().download('.')
model.get_port_data_reference().download('.)
```

Each `Run` object contains `StepRun` objects that contain information about the individual pipeline step run. The `run` is searched for the `StepRun` object for the `AutoMLStep`. The metrics and model are retrieved using their default names, which are available even if you don't pass `PipelineData` objects to the `outputs` parameter of the `AutoMLStep`.

Finally, the actual metrics and model are downloaded to your local machine, as was discussed in the "Examine pipeline results" section above.

Next Steps

- Run this Jupyter notebook showing a [complete example of automated ML in a pipeline](#) that uses regression to predict taxi fares
- [Create automated ML experiments without writing code](#)
- Explore a variety of [Jupyter notebooks demonstrating automated ML](#)
- Read about integrating your pipeline in to [End-to-end MLOps](#) or investigate the [MLOps GitHub repository](#)

Evaluate automated machine learning experiment results

12/23/2020 • 18 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 runs are created and each run 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	
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 run results

After your automated ML experiment completes, a history of the runs can be found via:

- A browser with [Azure Machine Learning studio](#)
- A Jupyter notebook using the [RunDetails 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 run.
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.

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.

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 combining the true positives and false positives from each class. • <code>AUC_weighted</code>, arithmetic mean of the score for each class, weighted by the number of true instances in each 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 combining the true positives and false positives at each cutoff. • <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. 	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

METRIC	DESCRIPTION	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. 	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>
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

METRIC	DESCRIPTION	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. 	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. 	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 doesn't differentiate between binary and multiclass metrics. The same validation metrics are reported whether a dataset has two classes or more than two classes. However, some 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 `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.

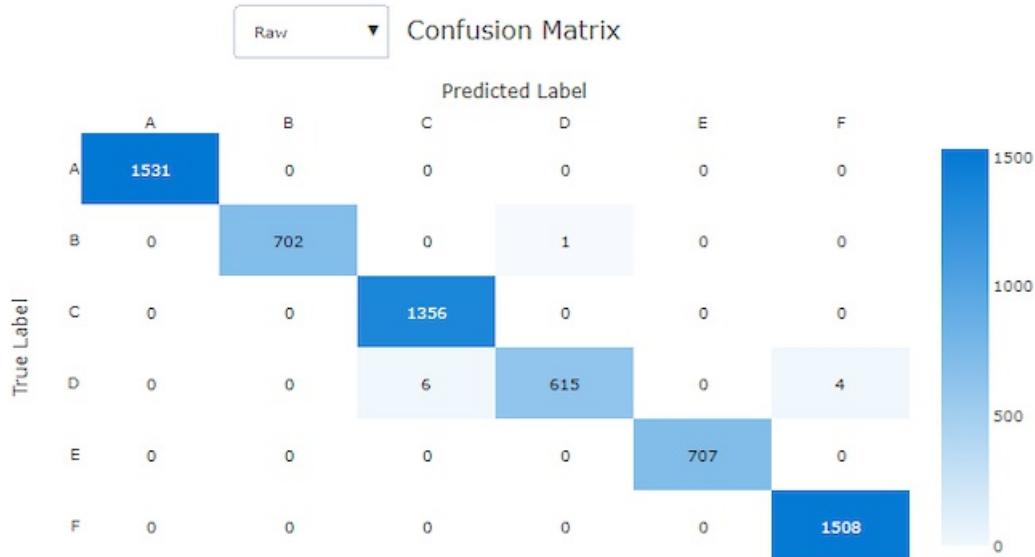
Confusion matrix

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 .

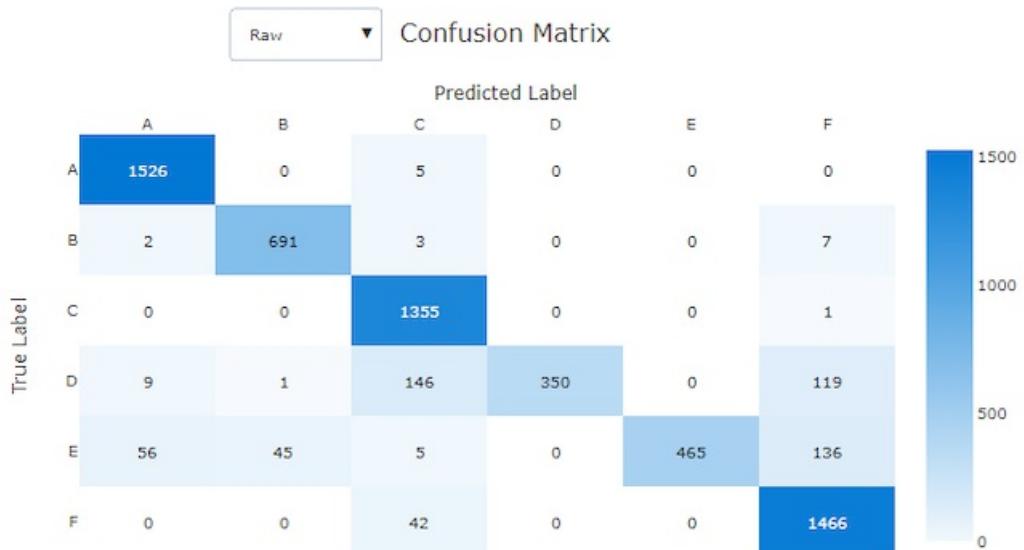
In the studio, a darker cell indicates a higher number of samples. Selecting **Normalized** view in the dropdown will normalize over each matrix row to show the percent of class c_i predicted to be class c_j . The benefit of the default **Raw** view is that you can see whether imbalance in the distribution 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.

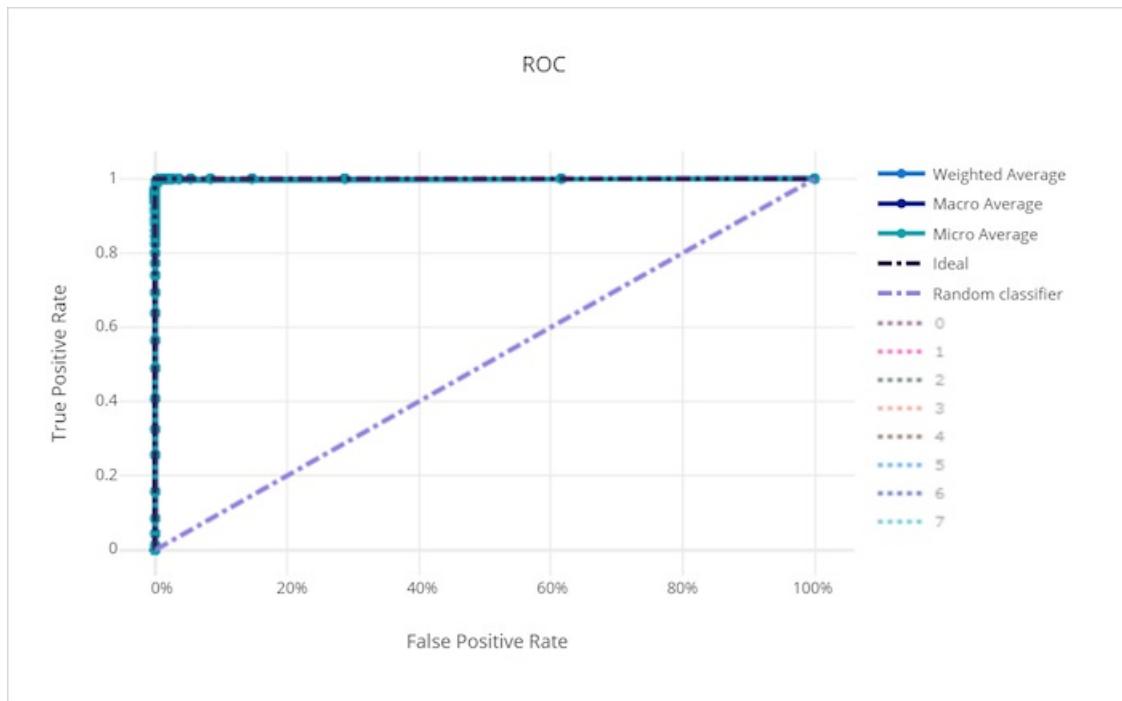
The area under the curve (AUC) can be interpreted as the proportion of correctly classified samples. More precisely, the AUC is the probability that the classifier ranks a randomly chosen positive sample higher than a randomly chosen negative sample. The shape of the curve gives an intuition for relationship 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.

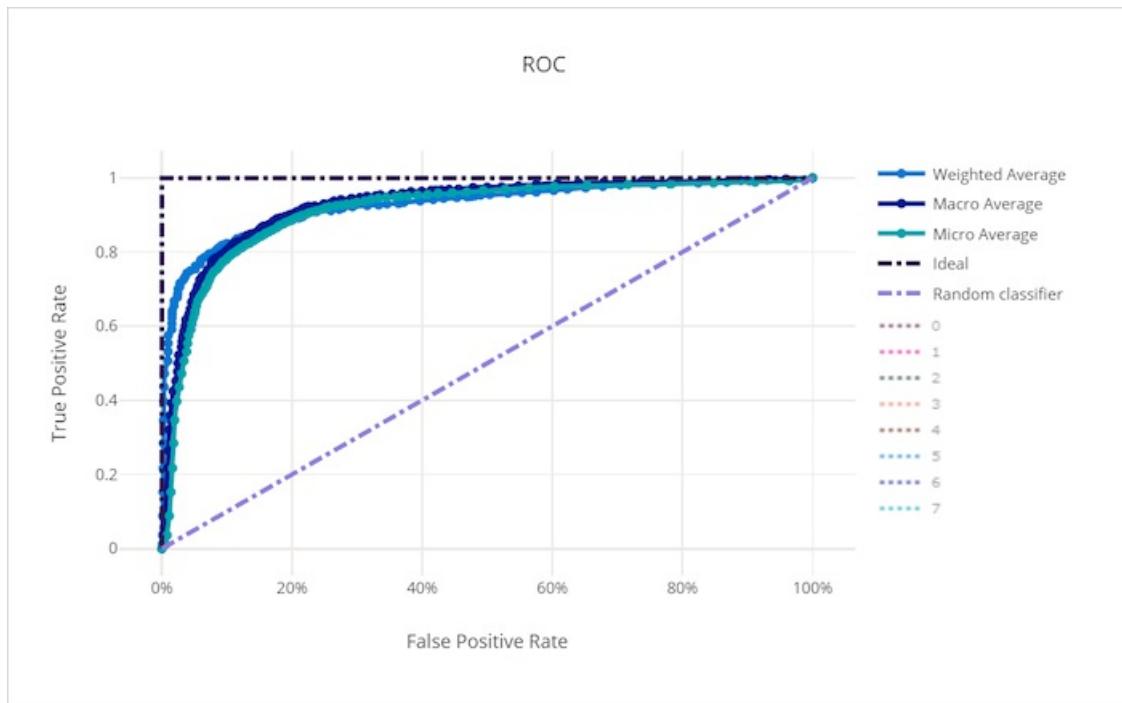
TIP

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



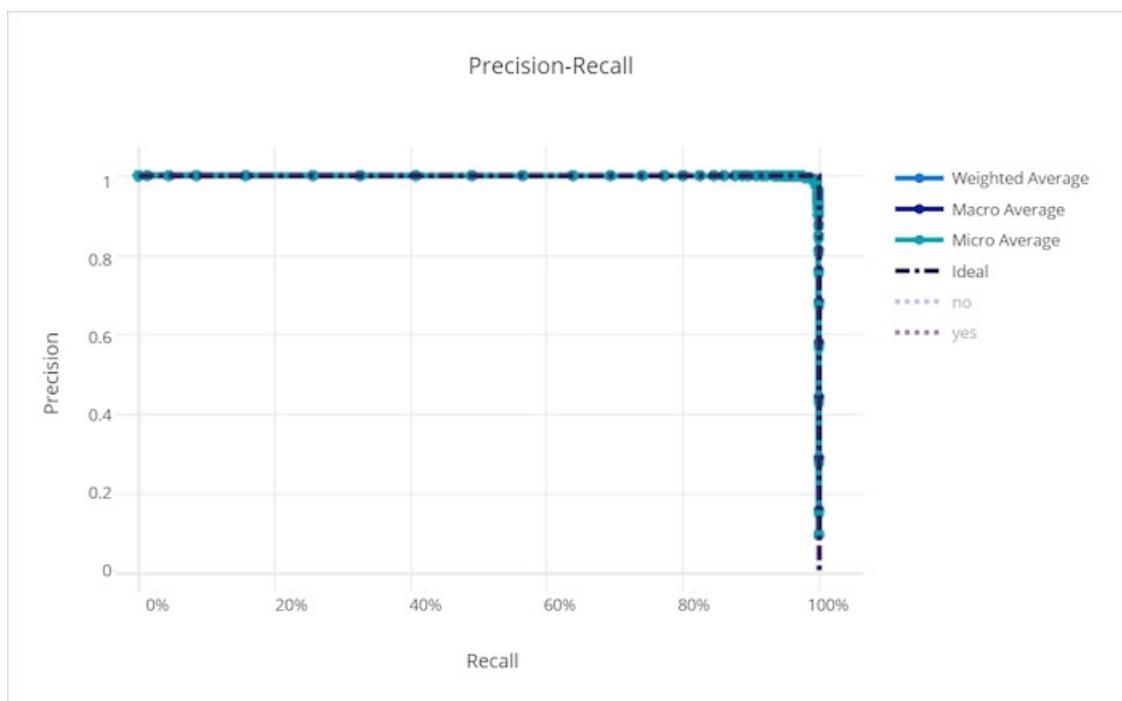
Precision-recall curve

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.

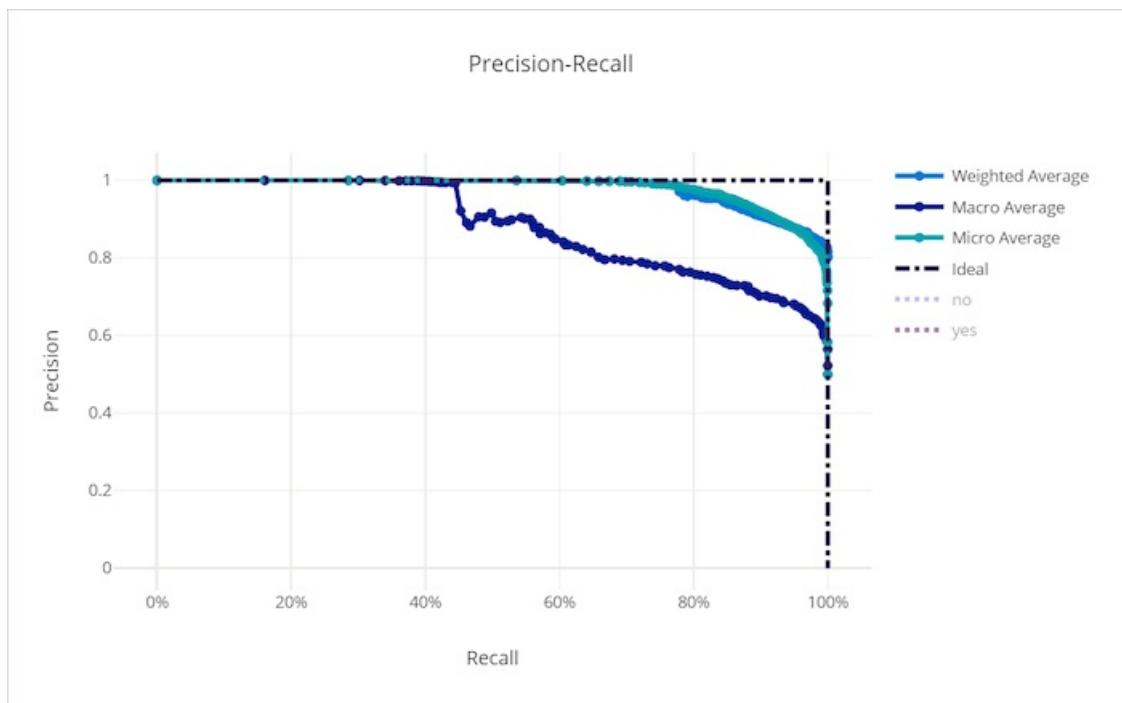
TIP

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.

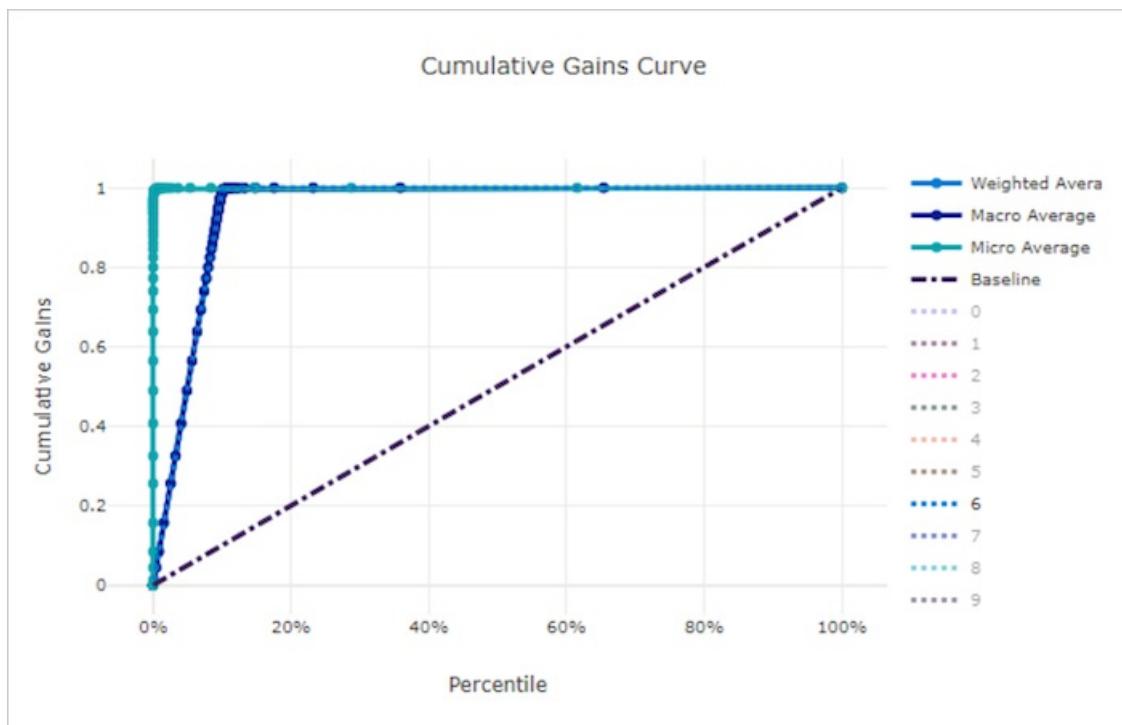
A perfect model will rank all positive samples above all negative samples giving a cumulative gains curve made up of two straight segments. The first is a line with slope $1 / x$ from $(0, 0)$ to $(x, 1)$ where x is the fraction of samples that belong to the positive class ($1 / \text{num_classes}$ if 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 will have a micro average curve that touches the top-left corner and a macro average line that has slope $1 / \text{num_classes}$ until cumulative gain is 100% and then horizontal until the data percent is 100.

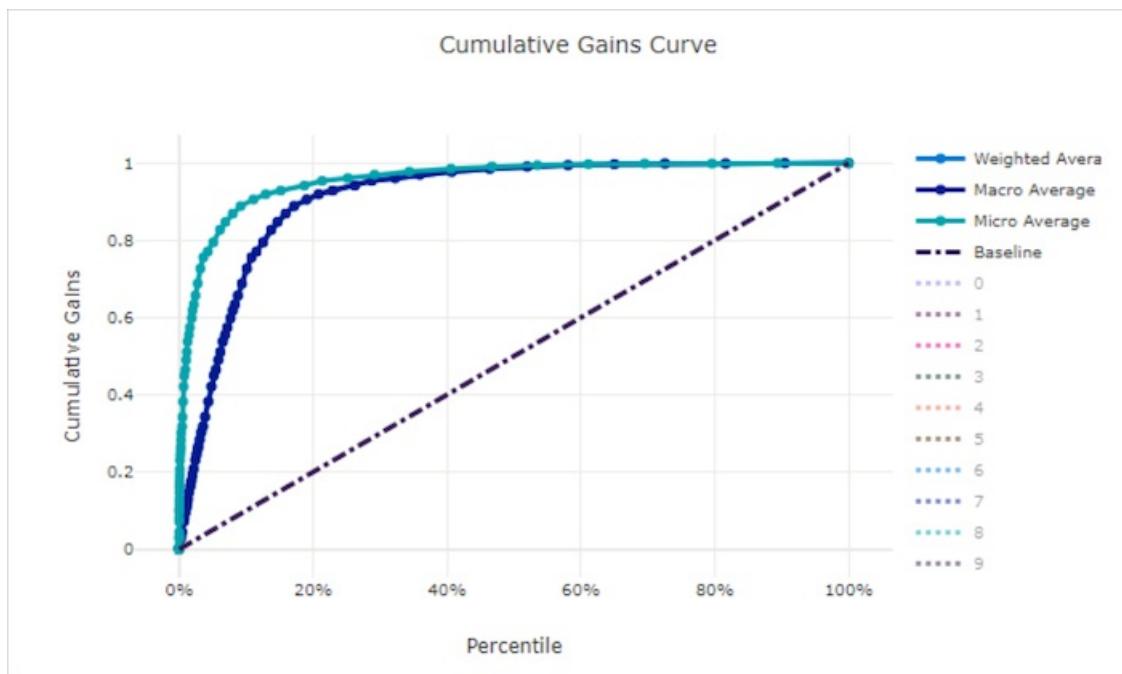
TIP

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.

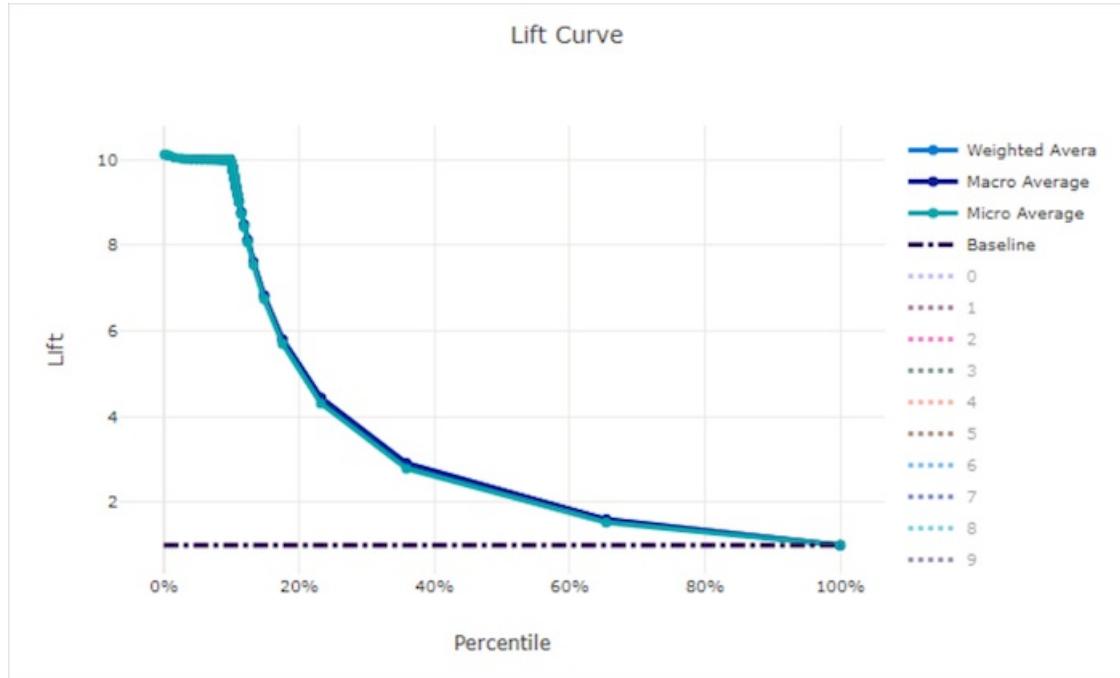
This relative performance takes into account the fact that classification gets harder as you increase the number of classes. (A random model incorrectly predicts a higher fraction of samples from a dataset with 10 classes compared to a dataset with two classes)

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.

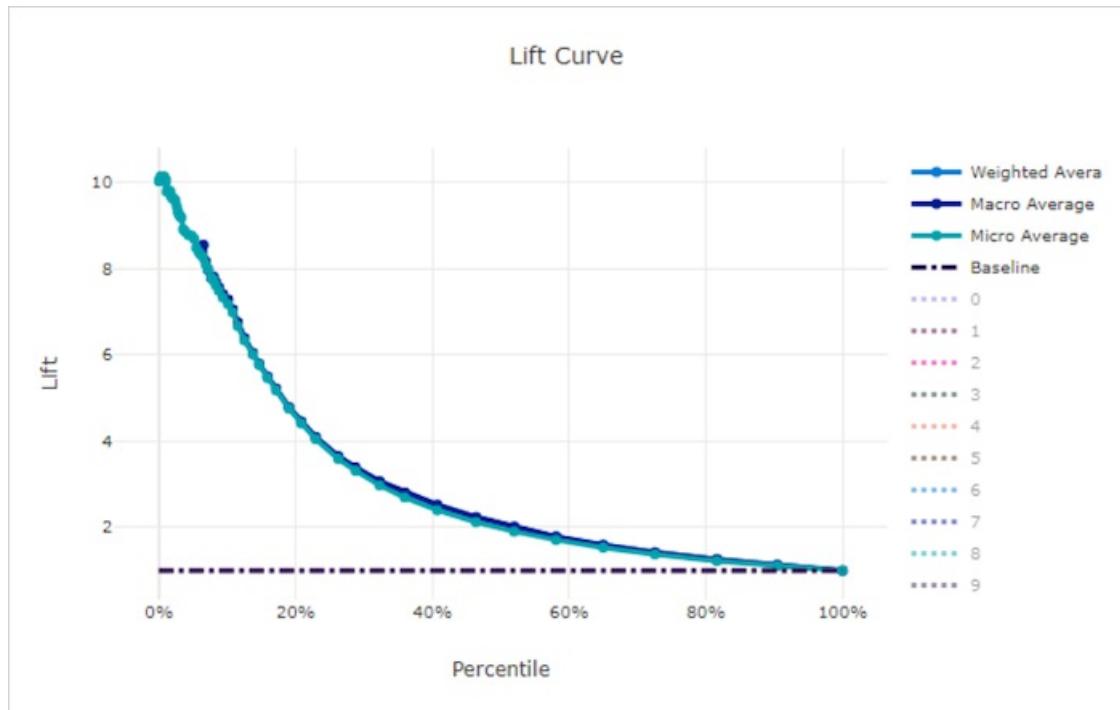
TIP

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

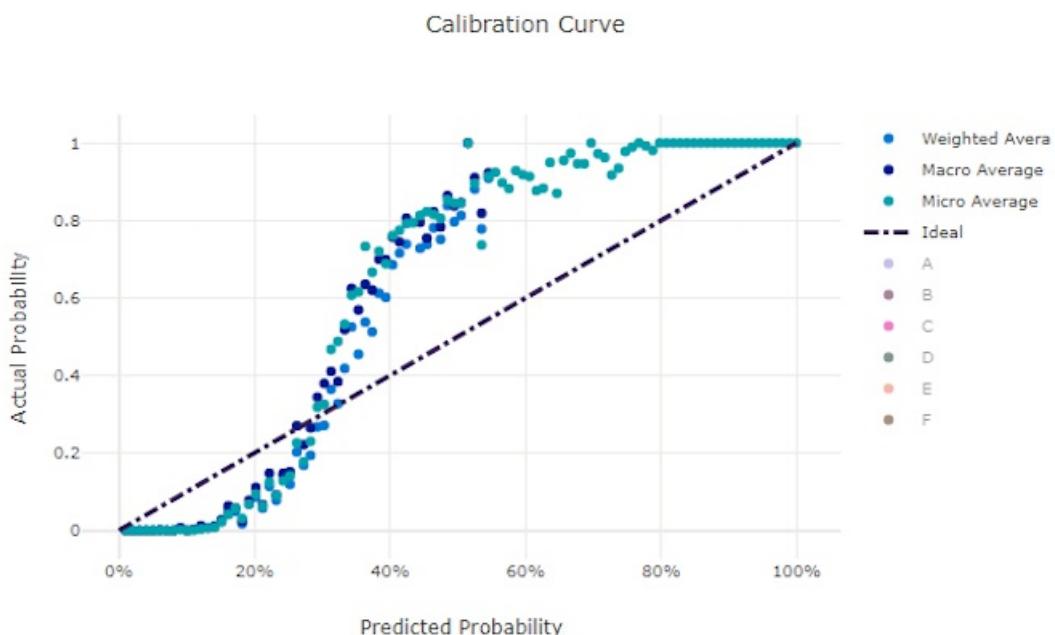
The calibration curve plots a model's confidence in its predictions against the proportion of positive samples at each confidence level. A well-calibrated model will correctly classify 100% of the predictions to which it assigns 100% confidence, 50% of the predictions it assigns 50% confidence, 20% of 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.

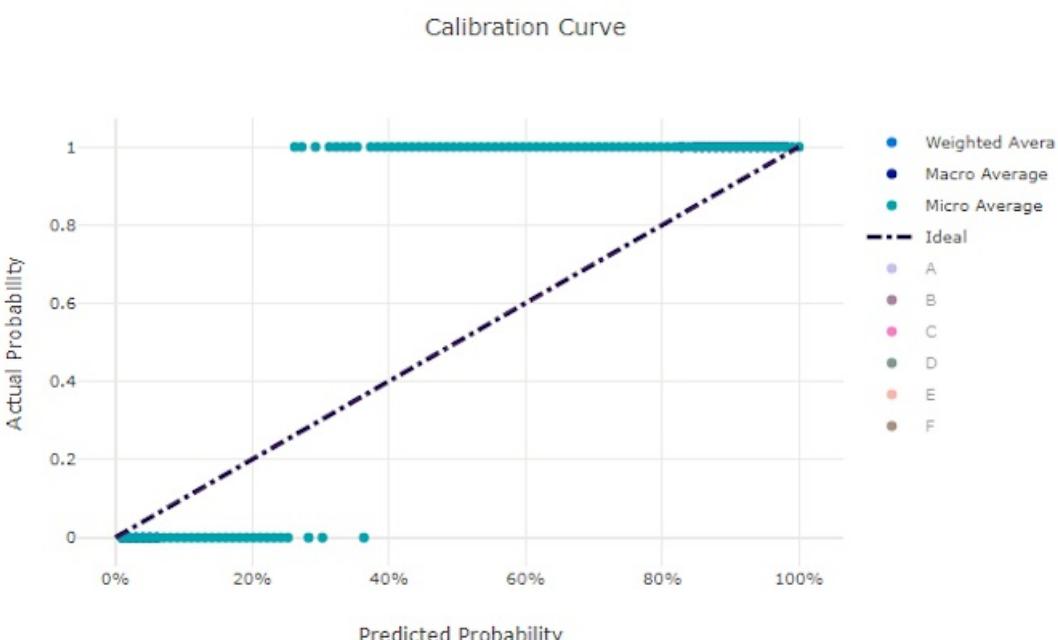
NOTE

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 <code>r2_score</code> 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: <code>mean_absolute_error</code> <code>normalized_mean_absolute_error</code> : the <code>mean_absolute_error</code> 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: <code>median_absolute_error</code> <code>normalized_median_absolute_error</code> : the <code>median_absolute_error</code> divided by the range of the data.</p>	Calculation

METRIC	DESCRIPTION	CALCULATION
r2_score	<p>R² is the coefficient of determination or the percent reduction in squared errors compared to a baseline model that outputs the mean.</p> <p>Objective: Closer to 1 the better Range: (-inf, 1]</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
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)`

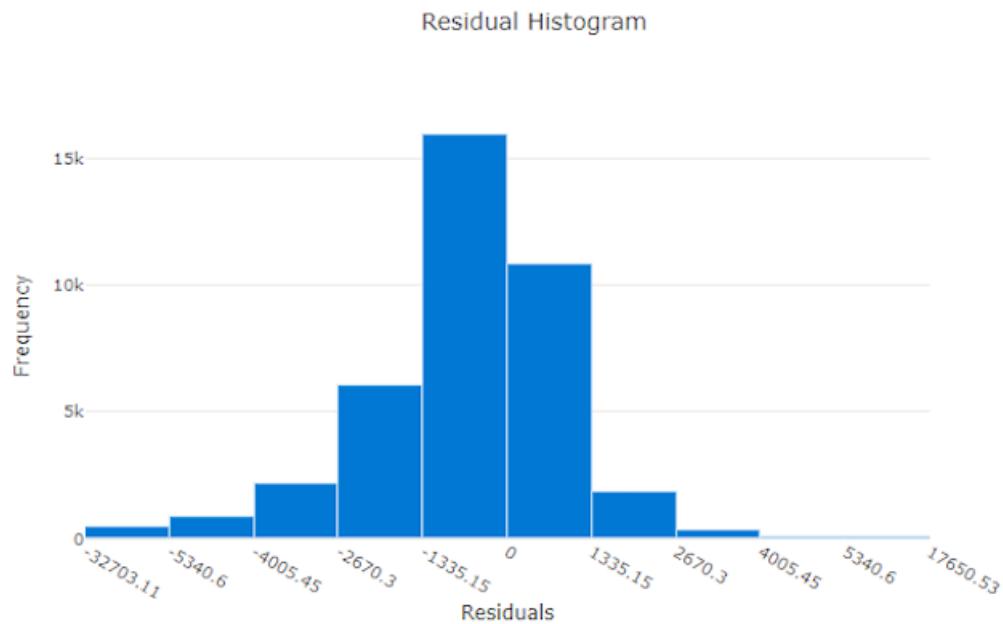
When evaluating a forecasting model on time series data, automated ML takes extra steps to ensure that normalization happens per time series ID (grain), because each time series likely has a different distribution of target values.

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.

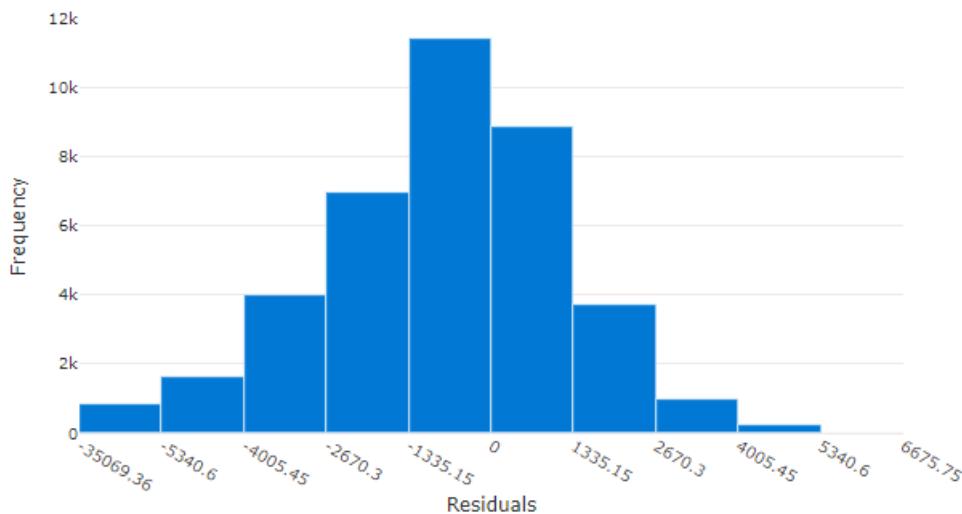
In this example, note that both models are slightly biased to predict lower than the actual value. This is not uncommon for a dataset with a skewed distribution of actual targets, but indicates worse model performance. A good model will have a residuals distribution that peaks at zero with few residuals at the extremes. A worse model will have a spread out residuals distribution with fewer samples around zero.

Residuals chart for a good model



Residuals chart for a bad model

Residual Histogram



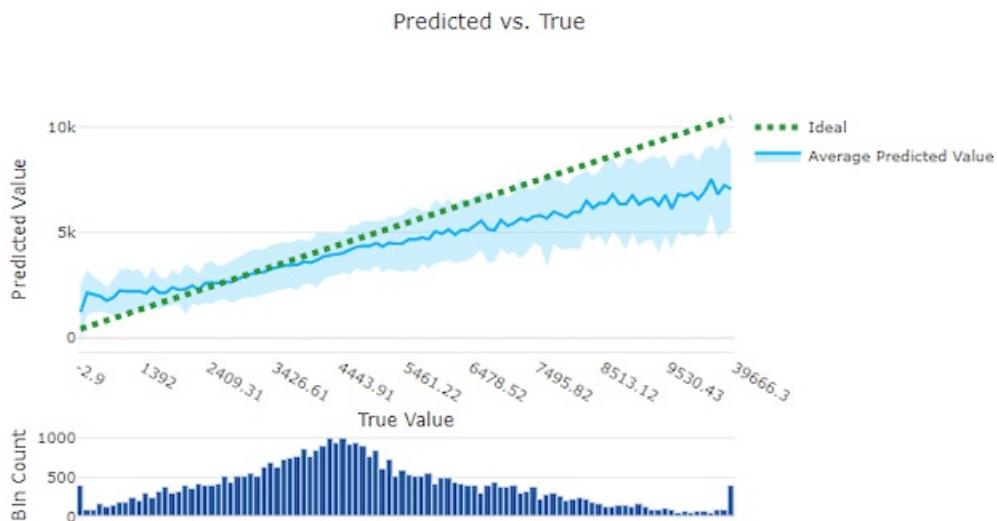
Predicted vs. true

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.

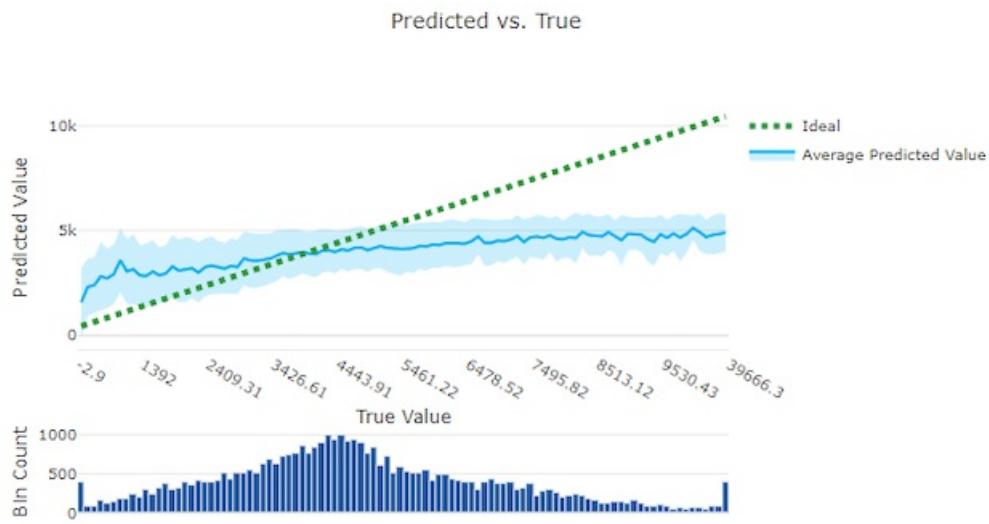
Often, the most common true value will have the most accurate predictions with the lowest variance. The distance of the trend line from the ideal $y = x$ line where there are few true values is a good measure of model performance on outliers. You can use the histogram at the bottom of the chart to reason 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

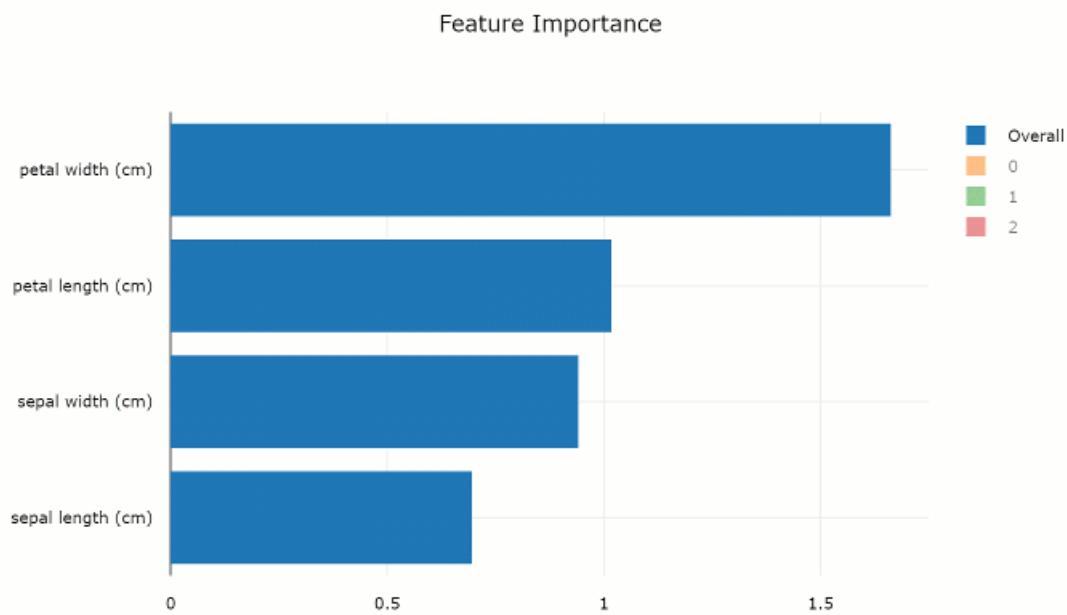


Predicted vs. true chart for a bad model



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 interpretability dashboard to measure and report the relative contributions of dataset features.



To view the interpretability dashboard 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 AutoML run
5. In the **Models** tab, select the **Algorithm name** for the model you want to explain
6. In the **Explanations** tab, you may see an explanation was already created if the model was the best

7. To create a new explanation, select **Explain model** and select the remote compute with which to compute explanations

NOTE

The ForecastTCN model is not currently supported by automated ML explanations and other forecasting models may have limited access to interpretability tools.

Next steps

- Try the [automated machine learning model explanation sample notebooks](#).
- Learn more about [responsible AI offerings in automated ML](#).
- For automated ML specific questions, reach out to askautomatedml@microsoft.com.

Make predictions with an AutoML ONNX model in .NET

12/23/2020 • 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.

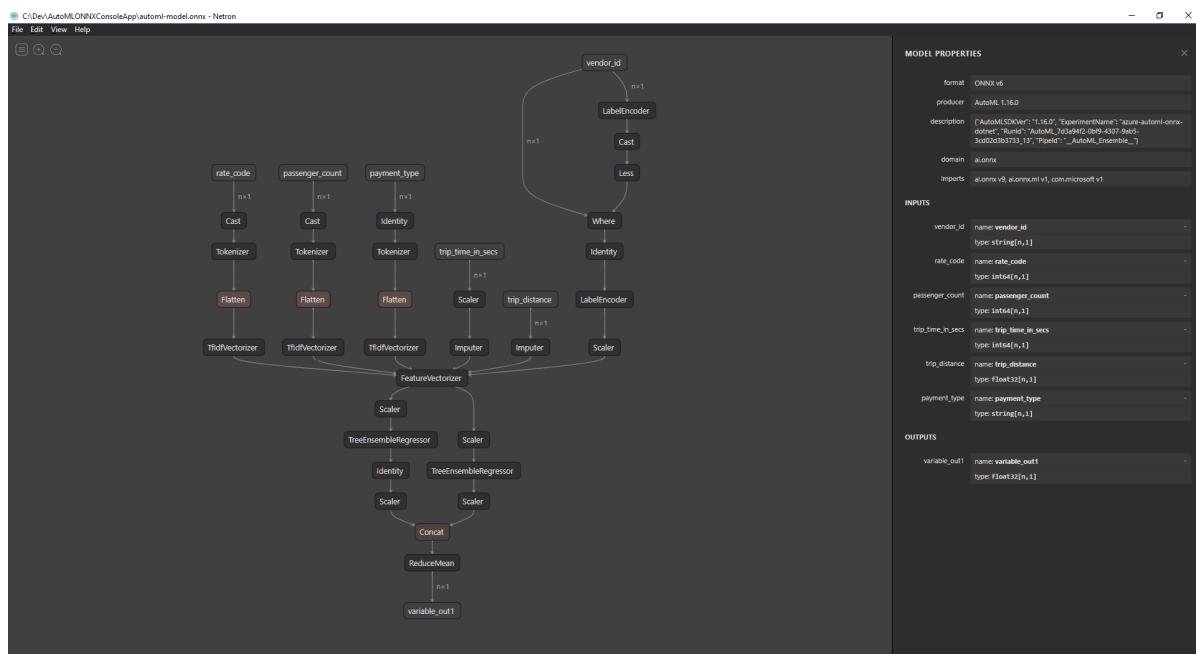
The model used in this sample uses data from the NYC TLC Taxi Trip dataset. A sample of the data can be seen below:

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](#)

Deploy models with Azure Machine Learning

12/23/2020 • 12 minutes to read • [Edit Online](#)

Learn how to deploy your machine learning model as a web service in the Azure cloud or to Azure IoT Edge devices.

The workflow is similar no matter where you deploy your model:

1. Register the model (optional, see below).
2. Prepare an inference configuration (unless using [no-code deployment](#)).
3. Prepare an entry script (unless using [no-code deployment](#)).
4. Choose a compute target.
5. Deploy the model to the compute target.
6. Test the resulting web service.

For more information on the concepts involved in the deployment workflow, see [Manage, deploy, and monitor models with Azure Machine Learning](#).

Prerequisites

- [Azure CLI](#)
- [Python](#)
- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- A model. If you don't have a trained model, you can use the model and dependency files provided in [this tutorial](#).
- The [Azure Command Line Interface \(CLI\) extension for the Machine Learning service](#).

Connect to your workspace

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

Follow the directions in the Azure CLI documentation for [setting your subscription context](#).

Then do:

```
az ml workspace list --resource-group=<my resource group>
```

to see the workspaces you have access to.

Register your model (optional)

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 the workspace. After you register the files, you can then download or deploy the registered model and receive all the files that you registered.

TIP

Registering a model for version tracking is recommended but not required. If you would rather proceed without registering a model, you will need to specify a source directory in your [InferenceConfig](#) or [inferenceconfig.json](#) and ensure your model resides within that source directory.

TIP

When you register a model, you provide the path of either a cloud location (from a training run) or a local directory. This path is just to locate the files for upload as part of the registration process. It doesn't need to match the path used in the entry script. For more information, see [Locate model files in your entry script](#).

IMPORTANT

When using Filter by `Tags` option on the Models page of Azure Machine Learning Studio, instead of using `TagName : TagValue` customers should use `TagName=TagValue` (without space)

The following examples demonstrate how to register a model.

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

Register a model from an Azure ML training run

```
az ml model register -n sklearn_mnist --asset-path outputs/sklearn_mnist_model.pkl --experiment-name myexperiment --run-id myrunid --tag area=mnist
```

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
```

The `--asset-path` parameter refers to the cloud location of the model. In this example, the path of a single file is used. To include multiple files in the model registration, set `--asset-path` to the path of a folder that contains the files.

Register a model from a local file

```
az ml model register -n onnx_mnist -p mnist/model.onnx
```

To include multiple files in the model registration, set `-p` to the path of a folder that contains the files.

For more information on `az ml model register`, consult the [reference documentation](#).

Define an entry script

The entry script receives data submitted to a deployed web service and passes it to the model. It

then takes the response returned by the model and returns that to the client. *The script is specific to your model.* It must understand the data that the model expects and returns.

The two things you need to accomplish in your entry script are:

1. Loading your model (using a function called `init()`)
2. Running your model on input data (using a function called `run()`)

Let's go through these steps in detail.

Writing init()

Loading a registered model

Your registered models are stored at a path pointed to by an environment variable called `AZUREML_MODEL_DIR`. For more information on the exact directory structure, see [Locate model files in your entry script](#)

Loading a local model

If you opted against registering your model and passed your model as part of your source directory, you can read it in like you would locally, by passing the path to the model relative to your scoring script. For example, if you had a directory structured as:

```
- source_dir
  - score.py
  - models
    - model1.onnx
```

you could load your models with the following Python code:

```
import os

model = open(os.path.join('.', 'models', 'model1.onnx'))
```

Writing run()

`run()` is executed every time your model receives a scoring request, and expects the body of the request to be a JSON document with the following structure:

```
{
  "data": <model-specific-data-structure>
}
```

The input to `run()` is a Python string containing whatever follows the "data" key.

The following example demonstrates how to load a registered scikit-learn model and score it with numpy data:

```

import json
import numpy as np
import os
from sklearn.externals import joblib

def init():
    global model
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'sklearn_mnist_model.pkl')
    model = joblib.load(model_path)

def run(data):
    try:
        data = np.array(json.loads(data))
        result = model.predict(data)
        # You can return any data type, as long as it is JSON serializable.
        return result.tolist()
    except Exception as e:
        error = str(e)
        return error

```

For more advanced examples, including automatic Swagger schema generation and binary (i.e. image) data, read [the article on advanced entry script authoring](#)

Define an inference configuration

An inference configuration describes how to set up the web-service containing your model. It's used later, when you deploy the model.

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

A minimal inference configuration can be written as:

```
{
    "entryScript": "score.py",
    "sourceDirectory": "./working_dir"
}
```

This specifies that the deployment will use the file `score.py` in the `./working_dir` directory to process incoming requests.

[See this article](#) for a more thorough discussion of inference configurations.

TIP

For information on using a custom Docker image with an inference configuration, see [How to deploy a model using a custom Docker image](#).

Choose 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	FPGA 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 Kubernetes Service (AKS)	Real-time inference	Yes (web service deployment)	Yes	<p>Use for high-scale production deployments. Provides fast response time and autoscaling of the deployed service. Cluster autoscaling isn't supported through the Azure Machine Learning SDK. To change the nodes in the AKS cluster, use the UI for your AKS cluster in the Azure portal.</p> <p>Supported in the designer.</p>
Azure Container Instances	Testing or development			<p>Use for low-scale CPU-based workloads that require less than 48 GB of RAM.</p> <p>Supported in the designer.</p>
Azure Machine Learning compute clusters	Batch inference	Yes (machine learning pipeline)		<p>Run batch scoring on serverless compute. Supports normal and low-priority VMs. No support for realtime inference.</p>

NOTE

Although compute targets like local, Azure Machine Learning compute, 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 AKS.

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.
- Use single-node AKS clusters for dev/test of larger models.

Define a deployment configuration

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

The options available for a deployment configuration differ depending on the compute target you choose.

The entries in the `deploymentconfig.json` document map to the parameters for [LocalWebservice.deploy_configuration](#). The following table describes the mapping between the entities in the JSON document and the parameters for the method:

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
<code>computeType</code>	NA	The compute target. For local targets, the value must be <code>local</code> .
<code>port</code>	<code>port</code>	The local port on which to expose the service's HTTP endpoint.

This JSON is an example deployment configuration for use with the CLI:

```
{  
    "computeType": "local",  
    "port": 32267  
}
```

For more information, see [this reference](#).

Deploy your model

You are now ready to deploy your model.

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

Using a registered model

If you registered your model in your Azure Machine Learning workspace, replace "mymodel:1" with the name of your model and its version number.

```
az ml model deploy -m mymodel:1 --ic inferenceconfig.json --dc deploymentconfig.json
```

Using a local model

If you would prefer not to register your model, you can pass the "sourceDirectory" parameter in your inferenceconfig.json to specify a local directory from which to serve your model.

```
az ml model deploy --ic inferenceconfig.json --dc deploymentconfig.json
```

Understanding service state

During model deployment, you may see the service state change while it fully deploys.

The following table describes the different service states:

WEBSERVICE STATE	DESCRIPTION	FINAL STATE?
Transitioning	The service is in the process of deployment.	No
Unhealthy	The service has deployed but is currently unreachable.	No
Unschedulable	The service cannot be deployed at this time due to lack of resources.	No
Failed	The service has failed to deploy due to an error or crash.	Yes
Healthy	The service is healthy and the endpoint is available.	Yes

Batch inference

Azure Machine Learning Compute targets are created and managed by Azure Machine Learning. They can be used for batch prediction from Azure Machine Learning pipelines.

For a walkthrough of batch inference with Azure Machine Learning Compute, see [How to run batch predictions](#).

IoT Edge inference

Support for deploying to the edge is in preview. For more information, see [Deploy Azure Machine Learning as an IoT Edge module](#).

Delete resources

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

To delete a deployed webservice, use `az ml service <name of webservice>`.

To delete a registered model from your workspace, use `az ml model delete <model id>`

Read more about [deleting a webservice](#) and [deleting a model](#).

Next steps

- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Create client applications to consume web services](#)
- [Update web service](#)
- [How to deploy a model using a custom Docker image](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Create event alerts and triggers for model deployments](#)

Deploy your existing model with Azure Machine Learning

12/23/2020 • 6 minutes to read • [Edit Online](#)

In this article, you learn how to register and deploy a machine learning model you trained outside Azure Machine Learning. You can deploy as a web service or to an IoT Edge device. Once deployed, you can monitor your model and detect data drift in Azure Machine Learning.

For more information on the concepts and terms in this article, see [Manage, deploy, and monitor machine learning models](#).

Prerequisites

- [An Azure Machine Learning workspace](#)
 - Python examples assume that the `ws` variable is set to your Azure Machine Learning workspace. For more information about how to connect to workspace, please refer to the [Azure Machine Learning SDK for Python documentation](#).
 - CLI examples use placeholders of `myworkspace` and `myresourcegroup`, which you should replace with the name of your workspace and the resource group that contains it.
- The [Azure Machine Learning Python SDK](#).
- The [Azure CLI](#) and [Machine Learning CLI extension](#).
- A trained model. The model must be persisted to one or more files on your development environment.

To demonstrate registering a model trained, the example code in this article uses the models from [Paolo Ripamonti's Twitter sentiment analysis project](#).

Register the model(s)

Registering a model allows you to store, version, and track metadata about models in your workspace. In the following Python and CLI examples, the `models` directory contains the `model.h5`, `model.w2v`, `encoder.pkl`, and `tokenizer.pkl` files. This example uploads the files contained in the `models` directory as a new model registration named `sentiment`:

```
from azureml.core.model import Model
# Tip: When model_path is set to a directory, you can use the child_paths parameter to include
#       only some of the files from the directory
model = Model.register(model_path = "./models",
                       model_name = "sentiment",
                       description = "Sentiment analysis model trained outside Azure Machine Learning",
                       workspace = ws)
```

For more information, see the [Model.register\(\)](#) reference.

```
az ml model register -p ./models -n sentiment -w myworkspace -g myresourcegroup
```

TIP

You can also set add `tags` and `properties` dictionary objects to the registered model. These values can be used later to help identify a specific model. For example, the framework used, training parameters, etc.

For more information, see the [az ml model register](#) reference.

For more information on model registration in general, see [Manage, deploy, and monitor machine learning models](#).

Define inference configuration

The inference configuration defines the environment used to run the deployed model. The inference configuration references the following entities, which are used to run the model when it's deployed:

- An entry script, named `score.py`, loads the model when the deployed service starts. This script is also responsible for receiving data, passing it to the model, and then returning a response.
- An Azure Machine Learning [environment](#). An environment defines the software dependencies needed to run the model and entry script.

The following example shows how to use the SDK to create an environment and then use it with an inference configuration:

```
from azureml.core.model import InferenceConfig
from azureml.core.environment import Environment
from azureml.core.conda_dependencies import CondaDependencies

# Create the environment
myenv = Environment(name="myenv")
conda_dep = CondaDependencies()

# Define the packages needed by the model and scripts
conda_dep.add_conda_package("tensorflow")
conda_dep.add_conda_package("numpy")
conda_dep.add_conda_package("scikit-learn")
# You must list azureml-defaults as a pip dependency
conda_dep.add_pip_package("azureml-defaults")
conda_dep.add_pip_package("keras")
conda_dep.add_pip_package("gensim")

# Adds dependencies to PythonSection of myenv
myenv.python.conda_dependencies=conda_dep

inference_config = InferenceConfig(entry_script="score.py",
                                    environment=myenv)
```

For more information, see the following articles:

- [How to use environments](#).
- [InferenceConfig](#) reference.

The CLI loads the inference configuration from a YAML file:

```
{
  "entryScript": "score.py",
  "runtime": "python",
  "condaFile": "myenv.yml"
}
```

With the CLI, the conda environment is defined in the `myenv.yml` file referenced by the inference configuration.

The following YAML is the contents of this file:

```
name: inference_environment
dependencies:
- python=3.6.2
- tensorflow
- numpy
- scikit-learn
- pip:
  - azureml-defaults
  - keras
  - gensim
```

For more information on inference configuration, see [Deploy models with Azure Machine Learning](#).

Entry script (`score.py`)

The entry script has only two required functions, `init()` and `run(data)`. These functions are used to initialize the service at startup and run the model using request data passed in by a client. The rest of the script handles loading and running the model(s).

IMPORTANT

There isn't a generic entry script that works for all models. It is always specific to the model that is used. It must understand how to load the model, the data format that the model expects, and how to score data using the model.

The following Python code is an example entry script (`score.py`):

```
import os
import pickle
import json
import time
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
from gensim.models.word2vec import Word2Vec

# SENTIMENT
POSITIVE = "POSITIVE"
NEGATIVE = "NEGATIVE"
NEUTRAL = "NEUTRAL"
SENTIMENT_THRESHOLDS = (0.4, 0.7)
SEQUENCE_LENGTH = 300

# Called when the deployed service starts
def init():
    global model
    global tokenizer
    global encoder
    global w2v_model

    # Get the path where the deployed model can be found.
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), './models')
    # load models
    model = load_model(model_path + '/model.h5')
    w2v_model = Word2Vec.load(model_path + '/model.w2v')

    with open(model_path + '/tokenizer.pkl','rb') as handle:
        tokenizer = pickle.load(handle)

    with open(model_path + '/encoder.pkl','rb') as handle:
        encoder = pickle.load(handle)
```

```

# Handle requests to the service
def run(data):
    try:
        # Pick out the text property of the JSON request.
        # This expects a request in the form of {"text": "some text to score for sentiment"}
        data = json.loads(data)
        prediction = predict(data['text'])
        #Return prediction
        return prediction
    except Exception as e:
        error = str(e)
        return error

# Determine sentiment from score
def decode_sentiment(score, include_neutral=True):
    if include_neutral:
        label = NEUTRAL
        if score <= SENTIMENT_THRESHOLDS[0]:
            label = NEGATIVE
        elif score >= SENTIMENT_THRESHOLDS[1]:
            label = POSITIVE
        return label
    else:
        return NEGATIVE if score < 0.5 else POSITIVE

# Predict sentiment using the model
def predict(text, include_neutral=True):
    start_at = time.time()
    # Tokenize text
    x_test = pad_sequences(tokenizer.texts_to_sequences([text]), maxlen=SEQUENCE_LENGTH)
    # Predict
    score = model.predict([x_test])[0]
    # Decode sentiment
    label = decode_sentiment(score, include_neutral=include_neutral)

    return {"label": label, "score": float(score),
            "elapsed_time": time.time()-start_at}

```

For more information on entry scripts, see [Deploy models with Azure Machine Learning](#).

Define deployment

The [Webservice](#) package contains the classes used for deployment. The class you use determines where the model is deployed. For example, to deploy as a web service on Azure Kubernetes Service, use [AksWebService.deploy_configuration\(\)](#) to create the deployment configuration.

The following Python code defines a deployment configuration for a local deployment. This configuration deploys the model as a web service to your local computer.

IMPORTANT

A local deployment requires a working installation of [Docker](#) on your local computer:

```

from azureml.core.webservice import LocalWebservice

deployment_config = LocalWebservice.deploy_configuration()

```

For more information, see the [LocalWebservice.deploy_configuration\(\)](#) reference.

The CLI loads the deployment configuration from a YAML file:

```
{  
    "computeType": "LOCAL"  
}
```

Deploying to a different compute target, such as Azure Kubernetes Service in the Azure cloud, is as easy as changing the deployment configuration. For more information, see [How and where to deploy models](#).

Deploy the model

The following example loads information on the registered model named `sentiment`, and then deploys it as a service named `sentiment`. During deployment, the inference configuration and deployment configuration are used to create and configure the service environment:

```
from azureml.core.model import Model  
  
model = Model(ws, name='sentiment')  
service = Model.deploy(ws, 'myservice', [model], inference_config, deployment_config)  
  
service.wait_for_deployment(True)  
print(service.state)  
print("scoring URI: " + service.scoring_uri)
```

For more information, see the [Model.deploy\(\)](#) reference.

To deploy the model from the CLI, use the following command. This command deploys version 1 of the registered model (`sentiment:1`) using the inference and deployment configuration stored in the `inferenceConfig.json` and `deploymentConfig.json` files:

```
az ml model deploy -n myservice -m sentiment:1 --ic inferenceConfig.json --dc deploymentConfig.json
```

For more information, see the [az ml model deploy](#) reference.

For more information on deployment, see [How and where to deploy models](#).

Request-response consumption

After deployment, the scoring URI is displayed. This URI can be used by clients to submit requests to the service. The following example is a simple Python client that submits data to the service and displays the response:

```
import requests  
import json  
  
scoring_uri = 'scoring uri for your service'  
headers = {'Content-Type':'application/json'}  
  
test_data = json.dumps({'text': 'Today is a great day!'})  
  
response = requests.post(scoring_uri, data=test_data, headers=headers)  
print(response.status_code)  
print(response.elapsed)  
print(response.json())
```

For more information on how to consume the deployed service, see [Create a client](#).

Next steps

- Monitor your Azure Machine Learning models with Application Insights
- Collect data for models in production
- How to create a client for a deployed model

Deploy a model to Azure Machine Learning compute instances

12/23/2020 • 2 minutes to read • [Edit Online](#)

Learn how to use Azure Machine Learning to deploy a model as a web service on your Azure Machine Learning compute instance. Use compute instances if one of the following conditions is true:

- You need to quickly deploy and validate your model.
- You are testing a model that is under development.

TIP

Deploying a model from a Jupyter Notebook on a compute instance, to a web service on the same VM is a *local deployment*. In this case, the 'local' computer is the compute instance. For more information on deployments, see [Deploy models with Azure Machine Learning](#).

Prerequisites

- An Azure Machine Learning workspace with a compute instance running. For more information, see [Setup environment and workspace](#).

Deploy to the compute instances

An example notebook that demonstrates local deployments is included on your compute instance. Use the following steps to load the notebook and deploy the model as a web service on the VM:

1. From [Azure Machine Learning studio](#), select your Azure Machine Learning compute instances.

2. Open the `samples-*` subdirectory, and then open

`how-to-use-azureml/deployment/deploy-to-local/register-model-deploy-local.ipynb`. Once open, run the notebook.

```
from azureml.core.webservice import LocalWebservice

#this is optional, if not provided we choose random port
deployment_config = LocalWebservice.deploy_configuration(port=6789)

local_service = Model.deploy(ws, "test", [model], inference_config, deployment_config)

local_service.wait_for_deployment()

Requirement already satisfied: pyjwt>=1.0.0 in /opt/miniconda/lib/python3.6/site-packages (from adal>=0.4.5>azureml-model-management-sdk==1.0.1b6.post1->-r /var/azureml-app/requirements.txt (line 1)) (1.7.1)

Requirement already satisfied: cffi!=1.11.3,>=1.8 in /opt/miniconda/lib/python3.6/site-packages (from cryptography>=1.1.0->adal>=0.4.5>azureml-model-management-sdk==1.0.1b6.post1->-r /var/azureml-app/requirements.txt (line 1)) (1.12.3)
Requirement already satisfied: asn1crypto>=0.21.0 in /opt/miniconda/lib/python3.6/site-packages (from cryptography>=1.1.0->adal>=0.4.5>azureml-model-management-sdk==1.0.1b6.post1->-r /var/azureml-app/requirements.txt (line 1)) (0.24.0)
Requirement already satisfied: pycparser in /opt/miniconda/lib/python3.6/site-packages (from cffi!=1.11.3,>=1.8->cryptography>1.1.0->adal>=0.4.5>azureml-model-management-sdk==1.0.1b6.post1->-r /var/azureml-app/requirements.txt (line 1)) (2.19)
--> fe3a3661fb0b
Step 7/7 : CMD ["runsvdir","/var/rununit"]
--> Running in 3611ce505d62
--> ece2403f94cc
Successfully built ece2403f94cc
Successfully tagged test:latest
Downloading model sklearn_regression_model.pkl:1 to /tmp/azureml_models_test/sklearn_regression_model.pkl:1
Starting Docker container...
Docker container running.
Checking container health...
Local webservice is running at http://localhost:6789
```

3. The notebook displays the URL and port that the service is running on. For example,

`https://localhost:6789`. You can also run the cell containing

```
print('Local service port: {}'.format(local_service.port))
```

```
print('Local service port: {}'.format(local_service.port))
```

```
Local service port: 6789
```

4. To test the service from a compute instance, use the `https://localhost:<local_service.port>` URL. To test from a remote client, get the public URL of the service running on the compute instance. The public URL can be determined using the following formula;

- Notebook VM:

```
https://<vm_name>-<local_service_port>.<azure_region_of_workspace>.notebooks.azureml.net/score
```

- Compute instance:

```
https://<vm_name>-<local_service_port>.<azure_region_of_workspace>.instances.azureml.net/score
```

For example,

- Notebook VM: `https://vm-name-6789.northcentralus.notebooks.azureml.net/score`

- Compute instance: `https://vm-name-6789.northcentralus.instances.azureml.net/score`

Test the service

To submit sample data to the running service, use the following code. Replace the value of `service_url` with the URL of from the previous step:

NOTE

When authenticating to a deployment on the compute instance, the authentication is made using Azure Active Directory.

The call to `interactive_auth.get_authentication_header()` in the example code authenticates you using AAD, and returns a header that can then be used to authenticate to the service on the compute instance. For more information, see [Set up authentication for Azure Machine Learning resources and workflows](#).

When authenticating to a deployment on Azure Kubernetes Service or Azure Container Instances, a different authentication method is used. For more information on, see [Configure authentication for Azure Machine models deployed as web services](#).

```
import requests
import json
from azureml.core.authentication import InteractiveLoginAuthentication

# Get a token to authenticate to the compute instance from remote
interactive_auth = InteractiveLoginAuthentication()
auth_header = interactive_auth.get_authentication_header()

# Create and submit a request using the auth header
headers = auth_header
# Add content type header
headers.update({'Content-Type': 'application/json'})

# Sample data to send to the service
test_sample = json.dumps({'data': [
    [1,2,3,4,5,6,7,8,9,10],
    [10,9,8,7,6,5,4,3,2,1]
]})
test_sample = bytes(test_sample,encoding = 'utf8')

# Replace with the URL for your compute instance, as determined from the previous section
service_url = "https://vm-name-6789.northcentralus.notebooks.azureml.net/score"
# for a compute instance, the url would be https://vm-name-6789.northcentralus.instances.azureml.net/score
resp = requests.post(service_url, test_sample, headers=headers)
print("prediction:", resp.text)
```

Next steps

- [How to deploy a model using a custom Docker image](#)
- [Deployment troubleshooting](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Consume a ML Model deployed as a web service](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)

Deploy a model to an Azure Kubernetes Service cluster

12/23/2020 • 15 minutes to read • [Edit Online](#)

Learn how to use Azure Machine Learning to deploy a model as a web service on Azure Kubernetes Service (AKS). Azure Kubernetes Service is good for high-scale production deployments. Use Azure Kubernetes service if you need one or more of the following capabilities:

- **Fast response time**
- **Autoscaling** of the deployed service
- **Logging**
- **Model data collection**
- **Authentication**
- **TLS termination**
- **Hardware acceleration** options such as GPU and field-programmable gate arrays (FPGA)

When deploying to Azure Kubernetes Service, you deploy to an AKS cluster that is **connected to your workspace**. For information on connecting an AKS cluster to your workspace, see [Create and attach an Azure Kubernetes Service cluster](#).

IMPORTANT

We recommend that you debug locally before deploying to the web service. For more information, see [Debug Locally](#)

You can also refer to [Azure Machine Learning - Deploy to Local Notebook](#)

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- A machine learning model registered in your workspace. If you don't have a registered model, see [How and where to deploy models](#).
- The [Azure CLI extension for Machine Learning service](#), [Azure Machine Learning Python SDK](#), or the [Azure Machine Learning Visual Studio Code extension](#).
- The **Python** code snippets in this article assume that the following variables are set:
 - `ws` - Set to your workspace.
 - `model` - Set to your registered model.
 - `inference_config` - Set to the inference configuration for the model.

For more information on setting these variables, see [How and where to deploy models](#).

- The **CLI** snippets in this article assume that you've created an `inferenceconfig.json` document. For more information on creating this document, see [How and where to deploy models](#).
- An Azure Kubernetes Service cluster connected to your workspace. For more information, see [Create and attach an Azure Kubernetes Service cluster](#).
 - If you want to deploy models to GPU nodes or FPGA nodes (or any specific SKU), then you must

create a cluster with the specific SKU. There is no support for creating a secondary node pool in an existing cluster and deploying models in the secondary node pool.

Understand the deployment processes

The word "deployment" is used in both Kubernetes and Azure Machine Learning. "Deployment" has different meanings in these two contexts. In Kubernetes, a `Deployment` is a concrete entity, specified with a declarative YAML file. A Kubernetes `Deployment` has a defined lifecycle and concrete relationships to other Kubernetes entities such as `Pods` and `ReplicaSets`. You can learn about Kubernetes from docs and videos at [What is Kubernetes?](#).

In Azure Machine Learning, "deployment" is used in the more general sense of making available and cleaning up your project resources. The steps that Azure Machine Learning considers part of deployment are:

1. Zipping the files in your project folder, ignoring those specified in `.amlignore` or `.gitignore`
2. Scaling up your compute cluster (Relates to Kubernetes)
3. Building or downloading the dockerfile to the compute node (Relates to Kubernetes)
 - 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 [Create & use software environments in Azure Machine Learning](#))
- 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 pushed to 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 (Relates to Kubernetes)

Azure ML router

The front-end component (`azureml-fe`) that routes incoming inference requests to deployed services automatically scales as needed. Scaling of `azureml-fe` is based on the AKS cluster purpose and size (number of nodes). The cluster purpose and nodes are configured when you [create or attach an AKS cluster](#). There is one `azureml-fe` service per cluster, which may be running on multiple pods.

IMPORTANT

When using a cluster configured as `dev-test`, the self-scaler is **disabled**.

`Azureml-fe` scales both up (vertically) to use more cores, and out (horizontally) to use more pods. When making the decision to scale up, the time that it takes to route incoming inference requests is used. If this time exceeds the threshold, a scale-up occurs. If the time to route incoming requests continues to exceed the threshold, a scale-out occurs.

When scaling down and in, CPU usage is used. If the CPU usage threshold is met, the front end will first be scaled down. If the CPU usage drops to the scale-in threshold, a scale-in operation happens. Scaling up and out will only occur if there are enough cluster resources available.

Deploy to AKS

To deploy a model to Azure Kubernetes Service, create a **deployment configuration** that describes the compute resources needed. For example, number of cores and memory. You also need an **inference configuration**, which describes the environment needed to host the model and web service. For more information on creating the inference configuration, see [How and where to deploy models](#).

NOTE

The number of models to be deployed is limited to 1,000 models per deployment (per container).

- [Python](#)
- [Azure CLI](#)
- [Visual Studio Code](#)

```
from azureml.core.webservice import AksWebservice, Webservice
from azureml.core.model import Model

aks_target = AksCompute(ws, "myaks")
# If deploying to a cluster configured for dev/test, ensure that it was created with enough
# cores and memory to handle this deployment configuration. Note that memory is also used by
# things such as dependencies and AML components.
deployment_config = AksWebservice.deploy_configuration(cpu_cores = 1, memory_gb = 1)
service = Model.deploy(ws, "myservice", [model], inference_config, deployment_config, aks_target)
service.wait_for_deployment(show_output = True)
print(service.state)
print(service.get_logs())
```

For more information on the classes, methods, and parameters used in this example, see the following reference documents:

- [AksCompute](#)
- [AksWebservice.deploy_configuration](#)
- [Model.deploy](#)
- [Webservice.wait_for_deployment](#)

Autoscaling

The component that handles autoscaling for Azure ML model deployments is `azureml-fe`, which is a smart request router. 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 Azure ML, 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 setting `autoscale_target_utilization`, `autoscale_min_replicas`, and `autoscale_max_replicas` for the AKS web service. The following example demonstrates how to enable

autoscaling:

```
aks_config = AksWebservice.deploy_configuration(autoscale_enabled=True,
                                                autoscale_target_utilization=30,
                                                autoscale_min_replicas=1,
                                                autoscale_max_replicas=4)
```

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 `autoscale_target_utilization`, then more replicas are created. If it is 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).

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)
```

For more information on setting `autoscale_target_utilization`, `autoscale_max_replicas`, and `autoscale_min_replicas`, see the [AksWebservice](#) module reference.

Deploy models to AKS using controlled rollout (preview)

Analyze and promote model versions in a controlled fashion using endpoints. You can deploy up to six versions behind a single endpoint. Endpoints provide the following capabilities:

- Configure the **percentage of scoring traffic sent to each endpoint**. For example, route 20% of the traffic to endpoint 'test' and 80% to 'production'.

NOTE

If you do not account for 100% of the traffic, any remaining percentage is routed to the **default** endpoint version. For example, if you configure endpoint version 'test' to get 10% of the traffic, and 'prod' for 30%, the remaining 60% is sent to the default endpoint version.

The first endpoint version created is automatically configured as the default. You can change this by setting `is_default=True` when creating or updating an endpoint version.

- Tag an endpoint version as either **control** or **treatment**. For example, the current production endpoint version might be the control, while potential new models are deployed as treatment versions. After evaluating performance of the treatment versions, if one outperforms the current control, it might be promoted to the new production/control.

NOTE

You can only have **one** control. You can have multiple treatments.

You can enable app insights to view operational metrics of endpoints and deployed versions.

Create an endpoint

Once you are ready to deploy your models, create a scoring endpoint and deploy your first version. The following example shows how to deploy and create the endpoint using the SDK. The first deployment will be defined as the default version, which means that unspecified traffic percentile across all versions will go to the default version.

TIP

In the following example, the configuration sets the initial endpoint version to handle 20% of the traffic. Since this is the first endpoint, it's also the default version. And since we don't have any other versions for the other 80% of traffic, it is routed to the default as well. Until other versions that take a percentage of traffic are deployed, this one effectively receives 100% of the traffic.

```
import azureml.core,
from azureml.core.webservice import AksEndpoint
from azureml.core.compute import AksCompute
from azureml.core.compute import ComputeTarget
# select a created compute
compute = ComputeTarget(ws, 'myaks')

# define the endpoint and version name
endpoint_name = "mynewendpoint"
version_name= "versiona"
# create the deployment config and define the scoring traffic percentile for the first deployment
endpoint_deployment_config = AksEndpoint.deploy_configuration(cpu_cores = 0.1, memory_gb = 0.2,
                                                               enable_app_insights = True,
                                                               tags = {'scikitlearn':'demo'},
                                                               description = "testing versions",
                                                               version_name = version_name,
                                                               traffic_percentile = 20)

# deploy the model and endpoint
endpoint = Model.deploy(ws, endpoint_name, [model], inference_config, endpoint_deployment_config, compute)
# Wait for he process to complete
endpoint.wait_for_deployment(True)
```

Update and add versions to an endpoint

Add another version to your endpoint and configure the scoring traffic percentile going to the version. There are two types of versions, a control and a treatment version. There can be multiple treatment versions to help compare against a single control version.

TIP

The second version, created by the following code snippet, accepts 10% of traffic. The first version is configured for 20%, so only 30% of the traffic is configured for specific versions. The remaining 70% is sent to the first endpoint version, because it is also the default version.

```

from azureml.core.webservice import AksEndpoint

# add another model deployment to the same endpoint as above
version_name_add = "versionb"
endpoint.create_version(version_name = version_name_add,
                        inference_config=inference_config,
                        models=[model],
                        tags = {'modelVersion':'b'},
                        description = "my second version",
                        traffic_percentile = 10)
endpoint.wait_for_deployment(True)

```

Update existing versions or delete them in an endpoint. You can change the version's default type, control type, and the traffic percentile. In the following example, the second version increases its traffic to 40% and is now the default.

TIP

After the following code snippet, the second version is now default. It is now configured for 40%, while the original version is still configured for 20%. This means that 40% of traffic is not accounted for by version configurations. The leftover traffic will be routed to the second version, because it is now default. It effectively receives 80% of the traffic.

```

from azureml.core.webservice import AksEndpoint

# update the version's scoring traffic percentage and if it is a default or control type
endpoint.update_version(version_name=endpoint.versions["versionb"].name,
                        description="my second version update",
                        traffic_percentile=40,
                        is_default=True,
                        is_control_version_type=True)
# Wait for the process to complete before deleting
endpoint.wait_for_deployment(true)
# delete a version in an endpoint
endpoint.delete_version(version_name="versionb")

```

Web service authentication

When deploying to Azure Kubernetes Service, **key-based** authentication is enabled by default. You can also enable **token-based** authentication. Token-based authentication requires clients to use an Azure Active Directory account to request an authentication token, which is used to make requests to the deployed service.

To **disable** authentication, set the `auth_enabled=False` parameter when creating the deployment configuration. The following example disables authentication using the SDK:

```
deployment_config = AksWebservice.deploy_configuration(cpu_cores=1, memory_gb=1, auth_enabled=False)
```

For information on authenticating from a client application, see the [Consume an Azure Machine Learning model deployed as a web service](#).

Authentication with keys

If key authentication is enabled, you can use the `get_keys` method to retrieve a primary and secondary authentication key:

```
primary, secondary = service.get_keys()  
print(primary)
```

IMPORTANT

If you need to regenerate a key, use `service.regen_key`

Authentication with tokens

To enable token authentication, set the `token_auth_enabled=True` parameter when you are creating or updating a deployment. The following example enables token authentication using the SDK:

```
deployment_config = AksWebservice.deploy_configuration(cpu_cores=1, memory_gb=1, token_auth_enabled=True)
```

If token authentication is enabled, you can use the `get_token` method to retrieve a JWT token and that token's expiration time:

```
token, refresh_by = service.get_token()  
print(token)
```

IMPORTANT

You will need to request a new token after the token's `refresh_by` time.

Microsoft strongly recommends that you create your Azure Machine Learning workspace in the same region as your Azure Kubernetes Service cluster. To authenticate with a token, the web service will make a call to the region in which your Azure Machine Learning workspace is created. If your workspace's region is unavailable, then you will not be able to fetch a token for your web service even, if your cluster is in a different region than your workspace. This effectively results in Token-based Authentication being unavailable until your workspace's region is available again. In addition, the greater the distance between your cluster's region and your workspace's region, the longer it will take to fetch a token.

To retrieve a token, you must use the Azure Machine Learning SDK or the `az ml service get-access-token` command.

Vulnerability scanning

Azure Security Center provides unified security management and advanced threat protection across hybrid cloud workloads. You should allow Azure Security Center to scan your resources and follow its recommendations. For more, see [Azure Kubernetes Services integration with Security Center](#).

Next steps

- [Use Azure RBAC for Kubernetes authorization](#)
- [Secure inferencing environment with Azure Virtual Network](#)
- [How to deploy a model using a custom Docker image](#)
- [Deployment troubleshooting](#)
- [Update web service](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Consume a ML Model deployed as a web service](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)

Deploy a model to Azure Container Instances

12/23/2020 • 4 minutes to read • [Edit Online](#)

Learn how to use Azure Machine Learning to deploy a model as a web service on Azure Container Instances (ACI). Use Azure Container Instances if one of the following conditions is true:

- You need to quickly deploy and validate your model. You do not need to create ACI containers ahead of time. They are created as part of the deployment process.
- You are testing a model that is under development.

For information on quota and region availability for ACI, see [Quotas and region availability for Azure Container Instances](#) article.

IMPORTANT

It is highly advised to debug locally before deploying to the web service, for more information see [Debug Locally](#)

You can also refer to [Azure Machine Learning - Deploy to Local Notebook](#)

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- A machine learning model registered in your workspace. If you don't have a registered model, see [How and where to deploy models](#).
- The [Azure CLI extension for Machine Learning service](#), [Azure Machine Learning Python SDK](#), or the [Azure Machine Learning Visual Studio Code extension](#).
- The **Python** code snippets in this article assume that the following variables are set:

- `ws` - Set to your workspace.
- `model` - Set to your registered model.
- `inference_config` - Set to the inference configuration for the model.

For more information on setting these variables, see [How and where to deploy models](#).

- The **CLI** snippets in this article assume that you've created an `inferenceconfig.json` document. For more information on creating this document, see [How and where to deploy models](#).

Limitations

- When using Azure Container Instances in a virtual network, the virtual network must be in the same resource group as your Azure Machine Learning workspace.
- When using Azure Container Instances inside the virtual network, the Azure Container Registry (ACR) for your workspace cannot also be in the virtual network.

For more information, see [How to secure inferencing with virtual networks](#).

Deploy to ACI

To deploy a model to Azure Container Instances, create a **deployment configuration** that describes the compute

resources needed. For example, number of cores and memory. You also need an [inference configuration](#), which describes the environment needed to host the model and web service. For more information on creating the inference configuration, see [How and where to deploy models](#).

NOTE

- ACI is suitable only for small models that are under 1 GB in size.
- We recommend using single-node AKS to dev-test larger models.
- The number of models to be deployed is limited to 1,000 models per deployment (per container).

Using the SDK

```
from azureml.core.webservice import AciWebservice, Webservice
from azureml.core.model import Model

deployment_config = AciWebservice.deploy_configuration(cpu_cores = 1, memory_gb = 1)
service = Model.deploy(ws, "aciservice", [model], inference_config, deployment_config)
service.wait_for_deployment(show_output = True)
print(service.state)
```

For more information on the classes, methods, and parameters used in this example, see the following reference documents:

- [AciWebservice.deploy_configuration](#)
- [Model.deploy](#)
- [Webservice.wait_for_deployment](#)

Using the CLI

To deploy using the CLI, use the following command. Replace `mymodel:1` with the name and version of the registered model. Replace `myservice` with the name to give this service:

```
az ml model deploy -m mymodel:1 -n myservice -ic inferenceconfig.json -dc deploymentconfig.json
```

The entries in the `deploymentconfig.json` document map to the parameters for [AciWebservice.deploy_configuration](#). The following table describes the mapping between the entities in the JSON document and the parameters for the method:

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
<code>computeType</code>	NA	The compute target. For ACI, the value must be <code>ACI</code> .
<code>containerResourceRequirements</code>	NA	Container for the CPU and memory entities.
<code>cpu</code>	<code>cpu_cores</code>	The number of CPU cores to allocate. Defaults, <code>0.1</code>
<code>memoryInGB</code>	<code>memory_gb</code>	The amount of memory (in GB) to allocate for this web service. Default, <code>0.5</code>

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
location	location	The Azure region to deploy this Webservice to. If not specified the Workspace location will be used. More details on available regions can be found here: ACI Regions
authEnabled	auth_enabled	Whether to enable auth for this Webservice. Defaults to False
sslEnabled	ssl_enabled	Whether to enable SSL for this Webservice. Defaults to False.
appInsightsEnabled	enable_app_insights	Whether to enable AppInsights for this Webservice. Defaults to False
sslCertificate	ssl_cert_pem_file	The cert file needed if SSL is enabled
sslKey	ssl_key_pem_file	The key file needed if SSL is enabled
cname	ssl_cname	The cname for if SSL is enabled
dnsNameLabel	dns_name_label	The dns name label for the scoring endpoint. If not specified a unique dns name label will be generated for the scoring endpoint.

The following JSON is an example deployment configuration for use with the CLI:

```
{
  "computeType": "aci",
  "containerResourceRequirements":
  {
    "cpu": 0.5,
    "memoryInGB": 1.0
  },
  "authEnabled": true,
  "sslEnabled": false,
  "appInsightsEnabled": false
}
```

For more information, see the [az ml model deploy](#) reference.

Using VS Code

See [deploy your models with VS Code](#).

IMPORTANT

You don't need to create an ACI container to test in advance. ACI containers are created as needed.

IMPORTANT

We append hashed workspace id to all underlying ACI resources which are created, all ACI names from same workspace will have same suffix. The Azure Machine Learning service name would still be the same customer provided "service_name" and all the user facing Azure Machine Learning SDK APIs do not need any change. We do not give any guarantees on the names of underlying resources being created.

Next steps

- [How to deploy a model using a custom Docker image](#)
- [Deployment troubleshooting](#)
- [Update the web service](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Consume a ML Model deployed as a web service](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)

Deploy a deep learning model for inference with GPU

12/23/2020 • 6 minutes to read • [Edit Online](#)

This article teaches you how to use Azure Machine Learning to deploy a GPU-enabled model as a web service. The information in this article is based on deploying a model on Azure Kubernetes Service (AKS). The AKS cluster provides a GPU resource that is used by the model for inference.

Inference, or model scoring, is the phase where the deployed model is used to make predictions. Using GPUs instead of CPUs offers performance advantages on highly parallelizable computation.

IMPORTANT

For web service deployments, GPU inference is only supported on Azure Kubernetes Service. For inference using a **machine learning pipeline**, GPUs are only supported on Azure Machine Learning Compute. For more information on using ML pipelines, see [Tutorial: Build an Azure Machine Learning pipeline for batch scoring](#).

TIP

Although the code snippets in this article use a TensorFlow model, you can apply the information to any machine learning framework that supports GPUs.

NOTE

The information in this article builds on the information in the [How to deploy to Azure Kubernetes Service](#) article. Where that article generally covers deployment to AKS, this article covers GPU specific deployment.

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- A Python development environment with the Azure Machine Learning SDK installed. For more information, see [Azure Machine Learning SDK](#).
- A registered model that uses a GPU.
 - To learn how to register models, see [Deploy Models](#).
 - To create and register the Tensorflow model used to create this document, see [How to Train a TensorFlow Model](#).
- A general understanding of [How and where to deploy models](#).

Connect to your workspace

To connect to an existing workspace, use the following code:

IMPORTANT

This code snippet expects the workspace configuration to be saved in the current directory or its parent. For more information on creating a workspace, see [Create and manage Azure Machine Learning workspaces](#). For more information on saving the configuration to file, see [Create a workspace configuration file](#).

```
from azureml.core import Workspace

# Connect to the workspace
ws = Workspace.from_config()
```

Create a Kubernetes cluster with GPUs

Azure Kubernetes Service provides many different GPU options. You can use any of them for model inference. See [the list of N-series VMs](#) for a full breakdown of capabilities and costs.

The following code demonstrates how to create a new AKS cluster for your workspace:

```
from azureml.core.compute import ComputeTarget, AksCompute
from azureml.exceptions import ComputeTargetException

# Choose a name for your cluster
aks_name = "aks-gpu"

# Check to see if the cluster already exists
try:
    aks_target = ComputeTarget(workspace=ws, name=aks_name)
    print('Found existing compute target')
except ComputeTargetException:
    print('Creating a new compute target...')
    # Provision AKS cluster with GPU machine
    prov_config = AksCompute.provisioning_configuration(vm_size="Standard_NC6")

    # Create the cluster
    aks_target = ComputeTarget.create(
        workspace=ws, name=aks_name, provisioning_configuration=prov_config
    )

    aks_target.wait_for_completion(show_output=True)
```

IMPORTANT

Azure will bill you as long as the AKS cluster exists. Make sure to delete your AKS cluster when you're done with it.

For more information on using AKS with Azure Machine Learning, see [How to deploy to Azure Kubernetes Service](#).

Write the entry script

The entry script receives data submitted to the web service, passes it to the model, and returns the scoring results. The following script loads the Tensorflow model on startup, and then uses the model to score data.

TIP

The entry script is specific to your model. For example, the script must know the framework to use with your model, data formats, etc.

```

import json
import numpy as np
import os
import tensorflow as tf

from azureml.core.model import Model

def init():
    global X, output, sess
    tf.reset_default_graph()
    model_root = os.getenv('AZUREML_MODEL_DIR')
    # the name of the folder in which to look for tensorflow model files
    tf_model_folder = 'model'
    saver = tf.train.import_meta_graph(
        os.path.join(model_root, tf_model_folder, 'mnist-tf.model.meta'))
    X = tf.get_default_graph().get_tensor_by_name("network/X:0")
    output = tf.get_default_graph().get_tensor_by_name("network/output/MatMul:0")

    sess = tf.Session()
    saver.restore(sess, os.path.join(model_root, tf_model_folder, 'mnist-tf.model'))

def run(raw_data):
    data = np.array(json.loads(raw_data)['data'])
    # make prediction
    out = output.eval(session=sess, feed_dict={X: data})
    y_hat = np.argmax(out, axis=1)
    return y_hat.tolist()

```

This file is named `score.py`. For more information on entry scripts, see [How and where to deploy](#).

Define the conda environment

The conda environment file specifies the dependencies for the service. It includes dependencies required by both the model and the entry script. Please note that you must indicate `azureml-defaults` with version $\geq 1.0.45$ as a pip dependency, because it contains the functionality needed to host the model as a web service. The following YAML defines the environment for a Tensorflow model. It specifies `tensorflow-gpu`, which will make use of the GPU used in this deployment:

```

name: project_environment
dependencies:
    # The python interpreter version.
    # Currently Azure ML only supports 3.5.2 and later.
    - python=3.6.2

    - pip:
        # You must list azureml-defaults as a pip dependency
        - azureml-defaults>=1.0.45
        - numpy
        - tensorflow-gpu=1.12
channels:
    - conda-forge

```

For this example, the file is saved as `myenv.yml`.

Define the deployment configuration

IMPORTANT

AKS does not allow pods to share GPUs, you can have only as many replicas of a GPU-enabled web service as there are GPUs in the cluster.

The deployment configuration defines the Azure Kubernetes Service environment used to run the web service:

```
from azureml.core.webservice import AksWebservice

gpu_aks_config = AksWebservice.deploy_configuration(autoscale_enabled=False,
                                                    num_replicas=3,
                                                    cpu_cores=2,
                                                    memory_gb=4)
```

For more information, see the reference documentation for [AksService.deploy_configuration](#).

Define the inference configuration

The inference configuration points to the entry script and an environment object, which uses a docker image with GPU support. Please note that the YAML file used for environment definition must list `azureml-defaults` with version `>= 1.0.45` as a pip dependency, because it contains the functionality needed to host the model as a web service.

```
from azureml.core.model import InferenceConfig
from azureml.core.environment import Environment, DEFAULT_GPU_IMAGE

myenv = Environment.from_conda_specification(name="myenv", file_path="myenv.yml")
myenv.docker.base_image = DEFAULT_GPU_IMAGE
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)
```

For more information on environments, see [Create and manage environments for training and deployment](#). For more information, see the reference documentation for [InferenceConfig](#).

Deploy the model

Deploy the model to your AKS cluster and wait for it to create your service.

```
from azureml.core.model import Model

# Name of the web service that is deployed
aks_service_name = 'aks-dnn-mnist'
# Get the registered model
model = Model(ws, "tf-dnn-mnist")
# Deploy the model
aks_service = Model.deploy(ws,
                           models=[model],
                           inference_config=inference_config,
                           deployment_config=gpu_aks_config,
                           deployment_target=aks_target,
                           name=aks_service_name)

aks_service.wait_for_deployment(show_output=True)
print(aks_service.state)
```

For more information, see the reference documentation for [Model](#).

Issue a sample query to your service

Send a test query to the deployed model. When you send a jpeg image to the model, it scores the image. The following code sample downloads test data and then selects a random test image to send to the service.

```
# Used to test your webservice
import os
import urllib
import gzip
import numpy as np
import struct
import requests

# load compressed MNIST gz files and return numpy arrays
def load_data(filename, label=False):
    with gzip.open(filename) as gz:
        struct.unpack('I', gz.read(4))
        n_items = struct.unpack('>I', gz.read(4))
        if not label:
            n_rows = struct.unpack('>I', gz.read(4))[0]
            n_cols = struct.unpack('>I', gz.read(4))[0]
            res = np.frombuffer(gz.read(n_items[0] * n_rows * n_cols), dtype=np.uint8)
            res = res.reshape(n_items[0], n_rows * n_cols)
        else:
            res = np.frombuffer(gz.read(n_items[0]), dtype=np.uint8)
            res = res.reshape(n_items[0], 1)
    return res

# one-hot encode a 1-D array
def one_hot_encode(array, num_of_classes):
    return np.eye(num_of_classes)[array.reshape(-1)]

# Download test data
os.makedirs('./data/mnist', exist_ok=True)
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz',
                           filename='./data/mnist/test-images.gz')
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz',
                           filename='./data/mnist/test-labels.gz')

# Load test data from model training
X_test = load_data('./data/mnist/test-images.gz', False) / 255.0
y_test = load_data('./data/mnist/test-labels.gz', True).reshape(-1)

# send a random row from the test set to score
random_index = np.random.randint(0, len(X_test)-1)
input_data = "{\"data\": [" + str(list(X_test[random_index])) + "]}""

api_key = aks_service.get_keys()[0]
headers = {'Content-Type': 'application/json',
           'Authorization': ('Bearer ' + api_key)}
resp = requests.post(aks_service.scoring_uri, input_data, headers=headers)

print("POST to url", aks_service.scoring_uri)
print("label:", y_test[random_index])
print("prediction:", resp.text)
```

For more information on creating a client application, see [Create client to consume deployed web service](#).

Clean up the resources

If you created the AKS cluster specifically for this example, delete your resources after you're done.

IMPORTANT

Azure bills you based on how long the AKS cluster is deployed. Make sure to clean it up after you are done with it.

```
aks_service.delete()  
aks_target.delete()
```

Next steps

- [Deploy model on FPGA](#)
- [Deploy model with ONNX](#)
- [Train Tensorflow DNN Models](#)

Deploy a machine learning model to Azure App Service (preview)

12/23/2020 • 7 minutes to read • [Edit Online](#)

Learn how to deploy a model from Azure Machine Learning as a web app in Azure App Service.

IMPORTANT

While both Azure Machine Learning and Azure App Service are generally available, the ability to deploy a model from the Machine Learning service to App Service is in preview.

With Azure Machine Learning, you can create Docker images from trained machine learning models. This image contains a web service that receives data, submits it to the model, and then returns the response. Azure App Service can be used to deploy the image, and provides the following features:

- Advanced [authentication](#) for enhanced security. Authentication methods include both Azure Active Directory and multi-factor auth.
- [Autoscale](#) without having to redeploy.
- [TLS support](#) for secure communications between clients and the service.

For more information on features provided by Azure App Service, see the [App Service overview](#).

IMPORTANT

If you need the ability to log the scoring data used with your deployed model, or the results of scoring, you should instead deploy to Azure Kubernetes Service. For more information, see [Collect data on your production models](#).

Prerequisites

- An Azure Machine Learning workspace. For more information, see the [Create a workspace](#) article.
- The [Azure CLI](#).
- A trained machine learning model registered in your workspace. If you do not have a model, use the [Image classification tutorial: train model](#) to train and register one.

IMPORTANT

The code snippets in this article assume that you have set the following variables:

- `ws` - Your Azure Machine Learning workspace.
- `model` - The registered model that will be deployed.
- `inference_config` - The inference configuration for the model.

For more information on setting these variables, see [Deploy models with Azure Machine Learning](#).

Prepare for deployment

Before deploying, you must define what is needed to run the model as a web service. The following list describes

the main items needed for a deployment:

- An **entry script**. This script accepts requests, scores the request using the model, and returns the results.

IMPORTANT

The entry script is specific to your model; it must understand the format of the incoming request data, the format of the data expected by your model, and the format of the data returned to clients.

If the request data is in a format that is not usable by your model, the script can transform it into an acceptable format. It may also transform the response before returning to it to the client.

IMPORTANT

The Azure Machine Learning SDK does not provide a way for the web service access your datastore or data sets. If you need the deployed model to access data stored outside the deployment, such as in an Azure Storage account, you must develop a custom code solution using the relevant SDK. For example, the [Azure Storage SDK for Python](#).

Another alternative that may work for your scenario is [batch predictions](#), which does provide access to datastores when scoring.

For more information on entry scripts, see [Deploy models with Azure Machine Learning](#).

- **Dependencies**, such as helper scripts or Python/Conda packages required to run the entry script or model

These entities are encapsulated into an **inference configuration**. The inference configuration references the entry script and other dependencies.

IMPORTANT

When creating an inference configuration for use with Azure App Service, you must use an [Environment](#) object. Please note that if you are defining a custom environment, you must add `azureml-defaults` with version $\geq 1.0.45$ as a pip dependency. This package contains the functionality needed to host the model as a web service. The following example demonstrates creating an environment object and using it with an inference configuration:

```
from azureml.core.environment import Environment
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core.model import InferenceConfig

# Create an environment and add conda dependencies to it
myenv = Environment(name="myenv")
# Enable Docker based environment
myenv.docker.enabled = True
# Build conda dependencies
myenv.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'],
                                                               pip_packages=['azureml-defaults'])
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)
```

For more information on environments, see [Create and manage environments for training and deployment](#).

For more information on inference configuration, see [Deploy models with Azure Machine Learning](#).

IMPORTANT

When deploying to Azure App Service, you do not need to create a **deployment configuration**.

Create the image

To create the Docker image that is deployed to Azure App Service, use [Model.package](#). The following code snippet demonstrates how to build a new image from the model and inference configuration:

NOTE

The code snippet assumes that `model` contains a registered model, and that `inference_config` contains the configuration for the inference environment. For more information, see [Deploy models with Azure Machine Learning](#).

```
from azureml.core import Model

package = Model.package(ws, [model], inference_config)
package.wait_for_creation(show_output=True)
# Display the package location/ACR path
print(package.location)
```

When `show_output=True`, the output of the Docker build process is shown. Once the process finishes, the image has been created in the Azure Container Registry for your workspace. Once the image has been built, the location in your Azure Container Registry is displayed. The location returned is in the format

`<acrinstance>.azurecr.io/package@sha256:<imagename>`. For example,

`myml08024f78fd10.azurecr.io/package@sha256:20190827151241`.

IMPORTANT

Save the location information, as it is used when deploying the image.

Deploy image as a web app

1. Use the following command to get the login credentials for the Azure Container Registry that contains the image. Replace `<acrinstance>` with the value returned previously from `package.location`:

```
az acr credential show --name <myacr>
```

The output of this command is similar to the following JSON document:

```
{
  "passwords": [
    {
      "name": "password",
      "value": "Iv01RZQ9762LUJrFiffo3P4sWgk4q+nW"
    },
    {
      "name": "password2",
      "value": "=pKCxHatX96jeoYBWZLsPR6opszr==mg"
    }
  ],
  "username": "myml08024f78fd10"
}
```

Save the value for **username** and one of the **passwords**.

2. If you do not already have a resource group or app service plan to deploy the service, the following commands demonstrate how to create both:

```
az group create --name myresourcegroup --location "West Europe"
az appservice plan create --name myplanname --resource-group myresourcegroup --sku B1 --is-linux
```

In this example, a **Basic** pricing tier (`--sku B1`) is used.

IMPORTANT

Images created by Azure Machine Learning use Linux, so you must use the `--is-linux` parameter.

3. To create the web app, use the following command. Replace `<app-name>` with the name you want to use. Replace `<acrinstance>` and `<imagename>` with the values from returned `package.location` earlier:

```
az webapp create --resource-group myresourcegroup --plan myplanname --name <app-name> --deployment-
container-image-name <acrinstance>.azurecr.io/package@sha256:<imagename>
```

This command returns information similar to the following JSON document:

```
{
  "adminSiteName": null,
  "appServicePlanName": "myplanname",
  "geoRegion": "West Europe",
  "hostingEnvironmentProfile": null,
  "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myplanname",
  "kind": "linux",
  "location": "West Europe",
  "maximumNumberOfWorkers": 1,
  "name": "myplanname",
  < JSON data removed for brevity. >
  "targetWorkerSizeId": 0,
  "type": "Microsoft.Web/serverfarms",
  "workerTierName": null
}
```

IMPORTANT

At this point, the web app has been created. However, since you haven't provided the credentials to the Azure Container Registry that contains the image, the web app is not active. In the next step, you provide the authentication information for the container registry.

4. To provide the web app with the credentials needed to access the container registry, use the following command. Replace `<app-name>` with the name you want to use. Replace `<acrinstance>` and `<imagename>` with the values from returned `package.location` earlier. Replace `<username>` and `<password>` with the ACR login information retrieved earlier:

```
az webapp config container set --name <app-name> --resource-group myresourcegroup --docker-custom-image-
name <acrinstance>.azurecr.io/package@sha256:<imagename> --docker-registry-server-url
https://<acrinstance>.azurecr.io --docker-registry-server-user <username> --docker-registry-server-
password <password>
```

This command returns information similar to the following JSON document:

```
[  
 {  
   "name": "WEBSITES_ENABLE_APP_SERVICE_STORAGE",  
   "slotSetting": false,  
   "value": "false"  
 },  
 {  
   "name": "DOCKER_REGISTRY_SERVER_URL",  
   "slotSetting": false,  
   "value": "https://myml08024f78fd10.azurecr.io"  
 },  
 {  
   "name": "DOCKER_REGISTRY_SERVER_USERNAME",  
   "slotSetting": false,  
   "value": "myml08024f78fd10"  
 },  
 {  
   "name": "DOCKER_REGISTRY_SERVER_PASSWORD",  
   "slotSetting": false,  
   "value": null  
 },  
 {  
   "name": "DOCKER_CUSTOM_IMAGE_NAME",  
   "value": "DOCKER|myml08024f78fd10.azurecr.io/package@sha256:20190827195524"  
 }  
 ]
```

At this point, the web app begins loading the image.

IMPORTANT

It may take several minutes before the image has loaded. To monitor progress, use the following command:

```
az webapp log tail --name <app-name> --resource-group myresourcegroup
```

Once the image has been loaded and the site is active, the log displays a message that states

```
Container <container name> for site <app-name> initialized successfully and is ready to serve requests .
```

Once the image is deployed, you can find the hostname by using the following command:

```
az webapp show --name <app-name> --resource-group myresourcegroup
```

This command returns information similar to the following hostname - `<app-name>.azurewebsites.net`. Use this value as part of the **base url** for the service.

Use the Web App

The web service that passes requests to the model is located at `{baseurl}/score`. For example, `https://<app-name>.azurewebsites.net(score)`. The following Python code demonstrates how to submit data to the URL and display the response:

```
import requests
import json

scoring_uri = "https://mywebapp.azurewebsites.net/score"

headers = {'Content-Type':'application/json'}

test_sample = json.dumps({'data': [
    [1,2,3,4,5,6,7,8,9,10],
    [10,9,8,7,6,5,4,3,2,1]
]})

response = requests.post(scoring_uri, data=test_sample, headers=headers)
print(response.status_code)
print(response.elapsed)
print(response.json())
```

Next steps

- Learn to configure your Web App in the [App Service on Linux](#) documentation.
- Learn more about scaling in [Get started with Autoscale in Azure](#).
- [Use a TLS/SSL certificate in your Azure App Service](#).
- [Configure your App Service app to use Azure Active Directory sign-in](#).
- [Consume a ML Model deployed as a web service](#)

Deploy a machine learning model to Azure Functions (preview)

12/23/2020 • 8 minutes to read • [Edit Online](#)

Learn how to deploy a model from Azure Machine Learning as a function app in Azure Functions.

IMPORTANT

While both Azure Machine Learning and Azure Functions are generally available, the ability to package a model from the Machine Learning service for Functions is in preview.

With Azure Machine Learning, you can create Docker images from trained machine learning models. Azure Machine Learning now has the preview functionality to build these machine learning models into function apps, which can be [deployed into Azure Functions](#).

Prerequisites

- An Azure Machine Learning workspace. For more information, see the [Create a workspace](#) article.
- The [Azure CLI](#).
- A trained machine learning model registered in your workspace. If you do not have a model, use the [Image classification tutorial: train model](#) to train and register one.

IMPORTANT

The code snippets in this article assume that you have set the following variables:

- `ws` - Your Azure Machine Learning workspace.
- `model` - The registered model that will be deployed.
- `inference_config` - The inference configuration for the model.

For more information on setting these variables, see [Deploy models with Azure Machine Learning](#).

Prepare for deployment

Before deploying, you must define what is needed to run the model as a web service. The following list describes the core items needed for a deployment:

- An **entry script**. This script accepts requests, scores the request using the model, and returns the results.

IMPORTANT

The entry script is specific to your model; it must understand the format of the incoming request data, the format of the data expected by your model, and the format of the data returned to clients.

If the request data is in a format that is not usable by your model, the script can transform it into an acceptable format. It may also transform the response before returning it to the client.

By default when packaging for functions, the input is treated as text. If you are interested in consuming the raw bytes of the input (for instance for Blob triggers), you should use [AMLRequest to accept raw data](#).

For more information on entry script, see [Define scoring code](#)

- **Dependencies**, such as helper scripts or Python/Conda packages required to run the entry script or model

These entities are encapsulated into an **inference configuration**. The inference configuration references the entry script and other dependencies.

IMPORTANT

When creating an inference configuration for use with Azure Functions, you must use an [Environment](#) object. Please note that if you are defining a custom environment, you must add `azureml-defaults` with version $\geq 1.0.45$ as a pip dependency. This package contains the functionality needed to host the model as a web service. The following example demonstrates creating an environment object and using it with an inference configuration:

```
from azureml.core.environment import Environment
from azureml.core.conda_dependencies import CondaDependencies

# Create an environment and add conda dependencies to it
myenv = Environment(name="myenv")
# Enable Docker based environment
myenv.docker.enabled = True
# Build conda dependencies
myenv.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'],
                                                          pip_packages=['azureml-defaults'])
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)
```

For more information on environments, see [Create and manage environments for training and deployment](#).

For more information on inference configuration, see [Deploy models with Azure Machine Learning](#).

IMPORTANT

When deploying to Functions, you do not need to create a **deployment configuration**.

Install the SDK preview package for functions support

To build packages for Azure Functions, you must install the SDK preview package.

```
pip install azureml-contrib-functions
```

Create the image

To create the Docker image that is deployed to Azure Functions, use `azureml.contrib.functions.package` or the specific package function for the trigger you are interested in using. The following code snippet demonstrates how to create a new package with a blob trigger from the model and inference configuration:

NOTE

The code snippet assumes that `model` contains a registered model, and that `inference_config` contains the configuration for the inference environment. For more information, see [Deploy models with Azure Machine Learning](#).

```

from azureml.contrib.functions import package
from azureml.contrib.functions import BLOB_TRIGGER
blob = package(ws, [model], inference_config, functions_enabled=True, trigger=BLOB_TRIGGER,
input_path="input/{blobname}.json", output_path="output/{blobname}_out.json")
blob.wait_for_creation(show_output=True)
# Display the package location/ACR path
print(blob.location)

```

When `show_output=True`, the output of the Docker build process is shown. Once the process finishes, the image has been created in the Azure Container Registry for your workspace. Once the image has been built, the location in your Azure Container Registry is displayed. The location returned is in the format

`<acrinstance>.azurecr.io/package@sha256:<imagename>`.

NOTE

Packaging for functions currently supports HTTP Triggers, Blob triggers and Service bus triggers. For more information on triggers, see [Azure Functions bindings](#).

IMPORTANT

Save the location information, as it is used when deploying the image.

Deploy image as a web app

1. Use the following command to get the login credentials for the Azure Container Registry that contains the image. Replace `<myacr>` with the value returned previously from `blob.location`:

```
az acr credential show --name <myacr>
```

The output of this command is similar to the following JSON document:

```
{
  "passwords": [
    {
      "name": "password",
      "value": "Iv01RZQ9762LUJrFiffo3P4sWgk4q+nW"
    },
    {
      "name": "password2",
      "value": "=pKCxHatX96jeoYBWZLsPR6opszr==mg"
    }
  ],
  "username": "mym108024f78fd10"
}
```

Save the value for `username` and one of the `passwords`.

2. If you do not already have a resource group or app service plan to deploy the service, the following commands demonstrate how to create both:

```
az group create --name myresourcegroup --location "West Europe"
az appservice plan create --name myplanname --resource-group myresourcegroup --sku B1 --is-linux
```

In this example, a *Linux basic* pricing tier (`--sku B1`) is used.

IMPORTANT

Images created by Azure Machine Learning use Linux, so you must use the `--is-linux` parameter.

3. Create the storage account to use for the web job storage and get its connection string. Replace `<webjobStorage>` with the name you want to use.

```
az storage account create --name <webjobStorage> --location westeurope --resource-group myresourcegroup  
--sku Standard_LRS
```

```
az storage account show-connection-string --resource-group myresourcegroup --name <webJobStorage> --  
query connectionString --output tsv
```

4. To create the function app, use the following command. Replace `<app-name>` with the name you want to use. Replace `<acrinstance>` and `<imagename>` with the values from returned `package.location` earlier. Replace `<webjobStorage>` with the name of the storage account from the previous step:

```
az functionapp create --resource-group myresourcegroup --plan myplanname --name <app-name> --deployment-  
container-image-name <acrinstance>.azurecr.io/package:<imagename> --storage-account <webjobStorage>
```

IMPORTANT

At this point, the function app has been created. However, since you haven't provided the connection string for the blob trigger or credentials to the Azure Container Registry that contains the image, the function app is not active. In the next steps, you provide the connection string and the authentication information for the container registry.

5. Create the storage account to use for the blob trigger storage and get its connection string. Replace `<triggerStorage>` with the name you want to use.

```
az storage account create --name <triggerStorage> --location westeurope --resource-group myresourcegroup  
--sku Standard_LRS
```

```
az storage account show-connection-string --resource-group myresourcegroup --name <triggerStorage> --  
query connectionString --output tsv
```

Record this connection string to provide to the function app. We will use it later when we ask for `<triggerConnectionString>`

6. Create the containers for the input and output in the storage account. Replace `<triggerConnectionString>` with the connection string returned earlier:

```
az storage container create -n input --connection-string <triggerConnectionString>
```

```
az storage container create -n output --connection-string <triggerConnectionString>
```

7. To associate the trigger connection string with the function app, use the following command. Replace `<app-name>` with the name of the function app. Replace `<triggerConnectionString>` with the connection string returned earlier:

```
az functionapp config appsettings set --name <app-name> --resource-group myresourcegroup --settings "TriggerConnectionString=<triggerConnectionString>"
```

8. You will need to retrieve the tag associated with the created container using the following command.

Replace `<username>` with the username returned earlier from the container registry:

```
az acr repository show-tags --repository package --name <username> --output tsv
```

Save the value returned, it will be used as the `<imagetag>` in the next step.

9. To provide the function app with the credentials needed to access the container registry, use the following command. Replace `<app-name>` with the name of the function app. Replace `<acrinstance>` and `<imagetag>` with the values from the AZ CLI call in the previous step. Replace `<username>` and `<password>` with the ACR login information retrieved earlier:

```
az functionapp config container set --name <app-name> --resource-group myresourcegroup --docker-custom-image-name <acrinstance>.azurecr.io/package:<imagetag> --docker-registry-server-url https://<acrinstance>.azurecr.io --docker-registry-server-user <username> --docker-registry-server-password <password>
```

This command returns information similar to the following JSON document:

```
[  
{  
    "name": "WEBSITES_ENABLE_APP_SERVICE_STORAGE",  
    "slotSetting": false,  
    "value": "false"  
},  
{  
    "name": "DOCKER_REGISTRY_SERVER_URL",  
    "slotSetting": false,  
    "value": "https://mym108024f78fd10.azurecr.io"  
},  
{  
    "name": "DOCKER_REGISTRY_SERVER_USERNAME",  
    "slotSetting": false,  
    "value": "mym108024f78fd10"  
},  
{  
    "name": "DOCKER_REGISTRY_SERVER_PASSWORD",  
    "slotSetting": false,  
    "value": null  
},  
{  
    "name": "DOCKER_CUSTOM_IMAGE_NAME",  
    "value": "DOCKER|mym108024f78fd10.azurecr.io/package:20190827195524"  
}  
]
```

At this point, the function app begins loading the image.

IMPORTANT

It may take several minutes before the image has loaded. You can monitor progress using the Azure Portal.

Test the deployment

Once the image has loaded and the app is available, use the following steps to trigger the app:

1. Create a text file that contains the data that the score.py file expects. The following example would work with a score.py that expects an array of 10 numbers:

```
{"data": [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]]}
```

IMPORTANT

The format of the data depends on what your score.py and model expects.

2. Use the following command to upload this file to the input container in the trigger storage blob created earlier. Replace <file> with the name of the file containing the data. Replace <triggerConnectionString> with the connection string returned earlier. In this example, `input` is the name of the input container created earlier. If you used a different name, replace this value:

```
az storage blob upload --container-name input --file <file> --name <file> --connection-string <triggerConnectionString>
```

The output of this command is similar to the following JSON:

```
{
  "etag": "\"0x8D7C21528E08844\"",
  "lastModified": "2020-03-06T21:27:23+00:00"
}
```

3. To view the output produced by the function, use the following command to list the output files generated. Replace <triggerConnectionString> with the connection string returned earlier. In this example, `output` is the name of the output container created earlier. If you used a different name, replace this value:

```
az storage blob list --container-name output --connection-string <triggerConnectionString> --query '[].name' --output tsv
```

The output of this command is similar to `sample_input_out.json`.

4. To download the file and inspect the contents, use the following command. Replace <file> with the file name returned by the previous command. Replace <triggerConnectionString> with the connection string returned earlier:

```
az storage blob download --container-name output --file <file> --name <file> --connection-string <triggerConnectionString>
```

Once the command completes, open the file. It contains the data returned by the model.

For more information on using blob triggers, see the [Create a function triggered by Azure Blob storage](#) article.

Next steps

- Learn to configure your Functions App in the [Functions documentation](#).
- Learn more about Blob storage triggers [Azure Blob storage bindings](#).
- [Deploy your model to Azure App Service](#).
- [Consume a ML Model deployed as a web service](#)

- [API Reference](#)

Deploy a model for use with Cognitive Search

12/23/2020 • 8 minutes to read • [Edit Online](#)

This article teaches you how to use Azure Machine Learning to deploy a model for use with [Azure Cognitive Search](#).

Cognitive Search performs content processing over heterogenous content, to make it queryable by humans or applications. This process can be enhanced by using a model deployed from Azure Machine Learning.

Azure Machine Learning can deploy a trained model as a web service. The web service is then embedded in a Cognitive Search *skill*, which becomes part of the processing pipeline.

IMPORTANT

The information in this article is specific to the deployment of the model. It provides information on the supported deployment configurations that allow the model to be used by Cognitive Search.

For information on how to configure Cognitive Search to use the deployed model, see the [Build and deploy a custom skill with Azure Machine Learning](#) tutorial.

For the sample that the tutorial is based on, see <https://github.com/Azure-Samples/azure-search-python-samples/tree/master/AzureML-Custom-Skill>.

When deploying a model for use with Azure Cognitive Search, the deployment must meet the following requirements:

- Use Azure Kubernetes Service to host the model for inference.
- Enable transport layer security (TLS) for the Azure Kubernetes Service. TLS is used to secure HTTPS communications between Cognitive Search and the deployed model.
- The entry script must use the `inference_schema` package to generate an OpenAPI (Swagger) schema for the service.
- The entry script must also accept JSON data as input, and generate JSON as output.

Prerequisites

- An Azure Machine Learning workspace. For more information, see [Create an Azure Machine Learning workspace](#).
- A Python development environment with the Azure Machine Learning SDK installed. For more information, see [Azure Machine Learning SDK](#).
- A registered model. If you do not have a model, use the example notebook at <https://github.com/Azure-Samples/azure-search-python-samples/tree/master/AzureML-Custom-Skill>.
- A general understanding of [How and where to deploy models](#).

Connect to your workspace

An Azure Machine Learning workspace provides 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.

To connect to an existing workspace, use the following code:

IMPORTANT

This code snippet expects the workspace configuration to be saved in the current directory or its parent. For more information, see [Create and manage Azure Machine Learning workspaces](#). For more information on saving the configuration to file, see [Create a workspace configuration file](#).

```
from azureml.core import Workspace

try:
    # Load the workspace configuration from local cached info
    ws = Workspace.from_config()
    print(ws.name, ws.location, ws.resource_group, ws.location, sep='\t')
    print('Library configuration succeeded')
except:
    print('Workspace not found')
```

Create a Kubernetes cluster

Time estimate: Approximately 20 minutes.

A Kubernetes cluster is a set of virtual machine instances (called nodes) that are used for running containerized applications.

When you deploy a model from Azure Machine Learning to Azure Kubernetes Service, the model and all the assets needed to host it as a web service are packaged into a Docker container. This container is then deployed onto the cluster.

The following code demonstrates how to create a new Azure Kubernetes Service (AKS) cluster for your workspace:

TIP

You can also attach an existing Azure Kubernetes Service to your Azure Machine Learning workspace. For more information, see [How to deploy models to Azure Kubernetes Service](#).

IMPORTANT

Notice that the code uses the `enable_ssl()` method to enable transport layer security (TLS) for the cluster. This is required when you plan on using the deployed model from Cognitive Services.

```

from azureml.core.compute import AksCompute, ComputeTarget
# Create or attach to an AKS inferencing cluster

# Create the provisioning configuration with defaults
prov_config = AksCompute.provisioning_configuration()

# Enable TLS (sometimes called SSL) communications
# Leaf domain label generates a name using the formula
# "<leaf-domain-label>#####.<azure-region>.cloudapp.azure.net"
# where "#####" is a random series of characters
prov_config.enable_ssl(leaf_domain_label = "contoso")

cluster_name = 'amlskills'
# Try to use an existing compute target by that name.
# If one doesn't exist, create one.
try:

    aks_target = ComputeTarget(ws, cluster_name)
    print("Attaching to existing cluster")
except Exception as e:
    print("Creating new cluster")
    aks_target = ComputeTarget.create(workspace = ws,
                                      name = cluster_name,
                                      provisioning_configuration = prov_config)
# Wait for the create process to complete
aks_target.wait_for_completion(show_output = True)

```

IMPORTANT

Azure will bill you as long as the AKS cluster exists. Make sure to delete your AKS cluster when you're done with it.

For more information on using AKS with Azure Machine Learning, see [How to deploy to Azure Kubernetes Service](#).

Write the entry script

The entry script receives data submitted to the web service, passes it to the model, and returns the scoring results. The following script loads the model on startup, and then uses the model to score data. This file is sometimes called `score.py`.

TIP

The entry script is specific to your model. For example, the script must know the framework to use with your model, data formats, etc.

IMPORTANT

When you plan on using the deployed model from Azure Cognitive Services, you must use the `inference_schema` package to enable schema generation for the deployment. This package provides decorators that allow you to define the input and output data format for the web service that performs inference using the model.

```

from azureml.core.model import Model
from nlp_architect.models.absa.inference.inference import SentimentInference
from spacy.cli.download import download as spacy_download
import traceback
import json
# Inference schema for schema discovery
from inference_schema.schema_decorators import input_schema, output_schema
from inference_schema.parameter_types.numpy_parameter_type import NumpyParameterType
from inference_schema.parameter_types.standard_py_parameter_type import StandardPythonParameterType

def init():
    """
    Set up the ABSA model for Inference
    """
    global SentInference
    spacy_download('en')
    aspect_lex = Model.get_model_path('hotel_aspect_lex')
    opinion_lex = Model.get_model_path('hotel_opinion_lex')
    SentInference = SentimentInference(aspect_lex, opinion_lex)

    # Use inference schema decorators and sample input/output to
    # build the OpenAPI (Swagger) schema for the deployment
    standard_sample_input = {'text': 'a sample input record containing some text' }
    standard_sample_output = {"sentiment": {"sentence": "This place makes false booking prices, when you get there, they say they do not have the reservation for that day.", "terms": [{"text": "hotels", "type": "AS", "polarity": "POS", "score": 1.0, "start": 300, "len": 6}, {"text": "nice", "type": "OP", "polarity": "POS", "score": 1.0, "start": 295, "len": 4}]}}
    @input_schema('raw_data', StandardPythonParameterType(standard_sample_input))
    @output_schema(StandardPythonParameterType(standard_sample_output))
    def run(raw_data):
        try:
            # Get the value of the 'text' field from the JSON input and perform inference
            input_txt = raw_data["text"]
            doc = SentInference.run(doc=input_txt)
            if doc is None:
                return None
            sentences = doc._sentences
            result = {"sentence": doc._doc_text}
            terms = []
            for sentence in sentences:
                for event in sentence._events:
                    for x in event:
                        term = {"text": x._text, "type": x._type.value, "polarity": x._polarity.value, "score": x._score, "start": x._start, "len": x._len }
                        terms.append(term)
            result["terms"] = terms
            print("Success!")
            # Return the results to the client as a JSON document
            return {"sentiment": result}
        except Exception as e:
            result = str(e)
            # return error message back to the client
            print("Failure!")
            print(traceback.format_exc())
            return json.dumps({"error": result, "tb": traceback.format_exc()})

```

For more information on entry scripts, see [How and where to deploy](#).

Define the software environment

The environment class is used to define the Python dependencies for the service. It includes dependencies required by both the model and the entry script. In this example, it installs packages from the regular pypi index, as well as from a GitHub repo.

```
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core import Environment

conda = None
pip = ["azureml-defaults", "azureml-monitoring",
       "git+https://github.com/NervanaSystems/nlp-architect.git@absa", 'nlp-architect', 'inference-schema',
       "spacy==2.0.18"]

conda_deps = CondaDependencies.create(conda_packages=None, pip_packages=pip)

myenv = Environment(name='myenv')
myenv.python.conda_dependencies = conda_deps
```

For more information on environments, see [Create and manage environments for training and deployment](#).

Define the deployment configuration

The deployment configuration defines the Azure Kubernetes Service hosting environment used to run the web service.

TIP

If you aren't sure about the memory, CPU, or GPU needs of your deployment, you can use profiling to learn these. For more information, see [How and where to deploy a model](#).

```
from azureml.core.model import Model
from azureml.core.webservice import Webservice
from azureml.core.image import ContainerImage
from azureml.core.webservice import AksWebservice, Webservice

# If deploying to a cluster configured for dev/test, ensure that it was created with enough
# cores and memory to handle this deployment configuration. Note that memory is also used by
# things such as dependencies and AML components.

aks_config = AksWebservice.deploy_configuration(autoscale_enabled=True,
                                                autoscale_min_replicas=1,
                                                autoscale_max_replicas=3,
                                                autoscale_refresh_seconds=10,
                                                autoscale_target_utilization=70,
                                                auth_enabled=True,
                                                cpu_cores=1, memory_gb=2,
                                                scoring_timeout_ms=5000,
                                                replica_max_concurrent_requests=2,
                                                max_request_wait_time=5000)
```

For more information, see the reference documentation for [AksService.deploy_configuration](#).

Define the inference configuration

The inference configuration points to the entry script and the environment object:

```
from azureml.core.model import InferenceConfig
inf_config = InferenceConfig(entry_script='score.py', environment=myenv)
```

For more information, see the reference documentation for [InferenceConfig](#).

Deploy the model

Deploy the model to your AKS cluster and wait for it to create your service. In this example, two registered models are loaded from the registry and deployed to AKS. After deployment, the `score.py` file in the deployment loads these models and uses them to perform inference.

```
from azureml.core.webservice import AksWebservice, Webservice

c_aspect_lex = Model(ws, 'hotel_aspect_lex')
c_opinion_lex = Model(ws, 'hotel_opinion_lex')
service_name = "hotel-absa-v2"

aks_service = Model.deploy(workspace=ws,
                           name=service_name,
                           models=[c_aspect_lex, c_opinion_lex],
                           inference_config=inference_config,
                           deployment_config=aks_config,
                           deployment_target=aks_target,
                           overwrite=True)

aks_service.wait_for_deployment(show_output = True)
print(aks_service.state)
```

For more information, see the reference documentation for [Model](#).

Issue a sample query to your service

The following example uses the deployment information stored in the `aks_service` variable by the previous code section. It uses this variable to retrieve the scoring URL and authentication token needed to communicate with the service:

```
import requests
import json

primary, secondary = aks_service.get_keys()

# Test data
input_data = '{"raw_data": {"text": "This is a nice place for a relaxing evening out with friends. The owners seem pretty nice, too. I have been there a few times including last night. Recommend."}}'

# Since authentication was enabled for the deployment, set the authorization header.
headers = {'Content-Type':'application/json', 'Authorization':('Bearer ' + primary)}

# Send the request and display the results
resp = requests.post(aks_service.scoring_uri, input_data, headers=headers)
print(resp.text)
```

The result returned from the service is similar to the following JSON:

```
{"sentiment": {"sentence": "This is a nice place for a relaxing evening out with friends. The owners seem pretty nice, too. I have been there a few times including last night. Recommend.", "terms": [{"text": "place", "type": "AS", "polarity": "POS", "score": 1.0, "start": 15, "len": 5}, {"text": "nice", "type": "OP", "polarity": "POS", "score": 1.0, "start": 10, "len": 4}]}}
```

Connect to Cognitive Search

For information on using this model from Cognitive Search, see the [Build and deploy a custom skill with Azure Machine Learning](#) tutorial.

Clean up the resources

If you created the AKS cluster specifically for this example, delete your resources after you're done testing it with Cognitive Search.

IMPORTANT

Azure bills you based on how long the AKS cluster is deployed. Make sure to clean it up after you are done with it.

```
aks_service.delete()  
aks_target.delete()
```

Next steps

- [Build and deploy a custom skill with Azure Machine Learning](#)

Deploy a model using a custom Docker base image

12/23/2020 • 11 minutes to read • [Edit Online](#)

Learn how to use a custom Docker base image when deploying trained models with Azure Machine Learning.

Azure Machine Learning will use a default base Docker image if none is specified. You can find the specific Docker image used with `azureml.core.runconfig.DEFAULT_CPU_IMAGE`. You can also use Azure Machine Learning environments to select a specific base image, or use a custom one that you provide.

A base image is used as the starting point when an image is created for a deployment. It provides the underlying operating system and components. The deployment process then adds additional components, such as your model, conda environment, and other assets, to the image.

Typically, you create a custom base image when you want to use Docker to manage your dependencies, maintain tighter control over component versions or save time during deployment. You might also want to install software required by your model, where the installation process takes a long time. Installing the software when creating the base image means that you don't have to install it for each deployment.

IMPORTANT

When you deploy a model, you cannot override core components such as the web server or IoT Edge components. These components provide a known working environment that is tested and supported by Microsoft.

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.

This document is broken into two sections:

- Create a custom base image: Provides information to admins and DevOps on creating a custom image and configuring authentication to an Azure Container Registry using the Azure CLI and Machine Learning CLI.
- Deploy a model using a custom base image: Provides information to Data Scientists and DevOps / ML Engineers on using custom images when deploying a trained model from the Python SDK or ML CLI.

Prerequisites

- An Azure Machine Learning workspace. For more information, see the [Create a workspace](#) article.
- The [Azure Machine Learning SDK](#).
- The [Azure CLI](#).
- The [CLI extension for Azure Machine Learning](#).
- An [Azure Container Registry](#) or other Docker registry that is accessible on the internet.
- The steps in this document assume that you are familiar with creating and using an **inference configuration** object as part of model deployment. For more information, see [Where to deploy and how](#).

Create a custom base image

The information in this section assumes that you are using an Azure Container Registry to store Docker images.

Use the following checklist when planning to create custom images for Azure Machine Learning:

- Will you use the Azure Container Registry created for the Azure Machine Learning workspace, or a standalone Azure Container Registry?

When using images stored in the **container registry for the workspace**, you do not need to authenticate to the registry. Authentication is handled by the workspace.

WARNING

The Azure Container Registry for your workspace is **created the first time you train or deploy a model** using the workspace. If you've created a new workspace, but not trained or created a model, no Azure Container Registry will exist for the workspace.

When using images stored in a **standalone container registry**, you will need to configure a service principal that has at least read access. You then provide the service principal ID (username) and password to anyone that uses images from the registry. The exception is if you make the container registry publicly accessible.

For information on creating a private Azure Container Registry, see [Create a private container registry](#).

For information on using service principals with Azure Container Registry, see [Azure Container Registry authentication with service principals](#).

- Azure Container Registry and image information: Provide the image name to anyone that needs to use it. For example, an image named `myimage`, stored in a registry named `myregistry`, is referenced as `myregistry.azurecr.io/myimage` when using the image for model deployment

Image requirements

Azure Machine Learning only supports Docker images that provide the following software:

- Ubuntu 16.04 or greater.
- Conda 4.5.# or greater.
- Python 3.5+.

To use Datasets, please install the libfuse-dev package. Also make sure to install any user space packages you may need.

Azure ML maintains a set of CPU and GPU base images published to Microsoft Container Registry that you can optionally leverage (or reference) instead of creating your own custom image. To see the Dockerfiles for those images, refer to the [Azure/AzureML-Containers](#) GitHub repository.

For GPU images, Azure ML currently offers both cuda9 and cuda10 base images. The major dependencies installed in these base images are:

DEPENDENCIES	INTELMPI CPU	OPENMPI CPU	INTELMPI GPU	OPENMPI GPU
miniconda	<code>==4.5.11</code>	<code>==4.5.11</code>	<code>==4.5.11</code>	<code>==4.5.11</code>
mpi	<code>intelmpi==2018.3.22</code> 2	<code>openmpi==3.1.2</code>	<code>intelmpi==2018.3.22</code> 2	<code>openmpi==3.1.2</code>
cuda	-	-	9.0/10.0	9.0/10.0/10.1
cudnn	-	-	7.4/7.5	7.4/7.5

DEPENDENCIES	INTELMPI CPU	OPENMPI CPU	INTELMPI GPU	OPENMPI GPU
nccl	-	-	2.4	2.4
git	2.7.4	2.7.4	2.7.4	2.7.4

The CPU images are built from ubuntu16.04. The GPU images for cuda9 are built from nvidia/cuda:9.0-cudnn7-devel-ubuntu16.04. The GPU images for cuda10 are built from nvidia/cuda:10.0-cudnn7-devel-ubuntu16.04.

IMPORTANT

When using custom Docker images, it is recommended that you pin package versions in order to better ensure reproducibility.

Get container registry information

In this section, learn how to get the name of the Azure Container Registry for your Azure Machine Learning workspace.

WARNING

The Azure Container Registry for your workspace is **created the first time you train or deploy a model** using the workspace. If you've created a new workspace, but not trained or created a model, no Azure Container Registry will exist for the workspace.

If you've already trained or deployed models using Azure Machine Learning, a container registry was created for your workspace. To find the name of this container registry, use the following steps:

1. Open a new shell or command-prompt and use the following command to authenticate to your Azure subscription:

```
az login
```

Follow the prompts to authenticate to the subscription.

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](#).

2. Use the following command to list the container registry for the workspace. Replace `<myworkspace>` with your Azure Machine Learning workspace name. Replace `<resourcegroup>` with the Azure resource group that contains your workspace:

```
az ml workspace show -w <myworkspace> -g <resourcegroup> --query containerRegistry
```

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
```

The information returned is similar to the following text:

```
/subscriptions/<subscription_id>/resourceGroups/<resource_group>/providers/Microsoft.ContainerRegistry/registries/<registry_name>
```

The <registry_name> value is the name of the Azure Container Registry for your workspace.

Build a custom base image

The steps in this section walk-through creating a custom Docker image in your Azure Container Registry. For sample dockerfiles, see the [Azure/AzureML-Containers GitHub repo](#).

1. Create a new text file named `Dockerfile`, and use the following text as the contents:

```
FROM ubuntu:16.04

ARG CONDA_VERSION=4.7.12
ARG PYTHON_VERSION=3.7
ARG AZUREML_SDK_VERSION=1.13.0
ARG INFERENCE_SCHEMA_VERSION=1.1.0

ENV LANG=C.UTF-8 LC_ALL=C.UTF-8
ENV PATH /opt/miniconda/bin:$PATH
ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update --fix-missing && \
    apt-get install -y wget bzip2 && \
    apt-get install -y fuse && \
    apt-get clean -y && \
    rm -rf /var/lib/apt/lists/*

RUN useradd --create-home dockeruser
WORKDIR /home/dockeruser
USER dockeruser

RUN wget --quiet https://repo.anaconda.com/miniconda/Miniconda3-${CONDA_VERSION}-Linux-x86_64.sh -O ~/miniconda.sh && \
    /bin/bash ~/miniconda.sh -b -p ~/miniconda && \
    rm ~/miniconda.sh && \
    ~/miniconda/bin/conda clean -tipsy
ENV PATH="/home/dockeruser/miniconda/bin/:${PATH}"

RUN conda install -y conda=${CONDA_VERSION} python=${PYTHON_VERSION} && \
    pip install azureml-defaults==${AZUREML_SDK_VERSION} inference-schema==${INFERENCE_SCHEMA_VERSION} && \
    conda clean -aqy && \
    rm -rf ~/miniconda/pkgs && \
    find ~/miniconda/ -type d -name __pycache__ -prune -exec rm -rf {} \;
```

2. From a shell or command-prompt, use the following to authenticate to the Azure Container Registry.

Replace the <registry_name> with the name of the container registry you want to store the image in:

```
az acr login --name <registry_name>
```

3. To upload the Dockerfile, and build it, use the following command. Replace <registry_name> with the name of the container registry you want to store the image in:

```
az acr build --image myimage:v1 --registry <registry_name> --file Dockerfile .
```

TIP

In this example, a tag of :v1 is applied to the image. If no tag is provided, a tag of :latest is applied.

During the build process, information is streamed to back to the command line. If the build is successful, you receive a message similar to the following text:

```
Run ID: cda was successful after 2m56s
```

For more information on building images with an Azure Container Registry, see [Build and run a container image using Azure Container Registry Tasks](#)

For more information on uploading existing images to an Azure Container Registry, see [Push your first image to a private Docker container registry](#).

Use a custom base image

To use a custom image, you need the following information:

- The **image name**. For example,

mcr.microsoft.com/azureml/o16n-sample-user-base/ubuntu-miniconda:latest is the path to a simple Docker Image provided by Microsoft.

IMPORTANT

For custom images that you've created, be sure to include any tags that were used with the image. For example, if your image was created with a specific tag, such as :v1. If you did not use a specific tag when creating the image, a tag of :latest was applied.

- If the image is in a **private repository**, you need the following information:

- The registry **address**. For example, myregistry.azurecr.io.
- A service principal **username** and **password** that has read access to the registry.

If you do not have this information, speak to the administrator for the Azure Container Registry that contains your image.

Publicly available base images

Microsoft provides several docker images on a publicly accessible repository, which can be used with the steps in this section:

IMAGE	DESCRIPTION
mcr.microsoft.com/azureml/o16n-sample-user-base/ubuntu-miniconda	Core image for Azure Machine Learning
mcr.microsoft.com/azureml/onnxruntime:latest	Contains ONNX Runtime for CPU inferencing

IMAGE	DESCRIPTION
mcr.microsoft.com/azureml/onnxruntime:latest-cuda	Contains the ONNX Runtime and CUDA for GPU
mcr.microsoft.com/azureml/onnxruntime:latest-tensorrt	Contains ONNX Runtime and TensorRT for GPU
mcr.microsoft.com/azureml/onnxruntime:latest-openvino-vadm	Contains ONNX Runtime and OpenVINO for Intel Vision Accelerator Design based on Movidius™ MyriadX VPUs
mcr.microsoft.com/azureml/onnxruntime:latest-openvino-myriad	Contains ONNX Runtime and OpenVINO for Intel Movidius™ USB sticks

For more information about the ONNX Runtime base images see the [ONNX Runtime dockerfile section](#) in the GitHub repo.

TIP

Since these images are publicly available, you do not need to provide an address, username or password when using them.

For more information, see [Azure Machine Learning containers](#) repository on GitHub.

Use an image with the Azure Machine Learning SDK

To use an image stored in the **Azure Container Registry for your workspace**, or a **container registry that is publicly accessible**, set the following [Environment](#) attributes:

- `docker.enabled=True`
- `docker.base_image` : Set to the registry and path to the image.

```
from azureml.core.environment import Environment
# Create the environment
myenv = Environment(name="myenv")
# Enable Docker and reference an image
myenv.docker.enabled = True
myenv.docker.base_image = "mcr.microsoft.com/azureml/o16n-sample-user-base/ubuntu-miniconda:latest"
```

To use an image from a **private container registry** that is not in your workspace, you must use `docker.base_image_registry` to specify the address of the repository and a user name and password:

```
# Set the container registry information
myenv.docker.base_image_registry.address = "myregistry.azurecr.io"
myenv.docker.base_image_registry.username = "username"
myenv.docker.base_image_registry.password = "password"

myenv.inferencing_stack_version = "latest" # This will install the inference specific apt packages.

# Define the packages needed by the model and scripts
from azureml.core.conda_dependencies import CondaDependencies
conda_dep = CondaDependencies()
# you must list azureml-defaults as a pip dependency
conda_dep.add_pip_package("azureml-defaults")
myenv.python.conda_dependencies=conda_dep
```

You must add `azureml-defaults` with version $\geq 1.0.45$ as a pip dependency. This package contains the functionality needed to host the model as a web service. You must also set `inferencing_stack_version` property

on the environment to "latest", this will install specific apt packages needed by web service.

After defining the environment, use it with an [InferenceConfig](#) object to define the inference environment in which the model and web service will run.

```
from azureml.core.model import InferenceConfig
# Use environment in InferenceConfig
inference_config = InferenceConfig(entry_script="score.py",
                                    environment=myenv)
```

At this point, you can continue with deployment. For example, the following code snippet would deploy a web service locally using the inference configuration and custom image:

```
from azureml.core.webservice import LocalWebservice, Webservice

deployment_config = LocalWebservice.deploy_configuration(port=8890)
service = Model.deploy(ws, "myservice", [model], inference_config, deployment_config)
service.wait_for_deployment(show_output = True)
print(service.state)
```

For more information on deployment, see [Deploy models with Azure Machine Learning](#).

For more information on customizing your Python environment, see [Create and manage environments for training and deployment](#).

Use an image with the Machine Learning CLI

IMPORTANT

Currently the Machine Learning CLI can use images from the Azure Container Registry for your workspace or publicly accessible repositories. It cannot use images from standalone private registries.

Before deploying a model using the Machine Learning CLI, create an [environment](#) that uses the custom image. Then create an inference configuration file that references the environment. You can also define the environment directly in the inference configuration file. The following JSON document demonstrates how to reference an image in a public container registry. In this example, the environment is defined inline:

```
{
    "entryScript": "score.py",
    "environment": {
        "docker": {
            "arguments": [],
            "baseDockerfile": null,
            "baseImage": "mcr.microsoft.com/azureml/o16n-sample-user-base/ubuntu-miniconda:latest",
            "enabled": false,
            "sharedVolumes": true,
            "shmSize": null
        },
        "environmentVariables": {
            "EXAMPLE_ENV_VAR": "EXAMPLE_VALUE"
        },
        "name": "my-deploy-env",
        "python": {
            "baseCondaEnvironment": null,
            "condaDependencies": {
                "channels": [
                    "conda-forge"
                ],
                "dependencies": [
                    "python=3.6.2",
                    {
                        "pip": [
                            "azureml-defaults",
                            "azureml-telemetry",
                            "scikit-learn",
                            "inference-schema[numpy-support]"
                        ]
                    }
                ],
                "name": "project_environment"
            },
            "condaDependenciesFile": null,
            "interpreterPath": "python",
            "userManagedDependencies": false
        },
        "version": "1"
    }
}
```

This file is used with the `az ml model deploy` command. The `--ic` parameter is used to specify the inference configuration file.

```
az ml model deploy -n myservice -m mymodel:1 --ic inferenceconfig.json --dc deploymentconfig.json --ct akscomputetarget
```

For more information on deploying a model using the ML CLI, see the "model registration, profiling, and deployment" section of the [CLI extension for Azure Machine Learning](#) article.

Next steps

- Learn more about [Where to deploy and how](#).
- Learn how to [Train and deploy machine learning models using Azure Pipelines](#).

Deploy ML models to field-programmable gate arrays (FPGAs) with Azure Machine Learning

12/23/2020 • 9 minutes to read • [Edit Online](#)

In this article, you learn about FPGAs and how to deploy your ML models to an Azure FPGA using the [hardware-accelerated models Python package](#) from [Azure Machine Learning](#).

What are FPGAs?

FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects. The interconnects allow these blocks to be configured in various ways after manufacturing. Compared to other chips, FPGAs provide a combination of programmability and performance.

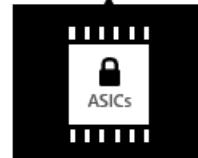
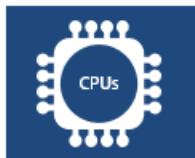
FPGAs make it possible to achieve low latency for real-time inference (or model scoring) requests. Asynchronous requests (batching) aren't needed. Batching can cause latency, because more data needs to be processed. Implementations of neural processing units don't require batching; therefore the latency can be many times lower, compared to CPU and GPU processors.

You can reconfigure FPGAs for different types of machine learning models. This flexibility makes it easier to accelerate the applications based on the most optimal numerical precision and memory model being used. Because FPGAs are reconfigurable, you can stay current with the requirements of rapidly changing AI algorithms.

Silicon alternatives

TRAINING
CPUs and GPUs, limited FPGAs,
ASICs under investigation

EVALUATION
CPUs and FPGAs,
ASICs under investigation



FLEXIBILITY

EFFICIENCY

PROCESSOR	ABBREVIATION	DESCRIPTION
Application-specific integrated circuits	ASICs	Custom circuits, such as Google's Tensor Processor Units (TPU), provide the highest efficiency. They can't be reconfigured as your needs change.
Field-programmable gate arrays	FPGAs	FPGAs, such as those available on Azure, provide performance close to ASICs. They are also flexible and reconfigurable over time, to implement new logic.

PROCESSOR	ABBREVIATION	DESCRIPTION
Graphics processing units	GPUs	A popular choice for AI computations. GPUs offer parallel processing capabilities, making it faster at image rendering than CPUs.
Central processing units	CPUs	General-purpose processors, the performance of which isn't ideal for graphics and video processing.

FPGA support in Azure

Microsoft Azure is the world's largest cloud investment in FPGAs. Microsoft uses FPGAs for deep neural networks (DNN) evaluation, Bing search ranking, and software defined networking (SDN) acceleration to reduce latency, while freeing CPUs for other tasks.

FPGAs on Azure are based on Intel's FPGA devices, which data scientists and developers use to accelerate real-time AI calculations. This FPGA-enabled architecture offers performance, flexibility, and scale, and is available on Azure.

Azure FPGAs are integrated with Azure Machine Learning. Azure can parallelize pre-trained DNN across FPGAs to scale out your service. The DNNs can be pre-trained, as a deep featurizer for transfer learning, or fine-tuned with updated weights.

SCENARIOS & CONFIGURATIONS ON AZURE	SUPPORTED DNN MODELS	REGIONAL SUPPORT
<ul style="list-style-type: none"> + Image classification and recognition scenarios + TensorFlow deployment (requires Tensorflow 1.x) + Intel FPGA hardware 	<ul style="list-style-type: none"> - ResNet 50 - ResNet 152 - DenseNet-121 - VGG-16 - SSD-VGG 	<ul style="list-style-type: none"> - East US - Southeast Asia - West Europe - West US 2

To optimize latency and throughput, your client sending data to the FPGA model should be in one of the regions above (the one you deployed the model to).

The **PBS Family of Azure VMs** contains Intel Arria 10 FPGAs. It will show as "Standard PBS Family vCPUs" when you check your Azure quota allocation. The PB6 VM has six vCPUs and one FPGA. PB6 VM is automatically provisioned by Azure Machine Learning during model deployment to an FPGA. It is only used with Azure ML, and it cannot run arbitrary bitstreams. For example, you will not be able to flash the FPGA with bitstreams to do encryption, encoding, etc.

Deploy models on FPGAs

You can deploy a model as a web service on FPGAs with [Azure Machine Learning Hardware Accelerated Models](#). Using FPGAs provides ultra-low latency inference, even with a single batch size.

In this example, you create a TensorFlow graph to preprocess the input image, make it a featurizer using ResNet 50 on an FPGA, and then run the features through a classifier trained on the ImageNet data set. Then, the model is deployed to an AKS cluster.

Prerequisites

- An Azure subscription. If you do not have one, create a [pay-as-you-go](#) account (free Azure accounts are not eligible for FPGA quota).
- An Azure Machine Learning workspace and the Azure Machine Learning SDK for Python installed, as

described in [Create a workspace](#).

- The hardware-accelerated models package: `pip install --upgrade azureml-accel-models[cpu]`
- The [Azure CLI](#)
- FPGA quota. Submit a [request for quota](#), or run this CLI command to check quota:

```
az vm list-usage --location "eastus" -o table --query "[?localName=='Standard PBS Family vCPUs']"
```

Make sure you have at least 6 vCPUs under the **CurrentValue** returned.

Define the TensorFlow model

Begin by using the [Azure Machine Learning SDK for Python](#) to create a service definition. A service definition is a file describing a pipeline of graphs (input, featurizer, and classifier) based on TensorFlow. The deployment command compresses the definition and graphs into a ZIP file, and uploads the ZIP to Azure Blob storage. The DNN is already deployed to run on the FPGA.

1. Load Azure Machine Learning workspace

```
import os
import tensorflow as tf

from azureml.core import Workspace

ws = Workspace.from_config()
print(ws.name, ws.resource_group, ws.location, ws.subscription_id, sep='\n')
```

2. Preprocess image.

The input to the web service is a JPEG image. The first step is to decode the JPEG image and preprocess it. The JPEG images are treated as strings and the result are tensors that will be the input to the ResNet 50 model.

```
# Input images as a two-dimensional tensor containing an arbitrary number of images represented as
# strings
import azureml.accel.models.utils as utils
tf.reset_default_graph()

in_images = tf.placeholder(tf.string)
image_tensors = utils.preprocess_array(in_images)
print(image_tensors.shape)
```

3. Load featurizer.

Initialize the model and download a TensorFlow checkpoint of the quantized version of ResNet50 to be used as a featurizer. Replace "QuantizedResnet50" in the code snippet to import other deep neural networks:

- QuantizedResnet152
- QuantizedVgg16
- Densenet121

```
from azureml.accel.models import QuantizedResnet50
save_path = os.path.expanduser('~/models')
model_graph = QuantizedResnet50(save_path, is_frozen=True)
feature_tensor = model_graph.import_graph_def(image_tensors)
print(model_graph.version)
print(feature_tensor.name)
print(feature_tensor.shape)
```

4. Add a classifier. This classifier was trained on the ImageNet data set.

```
classifier_output = model_graph.get_default_classifier(feature_tensor)
print(classifier_output)
```

5. Save the model. Now that the preprocessor, ResNet 50 featurizer, and the classifier have been loaded, save the graph and associated variables as a model.

```
model_name = "resnet50"
model_save_path = os.path.join(save_path, model_name)
print("Saving model in {}".format(model_save_path))

with tf.Session() as sess:
    model_graph.restore_weights(sess)
    tf.saved_model.simple_save(sess, model_save_path,
                               inputs={'images': in_images},
                               outputs={'output_alias': classifier_output})
```

6. Save input and output tensors as you will use them for model conversion and inference requests.

```
input_tensors = in_images.name
output_tensors = classifier_output.name

print(input_tensors)
print(output_tensors)
```

The following models are available listed with their classifier output tensors for inference if you used the default classifier.

- Resnet50, QuantizedResnet50

```
output_tensors = "classifier_1/resnet_v1_50/predictions/Softmax:0"
```

- Resnet152, QuantizedResnet152

```
output_tensors = "classifier/resnet_v1_152/predictions/Softmax:0"
```

- Densenet121, QuantizedDensenet121

```
output_tensors = "classifier/densenet121/predictions/Softmax:0"
```

- Vgg16, QuantizedVgg16

```
output_tensors = "classifier/vgg_16/fc8/squeezed:0"
```

- SsdVgg, QuantizedSsdVgg

```
output_tensors = ['ssd_300_vgg/block4_box/Reshape_1:0', 'ssd_300_vgg/block7_box/Reshape_1:0',
'ssd_300_vgg/block8_box/Reshape_1:0', 'ssd_300_vgg/block9_box/Reshape_1:0',
:ssd_300_vgg/block10_box/Reshape_1:0', 'ssd_300_vgg/block11_box/Reshape_1:0',
'ssd_300_vgg/block4_box/Reshape:0', 'ssd_300_vgg/block7_box/Reshape:0',
:ssd_300_vgg/block8_box/Reshape:0', 'ssd_300_vgg/block9_box/Reshape:0',
:ssd_300_vgg/block10_box/Reshape:0', 'ssd_300_vgg/block11_box/Reshape:0']
```

Convert the model to the Open Neural Network Exchange format (ONNX)

Before you can deploy to FPGAs, convert the model to the [ONNX](#) format.

1. Register the model by using the SDK with the ZIP file in Azure Blob storage. Adding tags and other metadata about the model helps you keep track of your trained models.

```
from azureml.core.model import Model

registered_model = Model.register(workspace=ws,
                                   model_path=model_save_path,
                                   model_name=model_name)

print("Successfully registered: ", registered_model.name,
      registered_model.description, registered_model.version, sep='\t')
```

If you've already registered a model and want to load it, you may retrieve it.

```
from azureml.core.model import Model
model_name = "resnet50"
# By default, the latest version is retrieved. You can specify the version, i.e. version=1
registered_model = Model(ws, name="resnet50")
print(registered_model.name, registered_model.description,
      registered_model.version, sep='\t')
```

2. Convert the TensorFlow graph to the ONNX format. You must provide the names of the input and output tensors, so your client can use them when you consume the web service.

```
from azureml.accel import AccelOnnxConverter

convert_request = AccelOnnxConverter.convert_tf_model(
    ws, registered_model, input_tensors, output_tensors)

# If it fails, you can run wait_for_completion again with show_output=True.
convert_request.wait_for_completion(show_output=False)

# If the above call succeeded, get the converted model
converted_model = convert_request.result
print("\nSuccessfully converted: ", converted_model.name, converted_model.url, converted_model.version,
      converted_model.id, converted_model.created_time, '\n')
```

Containerize and deploy the model

Next, create a Docker image from the converted model and all dependencies. This Docker image can then be deployed and instantiated. Supported deployment targets include Azure Kubernetes Service (AKS) in the cloud or an edge device such as [Azure Data Box Edge](#). You can also add tags and descriptions for your registered Docker image.

```
from azureml.core.image import Image
from azureml.accel import AccelContainerImage

image_config = AccelContainerImage.image_configuration()
# Image name must be lowercase
image_name = "{}-image".format(model_name)

image = Image.create(name=image_name,
                     models=[converted_model],
                     image_config=image_config,
                     workspace=ws)
image.wait_for_creation(show_output=False)
```

List the images by tag and get the detailed logs for any debugging.

```
for i in Image.list(workspace=ws):
    print('{}) v.{} [{}]) stored at {} with build log {}'.format(
        i.name, i.version, i.creation_state, i.image_location, i.image_build_log_uri))
```

Deploy to an Azure Kubernetes Service Cluster

1. To deploy your model as a high-scale production web service, use AKS. You can create a new one using the Azure Machine Learning SDK, CLI, or [Azure Machine Learning studio](#).

```
from azureml.core.compute import AksCompute, ComputeTarget

# Specify the Standard_PB6s Azure VM and location. Values for location may be "eastus",
# "southeastasia", "westeurope", or "westus2". If no value is specified, the default is "eastus".
prov_config = AksCompute.provisioning_configuration(vm_size = "Standard_PB6s",
                                                       agent_count = 1,
                                                       location = "eastus")

aks_name = 'my-aks-cluster'
# Create the cluster
aks_target = ComputeTarget.create(workspace=ws,
                                    name=aks_name,
                                    provisioning_configuration=prov_config)
```

The AKS deployment may take around 15 minutes. Check to see if the deployment succeeded.

```
aks_target.wait_for_completion(show_output=True)
print(aks_target.provisioning_state)
print(aks_target.provisioning_errors)
```

2. Deploy the container to the AKS cluster.

```
from azureml.core.webservice import Webservice, AksWebservice

# For this deployment, set the web service configuration without enabling auto-scaling or
# authentication for testing
aks_config = AksWebservice.deploy_configuration(autoscale_enabled=False,
                                                 num_replicas=1,
                                                 auth_enabled=False)

aks_service_name = 'my-aks-service'

aks_service = Webservice.deploy_from_image(workspace=ws,
                                            name=aks_service_name,
                                            image=image,
                                            deployment_config=aks_config,
                                            deployment_target=aks_target)
aks_service.wait_for_deployment(show_output=True)
```

Deploy to a local edge server

All [Azure Data Box Edge devices](#) contain an FPGA for running the model. Only one model can be running on the FPGA at one time. To run a different model, just deploy a new container. Instructions and sample code can be found in [this Azure Sample](#).

Consume the deployed model

Lastly, use the sample client to call into the Docker image to get predictions from the model. Sample client code is available:

- [Python](#)
- [C#](#)

The Docker image supports gRPC and the TensorFlow Serving "predict" API.

You can also download a sample client for TensorFlow Serving.

```
# Using the grpc client in Azure ML Accelerated Models SDK package
from azureml.accel import PredictionClient

address = aks_service.scoring_uri
ssl_enabled = address.startswith("https")
address = address[address.find('/')+2:].strip('/')
port = 443 if ssl_enabled else 80

# Initialize Azure ML Accelerated Models client
client = PredictionClient(address=address,
                           port=port,
                           use_ssl=ssl_enabled,
                           service_name=aks_service.name)
```

Since this classifier was trained on the [ImageNet](#) data set, map the classes to human-readable labels.

```
import requests
classes_entries = requests.get(
    "https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.txt").text.splitlines()

# Score image with input and output tensor names
results = client.score_file(path="./snowleopardgaze.jpg",
                             input_name=input_tensors,
                             outputs=output_tensors)

# map results [class_id] => [confidence]
results = enumerate(results)
# sort results by confidence
sorted_results = sorted(results, key=lambda x: x[1], reverse=True)
# print top 5 results
for top in sorted_results[:5]:
    print(classes_entries[top[0]], 'confidence:', top[1])
```

Clean up resources

To avoid unnecessary costs, clean up your resources **in this order**: web service, then image, and then the model.

```
aks_service.delete()
aks_target.delete()
image.delete()
registered_model.delete()
converted_model.delete()
```

Next steps

- Learn how to [secure your web services](#) document.
- Learn about FPGA and [Azure Machine Learning pricing and costs](#).
- [Hyperscale hardware: ML at scale on top of Azure + FPGA: Build 2018 \(video\)](#)
- [Microsoft FPGA-based configurable cloud \(video\)](#)

- Project Brainwave for real-time AI
- Automated optical inspection system

Troubleshooting remote model deployment

12/23/2020 • 7 minutes to read • [Edit Online](#)

Learn how to troubleshoot and solve, or work around, common errors you may encounter when deploying a model to Azure Container Instances (ACI) and Azure Kubernetes Service (AKS) using Azure Machine Learning.

Prerequisites

- An [Azure subscription](#). Try the [free or paid version of Azure Machine Learning](#).
- The [Azure Machine Learning SDK](#).
- The [Azure CLI](#).
- The [CLI extension for Azure Machine Learning](#).

Steps for Docker deployment of machine learning models

When you deploy a model to non-local compute in Azure Machine Learning, the following things happen:

1. The Dockerfile you specified in your Environments object in your InferenceConfig is sent to the cloud, along with the contents of your source directory
2. If a previously built image is not available in your container registry, a new Docker image is built in the cloud and stored in your workspace's default container registry.
3. The Docker image from your container registry is downloaded to your compute target.
4. Your workspace's default Blob store is mounted to your compute target, giving you access to registered models
5. Your web server is initialized by running your entry script's `init()` function
6. When your deployed model receives a request, your `run()` function handles that request

The main difference when using a local deployment is that the container image is built on your local machine, which is why you need to have Docker installed for a local deployment.

Understanding these high-level steps should help you understand where errors are happening.

Get deployment logs

The first step in debugging errors is to get your deployment logs. First, follow the [instructions here](#) to connect to your workspace.

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

To get the logs from a deployed webservice, do:

```
az ml service get-logs --verbose --workspace-name <my workspace name> --name <service name>
```

Debug locally

If you have problems when deploying a model to ACI or AKS, deploy it as a local web service. Using a local web service makes it easier to troubleshoot problems. To troubleshoot a deployment locally, see the [local troubleshooting article](#).

Container cannot be scheduled

When deploying a service to an Azure Kubernetes Service compute target, Azure Machine Learning will attempt to schedule the service with the requested amount of resources. If there are no nodes available in the cluster with the appropriate amount of resources after 5 minutes, the deployment will fail. The failure message is

```
Couldn't Schedule because the kubernetes cluster didn't have available resources after trying for 00:05:00.
```

You can address this error by either adding more nodes, changing the SKU of your nodes, or changing the resource requirements of your service.

The error message will typically indicate which resource you need more of - for instance, if you see an error message indicating `0/3 nodes are available: 3 Insufficient nvidia.com/gpu` that means that the service requires GPUs and there are three nodes in the cluster that do not have available GPUs. This could be addressed by adding more nodes if you are using a GPU SKU, switching to a GPU enabled SKU if you are not or changing your environment to not require GPUs.

Service launch fails

After the image is successfully built, the system attempts to start a container using your deployment configuration. As part of container starting-up process, the `init()` function in your scoring script is invoked by the system. If there are uncaught exceptions in the `init()` function, you might see **CrashLoopBackOff** error in the error message.

Use the info in the [Inspect the Docker log](#) article.

Function fails: get_model_path()

Often, in the `init()` function in the scoring script, `Model.get_model_path()` function is called to locate a model file or a folder of model files in the container. If the model file or folder cannot be found, the function fails. The easiest way to debug this error is to run the below Python code in the Container shell:

```
from azureml.core.model import Model
import logging
logging.basicConfig(level=logging.DEBUG)
print(Model.get_model_path(model_name='my-best-model'))
```

This example prints out the local path (relative to `/var/azureml-app`) in the container where your scoring script is expecting to find the model file or folder. Then you can verify if the file or folder is indeed where it is expected to be.

Setting the logging level to DEBUG may cause additional information to be logged, which may be useful in identifying the failure.

Function fails: run(input_data)

If the service is successfully deployed, but it crashes when you post data to the scoring endpoint, you can add error catching statement in your `run(input_data)` function so that it returns detailed error message instead. For example:

```

def run(input_data):
    try:
        data = json.loads(input_data)[ 'data' ]
        data = np.array(data)
        result = model.predict(data)
        return json.dumps({ "result": result.tolist() })
    except Exception as e:
        result = str(e)
        # return error message back to the client
        return json.dumps({ "error": result })

```

Note: Returning error messages from the `run(input_data)` call should be done for debugging purpose only. For security reasons, you should not return error messages this way in a production environment.

HTTP status code 502

A 502 status code indicates that the service has thrown an exception or crashed in the `run()` method of the `score.py` file. Use the information in this article to debug the file.

HTTP status code 503

Azure Kubernetes Service deployments support autoscaling, which allows replicas to be added to support additional load. The autoscaler is designed to handle **gradual** changes in load. If you receive large spikes in requests per second, clients may receive an HTTP status code 503. Even though the autoscaler reacts quickly, it takes AKS a significant amount of time to create additional containers.

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 `autoscale_target_utilization`, then more replicas are created. If it is lower, then replicas are reduced. Decisions to add replicas are eager and fast (around 1 second). Decisions to remove replicas are conservative (around 1 minute). By default, autoscaling target utilization is set to 70%, which means that the service can handle spikes in requests per second (RPS) of **up to 30%**.

There are two things that can help prevent 503 status codes:

TIP

These two approaches can be used individually or in combination.

- Change the utilization level at which autoscaling creates new replicas. You can adjust the utilization target by setting the `autoscale_target_utilization` to a lower value.

IMPORTANT

This change does not cause replicas to be created *faster*. Instead, they are created at a lower utilization threshold. Instead of waiting until the service is 70% utilized, changing the value to 30% causes replicas to be created when 30% utilization occurs.

If the web service is already using the current max replicas and you are still seeing 503 status codes, increase the `autoscale_max_replicas` value to increase the maximum number of replicas.

- Change the minimum number of replicas. Increasing the minimum replicas provides a larger pool to handle the incoming spikes.

To increase the minimum number of replicas, set `autoscale_min_replicas` to a higher value. You can

calculate the required replicas by using the following code, replacing values with values specific to your project:

```
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)
```

NOTE

If you receive request spikes larger than the new minimum replicas can handle, you may receive 503s again. For example, as traffic to your service increases, you may need to increase the minimum replicas.

For more information on setting `autoscale_target_utilization`, `autoscale_max_replicas`, and `autoscale_min_replicas` for, see the [AksWebservice](#) module reference.

HTTP status code 504

A 504 status code indicates that the request has timed out. The default timeout is 1 minute.

You can increase the timeout or try to speed up the service by modifying the score.py to remove unnecessary calls. If these actions do not correct the problem, use the information in this article to debug the score.py file. The code may be in a non-responsive state or an infinite loop.

Other error messages

Take these actions for the following errors:

ERROR	RESOLUTION
Image building failure when deploying web service	Add "pynacl==1.2.1" as a pip dependency to Conda file for image configuration
<code>['DaskOnBatch:context_managers.DaskOnBatch', 'setup.py']' died with <Signals.SIGKILL: 9></code>	Change the SKU for VMs used in your deployment to one that has more memory.
FPGA failure	You will not be able to deploy models on FPGAs until you have requested and been approved for FPGA quota. To request access, fill out the quota request form: https://aka.ms/aml-real-time-ai

Advanced debugging

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 cannot be determined by additional logging. By using Visual Studio Code and the debugpy, you can attach to the code running inside the Docker container.

For more information, visit the [interactive debugging in VS Code guide](#).

Model deployment user forum

Next steps

Learn more about deployment:

- [How to deploy and where](#)
- [Tutorial: Train & deploy models](#)
- [How to run and debug experiments locally](#)

Troubleshooting with a local model deployment

12/23/2020 • 3 minutes to read • [Edit Online](#)

Try a local model deployment as a first step in troubleshooting deployment to Azure Container Instances (ACI) or Azure Kubernetes Service (AKS). Using a local web service makes it easier to spot and fix common Azure Machine Learning Docker web service deployment errors.

Prerequisites

- An [Azure subscription](#). Try the [free or paid version of Azure Machine Learning](#).
- The [Azure Machine Learning SDK](#).
- The [Azure CLI](#).
- The [CLI extension for Azure Machine Learning](#).
- To debug locally, you can deploy model to [Azure Machine Learning Compute Instance](#) or have a working Docker installation on your local system.

To verify your Docker installation, use the command `docker run hello-world` from a terminal or command prompt. For information on installing Docker, or troubleshooting Docker errors, see the [Docker Documentation](#).

Debug locally

You can find a sample [local deployment notebook](#) in the [MachineLearningNotebooks](#) repo to explore a runnable example.

WARNING

Local web service deployments are not supported for production scenarios.

To deploy locally, modify your code to use `LocalWebservice.deploy_configuration()` to create a deployment configuration. Then use `Model.deploy()` to deploy the service. The following example deploys a model (contained in the `model` variable) as a local web service:

```
from azureml.core.environment import Environment
from azureml.core.model import InferenceConfig, Model
from azureml.core.webservice import LocalWebservice

# Create inference configuration based on the environment definition and the entry script
myenv = Environment.from_conda_specification(name="env", file_path="myenv.yml")
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)
# Create a local deployment, using port 8890 for the web service endpoint
deployment_config = LocalWebservice.deploy_configuration(port=8890)
# Deploy the service
service = Model.deploy(
    ws, "mymodel", [model], inference_config, deployment_config)
# Wait for the deployment to complete
service.wait_for_deployment(True)
# Display the port that the web service is available on
print(service.port)
```

If you are defining your own conda specification YAML, list `azureml-defaults` version $\geq 1.0.45$ as a pip dependency. This package is needed to host the model as a web service.

At this point, you can work with the service as normal. The following code demonstrates sending data to the service:

```
import json

test_sample = json.dumps({'data': [
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
]})

test_sample = bytes(test_sample, encoding='utf8')

prediction = service.run(input_data=test_sample)
print(prediction)
```

For more information on customizing your Python environment, see [Create and manage environments for training and deployment](#).

Update the service

During local testing, you may need to update the `score.py` file to add logging or attempt to resolve any problems that you've discovered. To reload changes to the `score.py` file, use `reload()`. For example, the following code reloads the script for the service, and then sends data to it. The data is scored using the updated `score.py` file:

IMPORTANT

The `reload` method is only available for local deployments. For information on updating a deployment to another compute target, see [how to update your webservice](#).

```
service.reload()
print(service.run(input_data=test_sample))
```

NOTE

The script is reloaded from the location specified by the `InferenceConfig` object used by the service.

To change the model, Conda dependencies, or deployment configuration, use `update()`. The following example updates the model used by the service:

```
service.update([different_model], inference_config, deployment_config)
```

Delete the service

To delete the service, use `delete()`.

Inspect the Docker log

You can print out detailed Docker engine log messages from the service object. You can view the log for ACI, AKS, and Local deployments. The following example demonstrates how to print the logs.

```
# if you already have the service object handy
print(service.get_logs())

# if you only know the name of the service (note there might be multiple services with the same name but
different version number)
print(ws.webservices['mysvc'].get_logs())
```

If you see the line `Booting worker with pid: <pid>` occurring multiple times in the logs, it means, there isn't enough memory to start the worker. You can address the error by increasing the value of `memory_gb` in `deployment_config`

Next steps

Learn more about deployment:

- [How to troubleshoot remote deployments](#)
- [How to deploy and where](#)
- [Tutorial: Train & deploy models](#)
- [How to run and debug experiments locally](#)

Consume an Azure Machine Learning model deployed as a web service

12/23/2020 • 12 minutes to read • [Edit Online](#)

Deploying an Azure Machine Learning model as a web service creates a REST API endpoint. You can send data to this endpoint and receive the prediction returned by the model. In this document, learn how to create clients for the web service by using C#, Go, Java, and Python.

You create a web service when you deploy a model to your local environment, Azure Container Instances, Azure Kubernetes Service, or field-programmable gate arrays (FPGA). You retrieve the URI used to access the web service by using the [Azure Machine Learning SDK](#). If authentication is enabled, you can also use the SDK to get the authentication keys or tokens.

The general workflow for creating a client that uses a machine learning web service is:

1. Use the SDK to get the connection information.
2. Determine the type of request data used by the model.
3. Create an application that calls the web service.

TIP

The examples in this document are manually created without the use of OpenAPI (Swagger) specifications. If you've enabled an OpenAPI specification for your deployment, you can use tools such as [swagger-codegen](#) to create client libraries for your service.

Connection information

NOTE

Use the Azure Machine Learning SDK to get the web service information. This is a Python SDK. You can use any language to create a client for the service.

The [azureml.core.Wbservice](#) class provides the information you need to create a client. The following `Wbservice` properties are useful for creating a client application:

- `auth_enabled` - If key authentication is enabled, `True`; otherwise, `False`.
- `token_auth_enabled` - If token authentication is enabled, `True`; otherwise, `False`.
- `scoring_uri` - The REST API address.
- `swagger_uri` - The address of the OpenAPI specification. This URI is available if you enabled automatic schema generation. For more information, see [Deploy models with Azure Machine Learning](#).

There are several ways to retrieve this information for deployed web services:

- [Python](#)
- [Azure CLI](#)
- [Portal](#)
- When you deploy a model, a `Wbservice` object is returned with information about the service:

```
service = Model.deploy(ws, "myservice", [model], inference_config, deployment_config)
service.wait_for_deployment(show_output = True)
print(service.scoring_uri)
print(service.swagger_uri)
```

- You can use `Webservice.list` to retrieve a list of deployed web services for models in your workspace. You can add filters to narrow the list of information returned. For more information about what can be filtered on, see the [Webservice.list](#) reference documentation.

```
services = Webservice.list(ws)
print(services[0].scoring_uri)
print(services[0].swagger_uri)
```

- If you know the name of the deployed service, you can create a new instance of `Webservice`, and provide the workspace and service name as parameters. The new object contains information about the deployed service.

```
service = Webservice(workspace=ws, name='myservice')
print(service.scoring_uri)
print(service.swagger_uri)
```

The following table shows what these URLs look like:

URI TYPE	EXAMPLE
Scoring URI	<code>http://104.214.29.152:80/api/v1/service/<service-name>/score</code>
Swagger URI	<code>http://104.214.29.152/api/v1/service/<service-name>/swagger.json</code>

TIP

The IP address will be different for your deployment. Each AKS cluster will have its own IP address that is shared by deployments to that cluster.

Secured web service

If you secured the deployed web service using a TLS/SSL certificate, you can use [HTTPS](#) to connect to the service using the scoring or swagger URI. HTTPS helps secure communications between a client and a web service by encrypting communications between the two. Encryption uses [Transport Layer Security \(TLS\)](#). TLS is sometimes still referred to as *Secure Sockets Layer* (SSL), which was the predecessor of TLS.

IMPORTANT

Web services deployed by Azure Machine Learning only support TLS version 1.2. When creating a client application, make sure that it supports this version.

For more information, see [Use TLS to secure a web service through Azure Machine Learning](#).

Authentication for services

Azure Machine Learning provides two ways to control access to your web services.

AUTHENTICATION METHOD	ACI	AKS
Key	Disabled by default	Enabled by default
Token	Not Available	Disabled by default

When sending a request to a service that is secured with a key or token, use the **Authorization** header to pass the key or token. The key or token must be formatted as `Bearer <key-or-token>`, where `<key-or-token>` is your key or token value.

The primary difference between keys and tokens is that **keys are static and can be regenerated manually**, and **tokens need to be refreshed upon expiration**. Key-based auth is supported for Azure Container Instance and Azure Kubernetes Service deployed web-services, and token-based auth is **only** available for Azure Kubernetes Service deployments. For more information on configuring authentication, see [Configure authentication for models deployed as web services](#).

Authentication with keys

When you enable authentication for a deployment, you automatically create authentication keys.

- Authentication is enabled by default when you are deploying to Azure Kubernetes Service.
- Authentication is disabled by default when you are deploying to Azure Container Instances.

To control authentication, use the `auth_enabled` parameter when you are creating or updating a deployment.

If authentication is enabled, you can use the `get_keys` method to retrieve a primary and secondary authentication key:

```
primary, secondary = service.get_keys()
print(primary)
```

IMPORTANT

If you need to regenerate a key, use `service.regen_key`.

Authentication with tokens

When you enable token authentication for a web service, a user must provide an Azure Machine Learning JWT token to the web service to access it.

- Token authentication is disabled by default when you are deploying to Azure Kubernetes Service.
- Token authentication is not supported when you are deploying to Azure Container Instances.

To control token authentication, use the `token_auth_enabled` parameter when you are creating or updating a deployment.

If token authentication is enabled, you can use the `get_token` method to retrieve a bearer token and that tokens expiration time:

```
token, refresh_by = service.get_token()
print(token)
```

If you have the [Azure CLI and the machine learning extension](#), you can use the following command to get a token:

```
az ml service get-access-token -n <service-name>
```

IMPORTANT

Currently the only way to retrieve the token is by using the Azure Machine Learning SDK or the Azure CLI machine learning extension.

You will need to request a new token after the token's `refresh_by` time.

Request data

The REST API expects the body of the request to be a JSON document with the following structure:

```
{
    "data":
        [
            <model-specific-data-structure>
        ]
}
```

IMPORTANT

The structure of the data needs to match what the scoring script and model in the service expect. The scoring script might modify the data before passing it to the model.

Binary data

For information on how to enable support for binary data in your service, see [Binary data](#).

TIP

Enabling support for binary data happens in the score.py file used by the deployed model. From the client, use the HTTP functionality of your programming language. For example, the following snippet sends the contents of a JPG file to a web service:

```
import requests
# Load image data
data = open('example.jpg', 'rb').read()
# Post raw data to scoring URI
res = request.post(url='<scoring-uri>', data=data, headers={'Content-Type': 'application/octet-stream'})
```

Cross-origin resource sharing (CORS)

For information on enabling CORS support in your service, see [Cross-origin resource sharing](#).

Call the service (C#)

This example demonstrates how to use C# to call the web service created from the [Train within notebook](#) example:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
```

```

using System.Net.Http.Headers;
using Newtonsoft.Json;

namespace MLWebServiceClient
{
    // The data structure expected by the service
    internal class InputData
    {
        [JsonProperty("data")]
        // The service used by this example expects an array containing
        // one or more arrays of doubles
        internal double[,] data;
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Set the scoring URI and authentication key or token
            string scoringUri = "<your web service URI>";
            string authKey = "<your key or token>";

            // Set the data to be sent to the service.
            // In this case, we are sending two sets of data to be scored.
            InputData payload = new InputData();
            payload.data = new double[,] {
                {
                    0.0199132141783263,
                    0.0506801187398187,
                    0.104808689473925,
                    0.0700725447072635,
                    -0.0359677812752396,
                    -0.0266789028311707,
                    -0.0249926566315915,
                    -0.00259226199818282,
                    0.00371173823343597,
                    0.0403433716478807
                },
                {
                    -0.0127796318808497,
                    -0.044641636506989,
                    0.0606183944448076,
                    0.0528581912385822,
                    0.0479653430750293,
                    0.0293746718291555,
                    -0.0176293810234174,
                    0.0343088588777263,
                    0.0702112981933102,
                    0.00720651632920303
                }
            };
        }

        // Create the HTTP client
        HttpClient client = new HttpClient();
        // Set the auth header. Only needed if the web service requires authentication.
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
        authKey);

        // Make the request
        try {
            var request = new HttpRequestMessage(HttpMethod.Post, new Uri(scoringUri));
            request.Content = new StringContent(JsonConvert.SerializeObject(payload));
            request.Content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
            var response = client.SendAsync(request).Result;
            // Display the response from the web service
            Console.WriteLine(response.Content.ReadAsStringAsync().Result);
        }
        catch (Exception e)
        {
            Console.Out.WriteLine(e.Message);
        }
    }
}

```

```
        }
    }
}
```

The results returned are similar to the following JSON document:

```
[217.67978776218715, 224.78937091757172]
```

Call the service (Go)

This example demonstrates how to use Go to call the web service created from the [Train within notebook](#) example:

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "net/http"
)

// Features for this model are an array of decimal values
type Features []float64

// The web service input can accept multiple sets of values for scoring
type InputData struct {
    Data []Features `json:"data",omitempty`
}

// Define some example data
var exampleData = []Features{
    []float64{
        0.0199132141783263,
        0.0506801187398187,
        0.104808689473925,
        0.0700725447072635,
        -0.0359677812752396,
        -0.0266789028311707,
        -0.0249926566315915,
        -0.00259226199818282,
        0.00371173823343597,
        0.0403433716478807,
    },
    []float64{
        -0.0127796318808497,
        -0.044641636506989,
        0.0606183944448076,
        0.0528581912385822,
        0.0479653430750293,
        0.0293746718291555,
        -0.0176293810234174,
        0.0343088588777263,
        0.0702112981933102,
        0.00720651632920303,
    },
}

// Set to the URI for your service
var serviceUri string = "<your web service URI>"
// Set to the authentication key or token (if any) for your service
var authKey string = "<your key or token>"
```

```

func main() {
    // Create the input data from example data
    jsonData := InputData{
        Data: exampleData,
    }
    // Create JSON from it and create the body for the HTTP request
    jsonValue, _ := json.Marshal(jsonData)
    body := bytes.NewBuffer(jsonValue)

    // Create the HTTP request
    client := &http.Client{}
    request, err := http.NewRequest("POST", serviceUri, body)
    request.Header.Add("Content-Type", "application/json")

    // These next two are only needed if using an authentication key
    bearer := fmt.Sprintf("Bearer %v", authKey)
    request.Header.Add("Authorization", bearer)

    // Send the request to the web service
    resp, err := client.Do(request)
    if err != nil {
        fmt.Println("Failure: ", err)
    }

    // Display the response received
    respBody, _ := ioutil.ReadAll(resp.Body)
    fmt.Println(string(respBody))
}

```

The results returned are similar to the following JSON document:

```
[217.67978776218715, 224.78937091757172]
```

Call the service (Java)

This example demonstrates how to use Java to call the web service created from the [Train within notebook](#) example:

```

import java.io.IOException;
import org.apache.http.client.fluent.*;
import org.apache.http.entity.ContentType;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

public class App {
    // Handle making the request
    public static void sendRequest(String data) {
        // Replace with the scoring_uri of your service
        String uri = "<your web service URI>";
        // If using authentication, replace with the auth key or token
        String key = "<your key or token>";
        try {
            // Create the request
            Content content = Request.Post(uri)
                .addHeader("Content-Type", "application/json")
                // Only needed if using authentication
                .addHeader("Authorization", "Bearer " + key)
                // Set the JSON data as the body
                .bodyString(data, ContentType.APPLICATION_JSON)
                // Make the request and display the response.
                .execute().returnContent();
            System.out.println(content);
        }
    }
}

```

```

        }
        catch (IOException e) {
            System.out.println(e);
        }
    }

    public static void main(String[] args) {
        // Create the data to send to the service
        JSONObject obj = new JSONObject();
        // In this case, it's an array of arrays
        JSONArray dataItems = new JSONArray();
        // Inner array has 10 elements
        JSONArray item1 = new JSONArray();
        item1.add(0.0199132141783263);
        item1.add(0.0506801187398187);
        item1.add(0.104808689473925);
        item1.add(0.0700725447072635);
        item1.add(-0.0359677812752396);
        item1.add(-0.0266789028311707);
        item1.add(-0.0249926566315915);
        item1.add(-0.00259226199818282);
        item1.add(0.00371173823343597);
        item1.add(0.0403433716478807);
        // Add the first set of data to be scored
        dataItems.add(item1);
        // Create and add the second set
        JSONArray item2 = new JSONArray();
        item2.add(-0.0127796318808497);
        item2.add(-0.044641636506989);
        item2.add(0.0606183944448076);
        item2.add(0.0528581912385822);
        item2.add(0.0479653430750293);
        item2.add(0.0293746718291555);
        item2.add(-0.0176293810234174);
        item2.add(0.0343088588777263);
        item2.add(0.0702112981933102);
        item2.add(0.00720651632920303);
        dataItems.add(item2);
        obj.put("data", dataItems);

        // Make the request using the JSON document string
        sendRequest(obj.toJSONString());
    }
}

```

The results returned are similar to the following JSON document:

```
[217.67978776218715, 224.78937091757172]
```

Call the service (Python)

This example demonstrates how to use Python to call the web service created from the [Train within notebook](#) example:

```

import requests
import json

# URL for the web service
scoring_uri = '<your web service URI>'
# If the service is authenticated, set the key or token
key = '<your key or token>'

# Two sets of data to score, so we get two results back
data = {"data":
    [
        [
            0.0199132141783263,
            0.0506801187398187,
            0.104808689473925,
            0.0700725447072635,
            -0.0359677812752396,
            -0.0266789028311707,
            -0.0249926566315915,
            -0.00259226199818282,
            0.00371173823343597,
            0.0403433716478807
        ],
        [
            -0.0127796318808497,
            -0.044641636506989,
            0.0606183944448076,
            0.0528581912385822,
            0.0479653430750293,
            0.0293746718291555,
            -0.0176293810234174,
            0.0343088588777263,
            0.0702112981933102,
            0.00720651632920303
        ]
    ]
}

# Convert to JSON string
input_data = json.dumps(data)

# Set the content type
headers = {'Content-Type': 'application/json'}
# If authentication is enabled, set the authorization header
headers['Authorization'] = f'Bearer {key}'

# Make the request and display the response
resp = requests.post(scoring_uri, input_data, headers=headers)
print(resp.text)

```

The results returned are similar to the following JSON document:

```
[217.67978776218715, 224.78937091757172]
```

Web service schema (OpenAPI specification)

If you used automatic schema generation with your deployment, you can get the address of the OpenAPI specification for the service by using the [swagger_uri property](#). (For example, `print(service.swagger_uri)`.) Use a GET request or open the URI in a browser to retrieve the specification.

The following JSON document is an example of a schema (OpenAPI specification) generated for a deployment:

```
{
```

```

"swagger": "2.0",
"info": {
    "title": "myservice",
    "description": "API specification for Azure Machine Learning myservice",
    "version": "1.0"
},
"schemes": [
    "https"
],
"consumes": [
    "application/json"
],
"produces": [
    "application/json"
],
"securityDefinitions": {
    "Bearer": {
        "type": "apiKey",
        "name": "Authorization",
        "in": "header",
        "description": "For example: Bearer abc123"
    }
},
"paths": {
    "/": {
        "get": {
            "operationId": "ServiceHealthCheck",
            "description": "Simple health check endpoint to ensure the service is up at any given point.",
            "responses": {
                "200": {
                    "description": "If service is up and running, this response will be returned with the content 'Healthy'",
                    "schema": {
                        "type": "string"
                    },
                    "examples": {
                        "application/json": "Healthy"
                    }
                },
                "default": {
                    "description": "The service failed to execute due to an error.",
                    "schema": {
                        "$ref": "#/definitions/ErrorResponse"
                    }
                }
            }
        }
    },
    "/score": {
        "post": {
            "operationId": "RunMLService",
            "description": "Run web service's model and get the prediction output",
            "security": [
                {
                    "Bearer": []
                }
            ],
            "parameters": [
                {
                    "name": "serviceInputPayload",
                    "in": "body",
                    "description": "The input payload for executing the real-time machine learning service.",
                    "schema": {
                        "$ref": "#/definitions/ServiceInput"
                    }
                }
            ]
        }
    }
}

```

```

    "responses": {
        "200": {
            "description": "The service processed the input correctly and provided a result prediction, if applicable.",
            "schema": {
                "$ref": "#/definitions/ServiceOutput"
            }
        },
        "default": {
            "description": "The service failed to execute due to an error.",
            "schema": {
                "$ref": "#/definitions/ErrorResponse"
            }
        }
    }
},
"definitions": {
    "ServiceInput": {
        "type": "object",
        "properties": {
            "data": {
                "type": "array",
                "items": {
                    "type": "array",
                    "items": {
                        "type": "integer",
                        "format": "int64"
                    }
                }
            }
        }
    },
    "example": {
        "data": [
            [ 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 ]
        ]
    }
},
"ServiceOutput": {
    "type": "array",
    "items": {
        "type": "number",
        "format": "double"
    }
},
"example": [
    3726.995
],
},
"ErrorResponse": {
    "type": "object",
    "properties": {
        "status_code": {
            "type": "integer",
            "format": "int32"
        },
        "message": {
            "type": "string"
        }
    }
}
}

```

For more information, see [OpenAPI specification](#).

For a utility that can create client libraries from the specification, see [swagger-codegen](#).

TIP

You can retrieve the schema JSON document after you deploy the service. Use the [swagger_uri](#) property from the deployed web service (for example, `service.swagger_uri`) to get the URI to the local web service's Swagger file.

Consume the service from Power BI

Power BI supports consumption of Azure Machine Learning web services to enrich the data in Power BI with predictions.

To generate a web service that's supported for consumption in Power BI, the schema must support the format that's required by Power BI. [Learn how to create a Power BI-supported schema](#).

Once the web service is deployed, it's consumable from Power BI dataflows. [Learn how to consume an Azure Machine Learning web service from Power BI](#).

Next steps

To view a reference architecture for real-time scoring of Python and deep learning models, go to the [Azure architecture center](#).

Update a deployed web service

12/23/2020 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to update a web service that was deployed with Azure Machine Learning.

Prerequisites

This tutorial assumes you have already deployed a web service with Azure Machine Learning. If you need to learn how to deploy a web service, [follow these steps](#).

Update web service

To update a web service, use the `update` method. You can update the web service to use a new model, a new entry script, or new dependencies that can be specified in an inference configuration. For more information, see the documentation for [Webservice.update](#).

See [AKS Service Update Method](#).

See [ACI Service Update Method](#).

IMPORTANT

When you create a new version of a model, you must manually update each service that you want to use it.

You can not use the SDK to update a web service published from the Azure Machine Learning designer.

Using the SDK

The following code shows how to use the SDK to update the model, environment, and entry script for a web service:

```

from azureml.core import Environment
from azureml.core.webservice import Webservice
from azureml.core.model import Model, InferenceConfig

# Register new model.
new_model = Model.register(model_path="outputs/sklearn_mnist_model.pkl",
                           model_name="sklearn_mnist",
                           tags={"key": "0.1"},
                           description="test",
                           workspace=ws)

# Use version 3 of the environment.
deploy_env = Environment.get(workspace=ws, name="myenv", version="3")
inference_config = InferenceConfig(entry_script="score.py",
                                    environment=deploy_env)

service_name = 'myservice'
# Retrieve existing service.
service = Webservice(name=service_name, workspace=ws)

# Update to new model(s).
service.update(models=[new_model], inference_config=inference_config)
service.wait_for_deployment(show_output=True)
print(service.state)
print(service.get_logs())

```

Using the CLI

You can also update a web service by using the ML CLI. The following example demonstrates registering a new model and then updating a web service to use the new model:

```

az ml model register -n sklearn_mnist --asset-path outputs/sklearn_mnist_model.pkl --experiment-name
myexperiment --output-metadata-file modelinfo.json
az ml service update -n myservice --model-metadata-file modelinfo.json

```

TIP

In this example, a JSON document is used to pass the model information from the registration command into the update command.

To update the service to use a new entry script or environment, create an [inference configuration file](#) and specify it with the `ic` parameter.

For more information, see the [az ml service update](#) documentation.

Next steps

- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Create client applications to consume web services](#)
- [How to deploy a model using a custom Docker image](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Create event alerts and triggers for model deployments](#)

Advanced entry script authoring

12/23/2020 • 8 minutes to read • [Edit Online](#)

This article shows how to write entry scripts for specialized use cases.

Prerequisites

This article assumes you already have a trained machine learning model that you intend to deploy with Azure Machine Learning. To learn more about model deployment, see [this tutorial](#).

Automatically generate a Swagger schema

To automatically generate a schema for your web service, provide a sample of the input and/or output in the constructor for one of the defined type objects. The type and sample are used to automatically create the schema. Azure Machine Learning then creates an [OpenAPI](#) (Swagger) specification for the web service during deployment.

WARNING

You must not use sensitive or private data for sample input or output. The Swagger page for AML-hosted inferencing exposes the sample data.

These types are currently supported:

- `pandas`
- `numpy`
- `pyspark`
- Standard Python object

To use schema generation, include the open-source `inference-schema` package version 1.1.0 or above in your dependencies file. For more information on this package, see <https://github.com/Azure/InferenceSchema>. In order to generate conforming swagger for automated web service consumption, scoring script `run()` function must have API shape of:

- A first parameter of type "StandardPythonParameterType", named **Inputs** and nested.
- An optional second parameter of type "StandardPythonParameterType", named **GlobalParameters**.
- Return a dictionary of type "StandardPythonParameterType" named **Results** and nested.

Define the input and output sample formats in the `input_sample` and `output_sample` variables, which represent the request and response formats for the web service. Use these samples in the input and output function decorators on the `run()` function. The following scikit-learn example uses schema generation.

Power BI compatible endpoint

The following example demonstrates how to define API shape according to above instruction. This method is supported for consuming the deployed web service from Power BI. ([Learn more about how to consume the web service from Power BI](#).)

```

import json
import pickle
import numpy as np
import pandas as pd
import azureml.train.automl
from sklearn.externals import joblib
from sklearn.linear_model import Ridge

from inference_schema.schema_decorators import input_schema, output_schema
from inference_schema.parameter_types.standard_py_parameter_type import StandardPythonParameterType
from inference_schema.parameter_types.numpy_parameter_type import NumpyParameterType
from inference_schema.parameter_types.pandas_parameter_type import PandasParameterType


def init():
    global model
    # Replace filename if needed.
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'sklearn_regression_model.pkl')
    # Deserialize the model file back into a sklearn model.
    model = joblib.load(model_path)

    # providing 3 sample inputs for schema generation
    numpy_sample_input = NumpyParameterType(np.array([[1,2,3,4,5,6,7,8,9,10],
    [10,9,8,7,6,5,4,3,2,1]],dtype='float64'))
    pandas_sample_input = PandasParameterType(pd.DataFrame({'name': ['Sarah', 'John'], 'age': [25, 26]}))
    standard_sample_input = StandardPythonParameterType(0.0)

    # This is a nested input sample, any item wrapped by `ParameterType` will be described by schema
    sample_input = StandardPythonParameterType({'input1': numpy_sample_input,
                                                'input2': pandas_sample_input,
                                                'input3': standard_sample_input})

    sample_global_parameters = StandardPythonParameterType(1.0) # this is optional
    sample_output = StandardPythonParameterType([1.0, 1.0])
    outputs = StandardPythonParameterType({'Results':sample_output}) # 'Results' is case sensitive

    @input_schema('Inputs', sample_input)
    # 'Inputs' is case sensitive

    @input_schema('GlobalParameters', sample_global_parameters)
    # this is optional, 'GlobalParameters' is case sensitive

    @output_schema(outputs)

def run(Inputs, GlobalParameters):
    # the parameters here have to match those in decorator, both 'Inputs' and
    # 'GlobalParameters' here are case sensitive
    try:
        data = Inputs['input1']
        # data will be convert to target format
        assert isinstance(data, np.ndarray)
        result = model.predict(data)
        return result.tolist()
    except Exception as e:
        error = str(e)
        return error

```

Binary (i.e. image) data

If your model accepts binary data, like an image, you must modify the `score.py` file used for your deployment to accept raw HTTP requests. To accept raw data, use the `AMLRequest` class in your entry script and add the `@rawhttp` decorator to the `run()` function.

Here's an example of a `score.py` that accepts binary data:

```
from azureml.contrib.services.aml_request import AMLRequest, rawhttp
from azureml.contrib.services.aml_response import AMLResponse
from PIL import Image
import json

def init():
    print("This is init()")

@rawhttp
def run(request):
    print("This is run()")

    if request.method == 'GET':
        # For this example, just return the URL for GETs.
        respBody = str.encode(request.full_path)
        return AMLResponse(respBody, 200)
    elif request.method == 'POST':
        file_bytes = request.files["image"]
        image = Image.open(file_bytes).convert('RGB')
        # For a real-world solution, you would load the data from reqBody
        # and send it to the model. Then return the response.

        # For demonstration purposes, this example just returns the size of the image as the response..
        return AMLResponse(json.dumps(image.size), 200)
    else:
        return AMLResponse("bad request", 500)
```

IMPORTANT

The `AMLRequest` class is in the `azureml.contrib` namespace. Entities in this namespace change frequently as we work to improve the service. Anything in this namespace should be considered a preview that's not fully supported by Microsoft.

If you need to test this in your local development environment, you can install the components by using the following command:

```
pip install azureml-contrib-services
```

The `AMLRequest` class only allows you to access the raw posted data in the `score.py`, there is no client-side component. From a client, you post data as normal. For example, the following Python code reads an image file and posts the data:

```
import requests

uri = service.scoring_uri
image_path = 'test.jpg'
files = {'image': open(image_path, 'rb').read()}
response = requests.post(url, files=files)

print(response.json)
```

Cross-origin resource sharing (CORS)

Cross-origin resource sharing is a way to allow resources on a webpage to be requested from another domain. CORS works via HTTP headers sent with the client request and returned with the service response. For more

information on CORS and valid headers, see [Cross-origin resource sharing](#) in Wikipedia.

To configure your model deployment to support CORS, use the `AMLResponse` class in your entry script. This class allows you to set the headers on the response object.

The following example sets the `Access-Control-Allow-Origin` header for the response from the entry script:

```
from azureml.contrib.services.aml_request import AMLRequest, rawhttp
from azureml.contrib.services.aml_response import AMLResponse

def init():
    print("This is init()")

@rawhttp
def run(request):
    print("This is run()")
    print("Request: [{0}]".format(request))
    if request.method == 'GET':
        # For this example, just return the URL for GETs.
        respBody = str.encode(request.full_path)
        return AMLResponse(respBody, 200)
    elif request.method == 'POST':
        reqBody = request.get_data(False)
        # For a real-world solution, you would load the data from reqBody
        # and send it to the model. Then return the response.

        # For demonstration purposes, this example
        # adds a header and returns the request body.
        resp = AMLResponse(reqBody, 200)
        resp.headers['Access-Control-Allow-Origin'] = "http://www.example.com"
        return resp
    else:
        return AMLResponse("bad request", 500)
```

IMPORTANT

The `AMLResponse` class is in the `azureml.contrib` namespace. Entities in this namespace change frequently as we work to improve the service. Anything in this namespace should be considered a preview that's not fully supported by Microsoft.

If you need to test this in your local development environment, you can install the components by using the following command:

```
pip install azureml-contrib-services
```

WARNING

Azure Machine Learning will route only POST and GET requests to the containers running the scoring service. This can cause errors due to browsers using OPTIONS requests to pre-flight CORS requests.

Load registered models

There are two ways to locate models in your entry script:

- `AZUREML_MODEL_DIR` : An environment variable containing the path to the model location.
- `Model.get_model_path` : An API that returns the path to model file using the registered model name.

AZUREML_MODEL_DIR

AZUREML_MODEL_DIR is an environment variable created during service deployment. You can use this environment variable to find the location of the deployed model(s).

The following table describes the value of AZUREML_MODEL_DIR depending on the number of models deployed:

DEPLOYMENT	ENVIRONMENT VARIABLE VALUE
Single model	The path to the folder containing the model.
Multiple models	The path to the folder containing all models. Models are located by name and version in this folder (\$MODEL_NAME/\$VERSION)

During model registration and deployment, Models are placed in the AZUREML_MODEL_DIR path, and their original filenames are preserved.

To get the path to a model file in your entry script, combine the environment variable with the file path you're looking for.

Single model example

```
# Example when the model is a file
model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'sklearn_regression_model.pkl')

# Example when the model is a folder containing a file
file_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'my_model_folder', 'sklearn_regression_model.pkl')
```

Multiple model example

In this scenario, two models are registered with the workspace:

- `my_first_model` : Contains one file (`my_first_model.pkl`) and there is only one version (`1`).
- `my_second_model` : Contains one file (`my_second_model.pkl`) and there are two versions; `1` and `2` .

When the service was deployed, both models are provided in the deploy operation:

```
first_model = Model(ws, name="my_first_model", version=1)
second_model = Model(ws, name="my_second_model", version=2)
service = Model.deploy(ws, "myservice", [first_model, second_model], inference_config, deployment_config)
```

In the Docker image that hosts the service, the `AZUREML_MODEL_DIR` environment variable contains the directory where the models are located. In this directory, each of the models is located in a directory path of `MODEL_NAME/VERSION`. Where `MODEL_NAME` is the name of the registered model, and `VERSION` is the version of the model. The files that make up the registered model are stored in these directories.

In this example, the paths would be `$AZUREML_MODEL_DIR/my_first_model/1/my_first_model.pkl` and `$AZUREML_MODEL_DIR/my_second_model/2/my_second_model.pkl`.

```
# Example when the model is a file, and the deployment contains multiple models
first_model_name = 'my_first_model'
first_model_version = '1'
first_model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), first_model_name, first_model_version,
'my_first_model.pkl')
second_model_name = 'my_second_model'
second_model_version = '2'
second_model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), second_model_name, second_model_version,
'my_second_model.pkl')
```

get_model_path

When you register a model, you provide a model name that's used for managing the model in the registry. You use this name with the [Model.get_model_path\(\)](#) method to retrieve the path of the model file or files on the local file system. If you register a folder or a collection of files, this API returns the path of the directory that contains those files.

When you register a model, you give it a name. The name corresponds to where the model is placed, either locally or during service deployment.

Framework-specific examples

More entry script examples for specific machine learning use cases can be found below:

- [PyTorch](#)
- [TensorFlow](#)
- [Keras](#)
- [AutoML](#)
- [ONNX](#)

Next steps

- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Create client applications to consume web services](#)
- [Update web service](#)
- [How to deploy a model using a custom Docker image](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Create event alerts and triggers for model deployments](#)

Collect data from models in production

12/23/2020 • 4 minutes to read • [Edit Online](#)

This article shows how to collect data from an Azure Machine Learning model deployed on an Azure Kubernetes Service (AKS) cluster. The collected data is then stored in Azure Blob storage.

Once collection is enabled, the data you collect helps you:

- [Monitor data drifts](#) on the production data you collect.
- Analyze collected data using [Power BI](#) or [Azure Databricks](#)
- Make better decisions about when to retrain or optimize your model.
- Retrain your model with the collected data.

What is collected and where it goes

The following data can be collected:

- Model input data from web services deployed in an AKS cluster. Voice audio, images, and video are *not* collected.
- Model predictions using production input data.

NOTE

Preaggregation and precalculations on this data are not currently part of the collection service.

The output is saved in Blob storage. Because the data is added to Blob storage, you can choose your favorite tool to run the analysis.

The path to the output data in the blob follows this syntax:

```
/modeldata/<subscriptionid>/<resourcegroup>/<workspace>/<webservice>/<model>/<version>/<designation>/<year>/<month>/<day>/data.csv  
# example: /modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myWorkspace/aks-w-  
collv9/best_model/10/inputs/2018/12/31/data.csv
```

NOTE

In versions of the Azure Machine Learning SDK for Python earlier than version 0.1.0a16, the `designation` argument is named `identifier`. If you developed your code with an earlier version, you need to update it accordingly.

Prerequisites

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- An Azure Machine Learning workspace, a local directory containing your scripts, and the Azure Machine Learning SDK for Python must be installed. To learn how to install them, see [How to configure a development environment](#).

- You need a trained machine-learning model to be deployed to AKS. If you don't have a model, see the [Train image classification model](#) tutorial.
- You need an AKS cluster. For information on how to create one and deploy to it, see [How to deploy and where](#).
- Set up your environment and install the [Azure Machine Learning Monitoring SDK](#).

Enable data collection

You can enable [data collection](#) regardless of the model you deploy through Azure Machine Learning or other tools.

To enable data collection, you need to:

1. Open the scoring file.
2. Add the following code at the top of the file:

```
from azureml.monitoring import ModelDataCollector
```

3. Declare your data collection variables in your `init` function:

```
global inputs_dc, prediction_dc
inputs_dc = ModelDataCollector("best_model", designation="inputs", feature_names=["feat1", "feat2",
"feat3", "feat4", "feat5", "feat6"])
prediction_dc = ModelDataCollector("best_model", designation="predictions", feature_names=
["prediction1", "prediction2"])
```

CorrelationId is an optional parameter. You don't need to use it if your model doesn't require it. Use of *CorrelationId* does help you more easily map with other data, such as *LoanNumber* or *CustomerId*.

The *Identifier* parameter is later used for building the folder structure in your blob. You can use it to differentiate raw data from processed data.

4. Add the following lines of code to the `run(input_df)` function:

```
data = np.array(data)
result = model.predict(data)
inputs_dc.collect(data) #this call is saving our input data into Azure Blob
prediction_dc.collect(result) #this call is saving our input data into Azure Blob
```

5. Data collection is *not* automatically set to **true** when you deploy a service in AKS. Update your configuration file, as in the following example:

```
aks_config = AksWebservice.deploy_configuration(collect_model_data=True)
```

You can also enable Application Insights for service monitoring by changing this configuration:

```
aks_config = AksWebservice.deploy_configuration(collect_model_data=True, enable_app_insights=True)
```

6. To create a new image and deploy the machine learning model, see [How to deploy and where](#).
7. Add the 'Azure-Monitoring' pip package to the conda-dependencies of the web service environment:

```
env = Environment('webserviceenv')
env.python.conda_dependencies = CondaDependencies.create(conda_packages=['numpy'],pip_packages=
['azureml-defaults','azureml-monitoring','inference-schema[numpy-support]'])
```

Disable data collection

You can stop collecting data at any time. Use Python code to disable data collection.

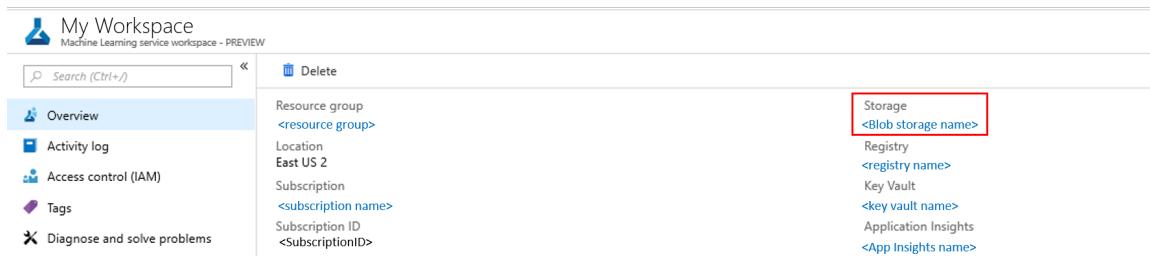
```
## replace <service_name> with the name of the web service
<service_name>.update(collect_model_data=False)
```

Validate and analyze your data

You can choose a tool of your preference to analyze the data collected in your Blob storage.

Quickly access your blob data

1. Sign in to [Azure portal](#).
2. Open your workspace.
3. Select **Storage**.



4. Follow the path to the blob's output data with this syntax:

```
/modeldata/<subscriptionid>/<resourcegroup>/<workspace>/<webservice>/<model>/<version>/<designation>/<year>/<month>/<day>/data.csv
# example: /modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myWorkspace/aks-w-
collv9/best_model/10/inputs/2018/12/31/data.csv
```

Analyze model data using Power BI

1. Download and open [Power BI Desktop](#).
2. Select **Get Data** and select [Azure Blob Storage](#).

Get Data

X

The screenshot shows the 'Get Data' dialog box with a search bar containing 'azure'. A sidebar on the left lists categories: 'All', 'Azure', and 'Online Services'. The 'All' category is selected and highlighted with a yellow border. Under 'All', a list of Azure services is shown, each with a small icon: Azure SQL database, Azure SQL Data Warehouse, Azure Analysis Services database, Azure Blob Storage (which is also highlighted with a yellow border), Azure Table Storage, Azure Cosmos DB (Beta), Azure Data Lake Store, Azure HDInsight (HDFS), Azure HDInsight Spark, Microsoft Azure Consumption Insights (Beta), and Azure KustoDB (Beta). At the bottom of the dialog are buttons for 'Certified Connectors', 'Connect' (in yellow), and 'Cancel'.

azure X

All

Azure

Online Services

Azure Blob Storage

Azure Table Storage

Azure Cosmos DB (Beta)

Azure Data Lake Store

Azure HDInsight (HDFS)

Azure HDInsight Spark

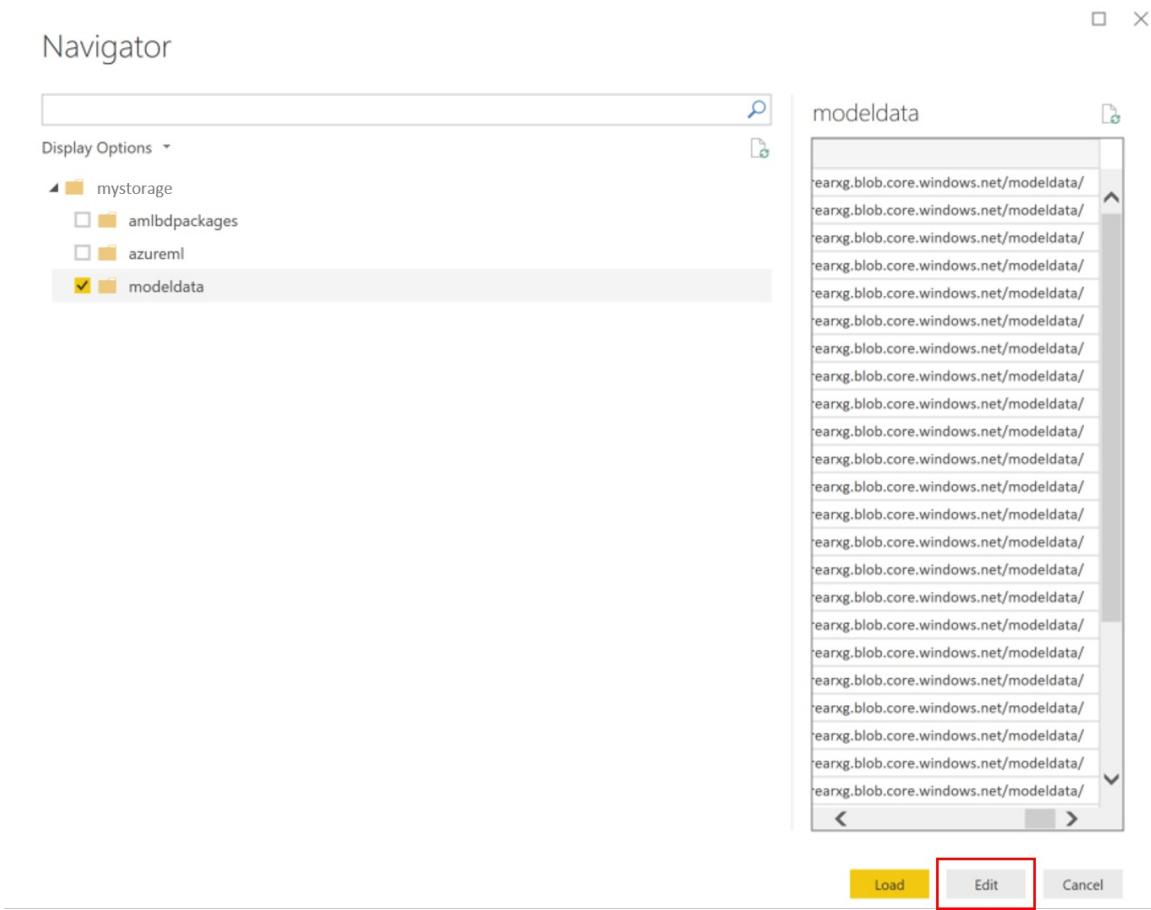
Microsoft Azure Consumption Insights (Beta)

Azure KustoDB (Beta)

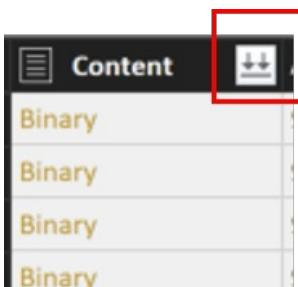
Certified Connectors

Connect Cancel

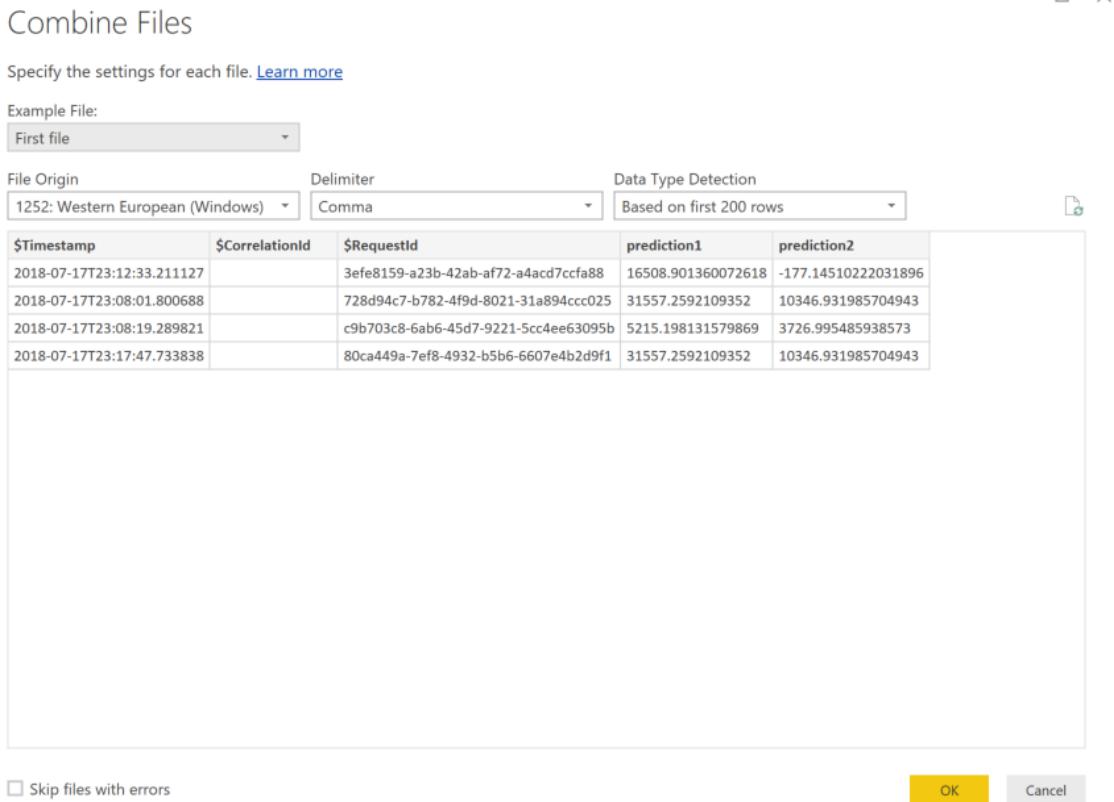
3. Add your storage account name and enter your storage key. You can find this information by selecting **Settings > Access keys** in your blob.
4. Select the **model data** container and select **Edit**.



- In the query editor, click under the **Name** column and add your storage account.
 - Enter your model path into the filter. If you want to look only into files from a specific year or month, just expand the filter path. For example, to look only into March data, use this filter path:
`/modeldata/<subscriptionid>/<resourcegroupname>/<workspacename>/<webservicename>/<modelname>/<modelversion>/<designation>/<year>/3`
 - Filter the data that is relevant to you based on **Name** values. If you stored predictions and inputs, you need to create a query for each.
 - Select the downward double arrows next to the **Content** column heading to combine the files.



9. Select OK. The data preloads.



10. Select Close and Apply.
11. If you added inputs and predictions, your tables are automatically ordered by **RequestId** values.
12. Start building your custom reports on your model data.

Analyze model data using Azure Databricks

1. Create an [Azure Databricks workspace](#).
2. Go to your Databricks workspace.
3. In your Databricks workspace, select **Upload Data**.



Azure Databricks



Explore the Quickstart Tutorial

Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.

Quickly im

Common Tasks

New Notebook

Upload Data

Create Table

New Cluster

New Job

Import Library

Read Documentation

Recents

2018-11-1

2018-06-1

setup2

2018-07-1

setup

4. Select **Create New Table** and select **Other Data Sources > Azure Blob Storage > Create Table in Notebook**.

Create New Table

Data source

Upload File DBFS Other Data Sources

Connector

Azure Blob Storage

Create Table in Notebook

5. Update the location of your data. Here is an example:

```
file_location = "wasbs://mycontainer@storageaccountname.blob.core.windows.net/modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myWorkspace/aks-w-collv9/best_model/10/inputs/2018/**/data.csv"
file_type = "csv"
```

Cmd 2

Step 1: Set the data location and type

There are two ways to access Azure Blob storage: account keys and shared access signatures (SAS).

To get started, we need to set the location and type of the file.

Cmd 3

```
1 storage_account_name = "mystorage"
2 storage_account_access_key = "3hsduhwe908rjjfoieudnf 98h v9udfhgb987yh g908ufgb98yfgb9 ufgb78gy8 gfo89ug98yfdg7yfg8ytg9fyg87"
```

Cmd 4

```
1 file_location = "wasbs://mycontainer@storageaccountname.blob.core.windows.net/modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myWorkspace/aks-w-collv9/best_model/10/inputs/2018/**/data.csv"
2 file_type = "csv"
```

Cmd 5

```
1 spark.conf.set(
2   "fs.azure.account.key."+storage_account_name+".blob.core.windows.net",
3   storage_account_access_key)
```

- Follow the steps on the template to view and analyze your data.

Next steps

[Detect data drift](#) on the data you have collected.

Monitor and collect data from ML web service endpoints

12/23/2020 • 5 minutes to read • [Edit Online](#)

In this article, you learn how to collect data from models deployed to web service endpoints in Azure Kubernetes Service (AKS) or Azure Container Instances (ACI). Use [Azure Application Insights](#) to collect the following data from an endpoint:

- Output data
- Responses
- Request rates, response times, and failure rates
- Dependency rates, response times, and failure rates
- Exceptions

The [enable-app-insights-in-production-service.ipynb](#) notebook demonstrates concepts in this article.

Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service](#).

Prerequisites

- An Azure subscription - try the [free or paid version of Azure Machine Learning](#).
- An Azure Machine Learning workspace, a local directory that contains your scripts, and the Azure Machine Learning SDK for Python installed. To learn more, see [How to configure a development environment](#).
- A trained machine learning model. To learn more, see the [Train image classification model](#) tutorial.

Configure logging with the Python SDK

In this section, you learn how to enable Application Insight logging by using the Python SDK.

Update a deployed service

Use the following steps to update an existing web service:

1. Identify the service in your workspace. The value for `ws` is the name of your workspace

```
from azureml.core.webservice import Webservice  
aks_service= Webservice(ws, "my-service-name")
```

2. Update your service and enable Azure Application Insights

```
aks_service.update(enable_app_insights=True)
```

Log custom traces in your service

IMPORTANT

Azure Application Insights only logs payloads of up to 64kb. If this limit is reached, you may see errors such as out of memory, or no information may be logged. If the data you want to log is larger 64kb, you should instead store it to blob storage using the information in [Collect Data for models in production](#).

For more complex situations, like model tracking within an AKS deployment, we recommend using a third-party library like [OpenCensus](#).

To log custom traces, follow the standard deployment process for AKS or ACI in the [How to deploy and where](#) document. Then, use the following steps:

1. Update the scoring file by adding print statements to send data to Application Insights during inference.

For more complex information, such as the request data and the response, use a JSON structure.

The following example `score.py` file logs when the model was initialized, input and output during inference, and the time any errors occur.

```
import pickle
import json
import numpy
from sklearn.externals import joblib
from sklearn.linear_model import Ridge
from azureml.core.model import Model
import time

def init():
    global model
    #Print statement for appinsights custom traces:
    print ("model initialized" + time.strftime("%H:%M:%S"))

    # note here "sklearn_regression_model.pkl" is the name of the model registered under the
    workspace
    # this call should return the path to the model.pkl file on the local disk.
    model_path = Model.get_model_path(model_name = 'sklearn_regression_model.pkl')

    # deserialize the model file back into a sklearn model
    model = joblib.load(model_path)

# note you can pass in multiple rows for scoring
def run(raw_data):
    try:
        data = json.loads(raw_data)['data']
        data = numpy.array(data)
        result = model.predict(data)
        # Log the input and output data to appinsights:
        info = {
            "input": raw_data,
            "output": result.tolist()
        }
        print(json.dumps(info))
        # you can return any datatype as long as it is JSON-serializable
        return result.tolist()
    except Exception as e:
        error = str(e)
        print (error + time.strftime("%H:%M:%S"))
        return error
```

2. Update the service configuration, and make sure to enable Application Insights.

```
config = Webservice.deploy_configuration(enable_app_insights=True)
```

3. Build an image and deploy it on AKS or ACI. For more information, see [How to deploy and where](#).

Disable tracking in Python

To disable Azure Application Insights, use the following code:

```
## replace <service_name> with the name of the web service
<service_name>.update(enable_app_insights=False)
```

Configure logging with Azure Machine Learning studio

You can also enable Azure Application Insights from Azure Machine Learning studio. When you're ready to deploy your model as a web service, use the following steps to enable Application Insights:

1. Sign in to the studio at <https://ml.azure.com>.
2. Go to **Models** and select the model you want to deploy.
3. Select **+Deploy**.
4. Populate the **Deploy model** form.
5. Expand the **Advanced** menu.

Deploy a model

i Customers should not include personal data or other sensitive information in fields marked with  because the content in these fields may be logged and shared across Microsoft systems to facilitate operations and troubleshooting. [Learn more](#) X

Name *



*

Description

Compute type *

▼

Models: AutoML6e225a7b963:1

Enable authentication



Entry script file * i

 *

Browse

Conda dependencies file *

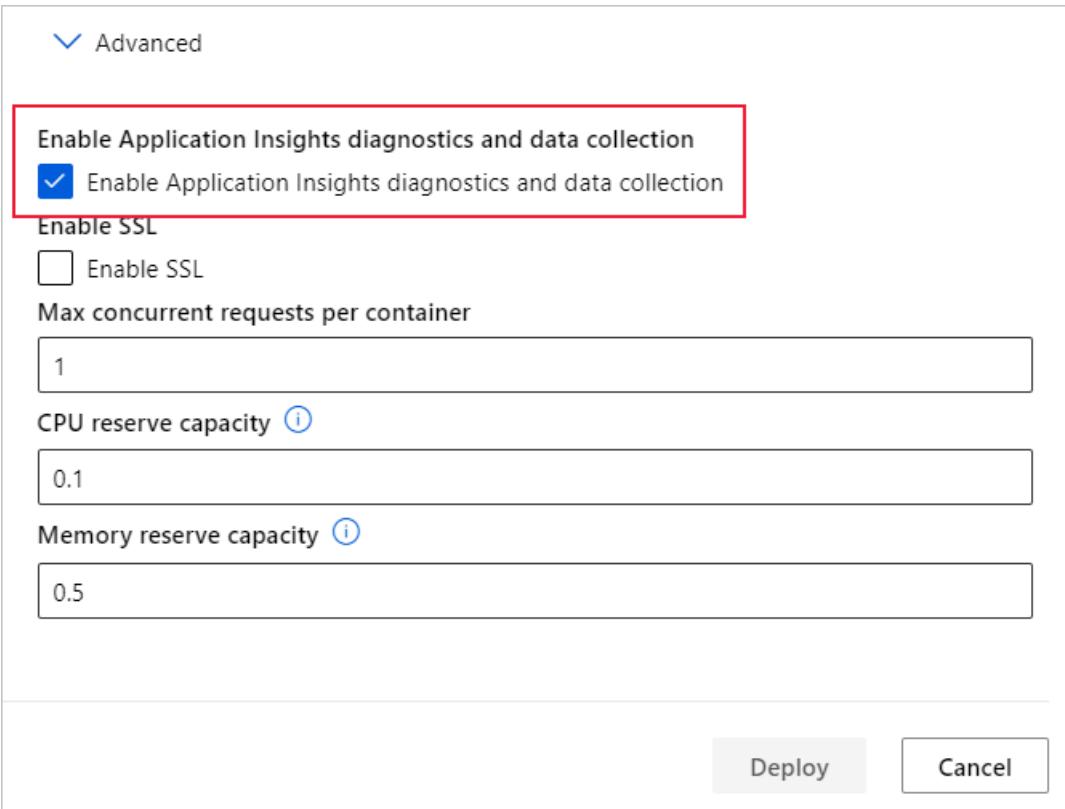
 *

Browse

Dependencies

> Advanced

6. Select Enable Application Insights diagnostics and data collection.



View metrics and logs

Query logs for deployed models

Logs of real-time endpoints are customer data. You can use the `get_logs()` function to retrieve logs from a previously deployed web service. The logs may contain detailed information about any errors that occurred during deployment.

```
from azureml.core import Workspace
from azureml.core.webservice import Webservice

ws = Workspace.from_config()

# load existing web service
service = Webservice(name="service-name", workspace=ws)
logs = service.get_logs()
```

If you have multiple Tenants, you may need to add the following authenticate code before

```
ws = Workspace.from_config()
```

```
from azureml.core.authentication import InteractiveLoginAuthentication
interactive_auth = InteractiveLoginAuthentication(tenant_id="the tenant_id in which your workspace resides")
```

View logs in the studio

Azure Application Insights stores your service logs in the same resource group as the Azure Machine Learning workspace. Use the following steps to view your data using the studio:

1. Go to your Azure Machine Learning workspace in the [studio](#).
2. Select **Endpoints**.
3. Select the deployed service.

4. Select the Application Insights url link.

The screenshot shows the Microsoft Azure Machine Learning interface. On the left, there's a sidebar with various options like 'New', 'Home', 'Notebooks', etc., and a red box highlights the 'Endpoints' option. The main area shows details for an endpoint named 'aks-service-appinsights'. It includes fields for CPU (0.1), Memory (0.5 GB), Application Insights enabled (true), and a red box highlights the 'Application Insights url' field, which contains a blue hyperlink: <https://portal.azure.com#@microsoft.onmicrosoft.com/resource/subscriptions/xxxXXXXXX..>

5. In Application Insights, from the Overview tab or the Monitoring section, select Logs.

The screenshot shows the Application Insights Overview page. The 'Logs' tab is selected. On the left, there's a sidebar with 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Investigate' (with 'Application map', 'Smart Detection', 'Live Metrics', 'Search', 'Availability', 'Failures', 'Performance', 'Troubleshooting guides'), 'Monitoring' (with 'Alerts' and 'Metrics'), and 'Logs'. The main area shows resource group details (Resource group: change, Location: West US, Subscription: documentationteam, Subscription ID: , Tags: Click here to add tags) and a chart titled 'Failed requests' showing a spike at 1:15 PM. Below the chart, it says 'Failed requests (Count)' and '10'. To the right are two more charts: 'Server response time' (Server response time (Avg) 0.69 ms) and 'Server requests' (Server requests (Count) 13).

6. To view information logged from the score.py file, look at the **traces** table. The following query searches for logs where the **input** value was logged:

```
traces
| where customDimensions contains "input"
| limit 10
```

The screenshot shows the Azure Application Insights Query Editor interface. On the left, there's a navigation sidebar with sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Investigate (Application map, Smart Detection, Live Metrics, Search, Availability, Failures, Performance, Troubleshooting guides), Monitoring (Alerts, Metrics, Logs, Workbooks), Usage (Users, Sessions, Events), and a Favorites section. The main area has tabs for Tables, Queries, and Filter. A search bar is at the top. Below it, a query editor window displays the following code:

```
traces
| where timestamp > ago(24h) and customDimensions contains "input"
| limit 10
```

The results pane shows a table with columns: timestamp [Local Time], message, severityLevel, itemType, and customDimensions. There are three records listed:

timestamp [Local Time]	message	severityLevel	itemType	customDimensions
6/8/2020, 11:56:42.718 AM	STDOUT	trace		{"Timestamp": "Jun 8 15:56:42", "Workspace Name": "larryml", "Service Name": "aks-w-dc5", "Container Id": "aks-w-dc5-5f68b86c9-gft56"}
6/8/2020, 11:56:49.561 AM	STDOUT	trace		{"Timestamp": "Jun 8 15:56:49", "Workspace Name": "larryml", "Service Name": "aks-w-dc5", "Container Id": "aks-w-dc5-5f68b86c9-gft56"}
6/8/2020, 2:08:54.950 PM	STDOUT	trace		{"Timestamp": "Jun 8 18:08:54", "Workspace Name": "larryml", "Service Name": "aks-w-dc5", "Container Id": "aks-w-dc5-5f68b86c9-gft56"}

Each record has a 'Contents' link under customDimensions, which expands to show 'input' and 'output' fields. The bottom of the results pane shows pagination controls.

For more information on how to use Azure Application Insights, see [What is Application Insights?](#).

Web service metadata and response data

IMPORTANT

Azure Application Insights only logs payloads of up to 64kb. If this limit is reached then you may see errors such as out of memory, or no information may be logged.

To log web service request information, add `print` statements to your score.py file. Each `print` statement results in one entry in the Application Insights trace table under the message `STDOUT`. Application Insights stores the `print` statement outputs in `customDimensions` and in the `Contents` trace table. Printing JSON strings produces a hierarchical data structure in the trace output under `Contents`.

Export data for retention and processing

IMPORTANT

Azure Application Insights only supports exports to blob storage. For more information on the limits of this implementation, see [Export telemetry from App Insights](#).

Use Application Insights' [continuous export](#) to export data to a blob storage account where you can define retention settings. Application Insights exports the data in JSON format.

Dashboard > azureml1588320359 - Continuous export

azureml1588320359 - Continuous export

Application Insights

Search (Ctrl+ /)

Add Refresh Help

Usage

- Users
- Sessions
- Events
- Funnels
- User Flows
- Retention
- Impact
- Cohorts
- More

Configure

- Properties
- Smart Detection settings
- Usage and estimated costs
- Continuous export
- Performance Testing
- API Access

STATUS	STORAGE ACCOUNT	LAST EXPORT
✓	copeterstest3321914124	9/27/2019, 8:55:10 AM
✓	chiworkspace3965959982	9/27/2019, 8:55:10 AM
✓	chichesstoragebmskrivc	9/27/2019, 8:55:47 AM
✓	azureml5386059871	9/27/2019, 8:55:47 AM

Next steps

In this article, you learned how to enable logging and view logs for web service endpoints. Try these articles for next steps:

- [How to deploy a model to an AKS cluster](#)
- [How to deploy a model to Azure Container Instances](#)
- [MLOps: Manage, deploy, and monitor models with Azure Machine Learning](#) to learn more about leveraging data collected from models in production. Such data can help to continually improve your machine learning process.

High-performance serving with Triton Inference Server (Preview)

12/23/2020 • 7 minutes to read • [Edit Online](#)

Learn how to use [NVIDIA Triton Inference Server](#) to improve the performance of the web service used for model inference.

One of the ways to deploy a model for inference is as a web service. For example, a deployment to Azure Kubernetes Service or Azure Container Instances. By default, Azure Machine Learning uses a single-threaded, *general purpose* web framework for web service deployments.

Triton is a framework that is *optimized for inference*. It provides better utilization of GPUs and more cost-effective inference. On the server-side, it batches incoming requests and submits these batches for inference. Batching better utilizes GPU resources, and is a key part of Triton's performance.

IMPORTANT

Using Triton for deployment from Azure Machine Learning is currently in [preview](#). Preview functionality may not be covered by customer support. For more information, see the [Supplemental terms of use for Microsoft Azure previews](#).

TIP

The code snippets in this document are for illustrative purposes and may not show a complete solution. For working example code, see the [end-to-end samples of Triton in Azure Machine Learning](#).

Prerequisites

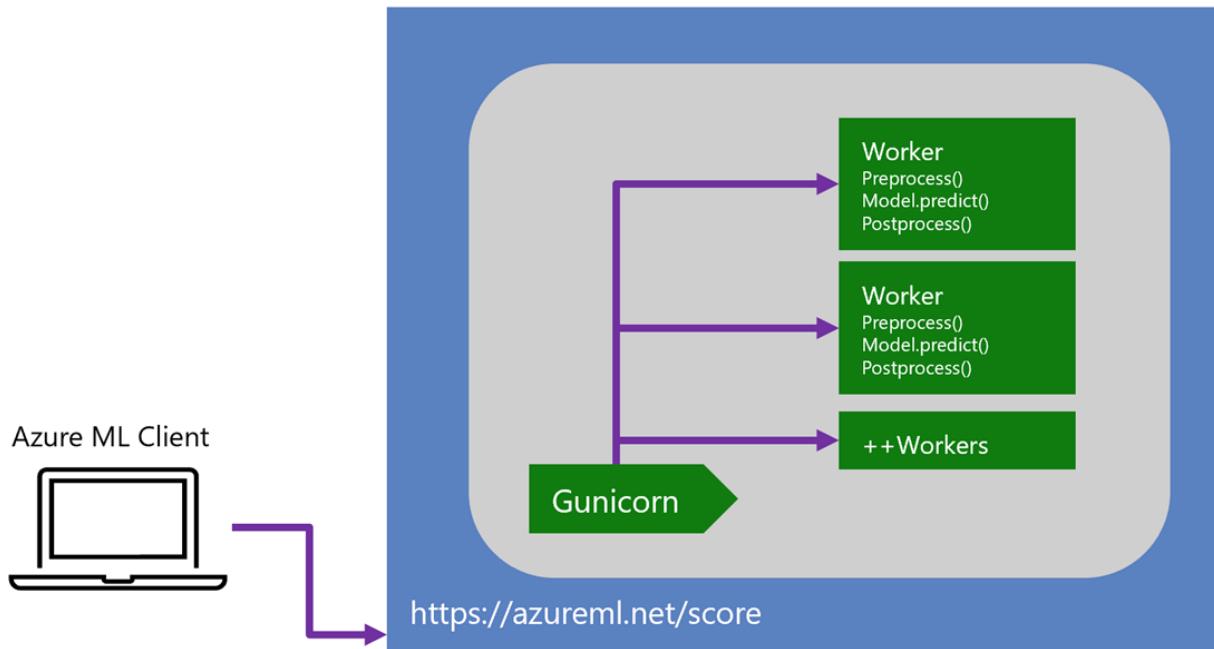
- An [Azure subscription](#). If you do not have one, try the [free or paid version of Azure Machine Learning](#).
- Familiarity with [how and where to deploy a model](#) with Azure Machine Learning.
- The [Azure Machine Learning SDK for Python](#) or the [Azure CLI](#) and [machine learning extension](#).
- A working installation of Docker for local testing. For information on installing and validating Docker, see [Orientation and setup](#) in the docker documentation.

Architectural overview

Before attempting to use Triton for your own model, it's important to understand how it works with Azure Machine Learning and how it compares to a default deployment.

Default deployment without Triton

- Multiple [Gunicorn](#) workers are started to concurrently handle incoming requests.
- These workers handle pre-processing, calling the model, and post-processing.
- Inference requests use the [scoring URI](#). For example, `https://myservice.azureml.net/score`.



Setting the number of workers

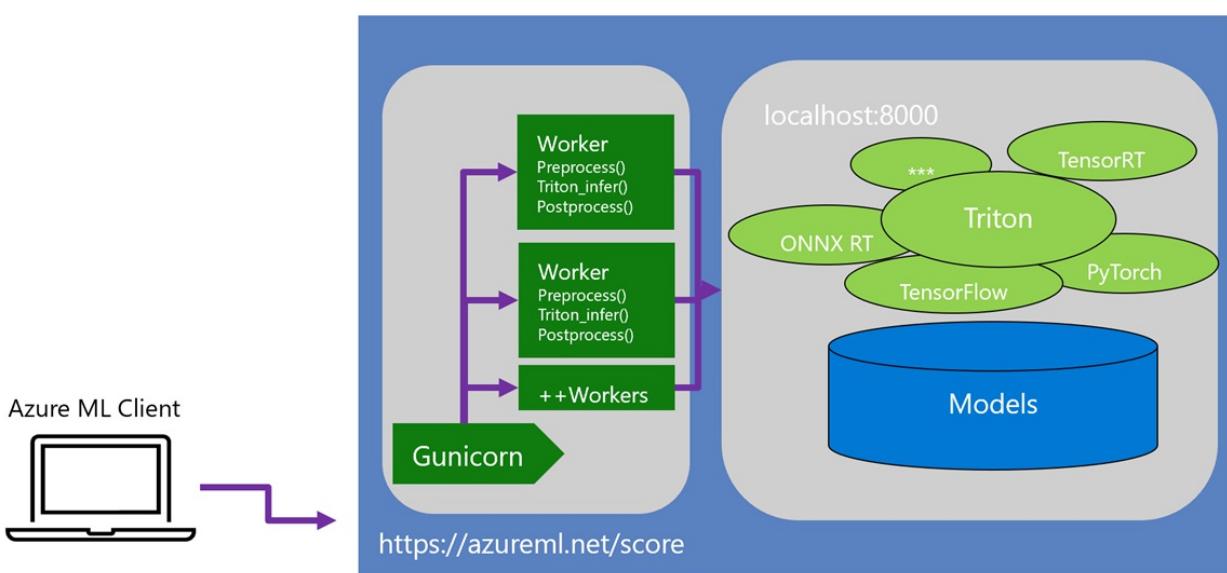
To set the number of workers in your deployment, set the environment variable `WORKER_COUNT`. Given you have an **Environment** object called `env`, you can do the following:

```
env.environment_variables["WORKER_COUNT"] = "1"
```

This will tell Azure ML to spin up the number of workers you specify.

Inference configuration deployment with Triton

- Multiple **Gunicorn** workers are started to concurrently handle incoming requests.
- The requests are forwarded to the **Triton server**.
- Triton processes requests in batches to maximize GPU utilization.
- The client uses the **scoring URI** to make requests. For example, [https://myservice.azureml.net\(score](https://myservice.azureml.net(score).



The workflow to use Triton for your model deployment is:

- Verify that Triton can serve your model.
- Verify you can send requests to your Triton-deployed model.

3. Incorporate your Triton-specific code into your AML deployment.

Verify that Triton can serve your model

First, follow the steps below to verify that the Triton Inference Server can serve your model.

(Optional) Define a model config file

The model configuration file tells Triton how many inputs to expects and of what dimensions those inputs will be. For more information on creating the configuration file, see [Model configuration](#) in the NVIDIA documentation.

TIP

We use the `--strict-model-config=false` option when starting the Triton Inference Server, which means you do not need to provide a `config.pbtxt` file for ONNX or TensorFlow models.

For more information on this option, see [Generated model configuration](#) in the NVIDIA documentation.

Use the correct directory structure

When registering a model with Azure Machine Learning, you can register either individual files or a directory structure. To use Triton, the model registration must be for a directory structure that contains a directory named `triton`. The general structure of this directory is:

```
models
  - triton
    - model_1
      - model_version
        - model_file
        - config_file
    - model_2
    ...
  ...
```

IMPORTANT

This directory structure is a Triton Model Repository and is required for your model(s) to work with Triton. For more information, see [Triton Model Repositories](#) in the NVIDIA documentation.

Test with Triton and Docker

To test your model to make sure it runs with Triton, you can use Docker. The following commands pull the Triton container to your local computer, and then start the Triton Server:

1. To pull the image for the Triton server to your local computer, use the following command:

```
docker pull nvcr.io/nvidia/tritonserver:20.09-py3
```

2. To start the Triton server, use the following command. Replace `<path-to-models/triton>` with the path to the Triton model repository that contains your models:

```
docker run --rm -ti -v<path-to-models/triton>:/models nvcr.io/nvidia/tritonserver:20.09-py3 tritonserver
--model-repository=/models --strict-model-config=false
```

IMPORTANT

If you are using Windows, you may be prompted to allow network connections to this process the first time you run the command. If so, select to enable access.

Once started, information similar to the following text is logged to the command line:

```
I0923 19:21:30.582866 1 http_server.cc:2705] Started HTTPService at 0.0.0.0:8000
I0923 19:21:30.626081 1 http_server.cc:2724] Started Metrics Service at 0.0.0.0:8002
```

The first line indicates the address of the web service. In this case, `0.0.0.0:8000`, which is the same as `localhost:8000`.

3. Use a utility such as curl to access the health endpoint.

```
curl -L -v -i localhost:8000/v2/health/ready
```

This command returns information similar to the following. Note the `200 OK`; this status means the web server is running.

```
* Trying 127.0.0.1:8000...
* Connected to localhost (127.0.0.1) port 8000 (#0)
> GET /v2/health/ready HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.71.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
```

Beyond a basic health check, you can create a client to send data to Triton for inference. For more information on creating a client, see the [client examples](#) in the NVIDIA documentation. There are also [Python samples at the Triton GitHub](#).

For more information on running Triton using Docker, see [Running Triton on a system with a GPU](#) and [Running Triton on a system without a GPU](#).

Register your model

Now that you've verified that your model works with Triton, register it with Azure Machine Learning. Model registration stores your model files in the Azure Machine Learning workspace, and are used when deploying with the Python SDK and Azure CLI.

The following examples demonstrate how to register the model(s):

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

```
from azureml.core.model import Model

model = Model.register(
    model_path=os.path.join("..", "triton"),
    model_name="bidaf_onnx",
    tags={'area': "Natural language processing", 'type': "Question answering"},
    description="Question answering model from ONNX model zoo",
    workspace=ws
```

Verify you can call into your model

After verifying that the Triton is able to serve your model, you can add pre and post-processing code by defining an *entry script*. This file is named `score.py`. For more information on entry scripts, see [Define an entry script](#).

The two main steps are to initialize a Triton HTTP client in your `init()` method, and to call into that client in your `run()` function.

Initialize the Triton Client

Include code like the following example in your `score.py` file. Triton in Azure Machine Learning expects to be addressed on localhost, port 8000. In this case, localhost is inside the Docker image for this deployment, not a port on your local machine:

TIP

The `tritonhttpclient` pip package is included in the curated `AzureML-Triton` environment, so there's no need to specify it as a pip dependency.

```
import tritonhttpclient

def init():
    global triton_client
    triton_client = tritonhttpclient.InferenceServerClient(url="localhost:8000")
```

Modify your scoring script to call into Triton

The following example demonstrates how to dynamically request the metadata for the model:

TIP

You can dynamically request the metadata of models that have been loaded with Triton by using the `.get_model_metadata` method of the Triton client. See the [sample notebook](#) for an example of its use.

```
input = tritonhttpclient.InferInput(input_name, data.shape, datatype)
input.set_data_from_numpy(data, binary_data=binary_data)

output = tritonhttpclient.InferRequestedOutput(
    output_name, binary_data=binary_data, class_count=class_count)

# Run inference
res = triton_client.infer(model_name,
                           [input]
                           request_id='0',
                           outputs=[output])
```

Redeploy with an inference configuration

An inference configuration allows you use an entry script, as well as the Azure Machine Learning deployment process using the Python SDK or Azure CLI.

IMPORTANT

You must specify the `AzureML-Triton` curated environment.

The Python code example clones `AzureML-Triton` into another environment called `My-Triton`. The Azure CLI code also uses this environment. For more information on cloning an environment, see the [Environment.Clone\(\)](#) reference.

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

```
from azureml.core.webservice import LocalWebservice
from azureml.core import Environment
from azureml.core.model import InferenceConfig

local_service_name = "triton-bidaf-onnx"
env = Environment.get(ws, "AzureML-Triton").clone("My-Triton")

for pip_package in ["nltk"]:
    env.python.conda_dependencies.add_pip_package(pip_package)

inference_config = InferenceConfig(
    entry_script="score_bidaf.py", # This entry script is where we dispatch a call to the Triton server
    source_directory=os.path.join("../", "scripts"),
    environment=env
)

local_config = LocalWebservice.deploy_configuration(
    port=6789
)

local_service = Model.deploy(
    workspace=ws,
    name=local_service_name,
    models=[model],
    inference_config=inference_config,
    deployment_config=local_config,
    overwrite=True)

local_service.wait_for_deployment(show_output = True)
print(local_service.state)
# Print the URI you can use to call the local deployment
print(local_service.scoring_uri)
```

After deployment completes, the scoring URI is displayed. For this local deployment, it will be

`http://localhost:6789/score`. If you deploy to the cloud, you can use the `az ml service show` CLI command to get the scoring URI.

For information on how to create a client that sends inference requests to the scoring URI, see [consume a model deployed as a web service](#).

Clean up resources

If you plan on continuing to use the Azure Machine Learning workspace, but want to get rid of the deployed service, use one of the following options:

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

```
local_service.delete()
```

Next steps

- See [end-to-end samples of Triton in Azure Machine Learning](#)
- Check out [Triton client examples](#)
- Read the [Triton Inference Server documentation](#)
- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Update web service](#)
- [Collect data for models in production](#)

Continuously deploy models

12/23/2020 • 2 minutes to read • [Edit Online](#)

This article shows how to use continuous deployment in Azure DevOps to automatically check for new versions of registered models and push those new models into production.

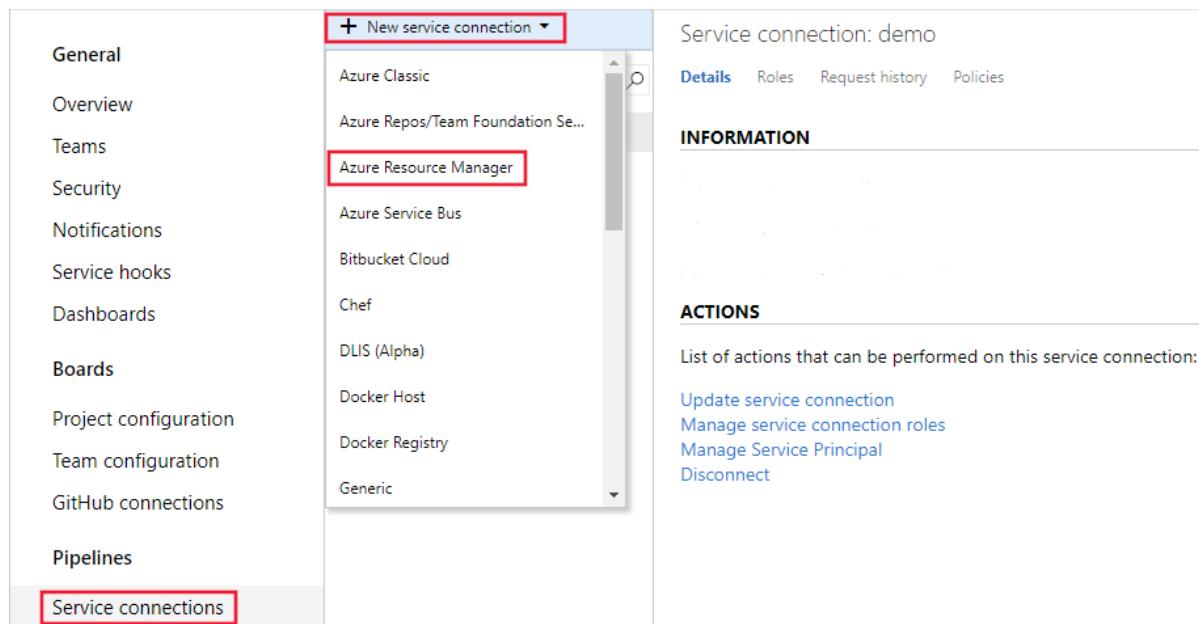
Prerequisites

This article assumes you have already registered a model in your Azure Machine Learning workspace. See [this tutorial](#) for an example of training and registering a scikit-learn model.

Continuously deploy models

You can continuously deploy models by using the Machine Learning extension for [Azure DevOps](#). You can use the Machine Learning extension for Azure DevOps to trigger a deployment pipeline when a new machine learning model is registered in an Azure Machine Learning workspace.

1. Sign up for [Azure Pipelines](#), which makes continuous integration and delivery of your application to any platform or cloud possible. (Note that Azure Pipelines isn't the same as [Machine Learning pipelines](#).)
2. [Create an Azure DevOps project](#).
3. Install the [Machine Learning extension for Azure Pipelines](#).
4. Use service connections to set up a service principal connection to your Azure Machine Learning workspace so you can access your artifacts. Go to project settings, select **Service connections**, and then select **Azure Resource Manager**:



5. In the Scope level list, select **AzureMLWorkspace**, and then enter the rest of the values:

Add an Azure Resource Manager service connection

Service Principal Authentication Managed Identity Authentication

Connection name	demo
Scope level	AzureMLWorkspace
Subscription	
Resource Group	
Machine Learning Workspace	

Machine Learning Workspaces listed are from Azure Cloud

A new Azure service principal will be created and assigned with the "Contributor" role, having access to all resources within the Workspace.

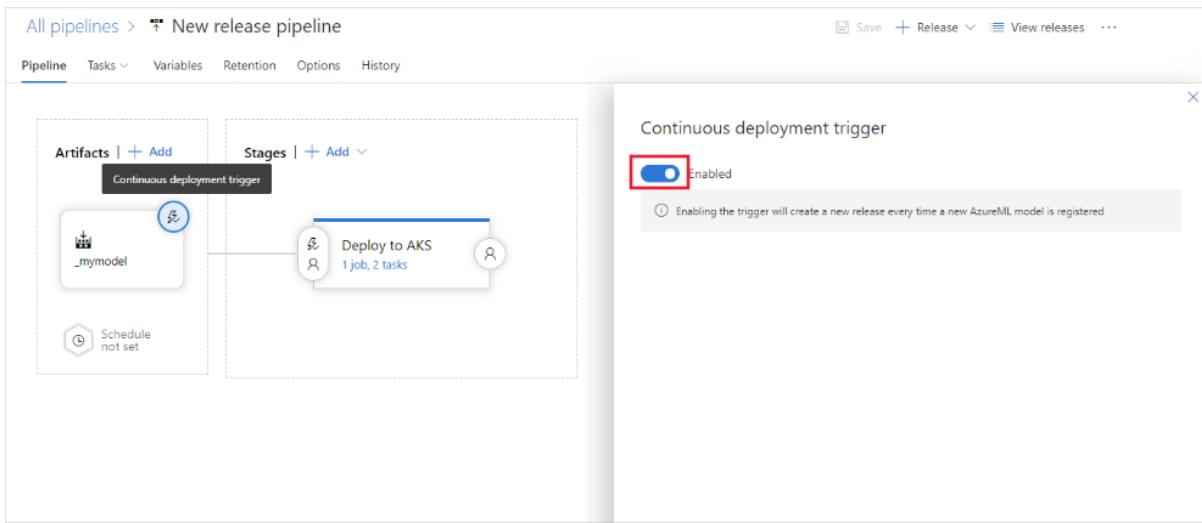
Allow all pipelines to use this connection.

OK **Close**

- To continuously deploy your machine learning model by using Azure Pipelines, under pipelines, select **release**. Add a new artifact, and then select the **AzureML Model** artifact and the service connection that you created earlier. Select the model and version to trigger a deployment:

The screenshot shows the 'Add an artifact' dialog in the Azure DevOps interface. On the left, there's a preview of a release pipeline with one stage named 'Stage 1'. On the right, the 'Source type' section is expanded, showing various options like Build, GitHub, TFVC, Azure Artifacts, Azure Container Registry, Docker Hub, and Jenkins. The 'AzureML Mo...' option is highlighted with a red box. Below it, there are fields for 'Service Endpoint', 'Model Names', 'Default version', and 'Source alias', all set to their respective values. At the bottom is a large blue 'Add' button.

- Enable the model trigger on your model artifact. When you turn on the trigger, every time the specified version (that is, the newest version) of that model is registered in your workspace, an Azure DevOps release pipeline is triggered.



Next steps

Check out the below projects on GitHub for more examples of continuous deployment for ML models.

- [Microsoft/MLOps](#)
- [Microsoft/MLOpsPython](#)

How to deploy an encrypted inferencing web service (preview)

12/23/2020 • 6 minutes to read • [Edit Online](#)

Learn how to deploy an image classification model as an encrypted inferencing web service in [Azure Container Instances](#) (ACI). The web service is a Docker container image that contains the model and scoring logic.

In this guide, you use Azure Machine Learning service to:

- Configure your environments
- Deploy encrypted inferencing web service
- Prepare test data
- Make encrypted predictions
- Clean up resources

ACI is a great solution for testing and understanding the model deployment workflow. For scalable production deployments, consider using Azure Kubernetes Service. For more information, see [how to deploy and where](#).

The encryption method used in this sample is [homomorphic encryption](#). Homomorphic encryption allows for computations to be done on encrypted data without requiring access to a secret (decryption) key. The results of the computations are encrypted and can be revealed only by the owner of the secret key.

Prerequisites

This guide assumes that you have an image classification model registered in Azure Machine Learning. If not, register the model using a [pretrained model](#) or create your own by completing the [train an image classification model with Azure Machine Learning](#) tutorial.

Configure local environment

In a Jupyter notebook

1. Import the Python packages needed for this sample.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

import azureml.core

# display the core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)
```

2. Install homomorphic encryption library for secure inferencing.

NOTE

The `encrypted-inference` package is currently in preview.

`encrypted-inference` is a library that contains bindings for encrypted inferencing based on [Microsoft SEAL](#).

```
!pip install encrypted-inference==0.9
```

Configure the inferencing environment

Create an environment for inferencing and add `encrypted-inference` package as a conda dependency.

```
from azureml.core.environment import Environment
from azureml.core.conda_dependencies import CondaDependencies

# to install required packages
env = Environment('tutorial-env')
cd = CondaDependencies.create(pip_packages=['azureml-databricks[pandas,fuse]>=1.1.14', 'azureml-defaults',
'azure-storage-blob', 'encrypted-inference==0.9'], conda_packages = ['scikit-learn==0.22.1'])

env.python.conda_dependencies = cd

# Register environment to re-use later
env.register(workspace = ws)
```

Deploy encrypted inferencing web service

Deploy the model as a web service hosted in ACI.

To build the correct environment for ACI, provide the following:

- A scoring script to show how to use the model
- A configuration file to build the ACI
- A trained model

Create scoring script

Create the scoring script `score.py` used by the web service for inferencing.

You must include two required functions into the scoring script:

- The `init()` function, which typically loads the model into a global object. This function is run only once when the Docker container is started.
- The `run(input_data)` function uses the model to predict a value based on the input data. Inputs and outputs to the run typically use JSON for serialization and de-serialization, but other formats are supported. The function fetches homomorphic encryption based public keys that are uploaded by the service caller.

```

%%writefile score.py
import json
import os
import pickle
import joblib
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient, PublicAccess
from encrypted.inference.eiserver import EIserver

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)
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'sklearn_mnist_model.pkl')
    model = joblib.load(model_path)

    global server
    server = EIserver(model.coef_, model.intercept_, verbose=True)

def run(raw_data):
    json_properties = json.loads(raw_data)

    key_id = json_properties['key_id']
    conn_str = json_properties['conn_str']
    container = json_properties['container']
    data = json_properties['data']

    # download the public keys from blob storage
    blob_service_client = BlobServiceClient.from_connection_string(conn_str=conn_str)
    blob_client = blob_service_client.get_blob_client(container=container, blob=key_id)
    public_keys = blob_client.download_blob().readall()

    result = {}
    # make prediction
    result = server.predict(data, public_keys)

    # you can return any data type as long as it is JSON-serializable
    return result

```

Create configuration file

Create a deployment configuration file and specify the number of CPUs and gigabyte of RAM needed for your ACI container. While it depends on your model, the default of 1 core and 1 gigabyte of RAM is usually sufficient for many models. If you feel you need more later, you would have to recreate the image and redeploy the service.

```

from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores=1,
                                                memory_gb=1,
                                                tags={"data": "MNIST", "method" : "sklearn"},
                                                description='Encrypted Predict MNIST with sklearn + SEAL')

```

Deploy to Azure Container Instances

Estimated time to complete: **about 2-5 minutes**

Configure the image and deploy. The following code goes through these steps:

1. Create environment object containing dependencies needed by the model using the environment file (`myenv.yml`)
2. Create inference configuration necessary to deploy the model as a web service using:
 - The scoring file (`score.py`)

- Environment object created in the previous step
3. Deploy the model to the ACI container.
 4. Get the web service HTTP endpoint.

```
%%time
from azureml.core.webservice import Webservice
from azureml.core.model import InferenceConfig
from azureml.core.environment import Environment
from azureml.core import Workspace
from azureml.core.model import Model

ws = Workspace.from_config()
model = Model(ws, 'sklearn_mnist')

myenv = Environment.get(workspace=ws, name="tutorial-env")
inference_config = InferenceConfig(entry_script="score.py", environment=myenv)

service = Model.deploy(workspace=ws,
                      name='sklearn-encrypted-mnist-svc',
                      models=[model],
                      inference_config=inference_config,
                      deployment_config=aciconfig)

service.wait_for_deployment(show_output=True)
```

Get the scoring web service's HTTP endpoint, which accepts REST client calls. This endpoint can be shared with anyone who wants to test the web service or integrate it into an application.

```
print(service.scoring_uri)
```

Prepare test data

1. Download the test data. In this case, it's saved into a directory called *data*.

```
import os
from azureml.core import Dataset
from azureml.opendatasets import MNIST

data_folder = os.path.join(os.getcwd(), 'data')
os.makedirs(data_folder, exist_ok=True)

mnist_file_dataset = MNIST.get_file_dataset()
mnist_file_dataset.download(data_folder, overwrite=True)
```

2. Load the test data from the *data* directory.

```
from utils import load_data
import os
import glob

data_folder = os.path.join(os.getcwd(), 'data')
# note we also shrink the intensity values (X) from 0-255 to 0-1. This helps the neural network
# converge faster
X_test = load_data(glob.glob(os.path.join(data_folder, "**/t10k-images-idx3-ubyte.gz"), recursive=True)
[0], False) / 255.0
y_test = load_data(glob.glob(os.path.join(data_folder, "**/t10k-labels-idx1-ubyte.gz"), recursive=True)
[0], True).reshape(-1)
```

Make encrypted predictions

Use the test dataset with the model to get predictions.

To make encrypted predictions:

1. Create a new `EILinearRegressionClient`, a homomorphic encryption based client, and public keys.

```
from encrypted.inference.eiclient import EILinearRegressionClient

# Create a new Encrypted inference client and a new secret key.
edp = EILinearRegressionClient(verbose=True)

public_keys_blob, public_keys_data = edp.get_public_keys()
```

2. Upload homomorphic encryption generated public keys to the workspace default blob store. This will allow you to share the keys with the inference server.

```
import azureml.core
from azureml.core import Workspace, Datastore
import os

ws = Workspace.from_config()

datastore = ws.get_default_datastore()
container_name=datastore.container_name

# Create a local file and write the keys to it
public_keys = open(public_keys_blob, "wb")
public_keys.write(public_keys_data)
public_keys.close()

# Upload the file to blob store
datastore.upload_files([public_keys_blob])

# Delete the local file
os.remove(public_keys_blob)
```

3. Encrypt the test data

```
#choose any one sample from the test data
sample_index = 1

#encrypt the data
raw_data = edp.encrypt(X_test[sample_index])
```

4. Use the SDK's `run` API to invoke the service and provide the test dataset to the model to get predictions.

We will need to send the connection string to the blob storage where the public keys were uploaded.

```

import json
from azureml.core import Webservice

service = Webservice(ws, 'sklearn-encrypted-mnist-svc')

#pass the connection string for blob storage to give the server access to the uploaded public keys
conn_str_template = 'DefaultEndpointsProtocol={}';AccountName={};AccountKey=
{};EndpointSuffix=core.windows.net'
conn_str = conn_str_template.format(datastore.protocol, datastore.account_name, datastore.account_key)

#build the json
data = json.dumps({'data': raw_data, "key_id" : public_keys_blob, "conn_str" : conn_str, "container" :
container_name })
data = bytes(data, encoding='ASCII')

print ('Making an encrypted inference web service call ')
eresult = service.run(input_data=data)

print ('Received encrypted inference results')

```

5. Use the client to decrypt the results.

```

import numpy as np

results = edp.decrypt(eresult)

print ('Decrypted the results ', results)

#Apply argmax to identify the prediction result
prediction = np.argmax(results)

print ( ' Prediction : ', prediction)
print ( ' Actual Label : ', y_test[sample_index])

```

Clean up resources

Delete the web service created in this sample:

```
service.delete()
```

If you no longer plan to use the Azure resources you've created, delete them. This prevents you from being charged for unutilized resources that are still running. See this guide on how to [clean up resources](#) to learn more.

Profile your model to determine resource utilization

12/23/2020 • 3 minutes to read • [Edit Online](#)

This article shows how to profile a machine learning model to determine how much CPU and memory you will need to allocate for the model when deploying it as a web service.

Prerequisites

This article assumes you have trained and registered a model with Azure Machine Learning. See the [sample tutorial here](#) for an example of training and registering a scikit-learn model with Azure Machine Learning.

Limitations

- Profiling will not work when the Azure Container Registry (ACR) for your workspace is behind a virtual network.

Run the profiler

Once you have registered your model and prepared the other components necessary for its deployment, you can determine the CPU and memory the deployed service will need. Profiling tests the service that runs your model and returns information such as the CPU usage, memory usage, and response latency. It also provides a recommendation for the CPU and memory based on resource usage.

In order to profile your model, you will need:

- A registered model.
- An inference configuration based on your entry script and inference environment definition.
- A single column tabular dataset, where each row contains a string representing sample request data.

IMPORTANT

At this point we only support profiling of services that expect their request data to be a string, for example: string serialized json, text, string serialized image, etc. The content of each row of the dataset (string) will be put into the body of the HTTP request and sent to the service encapsulating the model for scoring.

IMPORTANT

We only support profiling up to 2 CPUs in ChinaEast2 and USGovArizona region.

Below is an example of how you can construct an input dataset to profile a service that expects its incoming request data to contain serialized json. In this case, we created a dataset based 100 instances of the same request data content. In real world scenarios we suggest that you use larger datasets containing various inputs, especially if your model resource usage/behavior is input dependent.

```

import json
from azureml.core import Datastore
from azureml.core.dataset import Dataset
from azureml.data import dataset_type_definitions

input_json = {'data': [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                      [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]]}

# create a string that can be utf-8 encoded and
# put in the body of the request
serialized_input_json = json.dumps(input_json)
dataset_content = []
for i in range(100):
    dataset_content.append(serialized_input_json)
dataset_content = '\n'.join(dataset_content)
file_name = 'sample_request_data.txt'
f = open(file_name, 'w')
f.write(dataset_content)
f.close()

# upload the txt file created above to the Datastore and create a dataset from it
data_store = Datastore.get_default(ws)
data_store.upload_files(['./' + file_name], target_path='sample_request_data')
datastore_path = [(data_store, 'sample_request_data' + '/' + file_name)]
sample_request_data = Dataset.Tabular.from_delimited_files(
    datastore_path,
    separator='\n',
    infer_column_types=True,
    header=dataset_type_definitions.PromoteHeadersBehavior.NO_HEADERS)
sample_request_data = sample_request_data.register(workspace=ws,
                                                   name='sample_request_data',
                                                   create_new_version=True)

```

Once you have the dataset containing sample request data ready, create an inference configuration. Inference configuration is based on the score.py and the environment definition. The following example demonstrates how to create the inference configuration and run profiling:

```

from azureml.core.model import InferenceConfig, Model
from azureml.core.dataset import Dataset

model = Model(ws, id=model_id)
inference_config = InferenceConfig(entry_script='path-to-score.py',
                                    environment=myenv)
input_dataset = Dataset.get_by_name(workspace=ws, name='sample_request_data')
profile = Model.profile(ws,
                        'unique_name',
                        [model],
                        inference_config,
                        input_dataset=input_dataset)

profile.wait_for_completion(True)

# see the result
details = profile.get_details()

```

The following command demonstrates how to profile a model by using the CLI:

```
az ml model profile -g <resource-group-name> -w <workspace-name> --inference-config-file <path-to-inf-config.json> -m <model-id> --idi <input-dataset-id> -n <unique-name>
```

TIP

To persist the information returned by profiling, use tags or properties for the model. Using tags or properties stores the data with the model in the model registry. The following examples demonstrate adding a new tag containing the `requestedCpu` and `requestedMemoryInGb` information:

```
model.add_tags({'requestedCpu': details['requestedCpu'],
    'requestedMemoryInGb': details['requestedMemoryInGb']})
```

```
az ml model profile -g <resource-group-name> -w <workspace-name> --i <model-id> --add-tag requestedCpu=1 --
add-tag requestedMemoryInGb=0.5
```

Next steps

- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Create client applications to consume web services](#)
- [Update web service](#)
- [How to deploy a model using a custom Docker image](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Create event alerts and triggers for model deployments](#)

Use the studio to deploy models trained in the designer

12/23/2020 • 7 minutes to read • [Edit Online](#)

In this article, you learn how to deploy a designer model as a real-time endpoint in Azure Machine Learning studio.

Once registered or downloaded, you can use designer trained models just like any other model. Exported models can be deployed in use cases such as internet of things (IoT) and local deployments.

Deployment in the studio consists of the following steps:

1. Register the trained model.
2. Download the entry script and conda dependencies file for the model.
3. (Optional) Configure the entry script.
4. Deploy the model to a compute target.

You can also deploy models directly in the designer to skip model registration and file download steps. This can be useful for rapid deployment. For more information see, [Deploy a model with the designer](#).

Models trained in the designer can also be deployed through the SDK or command-line interface (CLI). For more information, see [Deploy your existing model with Azure Machine Learning](#).

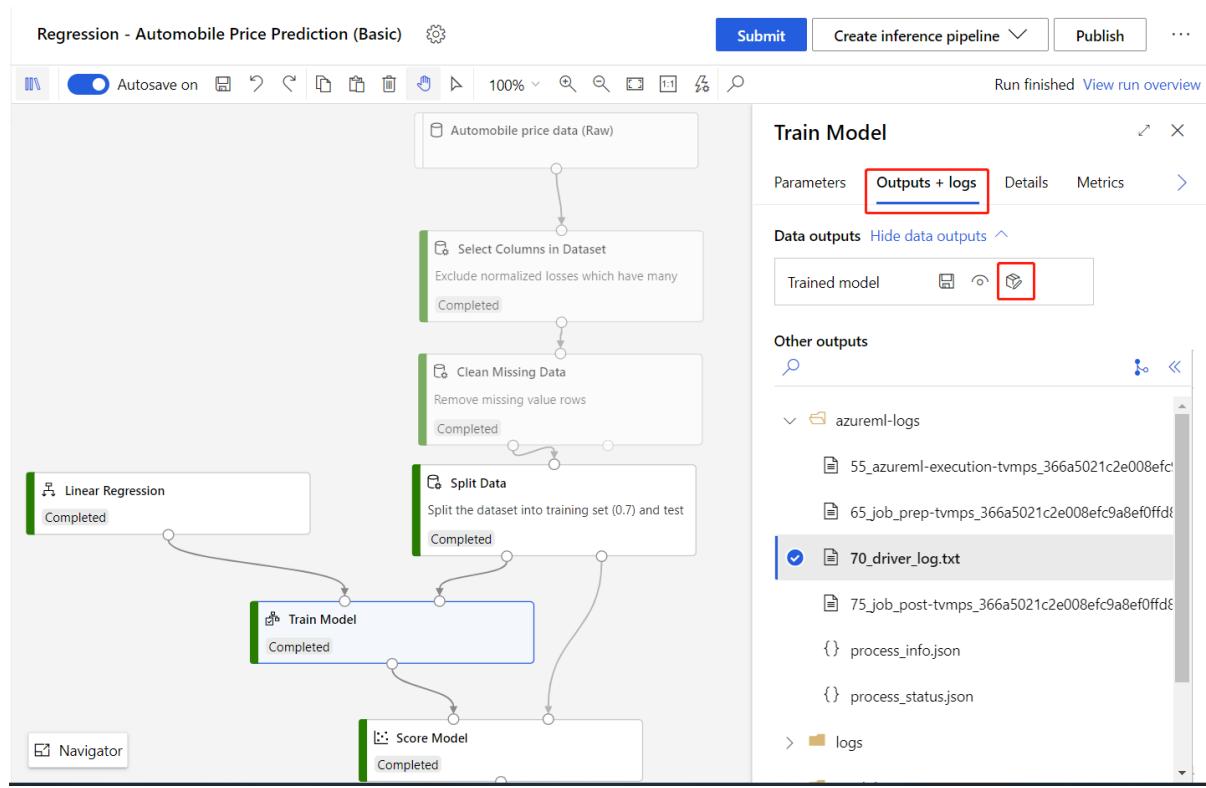
Prerequisites

- [An Azure Machine Learning workspace](#)
- A completed training pipeline containing one of following modules:
 - [Train Model module](#)
 - [Train Anomaly Detection Model module](#)
 - [Train Clustering Model module](#)
 - [Train Pytorch Model module](#)
 - [Train SVD Recommender module](#)
 - [Train Vowpal Wabbit Model module](#)
 - [Train Wide & Deep Model module](#)

Register the model

After the training pipeline completes, register the trained model to your Azure Machine Learning workspace to access the model in other projects.

1. Select the [Train Model module](#).
2. Select the **Outputs + logs** tab in the right pane.
3. Select the **Register Model** icon 



4. Enter a name for your model, then select **Save**.

After registering your model, you can find it in the **Models** asset page in the studio.

Name	Version	Experiment	Run ID	Created on	Tags
PricePredictionModel	1	Sample1	84a9bbe3-d387-40b7-8eb7-8d...	Aug 26, 2020 2:21 PM	CreatedByAMLStudio: true
amlstudio-rename-sample-test	1	Sample1	dd215f48-e5ea-4c8e-9c6a-26e...	Aug 24, 2020 1:44 PM	CreatedByAMLStudio: true
amlstudio-rename-sample-1	3	Sample1	dd215f48-e5ea-4c8e-9c6a-26e...	Aug 24, 2020 1:43 PM	CreatedByAMLStudio: true
amlstudio-testdeployssample1	1	Sample1	dd215f48-e5ea-4c8e-9c6a-26e...	Aug 24, 2020 1:41 PM	CreatedByAMLStudio: true
amlstudio-rename-sample-1	2	Sample1	dd215f48-e5ea-4c8e-9c6a-26e...	Aug 14, 2020 7:42 AM	CreatedByAMLStudio: true

Download the entry script file and conda dependencies file

You need the following files to deploy a model in Azure Machine Learning studio:

- **Entry script file** - loads the trained model, processes input data from requests, does real-time inferences, and returns the result. The designer automatically generates a `score.py` entry script file when the **Train Model** module completes.
- **Conda dependencies file** - specifies which pip and conda packages your webservice depends on. The designer automatically creates a `conda_env.yaml` file when the **Train Model** module completes.

You can download these two files in the right pane of the **Train Model** module:

1. Select the **Train Model** module.
2. In the **Outputs + logs** tab, select the folder `trained_model_outputs`.
3. Download the `conda_env.yaml` file and `score.py` file.

The screenshot shows the Azure Machine Learning Studio interface with the following details:

- Title:** Regression - Automobile Price Prediction (Basic)
- Toolbar:** Includes Save, Undo, Redo, Delete, Copy, Paste, 100% zoom, and search functions.
- Run Status:** Run finished, View run overview
- Pipeline Diagram:** A flowchart of the data processing steps:
 - Automobile price data (CSV) → Select Columns in Data (Exclude normalized losses) → Clean Missing Data (Remove missing value rows) → Split Data (Split the dataset into training and test sets).
 - Linear Regression (Completed) feeds into Train Model (Completed).
 - Train Model (Completed) feeds into the final output step.
- Train Model Panel:** Shows the status of the Train Model step as Completed.
- Outputs + logs Tab:** Active tab, showing the following outputs:
 - Data outputs:** Trained model (with icons for download, copy, and view).
 - Other outputs:** conda_env.yaml (highlighted with a red box), data.learner, data.metadata, model_specyaml, score.py (highlighted with a red box), and Trained_model.
- Bottom Navigation:** Navigator, Help, and a gear icon.

Alternatively, you can download the files from the **Models** asset page after registering your model:

1. Navigate to the **Models** asset page.
 2. Select the model you want to deploy.
 3. Select the **Artifacts** tab.
 4. Select the `trained_model_outputs` folder.
 5. Download the `conda_env.yaml` file and `score.py` file.

```
1 import os
2 import json
3
4 from azureml.studio.core.io.model_directory import ModelDirectory
5 from pathlib import Path
6 from azureml.studio.modules.ml.score.score_generic_module.score_generic_module import ScoreModelModule
7 from azureml.designer.serving.dagengine.converter import create_dfd_from_dict
8 from collections import defaultdict
9 from azureml.designer.serving.dagengine.utils import decode_nan
10 from azureml.studio.common.datatable.data_table import DataTable
11
12
13 model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'trained_model_outputs')
14 schema_file_path = Path(model_path) / '_schema.json'
15 with open(schema_file_path) as fp:
16     schema_data = json.load(fp)
17
18
19 def init():
20     global model
21     model = ModelDirectory.load(model_path).model
22
23
24 def run(data):
25     data = json.loads(data)
26     input_entry = defaultdict(list)
27     for row in data:
```

NOTE

The `score.py` file provides nearly the same functionality as the **Score Model** modules. However, some modules like [Score SVD Recommender](#), [Score Wide and Deep Recommender](#), and [Score Vowpal Wabbit Model](#) have parameters for different scoring modes. You can also change those parameters in the entry script.

For more information on setting parameters in the `score.py` file, see the section, [Configure the entry script](#).

Deploy the model

After downloading the necessary files, you're ready to deploy the model.

1. In the **Models** asset page, select the registered model.
2. Select the **Deploy** button.
3. In the configuration menu, enter the following information:
 - Input a name for the endpoint.
 - Select to deploy the model to [Azure Kubernetes Service](#) or [Azure Container Instance](#).
 - Upload the `score.py` for the **Entry script file**.
 - Upload the `conda_env.yml` for the **Conda dependencies file**.

TIP

In **Advanced** setting, you can set CPU/Memory capacity and other parameters for deployment. These settings are important for certain models such as PyTorch models, which consume considerable amount of memory (about 4 GB).

4. Select **Deploy** to deploy your model as a real-time endpoint.

Deploy a model

X

Name * ⓘ



*

Description ⓘ

Compute type * ⓘ

*

Models: PricePredictionModel:1

Enable authentication



Type



Entry script file * ⓘ

Browse

*

Conda dependencies file * ⓘ

Browse

*

Dependencies

Deploy

Cancel

Consume the real-time endpoint

After deployment succeeds, you can find the real-time endpoint in the **Endpoints** asset page. Once there, you will find a REST endpoint, which clients can use to submit requests to the real-time endpoint.

NOTE

The designer also generates a sample data json file for testing, you can download `_samples.json` in the `trained_model_outputs` folder.

Use the following code sample to consume a real-time endpoint.

```
import json
from pathlib import Path
from azureml.core.workspace import Workspace, Webservice

service_name = 'YOUR_SERVICE_NAME'
ws = Workspace.get(
    name='WORKSPACE_NAME',
    subscription_id='SUBSCRIPTION_ID',
    resource_group='RESOURCEGROUP_NAME'
)
service = Webservice(ws, service_name)
sample_file_path = '_samples.json'

with open(sample_file_path, 'r') as f:
    sample_data = json.load(f)
score_result = service.run(json.dumps(sample_data))
print(f'Inference result = {score_result}')
```

Consume computer vision related real-time endpoints

When consuming computer vision related real-time endpoints, you need to convert images to bytes, since web service only accepts string as input. Following is the sample code:

```

import base64
import json
from copy import deepcopy
from pathlib import Path
from azureml.studio.core.io.image_directory import (IMG_EXTS, image_from_file, image_to_bytes)
from azureml.studio.core.io.transformation_directory import ImageTransformationDirectory

# image path
image_path = Path('YOUR_IMAGE_FILE_PATH')

# provide the same parameter setting as in the training pipeline. Just an example here.
image_transform = [
    # format: (op, args). {} means using default parameter values of torchvision.transforms.
    # See https://pytorch.org/docs/stable/torchvision/transforms.html
    ('Resize', 256),
    ('CenterCrop', 224),
    # ('Pad', 0),
    # ('ColorJitter', {}),
    # ('Grayscale', {}),
    # ('RandomResizedCrop', 256),
    # ('RandomCrop', 224),
    # ('RandomHorizontalFlip', {}),
    # ('RandomVerticalFlip', {}),
    # ('RandomRotation', 0),
    # ('RandomAffine', 0),
    # ('RandomGrayscale', {}),
    # ('RandomPerspective', {}),
]
transform = ImageTransformationDirectory.create(transforms=image_transform).torch_transform

# download _samples.json file under Outputs+logs tab in the right pane of Train Pytorch Model module
sample_file_path = '_samples.json'
with open(sample_file_path, 'r') as f:
    sample_data = json.load(f)

# use first sample item as the default value
default_data = sample_data[0]
data_list = []
for p in image_path.iterdir():
    if p.suffix.lower() in IMG_EXTS:
        data = deepcopy(default_data)
        # convert image to bytes
        data['image'] = base64.b64encode(image_to_bytes(transform(image_from_file(p))).decode())
        data_list.append(data)

# use data.json as input of consuming the endpoint
data_file_path = 'data.json'
with open(data_file_path, 'w') as f:
    json.dump(data_list, f)

```

Configure the entry script

Some modules in the designer like [Score SVD Recommender](#), [Score Wide and Deep Recommender](#), and [Score Vowpal Wabbit Model](#) have parameters for different scoring modes.

In this section, you learn how to update these parameters in the entry script file too.

The following example updates the default behavior for a trained **Wide & Deep recommender** model. By default, the `score.py` file tells the web service to predict ratings between users and items.

You can modify the entry script file to make item recommendations, and return recommended items by changing the `recommender_prediction_kind` parameter.

```

import os
import json
from pathlib import Path
from collections import defaultdict
from azureml.studio.core.io.model_directory import ModelDirectory
from azureml.designer.modules.recommendation.dnn.wide_and_deep.score. \
    score_wide_and_deep_recommender import ScoreWideAndDeepRecommenderModule
from azureml.designer.serving.dagengine.utils import decode_nan
from azureml.designer.serving.dagengine.converter import create_dfd_from_dict

model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'), 'trained_model_outputs')
schema_file_path = Path(model_path) / '_schema.json'
with open(schema_file_path) as fp:
    schema_data = json.load(fp)

def init():
    global model
    model = ModelDirectory.load(load_from_dir=model_path)

def run(data):
    data = json.loads(data)
    input_entry = defaultdict(list)
    for row in data:
        for key, val in row.items():
            input_entry[key].append(decode_nan(val))

    data_frame_directory = create_dfd_from_dict(input_entry, schema_data)

    # The parameter names can be inferred from Score Wide and Deep Recommender module parameters:
    # convert the letters to lower cases and replace whitespaces to underscores.
    score_params = dict(
        trained_wide_and_deep_recommendation_model=model,
        dataset_to_score=data_frame_directory,
        training_data=None,
        user_features=None,
        item_features=None,
        ##### Note #####
        # Set 'Recommender prediction kind' parameter to enable item recommendation model
        recommender_prediction_kind='Item Recommendation',
        recommended_item_selection='From All Items',
        maximum_number_of_items_to_recommend_to_a_user=5,
        whether_to_return_the_predicted_ratings_of_the_items_along_with_the_labels='True')
    result_dfd, = ScoreWideAndDeepRecommenderModule().run(**score_params)
    result_df = result_dfd.data
    return json.dumps(result_df.to_dict("list"))

```

For **Wide & Deep recommender** and **Vowpal Wabbit** models, you can configure the scoring mode parameter using the following methods:

- The parameter names are the lowercase and underscore combinations of parameter names for [Score Vowpal Wabbit Model](#) and [Score Wide and Deep Recommender](#);
- Mode type parameter values are strings of the corresponding option names. Take **Recommender prediction kind** in the above codes as example, the value can be `'Rating Prediction'` or `'Item Recommendation'`. Other values are not allowed.

For **SVD recommender** trained model, the parameter names and values maybe less obvious, and you can look up the tables below to decide how to set parameters.

PARAMETER NAME IN SCORE SVD RECOMMENDER	PARAMETER NAME IN THE ENTRY SCRIPT FILE
Recommender prediction kind	prediction_kind
Recommended item selection	recommended_item_selection
Minimum size of the recommendation pool for a single user	min_recommendation_pool_size
Maximum number of items to recommend to a user	max_recommended_item_count
Whether to return the predicted ratings of the items along with the labels	return_ratings

The following code shows you how to set parameters for an SVD recommender, which uses all six parameters to recommend rated items with predicted ratings attached.

```
score_params = dict(
    learner=model,
    test_data=DataTable.from_dfd(data_frame_directory),
    training_data=None,
    # RecommenderPredictionKind has 2 members, 'RatingPrediction' and 'ItemRecommendation'. You
    # can specify prediction_kind parameter with one of them.
    prediction_kind=RecommenderPredictionKind.ItemRecommendation,
    # RecommendedItemSelection has 3 members, 'FromAllItems', 'FromRatedItems', 'FromUndatedItems'.
    # You can specify recommended_item_selection parameter with one of them.
    recommended_item_selection=RecommendedItemSelection.FromRatedItems,
    min_recommendation_pool_size=1,
    max_recommended_item_count=3,
    return_ratings=True,
)
```

Next steps

- [Train a model in the designer](#)
- [Deploy models with Azure Machine Learning SDK](#)
- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Create client applications to consume web services](#)
- [Update web service](#)

(Preview) No-code model deployment

12/23/2020 • 2 minutes to read • [Edit Online](#)

No-code model deployment is currently in preview and supports the following machine learning frameworks:

TensorFlow SavedModel format

TensorFlow models need to be registered in **SavedModel format** to work with no-code model deployment.

See [this link](#) for information on how to create a SavedModel.

We support any TensorFlow version that is listed under "Tags" at the [TensorFlow Serving DockerHub](#).

```
from azureml.core import Model

model = Model.register(workspace=ws,
                      model_name='flowers',           # Name of the registered model in your
workspace.                                workspace.
                      model_path='./flowers_model',   # Local Tensorflow SavedModel folder to
upload and register as a model.
                      model_framework=Model.Framework.TENSORFLOW, # Framework used to create the model.
                      model_framework_version='1.14.0',          # Version of Tensorflow used to create the
model.
                      description='Flowers model')

service_name = 'tensorflow-flower-service'
service = Model.deploy(ws, service_name, [model])
```

ONNX models

ONNX model registration and deployment is supported for any ONNX inference graph. Preprocess and postprocess steps are not currently supported.

Here is an example of how to register and deploy an MNIST ONNX model:

```
from azureml.core import Model

model = Model.register(workspace=ws,
                      model_name='mnist-sample',           # Name of the registered model in your
workspace.                                workspace.
                      model_path='mnist-model.onnx',        # Local ONNX model to upload and register
as a model.                                as a model.
                      model_framework=Model.Framework.ONNX , # Framework used to create the model.
                      model_framework_version='1.3',          # Version of ONNX used to create the model.
                      description='Onnx MNIST model')

service_name = 'onnx-mnist-service'
service = Model.deploy(ws, service_name, [model])
```

To score a model, see [Consume an Azure Machine Learning model deployed as a web service](#). Many ONNX projects use protobuf files to compactly store training and validation data, which can make it difficult to know what the data format expected by the service. As a model developer, you should document for your developers:

- Input format (JSON or binary)
- Input data shape and type (for example, an array of floats of shape [100,100,3])

- Domain information (for instance, for an image, the color space, component order, and whether the values are normalized)

If you're using Pytorch, [Exporting models from PyTorch to ONNX](#) has the details on conversion and limitations.

Scikit-learn models

No code model deployment is supported for all built-in scikit-learn model types.

Here is an example of how to register and deploy a sklearn model with no extra code:

```
from azureml.core import Model
from azureml.core.resource_configuration import ResourceConfiguration

model = Model.register(workspace=ws,
                      model_name='my-sklearn-model',           # Name of the registered model in your
workspace.                                         model_path='./sklearn_regression_model.pkl',   # Local file to upload and register as a
model.                                              model_framework=Model.Framework.SCIKITLEARN,    # Framework used to create the model.
model_framework_version='0.19.1',                  # Version of scikit-learn used to create
the model.                                         resource_configuration=ResourceConfiguration(cpu=1, memory_in_gb=0.5),
description='Ridge regression model to predict diabetes progression.', tags={'area': 'diabetes', 'type': 'regression'})'

service_name = 'my-sklearn-service'
service = Model.deploy(ws, service_name, [model])
```

NOTE

Models that support predict_proba will use that method by default. To override this to use predict you can modify the POST body as below:

```
import json

input_payload = json.dumps({
    'data': [
        [ 0.03807591,  0.05068012,  0.06169621,  0.02187235, -0.0442235,
        -0.03482076, -0.04340085, -0.00259226,  0.01990842, -0.01764613]
    ],
    'method': 'predict' # If you have a classification model, the default behavior is to run 'predict_proba'.
})

output = service.run(input_payload)

print(output)
```

NOTE

These dependencies are included in the prebuilt scikit-learn inference container:

```
- dll  
- azureml-defaults  
- inference-schema[numpy-support]  
- scikit-learn  
- numpy  
- joblib  
- pandas  
- scipy  
- sklearn_pandas
```

Next steps

- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Create client applications to consume web services](#)
- [Update web service](#)
- [How to deploy a model using a custom Docker image](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Create event alerts and triggers for model deployments](#)

How to package a registered model with Docker

12/23/2020 • 4 minutes to read • [Edit Online](#)

This article shows how to package a registered Azure Machine Learning model with Docker.

Prerequisites

This article assumes you have already trained and registered a model in your machine learning workspace. To learn how to train and register a scikit-learn model, [follow this tutorial](#).

Package models

In some cases, you might want to create a Docker image without deploying the model (if, for example, you plan to [deploy to Azure App Service](#)). Or you might want to download the image and run it on a local Docker installation. You might even want to download the files used to build the image, inspect them, modify them, and build the image manually.

Model packaging enables you to do these things. It packages all the assets needed to host a model as a web service and allows you to download either a fully built Docker image or the files needed to build one. There are two ways to use model packaging:

Download a packaged model: Download a Docker image that contains the model and other files needed to host it as a web service.

Generate a Dockerfile: Download the Dockerfile, model, entry script, and other assets needed to build a Docker image. You can then inspect the files or make changes before you build the image locally.

Both packages can be used to get a local Docker image.

TIP

Creating a package is similar to deploying a model. You use a registered model and an inference configuration.

IMPORTANT

To download a fully built image or build an image locally, you need to have [Docker](#) installed in your development environment.

Download a packaged model

The following example builds an image, which is registered in the Azure container registry for your workspace:

```
package = Model.package(ws, [model], inference_config)
package.wait_for_creation(show_output=True)
```

After you create a package, you can use `package.pull()` to pull the image to your local Docker environment. The output of this command will display the name of the image. For example:

```
Status: Downloaded newer image for myworkspacef78fd10.azurecr.io/package:20190822181338 .
```

After you download the model, use the `docker images` command to list the local images:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myworkspacef78fd10.azurecr.io/package	20190822181338	7ff48015d5bd	4 minutes ago	1.43 GB

To start a local container based on this image, use the following command to start a named container from the shell or command line. Replace the `<imageid>` value with the image ID returned by the `docker images` command.

```
docker run -p 6789:5001 --name mycontainer <imageid>
```

This command starts the latest version of the image named `myimage`. It maps local port 6789 to the port in the container on which the web service is listening (5001). It also assigns the name `mycontainer` to the container, which makes the container easier to stop. After the container is started, you can submit requests to `http://localhost:6789/score`.

Generate a Dockerfile and dependencies

The following example shows how to download the Dockerfile, model, and other assets needed to build an image locally. The `generate_dockerfile=True` parameter indicates that you want the files, not a fully built image.

```
package = Model.package(ws, [model], inference_config, generate_dockerfile=True)
package.wait_for_creation(show_output=True)
# Download the package.
package.save("./imagefiles")
# Get the Azure container registry that the model/Dockerfile uses.
acr=package.get_container_registry()
print("Address:", acr.address)
print("Username:", acr.username)
print("Password:", acr.password)
```

This code downloads the files needed to build the image to the `imagefiles` directory. The Dockerfile included in the saved files references a base image stored in an Azure container registry. When you build the image on your local Docker installation, you need to use the address, user name, and password to authenticate to the registry. Use the following steps to build the image by using a local Docker installation:

- From a shell or command-line session, use the following command to authenticate Docker with the Azure container registry. Replace `<address>`, `<username>`, and `<password>` with the values retrieved by `package.get_container_registry()`.

```
docker login <address> -u <username> -p <password>
```

- To build the image, use the following command. Replace `<imagefiles>` with the path of the directory where `package.save()` saved the files.

```
docker build --tag myimage <imagefiles>
```

This command sets the image name to `myimage`.

To verify that the image is built, use the `docker images` command. You should see the `myimage` image in the list:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<none>	<none>	2d5ee0bf3b3b	49 seconds ago	1.43 GB
myimage	latest	739f22498d64	3 minutes ago	1.43 GB

To start a new container based on this image, use the following command:

```
docker run -p 6789:5001 --name mycontainer myimage:latest
```

This command starts the latest version of the image named `myimage`. It maps local port 6789 to the port in the container on which the web service is listening (5001). It also assigns the name `mycontainer` to the container, which makes the container easier to stop. After the container is started, you can submit requests to <http://localhost:6789/score>.

Example client to test the local container

The following code is an example of a Python client that can be used with the container:

```
import requests
import json

# URL for the web service.
scoring_uri = 'http://localhost:6789/score'

# Two sets of data to score, so we get two results back.
data = {"data":
    [
        [ 1,2,3,4,5,6,7,8,9,10 ],
        [ 10,9,8,7,6,5,4,3,2,1 ]
    ]
}
# Convert to JSON string.
input_data = json.dumps(data)

# Set the content type.
headers = {'Content-Type': 'application/json'}

# Make the request and display the response.
resp = requests.post(scoring_uri, input_data, headers=headers)
print(resp.text)
```

For example clients in other programming languages, see [Consume models deployed as web services](#).

Stop the Docker container

To stop the container, use the following command from a different shell or command line:

```
docker kill mycontainer
```

Next steps

- [Troubleshoot a failed deployment](#)
- [Deploy to Azure Kubernetes Service](#)
- [Create client applications to consume web services](#)
- [Update web service](#)
- [How to deploy a model using a custom Docker image](#)
- [Use TLS to secure a web service through Azure Machine Learning](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Create event alerts and triggers for model deployments](#)

Create and run machine learning pipelines with Azure Machine Learning SDK

12/23/2020 • 13 minutes to read • [Edit Online](#)

In this article, you learn how to create and run [machine learning pipelines](#) by using the [Azure Machine Learning SDK](#). Use **ML pipelines** to create a workflow that stitches together various ML phases. Then, publish that pipeline for later access or sharing with others. Track ML pipelines to see how your model is performing in the real world and to detect data drift. ML pipelines are ideal for batch scoring scenarios, using various computes, reusing steps instead of rerunning them, as well as sharing ML workflows with others.

This article is not a tutorial. For guidance on creating your first pipeline, see [Tutorial: Build an Azure Machine Learning pipeline for batch scoring](#) or [Use automated ML in an Azure Machine Learning pipeline in Python](#).

While you can use a different kind of pipeline called an [Azure Pipeline](#) for CI/CD automation of ML tasks, that type of pipeline is not stored in your workspace. [Compare these different pipelines](#).

The ML pipelines you create are visible to the members of your Azure Machine Learning [workspace](#).

ML pipelines execute on compute targets (see [What are compute targets in Azure Machine Learning](#)). Pipelines can read and write data to and from supported [Azure Storage](#) locations.

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](#).

Prerequisites

- Create an [Azure Machine Learning workspace](#) to hold all your pipeline resources.
- [Configure your development environment](#) to install the Azure Machine Learning SDK, or use an [Azure Machine Learning compute instance](#) with the SDK already installed.

Start by attaching your workspace:

```
import azureml.core
from azureml.core import Workspace, Datastore

ws = Workspace.from_config()
```

Set up machine learning resources

Create the resources required to run an ML pipeline:

- Set up a datastore used to access the data needed in the pipeline steps.
- Configure a `Dataset` object to point to persistent data that lives in, or is accessible in, a datastore. Configure a `PipelineData` object for temporary data passed between pipeline steps.

TIP

An improved experience for passing temporary data between pipeline steps is available in the public preview class, `OutputFileDatasetConfig`. This class is an [experimental](#) preview feature, and may change at any time.

- Set up the [compute targets](#) on which your pipeline steps will run.

Set up a datastore

A datastore stores the data for the pipeline to access. Each workspace has a default datastore. You can register additional datastores.

When you create your workspace, [Azure Files](#) and [Azure Blob storage](#) are attached to the workspace. A default datastore is registered to connect to the Azure Blob storage. To learn more, see [Deciding when to use Azure Files, Azure Blobs, or Azure Disks](#).

```
# Default datastore
def_data_store = ws.get_default_datastore()

# Get the blob storage associated with the workspace
def_blob_store = Datastore(ws, "workspaceblobstore")

# Get file storage associated with the workspace
def_file_store = Datastore(ws, "workspacefilestore")
```

Steps generally consume data and produce output data. A step can create data such as a model, a directory with model and dependent files, or temporary data. This data is then available for other steps later in the pipeline. To learn more about connecting your pipeline to your data, see the articles [How to Access Data](#) and [How to Register Datasets](#).

Configure data with [Dataset](#) and [PipelineData](#) objects

The preferred way to provide data to a pipeline is a [Dataset](#) object. The [Dataset](#) object points to data that lives in or is accessible from a datastore or at a Web URL. The [Dataset](#) class is abstract, so you will create an instance of either a [FileDataset](#) (referring to one or more files) or a [TabularDataset](#) that's created by from one or more files with delimited columns of data.

You create a [Dataset](#) using methods like [from_files](#) or [from_delimited_files](#).

```
from azureml.core import Dataset

my_dataset = Dataset.File.from_files([(def_blob_store, 'train-images/')])
```

Intermediate data (or output of a step) is represented by a [PipelineData](#) object. [output_data1](#) is produced as the output of a step, and used as the input of one or more future steps. [PipelineData](#) introduces a data dependency between steps, and creates an implicit execution order in the pipeline. This object will be used later when creating pipeline steps.

```
from azureml.pipeline.core import PipelineData

output_data1 = PipelineData(
    "output_data1",
    datastore=def_blob_store,
    output_name="output_data1")
```

TIP

Persisting intermediate data between pipeline steps is also possible with the public preview class, [OutputFileDatasetConfig](#). For a code example using the [OutputFileDatasetConfig](#) class, see how to [build a two step ML pipeline](#).

TIP

Only upload files relevant to the job at hand. Any change in files within the data directory will be seen as reason to rerun the step the next time the pipeline is run even if reuse is specified.

Set up a compute target

In Azure Machine Learning, the term **compute** (or **compute target**) refers to the machines or clusters that perform the computational steps in your machine learning pipeline. See [compute targets for model training](#) for a full list of compute targets and [Create compute targets](#) for how to create and attach them to your workspace. The process for creating and or attaching a compute target is the same whether you are training a model or running a pipeline step. After you create and attach your compute target, use the `ComputeTarget` object in your [pipeline step](#).

IMPORTANT

Performing management operations on compute targets is not supported from inside remote jobs. Since machine learning pipelines are submitted as a remote job, do not use management operations on compute targets from inside the pipeline.

Azure Machine Learning compute

You can create an Azure Machine Learning compute for running your steps. The code for other compute targets is very similar, with slightly different parameters, depending on the type.

```
from azureml.core.compute import ComputeTarget, AmlCompute

compute_name = "aml-compute"
vm_size = "STANDARD_NC6"
if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('Found compute target: ' + compute_name)
else:
    print('Creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(vm_size=vm_size, # STANDARD_NC6 is GPU-enabled
                                                               min_nodes=0,
                                                               max_nodes=4)

    # create the compute target
    compute_target = ComputeTarget.create(
        ws, compute_name, provisioning_config)

    # Can poll for a minimum number of nodes and for a specific timeout.
    # If no min node count is provided it will use the scale settings for the cluster
    compute_target.wait_for_completion(
        show_output=True, min_node_count=None, timeout_in_minutes=20)

    # For a more detailed view of current cluster status, use the 'status' property
print(compute_target.status.serialize())
```

Configure the training run's environment

The next step is making sure that the remote training run has all the dependencies needed by the training steps. Dependencies and the runtime context are set by creating and configuring a `RunConfiguration` object.

```

from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core import Environment

aml_run_config = RunConfiguration()
# `compute_target` as defined in "Azure Machine Learning compute" section above
aml_run_config.target = compute_target

USE_CURATED_ENV = True
if USE_CURATED_ENV :
    curated_environment = Environment.get(workspace=ws, name="AzureML-Tutorial")
    aml_run_config.environment = curated_environment
else:
    aml_run_config.environment.python.user_managed_dependencies = False

# Add some packages relied on by data prep step
aml_run_config.environment.python.conda_dependencies = CondaDependencies.create(
    conda_packages=['pandas','scikit-learn'],
    pip_packages=['azureml-sdk', 'azureml-dataprep[fuse,pandas]'],
    pin_sdk_version=False)

```

The code above shows two options for handling dependencies. As presented, with `USE_CURATED_ENV = True`, the configuration is based on a curated environment. Curated environments are "prebaked" with common inter-dependent libraries and can be significantly faster to bring online. Curated environments have prebuilt Docker images in the [Microsoft Container Registry](#). For more information, see [Azure Machine Learning curated environments](#).

The path taken if you change `USE_CURATED_ENV` to `False` shows the pattern for explicitly setting your dependencies. In that scenario, a new custom Docker image will be created and registered in an Azure Container Registry within your resource group (see [Introduction to private Docker container registries in Azure](#)). Building and registering this image can take quite a few minutes.

Construct your pipeline steps

Once you have the compute resource and environment created, you are ready to define your pipeline's steps. There are many built-in steps available via the Azure Machine Learning SDK, as you can see on the [reference documentation for the `azureml.pipeline.steps` package](#). The most flexible class is `PythonScriptStep`, which runs a Python script.

```

from azureml.pipeline.steps import PythonScriptStep
dataprep_source_dir = "./dataprep_src"
entry_point = "prepare.py"
# `my_dataset` as defined above
ds_input = my_dataset.as_named_input('input1')

# `output_data1`, `compute_target`, `aml_run_config` as defined above
data_prep_step = PythonScriptStep(
    script_name=entry_point,
    source_directory=dataprep_source_dir,
    arguments=["--input", ds_input.as_download(), "--output", output_data1],
    inputs=[ds_input],
    outputs=[output_data1],
    compute_target=compute_target,
    runconfig=aml_run_config,
    allow_reuse=True
)

```

The above code shows a typical initial pipeline step. Your data preparation code is in a subdirectory (in this example, `"prepare.py"` in the directory `"./dataprep_src"`). As part of the pipeline creation process, this directory is zipped

and uploaded to the `compute_target` and the step runs the script specified as the value for `script_name`.

The `arguments`, `inputs`, and `outputs` values specify the inputs and outputs of the step. In the example above, the baseline data is the `my_dataset` dataset. The corresponding data will be downloaded to the compute resource since the code specifies it as `as_download()`. The script `prepare.py` does whatever data-transformation tasks are appropriate to the task at hand and outputs the data to `output_data1`, of type `PipelineData`. For more information, see [Moving data into and between ML pipeline steps \(Python\)](#).

The step will run on the machine defined by `compute_target`, using the configuration `aml_run_config`.

Reuse of previous results (`allow_reuse`) is key when using pipelines in a collaborative environment since eliminating unnecessary reruns offers agility. Reuse is the default behavior when the `script_name`, `inputs`, and the parameters of a step remain the same. When reuse is allowed, results from the previous run are immediately sent to the next step. If `allow_reuse` is set to `False`, a new run will always be generated for this step during pipeline execution.

It's possible to create a pipeline with a single step, but almost always you'll choose to split your overall process into several steps. For instance, you might have steps for data preparation, training, model comparison, and deployment. For instance, one might imagine that after the `data_prep_step` specified above, the next step might be training:

```
train_source_dir = "./train_src"
train_entry_point = "train.py"

training_results = PipelineData(name = "training_results",
                                datastore=def_blob_store,
                                output_name="training_results")

train_step = PythonScriptStep(
    script_name=train_entry_point,
    source_directory=train_source_dir,
    arguments=["--prepped_data", output_data1.as_input(), "--training_results", training_results],
    compute_target=compute_target,
    runconfig=aml_run_config,
    allow_reuse=True
)
```

The above code is very similar to that for the data preparation step. The training code is in a directory separate from that of the data preparation code. The `PipelineData` output of the data preparation step, `output_data1` is used as the *input* to the training step. A new `PipelineData` object, `training_results` is created to hold the results for a subsequent comparison or deployment step.

TIP

For an improved experience, and the ability to write intermediate data back to your datastores at the end of your pipeline run, use the public preview class, `OutputFileDatasetConfig`. For code examples, see how to [build a two step ML pipeline](#) and [how to write data back to datastores upon run completion](#).

After you define your steps, you build the pipeline by using some or all of those steps.

NOTE

No file or data is uploaded to Azure Machine Learning when you define the steps or build the pipeline. The files are uploaded when you call `Experiment.submit()`.

```

# list of steps to run (`compare_step` definition not shown)
compare_models = [data_prep_step, train_step, compare_step]

from azureml.pipeline.core import Pipeline

# Build the pipeline
pipeline1 = Pipeline(workspace=ws, steps=[compare_models])

```

How Python environments work with pipeline parameters

As discussed previously in [Configure the training run's environment](#), environment state and Python library dependencies are specified using an `Environment` object. Generally, you can specify an existing `Environment` by referring to its name and, optionally, a version:

```

aml_run_config = RunConfiguration()
aml_run_config.environment.name = 'MyEnvironment'
aml_run_config.environment.version = '1.0'

```

However, if you choose to use `PipelineParameter` objects to dynamically set variables at runtime for your pipeline steps, you cannot use this technique of referring to an existing `Environment`. Instead, if you want to use `PipelineParameter` objects, you must set the `environment` field of the `RunConfiguration` to an `Environment` object. It is your responsibility to ensure that such an `Environment` has its dependencies on external Python packages properly set.

Use a dataset

Datasets created from Azure Blob storage, Azure Files, Azure Data Lake Storage Gen1, Azure Data Lake Storage Gen2, Azure SQL Database, and Azure Database for PostgreSQL can be used as input to any pipeline step. You can write output to a [DataTransferStep](#), [DatabricksStep](#), or if you want to write data to a specific datastore use [PipelineData](#).

IMPORTANT

Writing output data back to a datastore using `PipelineData` is only supported for Azure Blob and Azure File share datastores.

To write output data back to Azure Blob, Azure File share, ADLS Gen 1 and ADLS Gen 2 datastores use the public preview class, [`OutputFileDatasetConfig`](#).

```

dataset_consumming_step = PythonScriptStep(
    script_name="iris_train.py",
    inputs=[iris_tabular_dataset.as_named_input("iris_data")],
    compute_target=compute_target,
    source_directory=project_folder
)

```

You then retrieve the dataset in your pipeline by using the `Run.input_datasets` dictionary.

```

# iris_train.py
from azureml.core import Run, Dataset

run_context = Run.get_context()
iris_dataset = run_context.input_datasets['iris_data']
dataframe = iris_dataset.to_pandas_dataframe()

```

The line `Run.get_context()` is worth highlighting. This function retrieves a `Run` representing the current experimental run. In the above sample, we use it to retrieve a registered dataset. Another common use of the `Run`

object is to retrieve both the experiment itself and the workspace in which the experiment resides:

```
# Within a PythonScriptStep  
  
ws = Run.get_context().experiment.workspace
```

For more detail, including alternate ways to pass and access data, see [Moving data into and between ML pipeline steps \(Python\)](#).

Caching & reuse

In order to optimize and customize the behavior of your pipelines, you can do a few things around caching and reuse. For example, you can choose to:

- **Turn off the default reuse of the step run output** by setting `allow_reuse=False` during [step definition](#). Reuse is key when using pipelines in a collaborative environment since eliminating unnecessary runs offers agility. However, you can opt out of reuse.
- **Force output regeneration for all steps in a run** with

```
pipeline_run = exp.submit(pipeline, regenerate_outputs=False)
```

By default, `allow_reuse` for steps is enabled and the `source_directory` specified in the step definition is hashed. So, if the script for a given step remains the same (`script_name`, inputs, and the parameters), and nothing else in the `source_directory` has changed, the output of a previous step run is reused, the job is not submitted to the compute, and the results from the previous run are immediately available to the next step instead.

```
step = PythonScriptStep(name="Hello World",  
                      script_name="hello_world.py",  
                      compute_target=aml_compute,  
                      source_directory=source_directory,  
                      allow_reuse=False,  
                      hash_paths=['hello_world.ipynb'])
```

Submit the pipeline

When you submit the pipeline, Azure Machine Learning checks the dependencies for each step and uploads a snapshot of the source directory you specified. If no source directory is specified, the current local directory is uploaded. The snapshot is also stored as part of the experiment in your workspace.

IMPORTANT

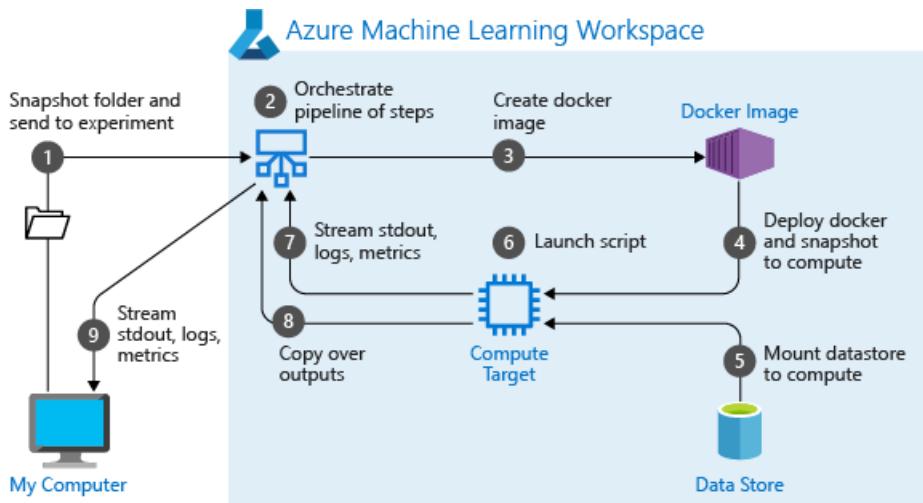
To prevent unnecessary files from being included in the snapshot, make an ignore file (`.gitignore` or `.amlinignore`) in the directory. Add the files and directories to exclude to this file. For more information on the syntax to use inside this file, see [syntax and patterns](#) for `.gitignore`. The `.amlinignore` file uses the same syntax. *If both files exist, the `.amlinignore` file takes precedence.*

For more information, see [Schemas](#).

```
from azureml.core import Experiment  
  
# Submit the pipeline to be run  
pipeline_run1 = Experiment(ws, 'Compare_Models_Exp').submit(pipeline1)  
pipeline_run1.wait_for_completion()
```

When you first run a pipeline, Azure Machine Learning:

- Downloads the project snapshot to the compute target from the Blob storage associated with the workspace.
- Builds a Docker image corresponding to each step in the pipeline.
- Downloads the Docker image for each step to the compute target from the container registry.
- Configures access to `Dataset` and `PipelineData` objects. For `as_mount()` access mode, FUSE is used to provide virtual access. If mount is not supported or if the user specified access as `as_download()`, the data is instead copied to the compute target.
- Runs the step in the compute target specified in the step definition.
- Creates artifacts, such as logs, stdout and stderr, metrics, and output specified by the step. These artifacts are then uploaded and kept in the user's default datastore.



For more information, see the [Experiment class](#) reference.

Use pipeline parameters for arguments that change at inference time

View results of a pipeline

See the list of all your pipelines and their run details in the studio:

1. Sign in to [Azure Machine Learning studio](#).
2. [View your workspace](#).
3. On the left, select **Pipelines** to see all your pipeline runs.

The screenshot shows the Azure Machine Learning studio interface. The left sidebar has sections for Home, Notebooks, Automated ML, Designer, Assets, Datasets, Experiments, and Pipelines. The Pipelines section is highlighted with a red box. The main area is titled 'Pipelines' and shows 'Pipeline runs'. Another red box highlights the 'Pipeline runs' tab. Below it is a table with columns: Run, Created time, Duration, Status, Description, Experiment, Submi..., and Tags. Two rows are listed: 'Run 1' (Created 11/08/2019, 5:27:22 PM, Duration 1m 12s, Status Finished, Description Estimator_sample, Experiment Estimator_sample, Tags +1) and 'Run 1' (Created 11/08/2019, 5:02:49 PM, Duration 10m 49s, Status Finished, Description Hello_World1, Experiment Hello_World1, Tags +1). A search bar at the top right says 'Search to filter items'.

Run	Created time	Duration	Status	Description	Experiment	Submi...	Tags
Run 1	11/08/2019, 5:27:22 PM	1m 12s	Finished	Estimator_sample	Estimator_sample	+1	
Run 1	11/08/2019, 5:02:49 PM	10m 49s	Finished	Hello_World1	Hello_World1	+1	

4. Select a specific pipeline to see the run results.

Git tracking and integration

When you start a training run where the source directory is a local Git repository, information about the repository is stored in the run history. For more information, see [Git integration for Azure Machine Learning](#).

Next steps

- To share your pipeline with colleagues or customers, see [Publish machine learning pipelines](#)
- Use [these Jupyter notebooks on GitHub](#) to explore machine learning pipelines further
- See the SDK reference help for the `azureml-pipelines-core` package and the `azureml-pipelines-steps` package
- See the [how-to](#) for tips on debugging and troubleshooting pipelines=
- Learn how to run notebooks by following the article [Use Jupyter notebooks to explore this service](#).

Moving data into and between ML pipeline steps (Python)

12/23/2020 • 8 minutes to read • [Edit Online](#)

This article provides code for importing, transforming, and moving data between steps in an Azure Machine Learning pipeline. For an overview of how data works in Azure Machine Learning, see [Access data in Azure storage services](#). For the benefits and structure of Azure Machine Learning pipelines, see [What are Azure Machine Learning pipelines?](#).

This article will show you how to:

- Use `Dataset` objects for pre-existing data
- Access data within your steps
- Split `Dataset` data into subsets, such as training and validation subsets
- Create `PipelineData` objects to transfer data to the next pipeline step
- Use `PipelineData` objects as input to pipeline steps
- Create new `Dataset` objects from `PipelineData` you wish to persist

TIP

An improved experience for passing temporary data between pipeline steps and persisting your data after pipeline runs is available in the public preview classes, `OutputFileDatasetConfig` and `OutputTabularDatasetConfig`. These classes are [experimental](#) preview features, and may change at any time.

Prerequisites

You'll need:

- 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](#), or access to [Azure Machine Learning studio](#).
- An Azure Machine Learning workspace.

Either [create an Azure Machine Learning workspace](#) or use an existing one via the Python SDK. Import the `Workspace` and `Datastore` class, and load your subscription information from the file `config.json` using the function `from_config()`. This function looks for the JSON file in the current directory by default, but you can also specify a path parameter to point to the file using `from_config(path="your/file/path")`.

```
import azureml.core
from azureml.core import Workspace, Datastore

ws = Workspace.from_config()
```

- Some pre-existing data. This article briefly shows the use of an [Azure blob container](#).
- Optional: An existing machine learning pipeline, such as the one described in [Create and run machine learning pipelines with Azure Machine Learning SDK](#).

Use `Dataset` objects for pre-existing data

The preferred way to ingest data into a pipeline is to use a `Dataset` object. `Dataset` objects represent persistent data available throughout a workspace.

There are many ways to create and register `Dataset` objects. Tabular datasets are for delimited data available in one or more files. File datasets are for binary data (such as images) or for data that you'll parse. The simplest programmatic ways to create `Dataset` objects are to use existing blobs in workspace storage or public URLs:

```
datastore = Datastore.get(workspace, 'training_data')
iris_dataset = Dataset.Tabular.from_delimited_files(DataPath(datastore, 'iris.csv'))

cats_dogs_dataset = Dataset.File.from_files(
    paths='https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-
6DEBA77B919F/kagglecatsanddogs_3367a.zip',
    archive_options=ArchiveOptions(archive_type=ArchiveType.ZIP, entry_glob='**/*.jpg')
)
```

For more options on creating datasets with different options and from different sources, registering them and reviewing them in the Azure Machine Learning UI, understanding how data size interacts with compute capacity, and versioning them, see [Create Azure Machine Learning datasets](#).

Pass datasets to your script

To pass the dataset's path to your script, use the `Dataset` object's `as_named_input()` method. You can either pass the resulting `DatasetConsumptionConfig` object to your script as an argument or, by using the `inputs` argument to your pipeline script, you can retrieve the dataset using `Run.get_context().input_datasets[]`.

Once you've created a named input, you can choose its access mode: `as_mount()` or `as_download()`. If your script processes all the files in your dataset and the disk on your compute resource is large enough for the dataset, the download access mode is the better choice. The download access mode will avoid the overhead of streaming the data at runtime. If your script accesses a subset of the dataset or it's too large for your compute, use the mount access mode. For more information, read [Mount vs. Download](#)

To pass a dataset to your pipeline step:

1. Use `TabularDataset.as_named_input()` or `FileDataset.as_named_input()` (no 's' at end) to create a `DatasetConsumptionConfig` object
2. Use `as_mount()` or `as_download()` to set the access mode
3. Pass the datasets to your pipeline steps using either the `arguments` or the `inputs` argument

The following snippet shows the common pattern of combining these steps within the `PythonScriptStep` constructor:

```
train_step = PythonScriptStep(
    name="train_data",
    script_name="train.py",
    compute_target=cluster,
    inputs=[iris_dataset.as_named_input('iris').as_mount()]
)
```

NOTE

You would need to replace the values for all these arguments (that is, "train_data", "train.py", cluster, and iris_dataset) with your own data. The above snippet just shows the form of the call and is not part of a Microsoft sample.

You can also use methods such as random_split() and take_sample() to create multiple inputs or reduce the amount of data passed to your pipeline step:

```
seed = 42 # PRNG seed
smaller_dataset = iris_dataset.take_sample(0.1, seed=seed) # 10%
train, test = smaller_dataset.random_split(percentage=0.8, seed=seed)

train_step = PythonScriptStep(
    name="train_data",
    script_name="train.py",
    compute_target=cluster,
    inputs=[train.as_named_input('train').as_download(), test.as_named_input('test').as_download()]
)
```

Access datasets within your script

Named inputs to your pipeline step script are available as a dictionary within the Run object. Retrieve the active Run object using Run.get_context() and then retrieve the dictionary of named inputs using input_datasets. If you passed the DatasetConsumptionConfig object using the arguments argument rather than the inputs argument, access the data using ArgParser code. Both techniques are demonstrated in the following snippet.

```
# In pipeline definition script:
# Code for demonstration only: It would be very confusing to split datasets between `arguments` and `inputs`
train_step = PythonScriptStep(
    name="train_data",
    script_name="train.py",
    compute_target=cluster,
    arguments=['--training-folder', train.as_named_input('train').as_download()]
    inputs=[test.as_named_input('test').as_download()]
)

# In pipeline script
parser = argparse.ArgumentParser()
parser.add_argument('--training-folder', type=str, dest='train_folder', help='training data folder mounting point')
args = parser.parse_args()
training_data_folder = args.train_folder

testing_data_folder = Run.get_context().input_datasets['test']
```

The passed value will be the path to the dataset file(s).

It's also possible to access a registered Dataset directly. Since registered datasets are persistent and shared across a workspace, you can retrieve them directly:

```
run = Run.get_context()
ws = run.experiment.workspace
ds = Dataset.get_by_name(workspace=ws, name='mnist_opendataset')
```

NOTE

The preceding snippets show the form of the calls and are not part of a Microsoft sample. You must replace the various arguments with values from your own project.

Use `PipelineData` for intermediate data

While `Dataset` objects represent persistent data, `PipelineData` objects are used for temporary data that is output from pipeline steps. Because the lifespan of a `PipelineData` object is longer than a single pipeline step, you define them in the pipeline definition script. When you create a `PipelineData` object, you must provide a name and a datastore at which the data will reside. Pass your `PipelineData` object(s) to your `PythonScriptStep` using *both* the `arguments` and the `outputs` arguments:

```
default_datastore = workspace.get_default_datastore()
dataprep_output = PipelineData("clean_data", datastore=default_datastore)

dataprep_step = PythonScriptStep(
    name="prep_data",
    script_name="dataprep.py",
    compute_target=cluster,
    arguments=["--output-path", dataprep_output]
    inputs=[Dataset.get_by_name(workspace, 'raw_data')],
    outputs=[dataprep_output]
)
```

You may choose to create your `PipelineData` object using an access mode that provides an immediate upload. In that case, when you create your `PipelineData`, set the `upload_mode` to `"upload"` and use the `output_path_on_compute` argument to specify the path to which you'll be writing the data:

```
PipelineData("clean_data", datastore=def_blob_store, output_mode="upload",
output_path_on_compute="clean_data_output/")
```

NOTE

The preceding snippets show the form of the calls and are not part of a Microsoft sample. You must replace the various arguments with values from your own project.

TIP

An improved experience for passing intermediate data between pipeline steps is available with the public preview class, `OutputFileDatasetConfig`. For a code example using `OutputFileDatasetConfig`, see how to [build a two step ML pipeline](#).

Use `PipelineData` as outputs of a training step

Within your pipeline's `PythonScriptStep`, you can retrieve the available output paths using the program's arguments. If this step is the first and will initialize the output data, you must create the directory at the specified path. You can then write whatever files you wish to be contained in the `PipelineData`.

```

parser = argparse.ArgumentParser()
parser.add_argument('--output_path', dest='output_path', required=True)
args = parser.parse_args()
# Make directory for file
os.makedirs(os.path.dirname(args.output_path), exist_ok=True)
with open(args.output_path, 'w') as f:
    f.write("Step 1's output")

```

If you created your `PipelineData` with the `is_directory` argument set to `True`, it would be enough to just perform the `os.makedirs()` call and then you would be free to write whatever files you wished to the path. For more details, see the [PipelineData](#) reference documentation.

Read `PipelineData` as inputs to non-initial steps

After the initial pipeline step writes some data to the `PipelineData` path and it becomes an output of that initial step, it can be used as an input to a later step:

```

step1_output_data = PipelineData("processed_data", datastore=def_blob_store, output_mode="upload")
# get adls gen 2 datastore already registered with the workspace
datastore = workspace.datastores['my_adlsgen2']

step1 = PythonScriptStep(
    name="generate_data",
    script_name="step1.py",
    runconfig = aml_run_config,
    arguments = ["--output_path", step1_output_data],
    inputs=[],
    outputs=[step1_output_data]
)

step2 = PythonScriptStep(
    name="read_pipeline_data",
    script_name="step2.py",
    compute_target=compute,
    runconfig = aml_run_config,
    arguments = ["--pd", step1_output_data],
    inputs=[step1_output_data]
)
pipeline = Pipeline(workspace=ws, steps=[step1, step2])

```

The value of a `PipelineData` input is the path to the previous output.

NOTE

The preceding snippets show the form of the calls and are not part of a Microsoft sample. You must replace the various arguments with values from your own project.

TIP

An improved experience for passing intermediate data between pipeline steps is available with the public preview class, `OutputFileDatasetConfig`. For a code example using `OutputFileDatasetConfig`, see how to [build a two step ML pipeline](#).

If, as shown previously, the first step wrote a single file, consuming it might look like:

```
parser = argparse.ArgumentParser()
parser.add_argument('--pd', dest='pd', required=True)
args = parser.parse_args()
with open(args.pd) as f:
    print(f.read())
```

Convert PipelineData objects to Datasets

If you'd like to make your `PipelineData` available for longer than the duration of a run, use its `as_dataset()` function to convert it to a `Dataset`. You may then register the `Dataset`, making it a first-class citizen in your workspace. Since your `PipelineData` object will have a different path every time the pipeline runs, it's highly recommended that you set `create_new_version` to `True` when registering a `Dataset` created from a `PipelineData` object.

```
step1_output_ds = step1_output_data.as_dataset()
step1_output_ds.register(name="processed_data", create_new_version=True)
```

TIP

An improved experience for persisting your intermediate data outside of your pipeline runs is available with the public preview class, `outputFileDatasetConfig`. For a code example using `outputFileDatasetConfig`, see how to [build a two-step ML pipeline](#).

Next steps

- [Create an Azure machine learning dataset](#)
- [Create and run machine learning pipelines with Azure Machine Learning SDK](#)

Publish and track machine learning pipelines

12/23/2020 • 7 minutes to read • [Edit Online](#)

This article will show you how to share a machine learning pipeline with your colleagues or customers.

Machine learning pipelines are reusable workflows for machine learning tasks. One benefit of pipelines is increased collaboration. You can also version pipelines, allowing customers to use the current model while you're working on a new version.

Prerequisites

- Create an [Azure Machine Learning workspace](#) to hold all your pipeline resources
- [Configure your development environment](#) to install the Azure Machine Learning SDK, or use an [Azure Machine Learning compute instance](#) with the SDK already installed
- Create and run a machine learning pipeline, such as by following [Tutorial: Build an Azure Machine Learning pipeline for batch scoring](#). For other options, see [Create and run machine learning pipelines with Azure Machine Learning SDK](#)

Publish a pipeline

Once you have a pipeline up and running, you can publish a pipeline so that it runs with different inputs. For the REST endpoint of an already published pipeline to accept parameters, you must configure your pipeline to use `PipelineParameter` objects for the arguments that will vary.

1. To create a pipeline parameter, use a `PipelineParameter` object with a default value.

```
from azureml.pipeline.core.graph import PipelineParameter

pipeline_param = PipelineParameter(
    name="pipeline_arg",
    default_value=10)
```

2. Add this `PipelineParameter` object as a parameter to any of the steps in the pipeline as follows:

```
compareStep = PythonScriptStep(
    script_name="compare.py",
    arguments=["--comp_data1", comp_data1, "--comp_data2", comp_data2, "--output_data", out_data3, "--param1", pipeline_param],
    inputs=[ comp_data1, comp_data2],
    outputs=[out_data3],
    compute_target=compute_target,
    source_directory=project_folder)
```

3. Publish this pipeline that will accept a parameter when invoked.

```
published_pipeline1 = pipeline_run1.publish_pipeline(
    name="My_Published_Pipeline",
    description="My Published Pipeline Description",
    version="1.0")
```

Run a published pipeline

All published pipelines have a REST endpoint. With the pipeline endpoint, you can trigger a run of the pipeline from any external systems, including non-Python clients. This endpoint enables "managed repeatability" in batch scoring and retraining scenarios.

IMPORTANT

If you are using Azure role-based access control (Azure RBAC) to manage access to your pipeline, [set the permissions for your pipeline scenario \(training or scoring\)](#).

To invoke the run of the preceding pipeline, you need an Azure Active Directory authentication header token.

Getting such a token is described in the [AzureCliAuthentication class reference](#) and in the [Authentication in Azure Machine Learning](#) notebook.

```
from azureml.pipeline.core import PublishedPipeline
import requests

response = requests.post(published_pipeline1.endpoint,
                           headers=aad_token,
                           json={"ExperimentName": "My_Pipeline",
                                  "ParameterAssignments": {"pipeline_arg": 20}})
```

The `json` argument to the POST request must contain, for the `ParameterAssignments` key, a dictionary containing the pipeline parameters and their values. In addition, the `json` argument may contain the following keys:

KEY	DESCRIPTION
<code>ExperimentName</code>	The name of the experiment associated with this endpoint
<code>Description</code>	Freeform text describing the endpoint
<code>Tags</code>	Freeform key-value pairs that can be used to label and annotate requests
<code>DataSetDefinitionValueAssignments</code>	Dictionary used for changing datasets without retraining (see discussion below)
<code>DataPathAssignments</code>	Dictionary used for changing datapaths without retraining (see discussion below)

Run a published pipeline using C#

The following code shows how to call a pipeline asynchronously from C#. The partial code snippet just shows the call structure and isn't part of a Microsoft sample. It doesn't show complete classes or error handling.

```

[DataContract]
public class SubmitPipelineRunRequest
{
    [DataMember]
    public string ExperimentName { get; set; }

    [DataMember]
    public string Description { get; set; }

    [DataMember(IsRequired = false)]
    public IDictionary<string, string> ParameterAssignments { get; set; }
}

// ... in its own class and method ...
const string RestEndpoint = "your-pipeline-endpoint";

using (HttpClient client = new HttpClient())
{
    var submitPipelineRunRequest = new SubmitPipelineRunRequest()
    {
        ExperimentName = "YourExperimentName",
        Description = "Asynchronous C# REST api call",
        ParameterAssignments = new Dictionary<string, string>
        {
            {
                // Replace with your pipeline parameter keys and values
                "your-pipeline-parameter", "default-value"
            }
        }
    };
}

string auth_key = "your-auth-key";
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", auth_key);

// submit the job
var requestPayload = JsonConvert.SerializeObject(submitPipelineRunRequest);
var httpContent = new StringContent(requestPayload, Encoding.UTF8, "application/json");
var submitResponse = await client.PostAsync(RestEndpoint, httpContent).ConfigureAwait(false);
if (!submitResponse.IsSuccessStatusCode)
{
    await WriteFailedResponse(submitResponse); // ... method not shown ...
    return;
}

var result = await submitResponse.Content.ReadAsStringAsync().ConfigureAwait(false);
var obj = JObject.Parse(result);
// ... use `obj` dictionary to access results
}

```

Run a published pipeline using Java

The following code shows a call to a pipeline that requires authentication (see [Set up authentication for Azure Machine Learning resources and workflows](#)). If your pipeline is deployed publicly, you don't need the calls that produce `authKey`. The partial code snippet doesn't show Java class and exception-handling boilerplate. The code uses `Optional.flatMap` for chaining together functions that may return an empty `Optional`. The use of `flatMap` shortens and clarifies the code, but note that `getRequestBody()` swallows exceptions.

```

import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.util.Optional;
// JSON library
import com.google.gson.Gson;

```

```

String scoringUri = "scoring-endpoint";
String tenantId = "your-tenant-id";
String clientId = "your-client-id";
String clientSecret = "your-client-secret";
String resourceManagerUrl = "https://management.azure.com";
String dataToBeScored = "{ \"ExperimentName\" : \"My_Pipeline\", \"ParameterAssignments\" : { \"pipeline_arg\" : \"20\" }}";

HttpClient client = HttpClient.newBuilder().build();
Gson gson = new Gson();

HttpRequest tokenAuthenticationRequest = tokenAuthenticationRequest(tenantId, clientId, clientSecret,
resourceManagerUrl);
Optional<String> authBody = getRequestBody(client, tokenAuthenticationRequest);
Optional<String> authKey = authBody.flatMap(body -> Optional.of(gson.fromJson(body,
AuthenticationBody.class).access_token));
Optional<HttpRequest> scoringRequest = authKey.flatMap(key -> Optional.of(scoringRequest(key, scoringUri,
dataToBeScored)));
Optional<String> scoringResult = scoringRequest.flatMap(req -> getRequestBody(client, req));
// ... etc (`scoringResult.orElse()`) ...

static HttpRequest tokenAuthenticationRequest(String tenantId, String clientId, String clientSecret, String
resourceManagerUrl)
{
    String authUrl = String.format("https://login.microsoftonline.com/%s/oauth2/token", tenantId);
    String clientIdParam = String.format("client_id=%s", clientId);
    String resourceParam = String.format("resource=%s", resourceManagerUrl);
    String clientSecretParam = String.format("client_secret=%s", clientSecret);

    String bodyString = String.format("grant_type=client_credentials&%s&%s&%s", clientIdParam, resourceParam,
clientSecretParam);

    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(authUrl))
        .POST(HttpRequest.BodyPublishers.ofString(bodyString))
        .build();
    return request;
}

static HttpRequest scoringRequest(String authKey, String scoringUri, String dataToBeScored)
{
    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(scoringUri))
        .header("Authorization", String.format("Token %s", authKey))
        .POST(HttpRequest.BodyPublishers.ofString(dataToBeScored))
        .build();
    return request;
}

static Optional<String> getRequestBody(HttpClient client, HttpRequest request) {
    try {
        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
        if (response.statusCode() != 200) {
            System.out.println(String.format("Unexpected server response %d", response.statusCode()));
            return Optional.empty();
        }
        return Optional.of(response.body());
    } catch(Exception x) {
        System.out.println(x.toString());
        return Optional.empty();
    }
}

class AuthenticationBody {
    String access_token;
    String token_type;
    int expires_in;
}

```

```

        String scope;
        String refresh_token;
        String id_token;

        AuthenticationBody() {}
    }
}

```

Changing datasets and datapaths without retraining

You may want to train and inference on different datasets and datapaths. For instance, you may wish to train on a smaller dataset but inference on the complete dataset. You switch datasets with the

`DataSetDefinitionValueAssignments` key in the request's `json` argument. You switch datapaths with `DataPathAssignments`. The technique for both is similar:

1. In your pipeline definition script, create a `PipelineParameter` for the dataset. Create a

`DatasetConsumptionConfig` or `DataPath` from the `PipelineParameter`:

```

tabular_dataset =
Dataset.Tabular.from_delimited_files('https://dprepdata.blob.core.windows.net/demo/Titanic.csv')
tabular_pipeline_param = PipelineParameter(name="tabular_ds_param", default_value=tabular_dataset)
tabular_ds_consumption = DatasetConsumptionConfig("tabular_dataset", tabular_pipeline_param)

```

2. In your ML script, access the dynamically specified dataset using `Run.get_context().input_datasets`:

```

from azureml.core import Run

input_tabular_ds = Run.get_context().input_datasets['tabular_dataset']
dataframe = input_tabular_ds.to_pandas_dataframe()
# ... etc ...

```

Notice that the ML script accesses the value specified for the `DatasetConsumptionConfig` (`tabular_dataset`) and not the value of the `PipelineParameter` (`tabular_ds_param`).

3. In your pipeline definition script, set the `DatasetConsumptionConfig` as a parameter to the

`PipelineScriptStep`:

```

train_step = PythonScriptStep(
    name="train_step",
    script_name="train_with_dataset.py",
    arguments=["--param1", tabular_ds_consumption],
    inputs=[tabular_ds_consumption],
    compute_target=compute_target,
    source_directory=source_directory)

pipeline = Pipeline(workspace=ws, steps=[train_step])

```

4. To switch datasets dynamically in your inferencing REST call, use `DataSetDefinitionValueAssignments`:

```

tabular_ds1 = Dataset.Tabular.from_delimited_files('path_to_training_dataset')
tabular_ds2 = Dataset.Tabular.from_delimited_files('path_to_inference_dataset')
ds1_id = tabular_ds1.id
ds2_id = tabular_ds2.id

response = requests.post(rest_endpoint,
                         headers=aad_token,
                         json={
                               "ExperimentName": "MyRestPipeline",
                               "DataSetDefinitionValueAssignments": [
                                   {
                                       "tabular_ds_param": {
                                           "SavedDataSetReference": {"Id": ds1_id #or ds2_id
                                         }}}]})
```

The notebooks [Showcasing Dataset and PipelineParameter](#) and [Showcasing DataPath and PipelineParameter](#) have complete examples of this technique.

Create a versioned pipeline endpoint

You can create a Pipeline Endpoint with multiple published pipelines behind it. This technique gives you a fixed REST endpoint as you iterate on and update your ML pipelines.

```

from azureml.pipeline.core import PipelineEndpoint

published_pipeline = PipelineEndpoint.get(workspace=ws, name="My_Published_Pipeline")
pipeline_endpoint = PipelineEndpoint.publish(workspace=ws, name="PipelineEndpointTest",
                                              pipeline=published_pipeline, description="Test description
Notebook")
```

Submit a job to a pipeline endpoint

You can submit a job to the default version of a pipeline endpoint:

```

pipeline_endpoint_by_name = PipelineEndpoint.get(workspace=ws, name="PipelineEndpointTest")
run_id = pipeline_endpoint_by_name.submit("PipelineEndpointExperiment")
print(run_id)
```

You can also submit a job to a specific version:

```

run_id = pipeline_endpoint_by_name.submit("PipelineEndpointExperiment", pipeline_version="0")
print(run_id)
```

The same can be accomplished using the REST API:

```

rest_endpoint = pipeline_endpoint_by_name.endpoint
response = requests.post(rest_endpoint,
                         headers=aad_token,
                         json={"ExperimentName": "PipelineEndpointExperiment",
                               "RunSource": "API",
                               "ParameterAssignments": {"1": "united", "2": "city"}})
```

Use published pipelines in the studio

You can also run a published pipeline from the studio:

1. Sign in to [Azure Machine Learning studio](#).

2. [View your workspace](#).

3. On the left, select **Endpoints**.

4. On the top, select **Pipeline endpoints**.

The screenshot shows the Azure Machine Learning studio interface. The left sidebar has sections for Home, Notebooks, Automated ML, Designer, Datasets, Experiments, Pipelines, Models, and Endpoints. The 'Endpoints' section is highlighted with a red box. The main content area is titled 'Endpoints' and shows a table of pipeline endpoints. The table has columns for Name, Description, Modified on, Modified by, Last run submit time, and more. A single row is visible: 'My_New_Pipeline' with the description 'My Published Pipeline D...', modified on 11/11/2019, 3:41:03 PM, and last run at 11/11/2019, 3:42:41 PM. The 'Pipeline endpoints' tab in the top navigation bar is also highlighted with a red box.

5. Select a specific pipeline to run, consume, or review results of previous runs of the pipeline endpoint.

Disable a published pipeline

To hide a pipeline from your list of published pipelines, you disable it, either in the studio or from the SDK:

```
# Get the pipeline by using its ID from Azure Machine Learning studio
p = PublishedPipeline.get(ws, id="068f4885-7088-424b-8ce2-eeb9ba5381a6")
p.disable()
```

You can enable it again with `p.enable()`. For more information, see [PublishedPipeline class](#) reference.

Next steps

- Use [these Jupyter notebooks on GitHub](#) to explore machine learning pipelines further.
- See the SDK reference help for the `azureml-pipelines-core` package and the `azureml-pipelines-steps` package.
- See the [how-to](#) for tips on debugging and troubleshooting pipelines.

Trigger machine learning pipelines with Azure Machine Learning SDK for Python

12/23/2020 • 6 minutes to read • [Edit Online](#)

In this article, you'll learn how to programmatically schedule a pipeline to run on Azure. You can choose to create a schedule based on elapsed time or on file-system changes. Time-based schedules can be used to take care of routine tasks, such as monitoring for data drift. Change-based schedules can be used to react to irregular or unpredictable changes, such as new data being uploaded or old data being edited. After learning how to create schedules, you'll learn how to retrieve and deactivate them. Finally, you'll learn how to use an Azure Logic App to allow more complex triggering logic or behavior.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#).
- A Python environment in which the Azure Machine Learning SDK for Python is installed. For more information, see [Create and manage reusable environments for training and deployment with Azure Machine Learning](#).
- A Machine Learning workspace with a published pipeline. You can use the one built in [Create and run machine learning pipelines with Azure Machine Learning SDK](#).

Initialize the workspace & get data

To schedule a pipeline, you'll need a reference to your workspace, the identifier of your published pipeline, and the name of the experiment in which you wish to create the schedule. You can get these values with the following code:

```
import azureml.core
from azureml.core import Workspace
from azureml.pipeline.core import Pipeline, PublishedPipeline
from azureml.core.experiment import Experiment

ws = Workspace.from_config()

experiments = Experiment.list(ws)
for experiment in experiments:
    print(experiment.name)

published_PIPELINES = PublishedPipeline.list(ws)
for published_PIPELINE in published_PIPELINES:
    print(f"{published_PIPELINE.name},{published_PIPELINE.id}")

experiment_name = "MyExperiment"
pipeline_id = "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee"
```

Create a schedule

To run a pipeline on a recurring basis, you'll create a schedule. A `Schedule` associates a pipeline, an experiment, and a trigger. The trigger can either be a `ScheduleRecurrence` that describes the wait between runs or a Datastore path that specifies a directory to watch for changes. In either case, you'll need the pipeline identifier and the name of the experiment in which to create the schedule.

At the top of your python file, import the `Schedule` and `ScheduleRecurrence` classes:

```
from azureml.pipeline.core.schedule import ScheduleRecurrence, Schedule
```

Create a time-based schedule

The `ScheduleRecurrence` constructor has a required `frequency` argument that must be one of the following strings: "Minute", "Hour", "Day", "Week", or "Month". It also requires an integer `interval` argument specifying how many of the `frequency` units should elapse between schedule starts. Optional arguments allow you to be more specific about starting times, as detailed in the [ScheduleRecurrence SDK docs](#).

Create a `Schedule` that begins a run every 15 minutes:

```
recurrence = ScheduleRecurrence(frequency="Minute", interval=15)
recurring_schedule = Schedule.create(ws, name="MyRecurringSchedule",
                                      description="Based on time",
                                      pipeline_id=pipeline_id,
                                      experiment_name=experiment_name,
                                      recurrence=recurrence)
```

Create a change-based schedule

Pipelines that are triggered by file changes may be more efficient than time-based schedules. For instance, you may want to perform a preprocessing step when a file is changed, or when a new file is added to a data directory. You can monitor any changes to a datastore or changes within a specific directory within the datastore. If you monitor a specific directory, changes within subdirectories of that directory will *not* trigger a run.

To create a file-reactive `Schedule`, you must set the `datastore` parameter in the call to `Schedule.create`. To monitor a folder, set the `path_on_datastore` argument.

The `polling_interval` argument allows you to specify, in minutes, the frequency at which the datastore is checked for changes.

If the pipeline was constructed with a [DataPath PipelineParameter](#), you can set that variable to the name of the changed file by setting the `data_path_parameter_name` argument.

```
datastore = Datastore(workspace=ws, name="workspaceblobstore")

reactive_schedule = Schedule.create(ws, name="MyReactiveSchedule", description="Based on input file change.",
                                      pipeline_id=pipeline_id, experiment_name=experiment_name, datastore=datastore,
                                      data_path_parameter_name="input_data")
```

Optional arguments when creating a schedule

In addition to the arguments discussed previously, you may set the `status` argument to `"Disabled"` to create an inactive schedule. Finally, the `continue_on_step_failure` allows you to pass a Boolean that will override the pipeline's default failure behavior.

View your scheduled pipelines

In your Web browser, navigate to Azure Machine Learning. From the **Endpoints** section of the navigation panel, choose **Pipeline endpoints**. This takes you to a list of the pipelines published in the Workspace.

Name	Description	Modified on	Modified by	Last run submit time	Last run status	Status
My published pipeline		-	Joanna Mooney	-	-	Active

In this page you can see summary information about all the pipelines in the Workspace: names, descriptions, status, and so forth. Drill in by clicking in your pipeline. On the resulting page, there are more details about your pipeline and you may drill down into individual runs.

Deactivate the pipeline

If you have a `Pipeline` that is published, but not scheduled, you can disable it with:

```
pipeline = PublishedPipeline.get(ws, id=pipeline_id)
pipeline.disable()
```

If the pipeline is scheduled, you must cancel the schedule first. Retrieve the schedule's identifier from the portal or by running:

```
ss = Schedule.list(ws)
for s in ss:
    print(s)
```

Once you have the `schedule_id` you wish to disable, run:

```
def stop_by_schedule_id(ws, schedule_id):
    s = next(s for s in Schedule.list(ws) if s.id == schedule_id)
    s.disable()
    return s

stop_by_schedule_id(ws, schedule_id)
```

If you then run `Schedule.list(ws)` again, you should get an empty list.

Use Azure Logic Apps for complex triggers

More complex trigger rules or behavior can be created using an [Azure Logic App](#).

To use an Azure Logic App to trigger a Machine Learning pipeline, you'll need the REST endpoint for a published

Machine Learning pipeline. [Create and publish your pipeline](#). Then find the REST endpoint of your `PublishedPipeline` by using the pipeline ID:

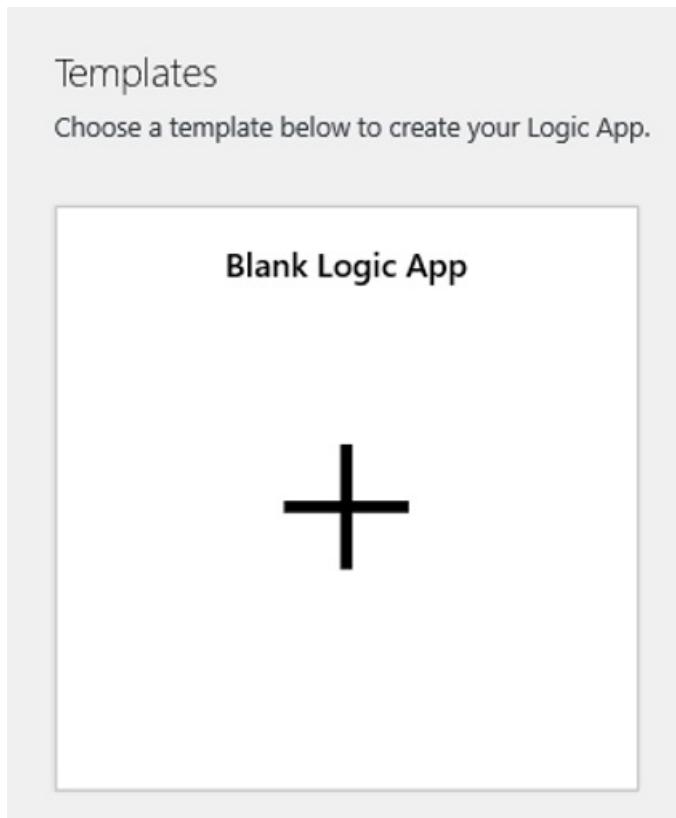
```
# You can find the pipeline ID in Azure Machine Learning studio  
  
published_pipeline = PublishedPipeline.get(ws, id=<pipeline-id-here>)  
published_pipeline.endpoint
```

Create a Logic App

Now create an [Azure Logic App](#) instance. If you wish, [use an integration service environment \(ISE\)](#) and [set up a customer-managed key](#) for use by your Logic App.

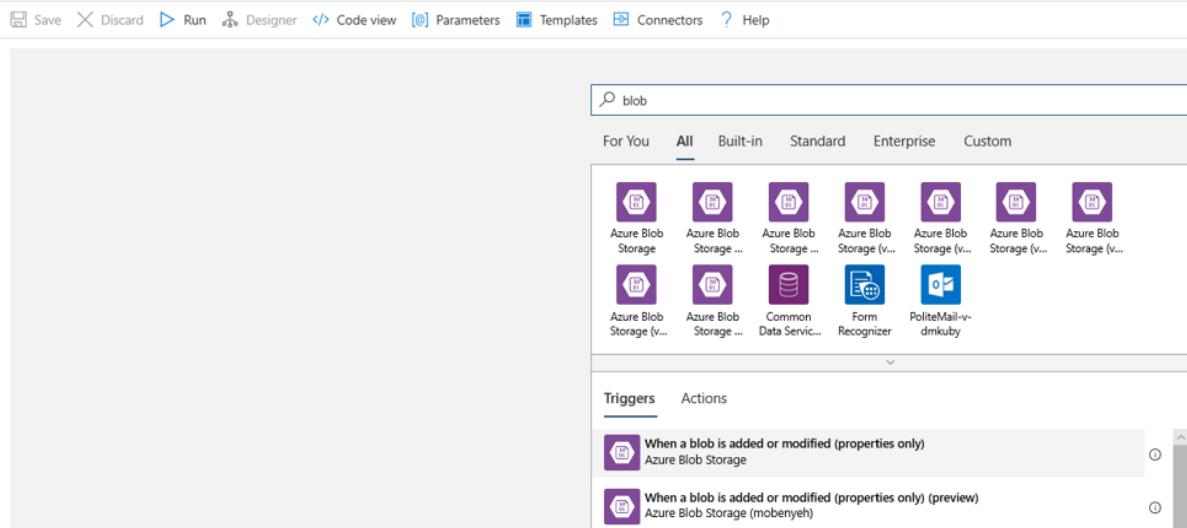
Once your Logic App has been provisioned, use these steps to configure a trigger for your pipeline:

1. [Create a system-assigned managed identity](#) to give the app access to your Azure Machine Learning Workspace.
2. Navigate to the Logic App Designer view and select the Blank Logic App template.



3. In the Designer, search for **blob**. Select the **When a blob is added or modified (properties only)** trigger and add this trigger to your Logic App.

Logic Apps Designer



- Fill in the connection info for the Blob storage account you wish to monitor for blob additions or modifications. Select the Container to monitor.

Choose the **Interval** and **Frequency** to poll for updates that work for you.

NOTE

This trigger will monitor the selected Container but will not monitor subfolders.

- Add an HTTP action that will run when a new or modified blob is detected. Select **+ New Step**, then search for and select the HTTP action.

Choose an action

HTTP

For You All Built-in Standard Enterprise Custom

 HTTP	 Office 365 Outlook	 Azure Blob Storage	 Aquaforest PDF	 Azure Blob Storage ...	 Azure Blob Storage ...	 Azure Blob Storage (v...
--	--	--	--	--	--	--

Triggers Actions See more

-  Extract PDF pages by barcode
Aquaforest PDF
-  Extract PDF pages by text
Aquaforest PDF
-  Get barcode value
Aquaforest PDF
-  Get text from PDF
Aquaforest PDF
-  OCR PDF or images
Aquaforest PDF

Use the following settings to configure your action:

SETTING	VALUE
HTTP action	POST
URI	the endpoint to the published pipeline that you found as a Prerequisite
Authentication mode	Managed Identity

- Set up your schedule to set the value of any [DataPath PipelineParameters](#) you may have:

```
"DataPathAssignments":{  
    "input_datapath":{  
        "DataStoreName":"<datastore-name>",  
        "RelativePath":"@triggerBody()?[ 'Name' ]"  
    },  
    "ExperimentName":"MyRestPipeline",  
    "ParameterAssignments":{  
        "input_string":"sample_string3"  
    },
```

Use the `DataStoreName` you added to your workspace as a [Prerequisite](#).

The screenshot shows the Azure Logic App builder interface with an 'HTTP' trigger configuration. The fields are as follows:

- * Method: POST
- * URI: `https://eastus2.aether.ms/api/v1.0/subscriptions/b8c23406-f9b5-4ccb-8a65-a8cb5dc6a5a/resourceGroups/aesviennatesteuap/providers/Microsoft.MachineLearningServices/workspaces/elihop-cuseuap/PipelineRuns/PipelineSubmit/7b1817a3-593a-42fe-a3f9-8b149860fe95`
- Headers: Enter key | Enter value
- Queries: Enter key | Enter value
- Body:

```
{  
    "DataPathAssignments": {  
        "input_datapath": {  
            "DataStoreName": "workspaceblobstore",  
            "RelativePath": "[List of Files Name]"  
        }  
    },  
    "ExperimentName": "MyRestPipeline",  
    "ParameterAssignments": {  
        "input_string": "sample_string3"  
    },  
    "RunSource": "SDK"  
}
```
- * Authentication: Managed Identity

2. Select **Save** and your schedule is now ready.

IMPORTANT

If you are using Azure role-based access control (Azure RBAC) to manage access to your pipeline, set the permissions for your pipeline scenario (training or scoring).

Next steps

In this article, you used the Azure Machine Learning SDK for Python to schedule a pipeline in two different ways. One schedule recurs based on elapsed clock time. The other schedule runs if a file is modified on a specified `Datastore` or within a directory on that store. You saw how to use the portal to examine the pipeline and individual runs. You learned how to disable a schedule so that the pipeline stops running. Finally, you created an Azure Logic App to trigger a pipeline.

For more information, see:

Use Azure Machine Learning Pipelines for batch scoring

- Learn more about [pipelines](#)
- Learn more about [exploring Azure Machine Learning with Jupyter](#)

Enable logging in Azure Machine Learning designer pipelines

12/23/2020 • 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.

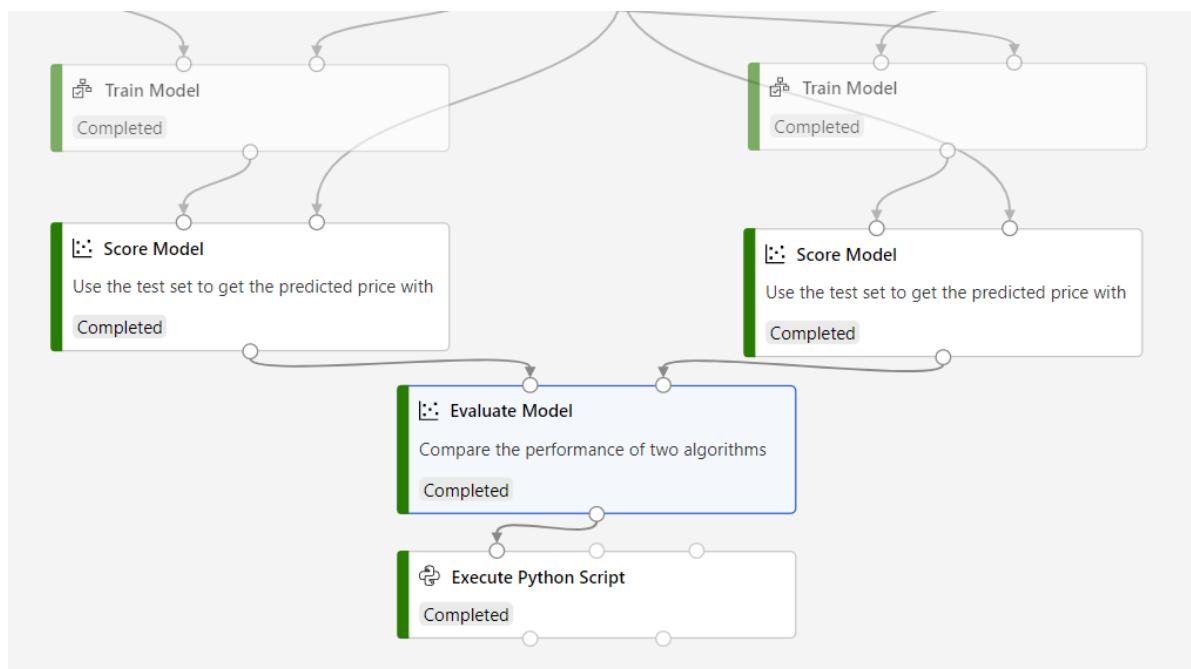
For more information on logging metrics using the SDK authoring experience, see [Monitor Azure ML experiment runs and metrics](#).

Enable logging with Execute Python Script

Use the **Execute Python Script** module 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** module 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 modules.

1. Connect an **Execute Python Script** module to the output of the **Evaluate Model** module.



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:

```

# 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.
    parent_run.log(name='Mean_Absolute_Error (right port)', value=dataframe1['Mean_Absolute_Error'][1])

    return dataframe1,

```

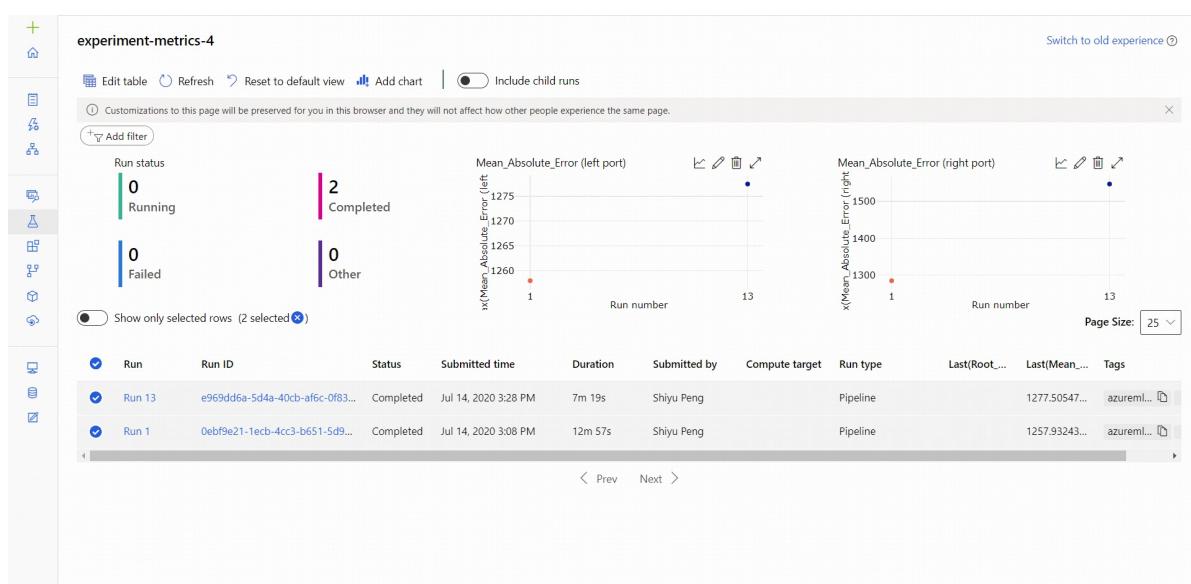
This code uses the Azure Machine Learning Python SDK to log values. It uses `Run.get_context()` to get the context of the current run. It then logs values to that context with the `run.parent.log()` method. It uses `parent` to log values to the parent pipeline run rather than the module run.

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 **Experiments** section.
2. Select your experiment.
3. Select the run 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](#).

Transform data in Azure Machine Learning designer

12/23/2020 • 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 modules** 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 modules. Select **Datasets**. Then view the **Samples** section.
5. Drag and drop the **Adult Census Income Binary classification** dataset onto the canvas.
6. Select the **Adult Census Income** dataset module.
7. In the details pane that appears to the right of the canvas, select **Outputs**.
8. Select the visualize icon .
9. Use the data preview window to explore the dataset. Take special note of the "native-country" column values.

Split the data

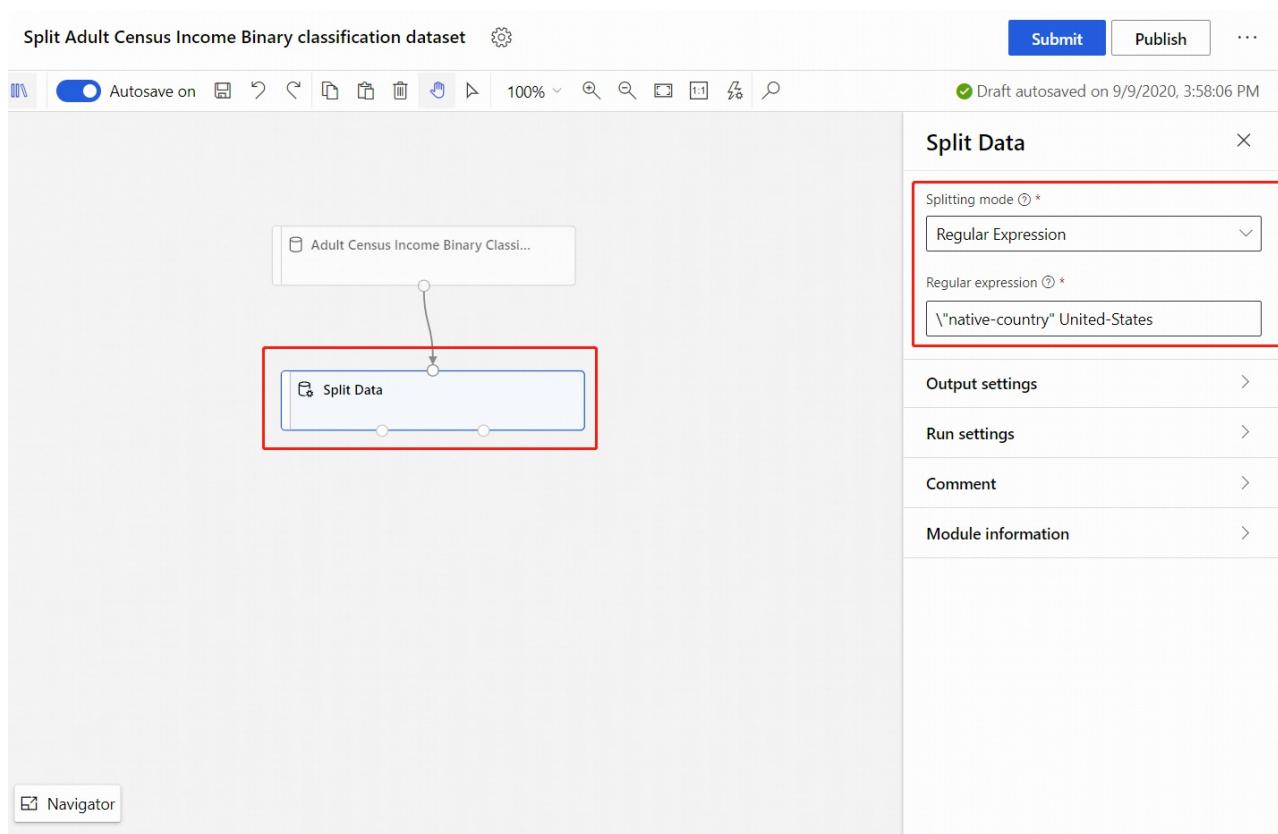
In this section, you use the [Split Data module](#) to identify and split rows that contain "United-States" in the "native-

country" column.

1. In the module palette to the left of the canvas, expand the **Data Transformation** section and find the **Split Data** module.
2. Drag the **Split Data** module onto the canvas, and drop the module below the dataset module.
3. Connect the dataset module to the **Split Data** module.
4. Select the **Split Data** module.
5. In the module 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 module, see the related [algorithm module reference page](#).

Your pipeline should look like this:

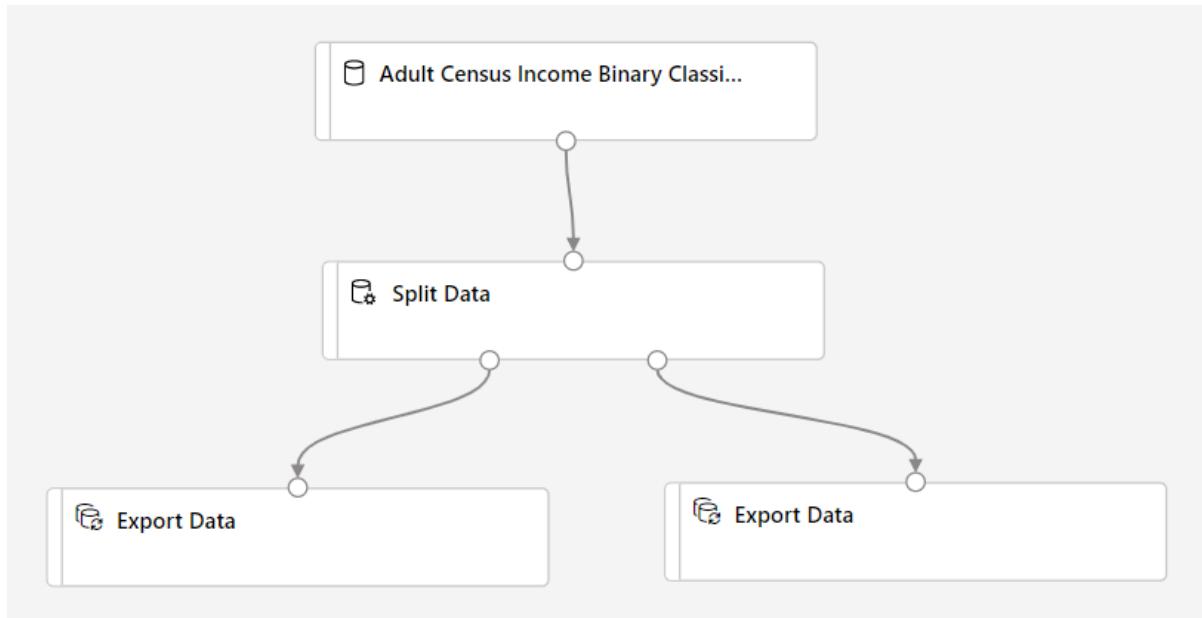


Save the datasets

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** module to save your dataset to a datastore. For more information on datastores, see [Connect to Azure storage services](#)

1. In the module palette to the left of the canvas, expand the **Data Input and Output** section and find the **Export Data** module.
2. Drag and drop two **Export Data** modules below the **Split Data** module.
3. Connect each output port of the **Split Data** module to a different **Export Data** module.

Your pipeline should look something like this:



- Select the **Export Data** module that is connected to the *left*-most port of the **Split Data** module.

The order of the output ports matter for the **Split Data** module. 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.

- In the module 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`.

- Select the **Export Data** module connected to the *right*-most port of the **Split Data** module.

- In the module details pane to the right of the canvas, set the following options:

Datastore type: Azure Blob Storage

Datastore: Select the same datastore as above

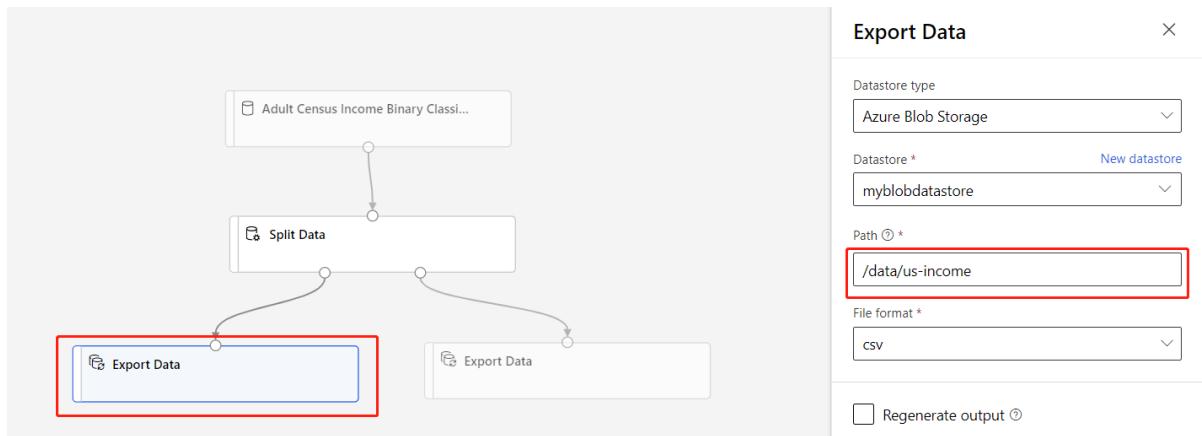
Path: `/data/non-us-income`

File format: csv

- Confirm the **Export Data** module connected to the left port of the **Split Data** has the **Path** `/data/us-income`.

9. Confirm the **Export Data** module connected to the right port has the **Path** `/data/us-income`.

Your pipeline and settings should look like this:



Submit the run

Now that your pipeline is setup to split and export the data, submit a pipeline run.

1. At the top of the canvas, select **Submit**.
2. In the **Set up pipeline run** dialog, select **Create new** to create an experiment.

Experiments logically group together related pipeline runs. 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** module to confirm that your data has been split correctly.

1. Select the **Split Data** module.
2. In the module 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 interface. On the left, the 'Resource groups' blade is open, displaying a list of resource groups. One group, 'my-rg', is selected and highlighted with a red box. On the right, the 'Overview' blade for the selected resource group is displayed, showing its subscription information ('documentationteam'), subscription ID ('aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa'), and tags ('Click here to add tags'). At the top of the overview blade, there is 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 navigation sidebar with sections like Home, Author, Automated ML, Designer, Notebooks, Assets (Datasets, Experiments, Pipelines, Models, Endpoints), Manage, and Compute. The Compute section is highlighted with a red box. The main area is titled 'Compute' and has tabs for Compute Instances, Training Clusters, Inference Clusters (which is underlined in blue), and Attached Compute. Below the tabs is a toolbar with New, Refresh, Delete (highlighted with a red box), and a search bar. A table lists datasets, with one row for 'aks-compute' showing its type as 'Kubernetes Serv...', provisioning status as 'Succe...', creation date as '10/17/...', and status as 'STAN...'. Navigation arrows for 'Prev' and 'Next' are also present.

You can unregister datasets from your workspace by selecting each dataset and selecting **Unregister**.

The screenshot shows the Azure Machine Learning Datasets blade for a dataset named 'TD-Sample_1: Regression - Automobile_Price_Prediction_(Basic)-Clean_Missing_Data-Cleaning_transformation-f6dc0eb1'. The sidebar on the left has sections like New, Home, Author, Notebooks, Automated ML, Designer, Assets (Datasets, Experiments, Pipelines, Models, Endpoints), Manage, Compute, Environments, Datastores, and Data labeling. The 'Datasets' section is highlighted with a red box. The main content area shows the dataset details: Attributes (Properties, File, Description: 'This is a dataset promoted by inference graph generation automatically on 11/12/2023.', Datastore: 'workspaceblobstore', Relative path: 'azurerm/4393076b-19ff-4e41-81d9-a1146d905696/Cleaning_transformation'), Profile (No profile generated), Files in dataset (4), Current version (1), and Latest version (1). To the right, there are Tags (CreatedByAMLStudio: true) and Sample usage (Python code snippet for workspace setup and dataset download).

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 to retrain models in the designer

12/23/2020 • 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](#)

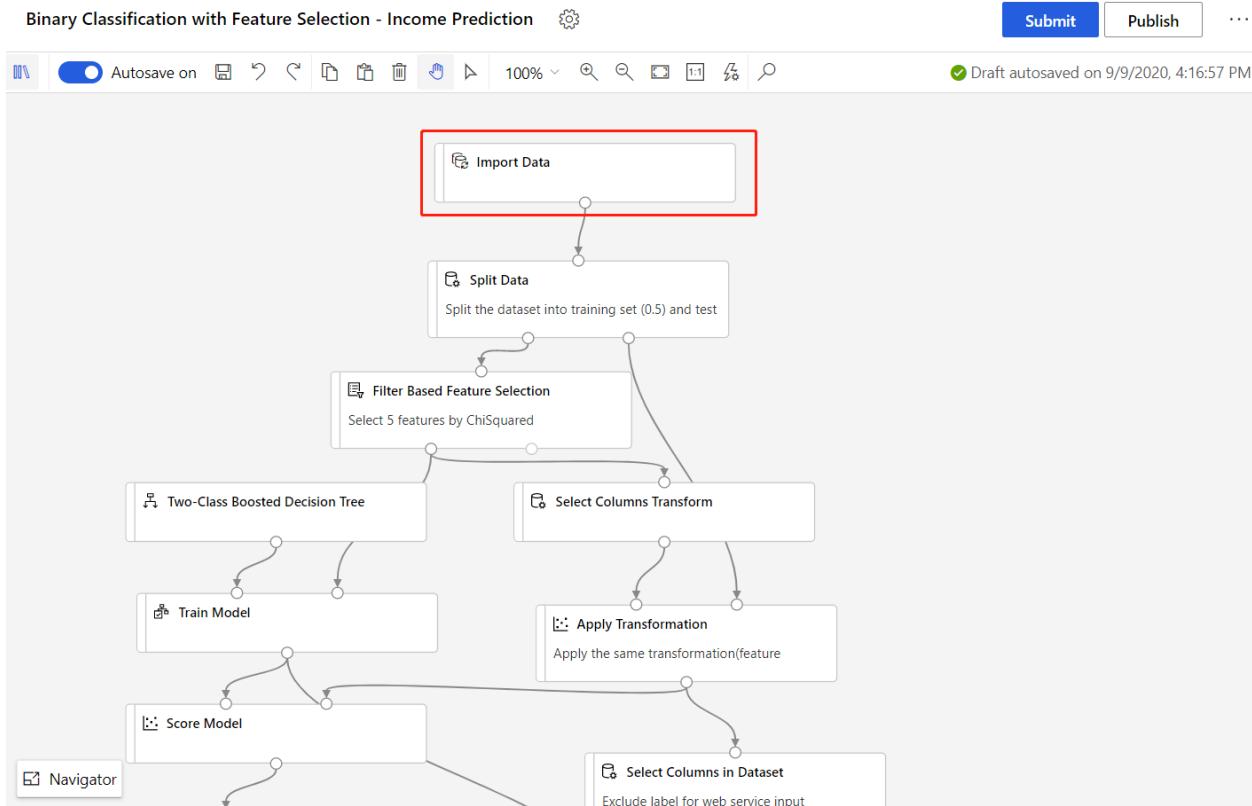
IMPORTANT

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](#) module 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 module parameters in a pipeline. When the pipeline is resubmitted, the values of these parameters can be specified.

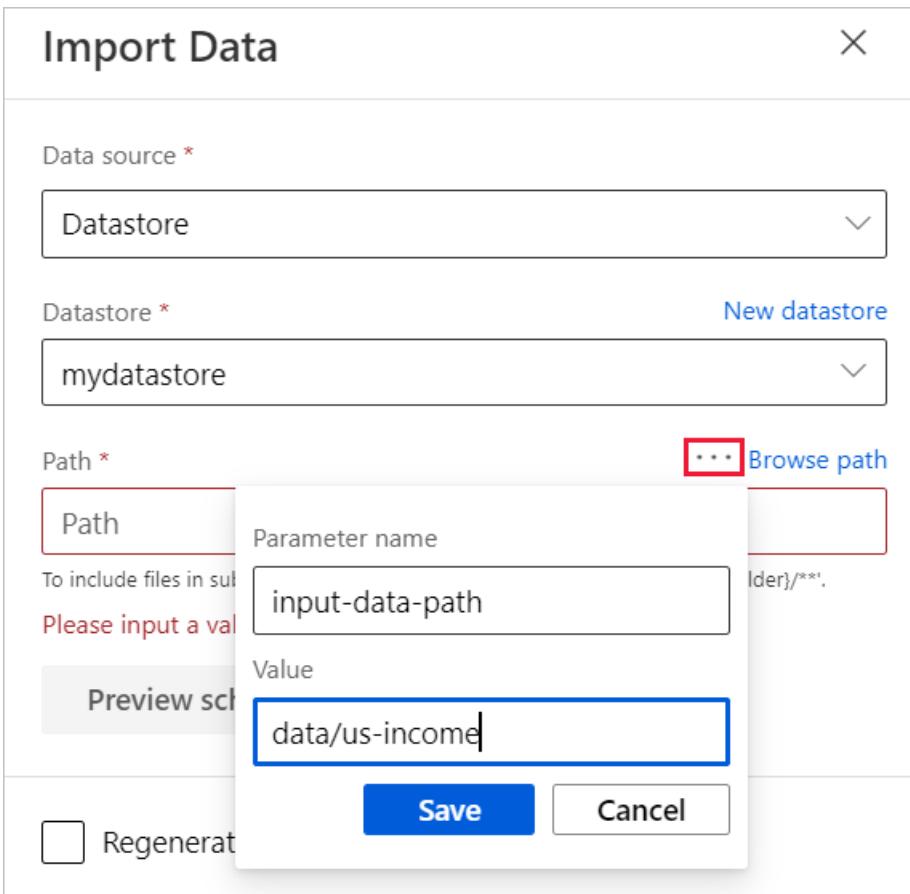
For this example, you will change the training data path from a fixed value to a parameter, so that you can retrain your model on different data. You can also add other module parameters as pipeline parameters according to your use case.

1. Select the **Import Data** module.

NOTE

This example uses the Import Data module to access data in a registered datastore. However, you can follow similar steps if you use alternative data access patterns.

2. In the module 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.



7. Select **Save**.

NOTE

You can also detach a module parameter from pipeline parameter in the module 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 module parameter.

8. Submit the pipeline run.

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**.

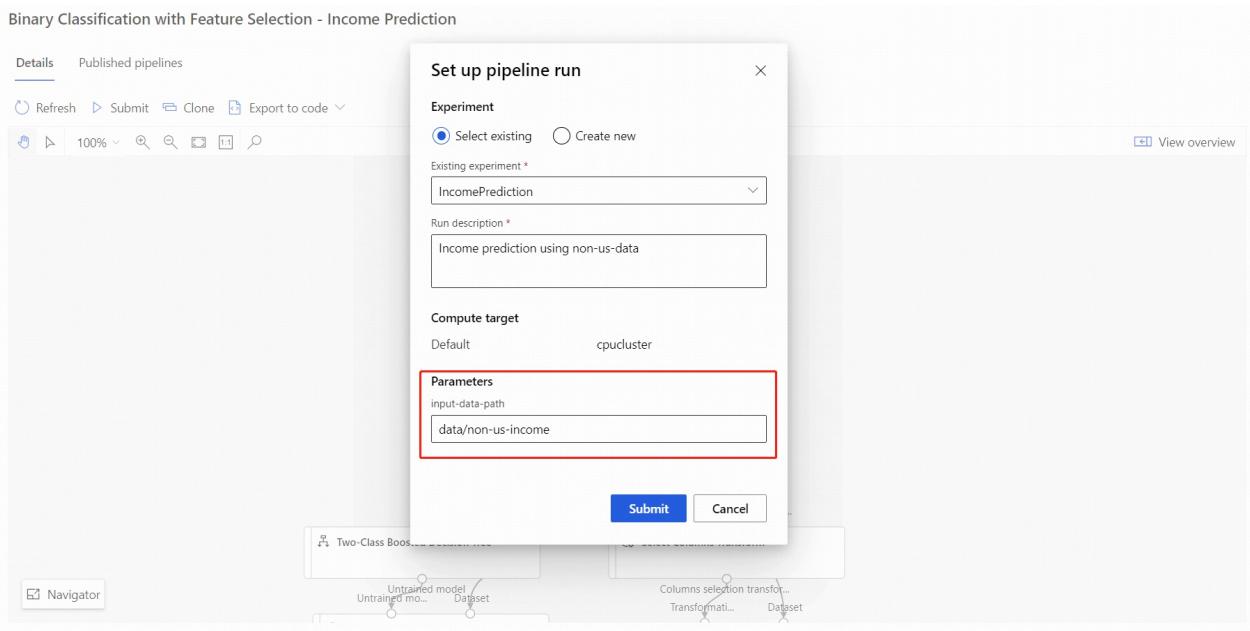
Retrain your model

Now that you have a published training pipeline, you can use it to retrain your model on new data. You can submit runs from a pipeline endpoint from the studio workspace or programmatically.

Submit runs by using the studio portal

Use the following steps to submit a parameterized pipeline endpoint run 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 run. For this example, update the data path to train your model using a non-US dataset.



Submit runs 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.

Run batch predictions using Azure Machine Learning designer

12/23/2020 • 3 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 [how-to](#).

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](#).

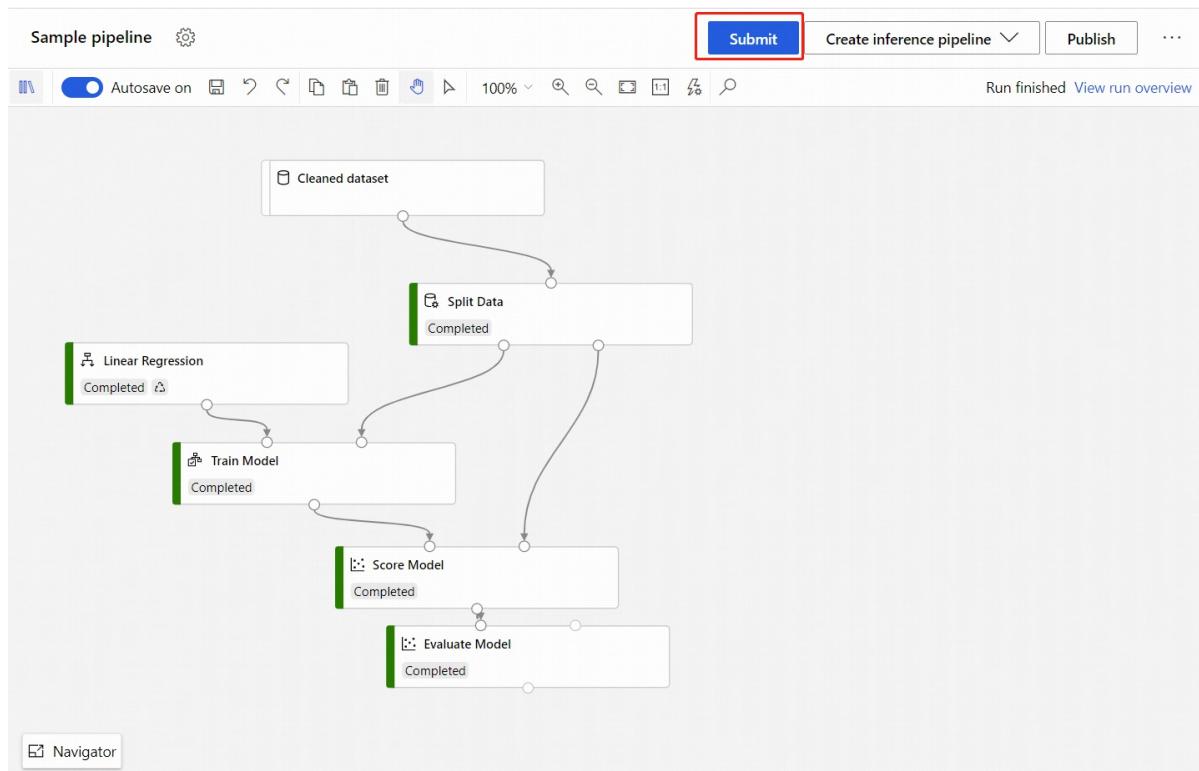
IMPORTANT

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

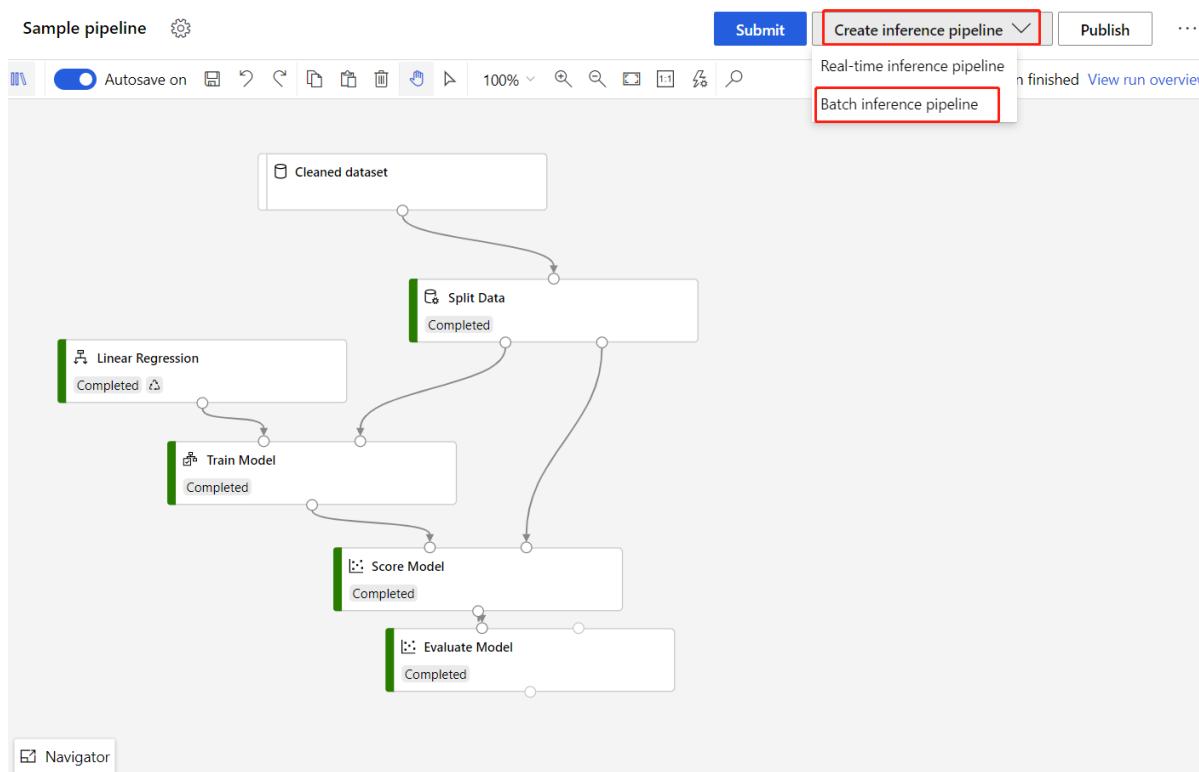
Your training pipeline must be run at least once to be able to create an inferencing pipeline.

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.



Now that the training pipeline has been run, you can create a batch inference pipeline.

1. Next to **Submit**, select the new dropdown **Create inference pipeline**.
2. Select **Batch inference pipeline**.



The result is a default batch inference pipeline.

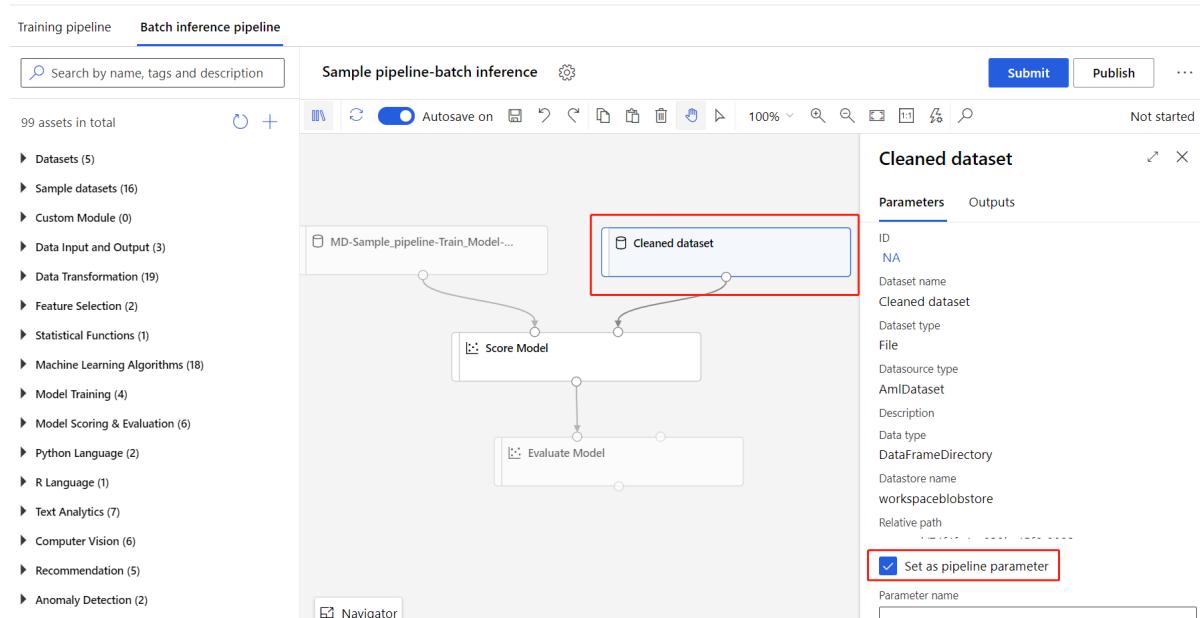
Add a pipeline parameter

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 module.
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.



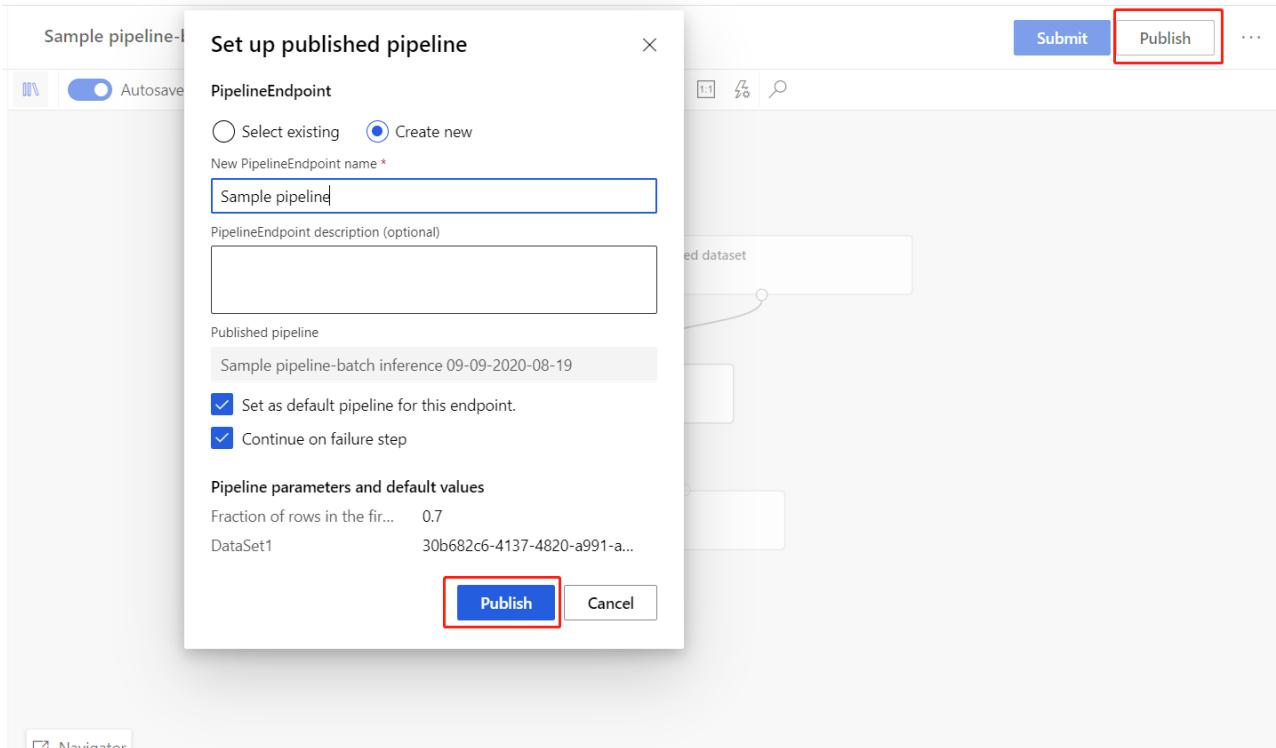
Publish your batch inference pipeline

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 run

In this section, you will set up a manual pipeline run 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.

The screenshot shows the Azure Machine Learning studio interface. On the left, there is a navigation sidebar with the following items:

- New
- Home
- Author
 - Notebooks
 - Automated ML
 - Designer
- Assets
 - Datasets
 - Experiments
 - Pipelines
 - Models
- Endpoints** (This item is highlighted with a red box)
- Manage
 - Compute
 - Datastores
 - Data labeling

The main content area has a breadcrumb navigation bar: `docs-ws > Endpoints`. The title is **Endpoints**. Below the title, there are two tabs: **Real-time endpoints** and **Pipeline endpoints**. The **Pipeline endpoints** tab is selected and highlighted with a red box. There are buttons for Refresh, Disable, Enable, and View disabled. A table lists the published pipelines:

Name ↓	Description
Sample pipeline	Inferencing pipeline. Adj...
income-prediction	
batch-inference-en...	Endpoint for triggering b...

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 run 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 run overview panel. By calling the endpoint, you are 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 will 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

5c9d9f61-78d8-44a5-90b9-7...

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. The top navigation bar is blue with the text "Preview Microsoft Azure Machine Learning". Below it, the breadcrumb navigation shows "docs-ws > Endpoints > Sample pipeline". On the left, there's a sidebar with icons for workspace, endpoints, datasets, models, and experiments. The main area has a title "Sample pipeline" and tabs for "Details" and "Published pipelines". Under "Published pipelines", there are buttons for "Refresh", "Disable", "Enable", and "Set as Default" (which is highlighted with a red box). A table lists three pipelines:

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 right of the table, there are "Prev" and "Next" navigation buttons.

Next steps

Follow the designer [tutorial](#) to train and deploy a regression model."

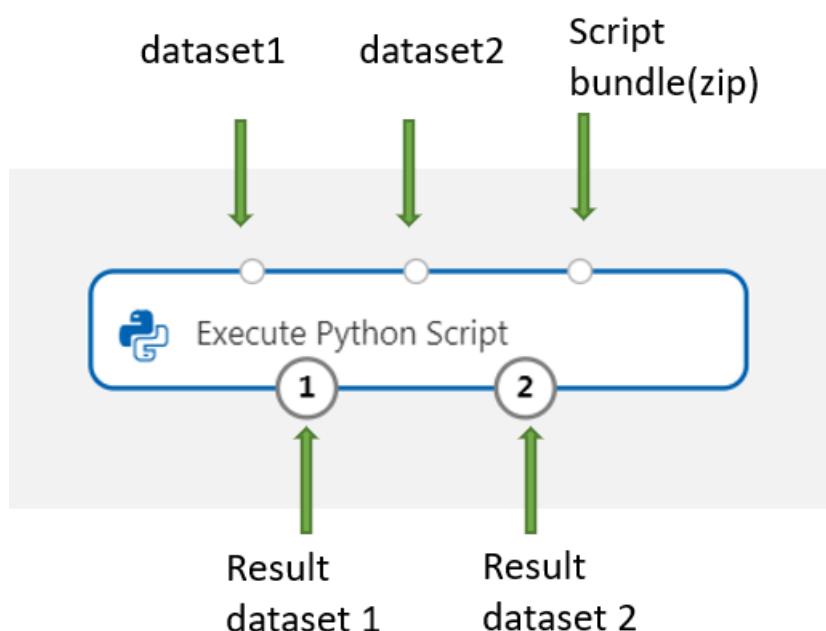
Run Python code in Azure Machine Learning designer

12/23/2020 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to use the [Execute Python Script](#) module 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 module 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 module

1. Find the **Execute Python Script** module in the designer palette. It can be found in the **Python Language** section.
2. Drag and drop the module onto the pipeline canvas.

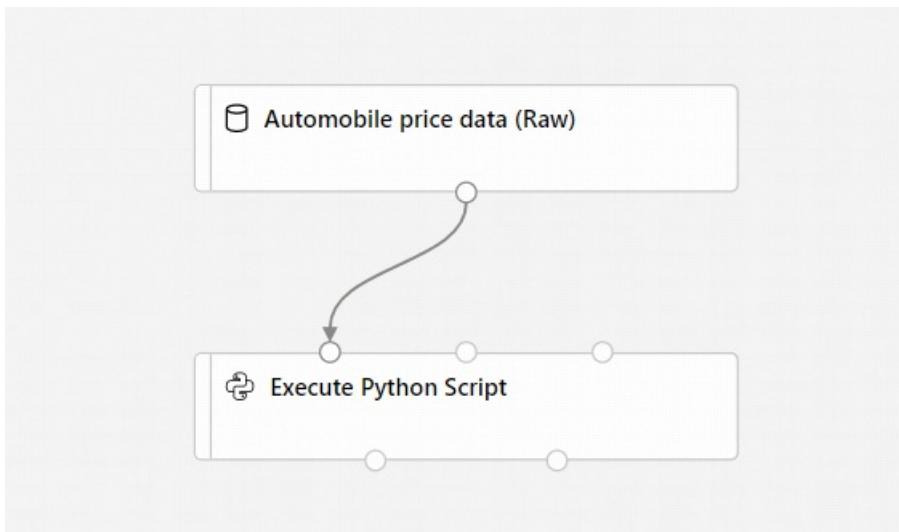
Connect input datasets

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** module. 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 modules are optional since you can generate or import data directly in the **Execute Python Script** module.

Write your Python code

The designer provides an initial entry point script for you to edit and enter your own Python code.

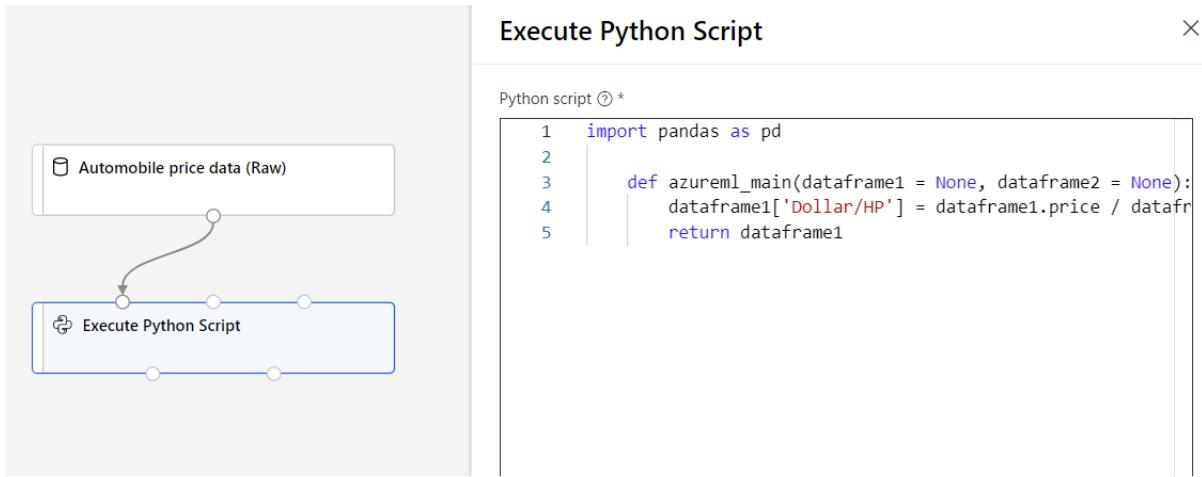
In this example, you use Pandas to combine two columns found in the automobile dataset, **Price** and **Horsepower**, to create a new column, **Dollars per horsepower**. This column represents how much you pay for each horsepower, which could be a useful feature to decide if a car is a good deal for the money.

1. Select the **Execute Python Script** module.
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** module.

The return value must be a Pandas Dataframe. You can return up to two dataframes as module 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.

Troubleshooting machine learning pipelines

12/23/2020 • 6 minutes to read • [Edit Online](#)

In this article, you learn how to troubleshoot when you get errors running a [machine learning pipeline](#) in the [Azure Machine Learning SDK](#) and [Azure Machine Learning designer](#).

Troubleshooting tips

The following table contains common problems during pipeline development, with potential solutions.

PROBLEM	POSSIBLE SOLUTION
Unable to pass data to <code>PipelineData</code> directory	Ensure you have created a directory in the script that corresponds to where your pipeline expects the step output data. In most cases, an input argument will define the output directory, and then you create the directory explicitly. Use <code>os.makedirs(args.output_dir, exist_ok=True)</code> to create the output directory. See the tutorial for a scoring script example that shows this design pattern.
Dependency bugs	If you see dependency errors in your remote pipeline that did not occur when locally testing, confirm your remote environment dependencies and versions match those in your test environment. (See Environment building, caching, and reuse)
Ambiguous errors with compute targets	Try deleting and re-creating compute targets. Re-creating compute targets is quick and can solve some transient issues.
Pipeline not reusing steps	Step reuse is enabled by default, but ensure you haven't disabled it in a pipeline step. If reuse is disabled, the <code>allow_reuse</code> parameter in the step will be set to <code>False</code> .
Pipeline is rerunning unnecessarily	To ensure that steps only rerun when their underlying data or scripts change, decouple your source-code directories for each step. If you use the same source directory for multiple steps, you may experience unnecessary reruns. Use the <code>source_directory</code> parameter on a pipeline step object to point to your isolated directory for that step, and ensure you aren't using the same <code>source_directory</code> path for multiple steps.
Step slowing down over training epochs or other looping behavior	Try switching any file writes, including logging, from <code>as_mount()</code> to <code>as_upload()</code> . The <code>mount</code> mode uses a remote virtualized filesystem and uploads the entire file each time it is appended to.

Authentication errors

If you perform a management operation on a compute target from a remote job, you will receive one of the following errors:

```
{"code": "Unauthorized", "statusCode": 401, "message": "Unauthorized", "details": [{"code": "InvalidOrExpiredToken", "message": "The request token was either invalid or expired. Please try again with a valid token."}]}
```

```
{"error": {"code": "AuthenticationFailed", "message": "Authentication failed."}}
```

For example, you will receive an error if you try to create or attach a compute target from an ML Pipeline that is submitted for remote execution.

Debugging techniques

There are three major techniques for debugging pipelines:

- Debug individual pipeline steps on your local computer
- Use logging and Application Insights to isolate and diagnose the source of the problem
- Attach a remote debugger to a pipeline running in Azure

Debug scripts locally

One of the most common failures in a pipeline is that the domain script does not run as intended, or contains runtime errors in the remote compute context that are difficult to debug.

Pipelines themselves cannot be run locally, but running the scripts in isolation on your local machine allows you to debug faster because you don't have to wait for the compute and environment build process. Some development work is required to do this:

- If your data is in a cloud datastore, you will need to download data and make it available to your script. Using a small sample of your data is a good way to cut down on runtime and quickly get feedback on script behavior
- If you are attempting to simulate an intermediate pipeline step, you may need to manually build the object types that the particular script is expecting from the prior step
- You will also need to define your own environment, and replicate the dependencies defined in your remote compute environment

Once you have a script setup to run on your local environment, it is much easier to do debugging tasks like:

- Attaching a custom debug configuration
- Pausing execution and inspecting object-state
- Catching type or logical errors that won't be exposed until runtime

TIP

Once you can verify that your script is running as expected, a good next step is running the script in a single-step pipeline before attempting to run it in a pipeline with multiple steps.

Configure, write to, and review pipeline logs

Testing scripts locally is a great way to debug major code fragments and complex logic before you start building a pipeline, but at some point you will likely need to debug scripts during the actual pipeline run itself, especially when diagnosing behavior that occurs during the interaction between pipeline steps. We recommend liberal use of `print()` statements in your step scripts so that you can see object state and expected values during remote execution, similar to how you would debug JavaScript code.

Logging options and behavior

The table below provides information for different debug options for pipelines. It isn't an exhaustive list, as other options exist besides just the Azure Machine Learning, Python, and OpenCensus ones shown here.

LIBRARY	TYPE	EXAMPLE	DESTINATION	RESOURCES
Azure Machine Learning SDK	Metric	run.log(name, val)	Azure Machine Learning Portal UI	How to track experiments <code>azureml.core.Run</code> class
Python printing/logging	Log	print(val) logging.info(message)	Driver logs, Azure Machine Learning designer	How to track experiments Python logging
OpenCensus Python	Log	logger.addHandler(AzureApplicationInsights - logging.log(message))	Application Insights - traces	Debug pipelines in Application Insights OpenCensus Azure Monitor Exporters Python logging cookbook

Logging options example

```
import logging

from azureml.core.run import Run
from opencensus.ext.azure.log_exporter import AzureLogHandler

run = Run.get_context()

# Azure ML Scalar value logging
run.log("scalar_value", 0.95)

# Python print statement
print("I am a python print statement, I will be sent to the driver logs.")

# Initialize python logger
logger = logging.getLogger(__name__)
logger.setLevel(args.log_level)

# Plain python logging statements
logger.debug("I am a plain debug statement, I will be sent to the driver logs.")
logger.info("I am a plain info statement, I will be sent to the driver logs.")

handler = AzureLogHandler(connection_string='<connection string>')
logger.addHandler(handler)

# Python logging with OpenCensus AzureLogHandler
logger.warning("I am an OpenCensus warning statement, find me in Application Insights!")
logger.error("I am an OpenCensus error statement with custom dimensions", {'step_id': run.id})
```

Azure Machine Learning designer

For pipelines created in the designer, you can find the `70_driver_log` file in either the authoring page, or in the pipeline run detail page.

Enable logging for real-time endpoints

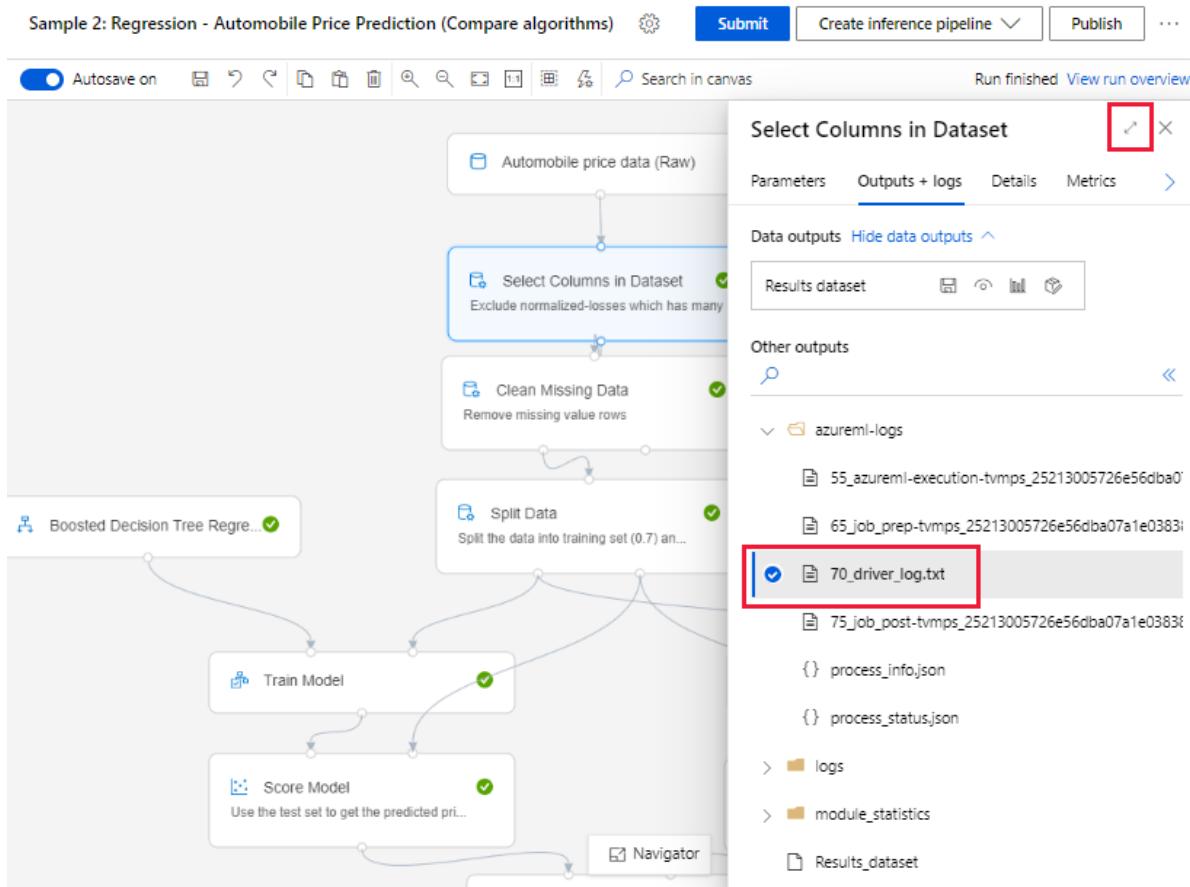
In order to troubleshoot and debug real-time endpoints in the designer, you must enable Application Insight logging using the SDK. Logging lets you troubleshoot and debug model deployment and usage issues. For more

information, see [Logging for deployed models](#).

Get logs from the authoring page

When you submit a pipeline run and stay in the authoring page, you can find the log files generated for each module as each module finishes running.

1. Select a module that has finished running in the authoring canvas.
2. In the right pane of the module, go to the **Outputs + logs** tab.
3. Expand the right pane, and select the **70_driver_log.txt** to view the file in browser. You can also download logs locally.



Get logs from pipeline runs

You can also find the log files for specific runs in the pipeline run detail page, which can be found in either the **Pipelines** or **Experiments** section of the studio.

1. Select a pipeline run created in the designer.

The screenshot shows the 'Pipelines' section of the Azure Machine Learning Studio. On the left is a navigation sidebar with icons for Home, Pipelines, Experiments, Datasets, Models, Metrics, and Help. The main area shows a table of pipeline runs. The first four rows are highlighted with red boxes:

- Run 1, Created time: 12/12/2019, 2:41:00 PM, Status: Completed, Description: Sample 2: Regression - Autom... Experiment: Sample2, Submitted by: Blanca Li, Tags: azureml.Designer: true, azure...
- Run 1, Created time: 12/12/2019, 12:57:12 PM, Status: Failed, Description: NYCTaxi_Tutorial_Pipelines Experiment: NYCTaxi_Tutorial_Pipelines, Submitted by: Blanca Li, Tags: azureml.pipelineComponent: pi...
- Run 1, Created time: 12/11/2019, 5:23:15 PM, Status: Completed, Description: Sample 1: Regression - Autom... Experiment: Sample1, Submitted by: Blanca Li, Tags: azureml.Designer: true, azure...
- Run 1, Created time: 12/11/2019, 5:00:21 PM, Status: Completed, Description: has updated train step Experiment: helloworld, Submitted by: Blanca Li, Tags: azureml.pipelineComponent: pi...

At the bottom of the table, there are navigation arrows for 'Prev' and 'Next'.

2. Select a module in the preview pane.

3. In the right pane of the module, go to the **Outputs + logs** tab.
4. Expand the right pane to view the **70_driver_log.txt** file in browser, or select the file to download the logs locally.

IMPORTANT

To update a pipeline from the pipeline run details page, you must **clone** the pipeline run to a new pipeline draft. A pipeline run is a snapshot of the pipeline. It's similar to a log file, and cannot be altered.

Application Insights

For more information on using the OpenCensus Python library in this manner, see this guide: [Debug and troubleshoot machine learning pipelines in Application Insights](#)

Interactive debugging with Visual Studio Code

In some cases, you may need to interactively debug the Python code used in your ML pipeline. By using Visual Studio Code (VS Code) and debugpy, you can attach to the code as it runs in the training environment. For more information, visit the [interactive debugging in VS Code guide](#).

Next steps

- [Debug and troubleshoot ParallelRunStep](#)
- For a complete tutorial using `ParallelRunStep`, see [Tutorial: Build an Azure Machine Learning pipeline for batch scoring](#).
- For a complete example showing automated machine learning in ML pipelines, see [Use automated ML in an Azure Machine Learning pipeline in Python](#).
- See the SDK reference for help with the `azureml-pipelines-core` package and the `azureml-pipelines-steps` package.
- See the list of [designer exceptions and error codes](#).

Collect machine learning pipeline log files in Application Insights for alerts and debugging

12/23/2020 • 4 minutes to read • [Edit Online](#)

The [OpenCensus python library](#) can be used to route logs to Application Insights from your scripts. Aggregating logs from pipeline runs in one place allows you to build queries and diagnose issues. Using Application Insights will allow you to track logs over time and compare pipeline logs across runs.

Having your logs in once place will provide a history of exceptions and error messages. Since Application Insights integrates with Azure Alerts, you can also create alerts based on Application Insights queries.

Prerequisites

- Follow the steps to create an [Azure Machine Learning](#) workspace and [create your first pipeline](#)
- [Configure your development environment](#) to install the Azure Machine Learning SDK.
- Install the [OpenCensus Azure Monitor Exporter](#) package locally:

```
pip install opencensus-ext-azure
```

- Create an [Application Insights instance](#) (this doc also contains information on getting the connection string for the resource)

Getting Started

This section is an introduction specific to using OpenCensus from an Azure Machine Learning pipeline. For a detailed tutorial, see [OpenCensus Azure Monitor Exporters](#)

Add a PythonScriptStep to your Azure ML Pipeline. Configure your [RunConfiguration](#) with the dependency on `opencensus-ext-azure`. Configure the `APPLICATIONINSIGHTS_CONNECTION_STRING` environment variable.

```

from azureml.core.conda_dependencies import CondaDependencies
from azureml.core.runconfig import RunConfiguration
from azureml.pipeline.core import Pipeline
from azureml.pipeline.steps import PythonScriptStep

# Connecting to the workspace and compute target not shown

# Add pip dependency on OpenCensus
dependencies = CondaDependencies()
dependencies.add_pip_package("opencensus-ext-azure>=1.0.1")
run_config = RunConfiguration(conda_dependencies=dependencies)

# Add environment variable with Application Insights Connection String
# Replace the value with your own connection string
run_config.environment.environment_variables = {
    "APPLICATIONINSIGHTS_CONNECTION_STRING": 'InstrumentationKey=00000000-0000-0000-0000-000000000000'
}

# Configure step with runconfig
sample_step = PythonScriptStep(
    script_name="sample_step.py",
    compute_target=compute_target,
    runconfig=run_config
)

# Submit new pipeline run
pipeline = Pipeline(workspace=ws, steps=[sample_step])
pipeline.submit(experiment_name="Logging_Experiment")

```

Create a file called `sample_step.py`. Import the AzureLogHandler class to route logs to Application Insights. You'll also need to import the Python Logging library.

```

from opencensus.ext.azure.log_exporter import AzureLogHandler
import logging

```

Next, add the AzureLogHandler to the python logger.

```

logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
logger.addHandler(logging.StreamHandler())

# Assumes the environment variable APPLICATIONINSIGHTS_CONNECTION_STRING is already set
logger.addHandler(AzureLogHandler())
logger.warning("I will be sent to Application Insights")

```

Logging with Custom Dimensions

By default, logs forwarded to Application Insights won't have enough context to trace back to the run or experiment. To make the logs actionable for diagnosing issues, additional fields are needed.

To add these fields, Custom Dimensions can be added to provide context to a log message. One example is when someone wants to view logs across multiple steps in the same pipeline run.

Custom Dimensions make up a dictionary of key-value (stored as string, string) pairs. The dictionary is then sent to Application Insights and displayed as a column in the query results. Its individual dimensions can be used as [query parameters](#).

Helpful Context to include

FIELD	REASONING/EXAMPLE
parent_run_id	Can query logs for ones with the same parent_run_id to see logs over time for all steps, instead of having to dive into each individual step
step_id	Can query logs for ones with the same step_id to see where an issue occurred with a narrow scope to just the individual step
step_name	Can query logs to see step performance over time. Also helps to find a step_id for recent runs without diving into the portal UI
experiment_name	Can query across logs to see experiment performance over time. Also helps find a parent_run_id or step_id for recent runs without diving into the portal UI
run_url	Can provide a link directly back to the run for investigation.

Other helpful fields

These fields may require additional code instrumentation, and aren't provided by the run context.

FIELD	REASONING/EXAMPLE
build_url/build_version	If using CI/CD to deploy, this field can correlate logs to the code version that provided the step and pipeline logic. This link can further help to diagnose issues, or identify models with specific traits (log/metric values)
run_type	Can differentiate between different model types, or training vs. scoring runs

Creating a Custom Dimensions dictionary

```
from azureml.core import Run

run = Run.get_context(allow_offline=False)

custom_dimensions = {
    "parent_run_id": run.parent.id,
    "step_id": run.id,
    "step_name": run.name,
    "experiment_name": run.experiment.name,
    "run_url": run.parent.get_portal_url(),
    "run_type": "training"
}

# Assumes AzureLogHandler was already registered above
logger.info("I will be sent to Application Insights with Custom Dimensions", custom_dimensions)
```

OpenCensus Python logging considerations

The OpenCensus AzureLogHandler is used to route Python logs to Application Insights. As a result, Python logging nuances should be considered. When a logger is created, it has a default log level and will show logs greater than or equal to that level. A good reference for using Python logging features is the [Logging Cookbook](#).

The `APPLICATIONINSIGHTS_CONNECTION_STRING` environment variable is needed for the OpenCensus library. We recommend setting this environment variable instead of passing it in as a pipeline parameter to avoid passing around plaintext connection strings.

Querying logs in Application Insights

The logs routed to Application Insights will show up under 'traces' or 'exceptions'. Be sure to adjust your time window to include your pipeline run.

The screenshot shows the Azure Application Insights Log Analytics interface. At the top, there are buttons for 'Run', 'Time range : Last 24 hours', 'Save', 'Copy link', 'New alert rule', 'Export', 'Pin to dashboard', and 'Prettify query'. Below this is a search bar with the word 'traces'. The main area displays a table of log results. The table has columns: timestamp [UTC], message, severityLevel, itemType, and customDimensions. A dropdown menu is open over the 'customDimensions' column, showing a dictionary entry: `{"lineNumber": "67", "fileName": "`. Underneath this, the table lists various log entries with their corresponding values for each dimension. The log entries include:

customDimensions	value
build_id	12345
build_url	<a href="https://dev.azure.com/<your org here>/<your project here>/_build/results?buildId=12345&view=results">https://dev.azure.com/<your org here>/<your project here>/_build/results?buildId=12345&view=results
experiment_name	basemodel training
fileName	[redacted]
level	INFO
lineNumber	67
module	sample_step
parent_run_id	931024c2-3720-11ea-b247-c49deda841c1
process	MainProcess
run_type	training
step_id	93104bca-3720-11ea-9f85-c49deda841c1
step_name	preprocess_data

The result in Application Insights will show the log message and level, file path, and code line number. It will also show any custom dimensions included. In this image, the `customDimensions` dictionary shows the key/value pairs from the previous [code sample](#).

Additional helpful queries

Some of the queries below use 'customDimensions.Level'. These severity levels correspond to the level the Python log was originally sent with. For additional query information, see [Azure Monitor Log Queries](#).

USE CASE	QUERY
Log results for specific custom dimension, for example 'parent_run_id'	<pre>traces where customDimensions.parent_run_id == '931024c2- 3720-11ea-b247-c49deda841c1'</pre>
Log results for all training runs over the last 7 days	<pre>traces where timestamp > ago(7d) and customDimensions.run_type == 'training'</pre>

USE CASE	QUERY
Log results with severityLevel Error from the last 7 days	<pre>traces where timestamp > ago(7d) and customDimensions.Level == 'ERROR'</pre>
Count of log results with severityLevel Error over the last 7 days	<pre>traces where timestamp > ago(7d) and customDimensions.Level == 'ERROR' summarize count()</pre>

Next Steps

Once you have logs in your Application Insights instance, they can be used to set [Azure Monitor alerts](#) based on query results.

You can also add results from queries to an [Azure Dashboard](#) for additional insights.

Troubleshooting the ParallelRunStep

12/23/2020 • 9 minutes to read • [Edit Online](#)

In this article, you learn how to troubleshoot when you get errors using the `ParallelRunStep` class from the [Azure Machine Learning SDK](#).

For general tips on troubleshooting a pipeline, see [Troubleshooting machine learning pipelines](#).

Testing scripts locally

Your `ParallelRunStep` runs as a step in ML pipelines. You may want to [test your scripts locally](#) as a first step.

Script requirements

The script for a `ParallelRunStep` *must contain* two functions:

- `init()` : Use this function for any costly or common preparation for later inference. For example, use it to load the model into a global object. This function will be called only once at beginning of process.
- `run(mini_batch)` : The function will run for each `mini_batch` instance.
 - `mini_batch` : `ParallelRunStep` will invoke `run` method and pass either a list or pandas `DataFrame` as an argument to the method. Each entry in `mini_batch` will be a file path if input is a `FileDataset` or a pandas `DataFrame` if input is a `TabularDataset`.
 - `response` : `run()` method should return a pandas `DataFrame` or an array. For `append_row` `output_action`, these returned elements are appended into the common output file. For `summary_only`, the contents of the elements are ignored. For all output actions, each returned output element indicates one successful run of input element in the input mini-batch. Make sure that enough data is included in `run` result to map input to run output result. Run output will be written in output file and not guaranteed to be in order, you should use some key in the output to map it to input.

```

%%writefile digit_identification.py
# Snippets from a sample script.
# Refer to the accompanying digit_identification.py
# (https://github.com/Azure/MachineLearningNotebooks/tree/master/how-to-use-azureml/machine-learning-
pipelines/parallel-run)
# for the implementation script.

import os
import numpy as np
import tensorflow as tf
from PIL import Image
from azureml.core import Model

def init():
    global g_tf_sess

    # Pull down the model from the workspace
    model_path = Model.get_model_path("mnist")

    # Construct a graph to execute
    tf.reset_default_graph()
    saver = tf.train.import_meta_graph(os.path.join(model_path, 'mnist-tf.model.meta'))
    g_tf_sess = tf.Session()
    saver.restore(g_tf_sess, os.path.join(model_path, 'mnist-tf.model'))

def run(mini_batch):
    print(f'run method start: {__file__}, run({mini_batch})')
    resultList = []
    in_tensor = g_tf_sess.graph.get_tensor_by_name("network/X:0")
    output = g_tf_sess.graph.get_tensor_by_name("network/output/MatMul:0")

    for image in mini_batch:
        # Prepare each image
        data = Image.open(image)
        np_im = np.array(data).reshape((1, 784))
        # Perform inference
        inference_result = output.eval(feed_dict={in_tensor: np_im}, session=g_tf_sess)
        # Find the best probability, and add it to the result list
        best_result = np.argmax(inference_result)
        resultList.append("{}: {}".format(os.path.basename(image), best_result))

    return resultList

```

If you have another file or folder in the same directory as your inference script, you can reference it by finding the current working directory.

```

script_dir = os.path.realpath(os.path.join(__file__, '..'))
file_path = os.path.join(script_dir, "<file_name>")

```

Parameters for ParallelRunConfig

`ParallelRunConfig` is the major configuration for `ParallelRunStep` instance within the Azure Machine Learning pipeline. You use it to wrap your script and configure necessary parameters, including all of the following entries:

- `entry_script`: A user script as a local file path that will be run in parallel on multiple nodes. If `source_directory` is present, use a relative path. Otherwise, use any path that's accessible on the machine.
- `mini_batch_size`: The size of the mini-batch passed to a single `run()` call. (optional; the default value is `10` files for `FileDataset` and `1MB` for `TabularDataset`.)
 - For `FileDataset`, it's the number of files with a minimum value of `1`. You can combine multiple files into one mini-batch.

- For `TabularDataset`, it's the size of data. Example values are `1024`, `1024KB`, `10MB`, and `1GB`. The recommended value is `1MB`. The mini-batch from `TabularDataset` will never cross file boundaries. For example, if you have .csv files with various sizes, the smallest file is 100 KB and the largest is 10 MB. If you set `mini_batch_size = 1MB`, then files with a size smaller than 1 MB will be treated as one mini-batch. Files with a size larger than 1 MB will be split into multiple mini-batches.
- `error_threshold` : The number of record failures for `TabularDataset` and file failures for `FileDataset` that should be ignored during processing. If the error count for the entire input goes above this value, the job will be aborted. The error threshold is for the entire input and not for individual mini-batch sent to the `run()` method. The range is `[-1, int.max]`. The `-1` part indicates ignoring all failures during processing.
- `output_action` : One of the following values indicates how the output will be organized:
 - `summary_only` : The user script will store the output. `ParallelRunStep` will use the output only for the error threshold calculation.
 - `append_row` : For all inputs, only one file will be created in the output folder to append all outputs separated by line.
- `append_row_file_name` : To customize the output file name for `append_row` `output_action` (optional; default value is `parallel_run_step.txt`).
- `source_directory` : Paths to folders that contain all files to execute on the compute target (optional).
- `compute_target` : Only `AmlCompute` is supported.
- `node_count` : The number of compute nodes to be used for running the user script.
- `process_count_per_node` : The number of processes per node. Best practice is to set to the number of GPU or CPU one node has (optional; default value is `1`).
- `environment` : The Python environment definition. You can configure it to use an existing Python environment or to set up a temporary environment. The definition is also responsible for setting the required application dependencies (optional).
- `logging_level` : Log verbosity. Values in increasing verbosity are: `WARNING`, `INFO`, and `DEBUG`. (optional; the default value is `INFO`)
- `run_invocation_timeout` : The `run()` method invocation timeout in seconds. (optional; default value is `60`)
- `run_max_try` : Maximum try count of `run()` for a mini-batch. A `run()` is failed if an exception is thrown, or nothing is returned when `run_invocation_timeout` is reached (optional; default value is `3`).

You can specify `mini_batch_size`, `node_count`, `process_count_per_node`, `logging_level`, `run_invocation_timeout`, and `run_max_try` as `PipelineParameter`, so that when you resubmit a pipeline run, you can fine-tune the parameter values. In this example, you use `PipelineParameter` for `mini_batch_size` and `Process_count_per_node` and you will change these values when resubmit a run later.

Parameters for creating the ParallelRunStep

Create the `ParallelRunStep` by using the script, environment configuration, and parameters. Specify the compute target that you already attached to your workspace as the target of execution for your inference script. Use `ParallelRunStep` to create the batch inference pipeline step, which takes all the following parameters:

- `name` : The name of the step, with the following naming restrictions: unique, 3-32 characters, and regex `^[a-z]([-a-z0-9]*[a-z0-9])?$.`
- `parallel_run_config` : A `ParallelRunConfig` object, as defined earlier.
- `inputs` : One or more single-typed Azure Machine Learning datasets to be partitioned for parallel processing.
- `side_inputs` : One or more reference data or datasets used as side inputs without need to be partitioned.
- `output` : A `PipelineData` object that corresponds to the output directory.
- `arguments` : A list of arguments passed to the user script. Use `unknown_args` to retrieve them in your entry script (optional).
- `allow_reuse` : Whether the step should reuse previous results when run with the same settings/inputs. If this

parameter is `False`, a new run will always be generated for this step during pipeline execution. (optional; the default value is `True`.)

```
from azureml.pipeline.steps import ParallelRunStep

parallelrun_step = ParallelRunStep(
    name="predict-digits-mnist",
    parallel_run_config=parallel_run_config,
    inputs=[input_mnist_ds_consumption],
    output=output_dir,
    allow_reuse=True
)
```

Debugging scripts from remote context

The transition from debugging a scoring script locally to debugging a scoring script in an actual pipeline can be a difficult leap. For information on finding your logs in the portal, see [machine learning pipelines section on debugging scripts from a remote context](#). The information in that section also applies to a ParallelRunStep.

For example, the log file `70_driver_log.txt` contains information from the controller that launches the ParallelRunStep code.

Because of the distributed nature of ParallelRunStep jobs, there are logs from several different sources. However, two consolidated files are created that provide high-level information:

- `~/logs/job_progress_overview.txt`: This file provides a high-level info about the number of mini-batches (also known as tasks) created so far and number of mini-batches processed so far. At this end, it shows the result 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. Includes task creation, progress monitoring, the run result.

Logs generated from entry script using EntryScript helper and print statements will be found in following files:

- `~/logs/user/entry_script_log/<ip_address>/<process_name>.log.txt`: These files are the logs written from entry_script using EntryScript helper.
- `~/logs/user/stdout/<ip_address>/<process_name>.stdout.txt`: These files are the logs from stdout (e.g. print statement) of entry_script.
- `~/logs/user/stderr/<ip_address>/<process_name>.stderr.txt`: These files are the logs from stderr of entry_script.

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/`: Contains full stack traces of exceptions thrown while loading and running 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, successfully processed items count, and failed item count.

- The start time, duration, process time and run method time.

You can also find information on the resource usage of the processes for each worker. This information is in CSV format and is located at `~/logs/sys/perf/<ip_address>/node_resource_usage.csv`. Information about each process is available under `~/logs/sys/perf/<ip_address>/processes_resource_usage.csv`.

How do I log from my user script from a remote context?

ParallelRunStep may run multiple processes on one node based on `process_count_per_node`. In order to organize logs from each process on node and combine print and log statement, we recommend using ParallelRunStep logger as shown below. You get a logger from EntryScript and make the logs show up in `logs/user` folder in the portal.

A sample entry script using the logger:

```
from azureml_user.parallel_run import EntryScript

def init():
    """ Initialize the node."""
    entry_script = EntryScript()
    logger = entry_script.logger
    logger.debug("This will show up in files under logs/user on the Azure portal.")

def run(mini_batch):
    """ Accept and return the list back."""
    # This class is in singleton pattern and will return same instance as the one in init()
    entry_script = EntryScript()
    logger = entry_script.logger
    logger.debug(f"__file__: {mini_batch}.")
    ...

    return mini_batch
```

How could I pass a side input such as, a file or file(s) containing a lookup table, to all my workers?

User can pass reference data to script using `side_inputs` parameter of ParallelRunStep. All datasets provided as `side_inputs` will be mounted on each worker node. User can get the location of mount by passing argument.

Construct a [Dataset](#) containing the reference data and register it with your workspace. Pass it to the `side_inputs` parameter of your `ParallelRunStep`. Additionally, you can add its path in the `arguments` section to easily access its mounted path:

```
label_config = label_ds.as_named_input("labels_input")
batch_score_step = ParallelRunStep(
    name=parallel_step_name,
    inputs=[input_images.as_named_input("input_images")],
    output=output_dir,
    arguments=["--labels_dir", label_config],
    side_inputs=[label_config],
    parallel_run_config=parallel_run_config,
)
```

After that you can access it in your inference script (for example, in your `init()` method) as follows:

```
parser = argparse.ArgumentParser()
parser.add_argument('--labels_dir', dest="labels_dir", required=True)
args, _ = parser.parse_known_args()

labels_path = args.labels_dir
```

How to use input datasets with service principal authentication?

User can pass input datasets with service principal authentication used in workspace. Using such dataset in ParallelRunStep requires that dataset to be registered for it to construct ParallelRunStep configuration.

```
service_principal = ServicePrincipalAuthentication(  
    tenant_id="***",  
    service_principal_id="***",  
    service_principal_password="***")  
  
ws = Workspace(  
    subscription_id="***",  
    resource_group="***",  
    workspace_name="***",  
    auth=service_principal  
)  
  
default_blob_store = ws.get_default_datastore() # or Datastore(ws, '***datastore-name***')  
ds = Dataset.File.from_files(default_blob_store, '**path**')  
registered_ds = ds.register(ws, '***dataset-name***', create_new_version=True)
```

Next steps

- See these [Jupyter notebooks demonstrating Azure Machine Learning pipelines](#)
- See the SDK reference for help with the [azureml-pipeline-steps](#) package. View reference [documentation](#) for ParallelRunStep class.
- Follow the [advanced tutorial](#) on using pipelines with ParallelRunStep. The tutorial shows how to pass another file as a side input.

Use GitHub Actions with Azure Machine Learning

12/23/2020 • 4 minutes to read • [Edit Online](#)

Get started with [GitHub Actions](#) to train a model on Azure Machine Learning.

NOTE

GitHub Actions for Azure Machine Learning are provided as-is, and are not fully supported by Microsoft. If you encounter problems with a specific action, open an issue in the repository for the action. For example, if you encounter a problem with the aml-deploy action, report the problem in the <https://github.com/Azure/aml-deploy> repo.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub account. If you don't have one, sign up for [free](#).

Workflow file overview

A workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

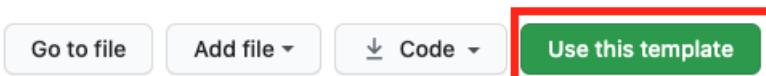
The file has four sections:

SECTION	TASKS
Authentication	1. Define a service principal. 2. Create a GitHub secret.
Connect	1. Connect to the machine learning workspace. 2. Connect to a compute target.
Run	1. Submit a training run.
Deploy	1. Register model in Azure Machine Learning registry. 1. Deploy the model.

Create repository

Create a new repository off the [ML Ops with GitHub Actions and Azure Machine Learning template](#).

1. Open the [template](#) on GitHub.
2. Select **Use this template**.



3. Create a new repository from the template. Set the repository name to `m1-learning` or a name of your choice.

Generate deployment credentials

You can 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>",
  (...),
}
```

Configure the GitHub secret

1. In [GitHub](#), browse your repository, select [Settings > Secrets > Add a new secret](#).
2. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.

Connect to the workspace

Use the [Azure Machine Learning Workspace action](#) to connect to your Azure Machine Learning workspace.

```
- name: Connect/Create Azure Machine Learning Workspace
  id: aml_workspace
  uses: Azure/aml-workspace@v1
  with:
    azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
```

By default, the action expects a `workspace.json` file. If your JSON file has a different name, you can specify it with the `parameters_file` input parameter. If there is not a file, a new one will be created with the repository name.

```
- name: Connect/Create Azure Machine Learning Workspace
  id: aml_workspace
  uses: Azure/aml-workspace@v1
  with:
    azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
    parameters_file: "alternate_workspace.json"
```

The action writes the workspace Azure Resource Manager (ARM) properties to a config file, which will be picked by all future Azure Machine Learning GitHub Actions. The file is saved to `GITHUB_WORKSPACE/ml_arm_config.json`.

Connect to a Compute Target in Azure Machine Learning

Use the [Azure Machine Learning Compute action](#) to connect to a compute target in Azure Machine Learning. If the compute target exists, the action will connect to it. Otherwise the action will create a new compute target. The [AML](#)

[Compute action](#) only supports the Azure ML compute cluster and Azure Kubernetes Service (AKS).

```
- name: Connect/Create Azure Machine Learning Compute Target
  id: aml_compute_training
  uses: Azure/aml-compute@v1
  with:
    azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
```

Submit training run

Use the [Azure Machine Learning Training action](#) to submit a ScriptRun, an Estimator or a Pipeline to Azure Machine Learning.

```
- name: Submit training run
  id: aml_run
  uses: Azure/aml-run@v1
  with:
    azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
```

Register model in registry

Use the [Azure Machine Learning Register Model action](#) to register a model to Azure Machine Learning.

```
- name: Register model
  id: aml_registermodel
  uses: Azure/aml-registermodel@v1
  with:
    azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
    run_id: ${{ stepsaml_run.outputs.run_id }}
    experiment_name: ${{ stepsaml_run.outputs.experiment_name }}
```

Deploy model to Azure Machine Learning to ACI

Use the [Azure Machine Learning Deploy action](#) to deploys a model and create an endpoint for the model. You can also use the Azure Machine Learning Deploy to deploy to Azure Kubernetes Service. See [this sample workflow](#) for a model that deploys to Azure Kubernetes Service.

```
- name: Deploy model
  id: aml_deploy
  uses: Azure/aml-deploy@v1
  with:
    azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
    model_name: ${{ stepsaml_registermodel.outputs.model_name }}
    model_version: ${{ stepsaml_registermodel.outputs.model_version }}
```

Complete example

Train your model and deploy to Azure Machine Learning.

```

# Actions train a model on Azure Machine Learning
name: Azure Machine Learning training and deployment
on:
  push:
    branches:
      - master
    # paths:
    #   - 'code/*'
jobs:
  train:
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
      - name: Check Out Repository
        id: checkout_repository
        uses: actions/checkout@v2

      # Connect or Create the Azure Machine Learning Workspace
      - name: Connect/Create Azure Machine Learning Workspace
        id: aml_workspace
        uses: Azure/aml-workspace@v1
        with:
          azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}

      # Connect or Create a Compute Target in Azure Machine Learning
      - name: Connect/Create Azure Machine Learning Compute Target
        id: aml_compute_training
        uses: Azure/aml-compute@v1
        with:
          azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}

      # Submit a training run to the Azure Machine Learning
      - name: Submit training run
        id: aml_run
        uses: Azure/aml-run@v1
        with:
          azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}

      # Register model in Azure Machine Learning model registry
      - name: Register model
        id: aml_registermodel
        uses: Azure/aml-registermodel@v1
        with:
          azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
          run_id: ${{ steps.aml_run.outputs.run_id }}
          experiment_name: ${{ steps.aml_run.outputs.experiment_name }}

      # Deploy model in Azure Machine Learning to ACI
      - name: Deploy model
        id: aml_deploy
        uses: Azure/aml-deploy@v1
        with:
          azure_credentials: ${{ secrets.AZURE_CREDENTIALS }}
          model_name: ${{ steps.aml_registermodel.outputs.model_name }}
          model_version: ${{ steps.aml_registermodel.outputs.model_version }}

```

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 and run machine learning pipelines with Azure Machine Learning SDK

Manage and increase quotas for resources with Azure Machine Learning

12/23/2020 • 8 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.
- Private endpoint and DNS quotas.

Along with managing quotas, you can learn how to [plan and manage costs for 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 pipelines
- Virtual machines
- Azure Container Instances
- Azure Storage

IMPORTANT

Limits are subject to change. For the latest information, see [Azure subscription and service limits, quotas, and constraints](#) for all of Azure.

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) as well as 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 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.
- **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. These are shared between a training cluster and a compute instance. (A compute instance is considered a single-node cluster for quota purposes.)

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 additional limits in the platform. Please 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 setup as a non communication-enabled pool (i.e. cannot run MPI jobs)	100 nodes but configurable up to 65000 nodes
Nodes in a single Parallel Run Step run on an Azure Machine Learning Compute (AmlCompute) cluster	100 nodes but configurable up to 65000 nodes if your cluster is setup to scale per above

RESOURCE OR ACTION	MAXIMUM LIMIT
Nodes in a single Azure Machine Learning Compute (AmlCompute) cluster setup as a communication-enabled pool	300 nodes but configurable up to 4000 nodes
Nodes in a single Azure Machine Learning Compute (AmlCompute) cluster setup 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
GPU MPI processes per node	1-4
GPU workers per node	1-4
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 run starts and when it finishes. Completed runs persist indefinitely. Data for runs not completed within the maximum lifetime is not 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 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 does not 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 per Azure Active Directory tenant	Unlimited
Coadministrators per subscription	Unlimited
Resource groups per subscription	980

RESOURCE	LIMIT
Azure Resource Manager API request size	4,194,304 bytes
Tags per subscription ¹	50
Unique tag calculations per subscription ¹	10,000
Subscription-level deployments per location	800 ²

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. Resource Manager returns a [list of unique tag name and values](#) in the subscription only when the number of tags is 10,000 or less. You still can find a resource by tag when the number exceeds 10,000.

²If you reach the limit of 800 deployments, delete deployments that are no longer needed from the history. To delete subscription-level deployments, use [Remove-AzDeployment](#) or [az deployment sub delete](#).

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	Unallocated cores: 0, Maximum: 100
build19-weurope (build19)	100	Unallocated cores: 0, Maximum: 100
danielsc (aml-workshop)	100	Unallocated cores: 0, Maximum: 100
ignite2019 (ignite2019)	100	Unallocated cores: 0, Maximum: 100
keras (azureml-webinar)	100	Unallocated cores: 0, Maximum: 100
maxluk-azml (azureml-webinar)	100	Unallocated cores: 0, Maximum: 100
pytorch-bert-squad (azureml-webinar)	100	Unallocated cores: 0, Maximum: 100
test (amlworkspace)	100	Unallocated cores: 0, Maximum: 100
test-sku-ws (azureml)	50	<input type="text" value="50"/>
test2 (aml-workshop)	100	Unallocated cores: 0, Maximum: 100
testws1 (azureml)	100	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 your usage and quotas

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](#).

Private endpoint and private DNS quota increases

There are limits on the number of private endpoints and private DNS zones that you can create in a subscription.

Azure Machine Learning creates resources in your (customer) subscription, but some scenarios create resources in a Microsoft-owned subscription.

In the following scenarios, you might need to request a quota allowance in the Microsoft-owned subscription:

- Azure Private Link enabled workspace with a customer-managed key (CMK)
- Azure Container Registry for the workspace behind your virtual network
- Attaching a Private Link enabled Azure Kubernetes Service cluster to your workspace

To request an allowance for these scenarios, use the following steps:

1. [Create an Azure support request](#) and select the following options in the **Basics** section:

FIELD	SELECTION
Issue type	Technical
Service	My services. Then select Machine Learning in the drop-down list.
Problem type	Workspace Configuration and Security
Problem subtype	Private Endpoint and Private DNS Zone allowance request

2. In the **Details** section, use the **Description** field to provide the Azure region and the scenario that you plan to use. If you need to request quota increases for multiple subscriptions, list the subscription IDs in this field.
3. Select **Create** to create the request.

Create a new support request to get assistance with billing, subscription, technical (including advisory) or quota management issues.

Complete the Basics tab by selecting the options that best describe your problem. Providing detailed, accurate information can help to solve your issues faster.

* Issue type

* Subscription Can't find your subscription? [Show more](#) ⓘ

* Service My services All services

* Resource

* Summary ✓

* Problem type

* Problem subtype ^

All problem types

Authentication & Role Based Access (RBAC)

Data Encryption

Private Endpoint and Private DNS Zone allowance request

Problem provisioning or managing workspace

Virtual Network and Private Link Configuration

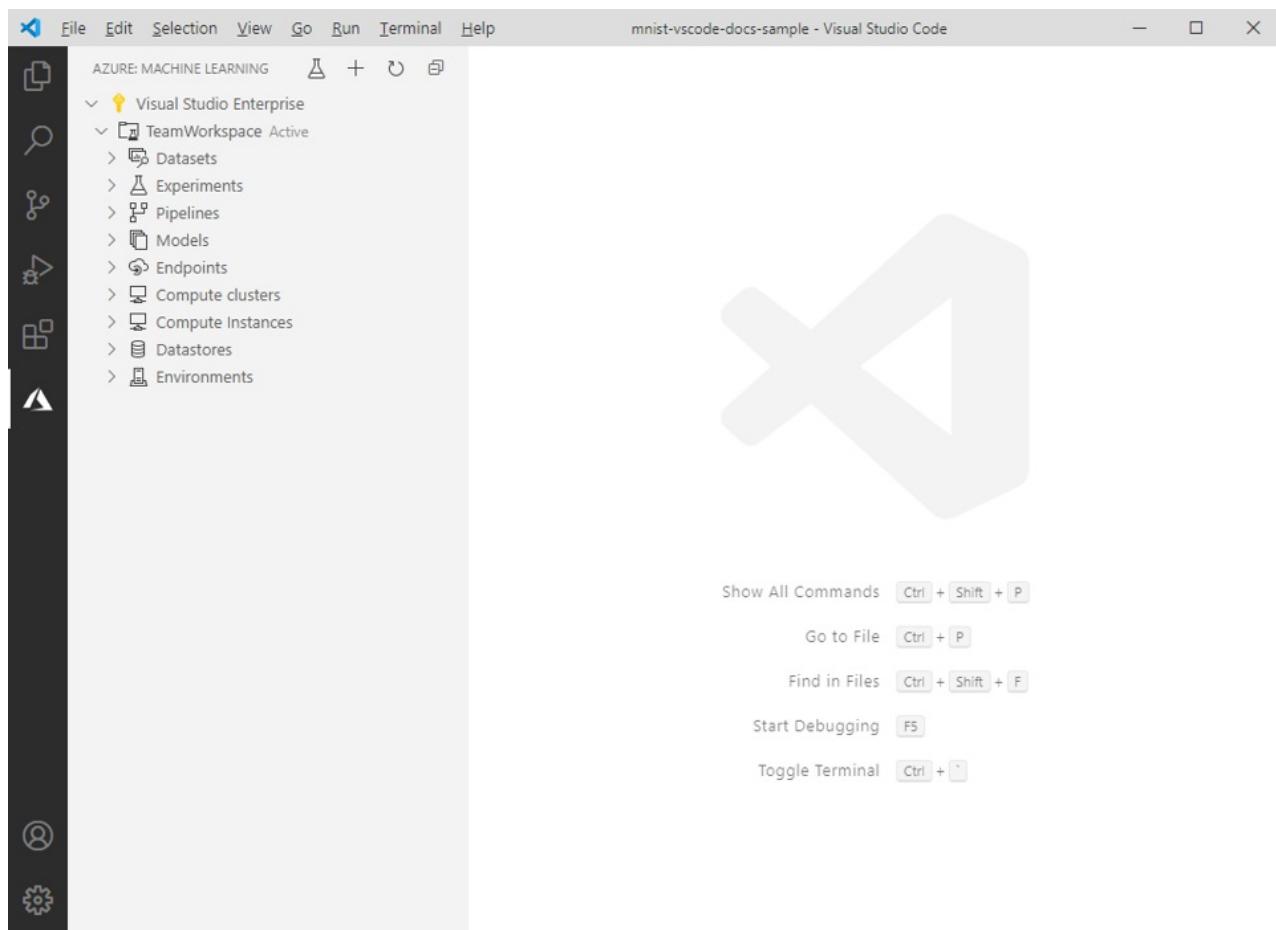
Next steps

- [Plan and manage costs for Azure Machine Learning](#)

Manage Azure Machine Learning resources with the VS Code Extension (preview)

12/23/2020 • 13 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](#).
- VS Code Azure Machine Learning Extension. Follow the [Azure Machine Learning VS Code extension installation guide](#) to install the extension.

All of the processes below assume that you are in the Azure Machine Learning view in Visual Studio Code. To launch the extension, select the **Azure** icon in the VS Code activity bar.

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. In the prompt:
 - a. Provide a name for your workspace

- b. Choose your Azure subscription
- c. Choose or create a new resource group to provision the workspace in
- d. Select the location where to provision the workspace.

Alternative methods to create a workspace include:

- Open the command palette **View > Command Palette** and enter into the text prompt **Azure ML: Create Workspace**.
- Click the  icon at the top of the Azure Machine Learning view.
- Create a new workspace when prompted to select a workspace during the provisioning of other resources.

Remove a 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.

Datastores

The VS Code extension currently supports datastores of the following types:

- Azure File Share
- Azure Blob Storage

When you create a workspace, a datastore is created for each of these types.

For more information, see [datastores](#).

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 **Register datastore**.
4. In the prompt:
 - a. Provide a name for your datastore.
 - b. Choose the datastore type.
 - c. Select your storage resource. You can either choose a storage resource that's associated with your workspace or select from any valid storage resource in your Azure subscriptions.
 - d. Choose the container where your data is inside the previously selected storage resource.
5. A configuration file appears in VS Code. If you're satisfied with your configuration file, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

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. Select the datastore you want to:
 - *Set as default*: Whenever you run experiments, this is the datastore that will be used.
 - *Inspect read-only settings*.
 - *Modify*: Change the authentication type and credentials. Supported authentication types include account

key and SAS token.

Datasets

The extension currently supports the following dataset types:

- *Tabular*: Allows you to materialize data into a DataFrame (Pandas or PySpark).
- *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 datastore under.
3. Right-click the **Datasets** node and select **Create dataset**.
4. In the prompt:
 - a. Choose the dataset type
 - b. Define whether the data is located on your PC or on the web
 - c. Provide the location of your data. This can either be a single file or a directory containing your data files.
 - d. Choose the datastore you want to upload your data to.
 - e. Provide a prefix that helps identify your dataset in the datastore.

Version datasets

When building machine learning models, as data changes, you may want to version your dataset. To do so in the VS Code extension:

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 version and select **Create New Version**.
5. In the prompt:
 - a. Select the dataset type
 - b. Define whether the data is located on your PC or on the web.
 - c. Provide the location of your data. This can either be a single file or a directory containing your data files.
 - d. Choose the datastore you want to upload your data to.
 - e. Provide a prefix that helps identify your dataset in the datastore.

View dataset properties

This option allows you to see metadata associated with a specific dataset. To do so in the VS Code extension:

1. Expand your workspace node.
2. Expand the **Datasets** node.
3. Right-click the dataset you want to inspect and select **View Dataset Properties**. This will display a configuration file with the properties of the latest dataset version.

NOTE

If you have multiple version of your 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.

Unregister datasets

To remove a dataset and all version of it, unregister it. To do so in the VS Code extension:

1. Expand your workspace node.
2. Expand the **Datasets** node.
3. Right-click the dataset you want to unregister and select **Unregister dataset**.

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. In the prompt:
 - a. Provide a name for your environment
 - b. Define your environment configuration:
 - *Curated environments*: Preconfigured environments in Azure Machine Learning. You can further customize the environment by modifying the `dependencies` property in the JSON file. Learn more about [curated environments](#).
 - *Conda dependencies file*: For Anaconda environments, the file containing your environment definition can be provided.
 - *Pip requirements file*: For pip environments, the file containing your environment definition can be provided.
 - *Existing Conda environment*: This option looks for the conda environments in your local PC and tries to build an environment from the selected environment.
 - *Custom*: Define your own channels and dependencies
 - c. A configuration file opens in the editor. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

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**.

Edit environment configurations

To edit the dependencies and configurations for a specific environment in the extension:

1. Expand the subscription node that contains your workspace.
2. Expand the **Environments** node inside your workspace.
3. Right-click the environment you want to view and select **Edit Environment**.
4. After making the modifications, if you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

Experiments

For more information, see [experiments](#).

Create experiment

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 experiment**.
4. In the prompt, provide a name for your experiment.

Run experiment

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 run.
4. Select the **Run Experiment** icon in the activity bar.
5. Select whether you want to run your experiment locally or remotely. See the [debugging guide](#) for more information on running and debugging experiments locally.
6. Choose your subscription.
7. Choose the Azure ML Workspace to run the experiment under.
8. Choose your experiment.
9. Choose or create a compute to run the experiment on.
10. Choose or create a run configuration for your experiment.

Alternatively, you can select the **Run Experiment** button at the top of the extension and configure your experiment run in the prompt.

View experiment

To view your experiment 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**.
4. A prompt appears asking you to open the experiment URL in Azure Machine Learning studio. Select **Open**.

Track run progress

As you're running your experiment, you may want to see its progress. To track the progress of a run 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 experiment node you want to track progress for.
4. Right-click the run and select **View Run in Azure portal**.
5. A prompt appears asking you to open the run URL in Azure Machine Learning studio. Select **Open**.

Download run logs & outputs

Once a run is complete, you may want to download the logs and assets such as the model generated as part of an experiment run.

1. Expand the subscription node that contains your workspace.
2. Expand the **Experiments** node inside your workspace.
3. Expand the experiment node you want to track progress for.
4. Right-click the run:
 - To download the outputs, select **Download outputs**.
 - To download the logs, select **Download logs**.

View run metadata

In the extension, you can inspect metadata such as the run configuration used for the run as well as run details.

Compute instances

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

Create compute instance

1. Expand the subscription node that contains your workspace.
2. Expand the workspace node you want to create the compute instance under.
3. Right-click the **Compute instances** node and select **Create compute instance**.
4. In the prompt:
 - a. Provide a name for your compute instance.
 - b. Select a VM size from the list.
 - c. Choose whether you want to enable SSH access.
 - a. If you enable SSH access, you'll have to also provide the public SSH key or the file containing the key. For more information, see the [guide on creating and using SSH keys on Azure](#).

Stop or restart compute instance

1. Expand the subscription node that contains your workspace.
2. Expand the **Compute instance** node inside your workspace.
3. Right-click the compute instance you want to stop or restart and select **Stop Compute instance** or **Restart compute instance** respectively.

View compute instance configuration

1. Expand the subscription node that contains your workspace.
2. Expand the **Compute instance** node inside your workspace.
3. Right-click the compute instance you want to inspect and select **View Compute instance Properties**.

Delete compute instance

1. Expand the subscription node that contains your workspace.
2. Expand the **Compute instance** node inside your workspace.
3. Right-click the compute instance you want to delete and select **Delete compute instance**.

Compute clusters

The extension supports the following compute types:

- Azure Machine Learning compute cluster
- Azure Kubernetes Service

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

Create compute

1. Expand the subscription node that contains your workspace.
2. Expand the workspace node you want to create the compute cluster under.
3. Right-click the **Compute clusters** node and select **Create Compute**.
4. In the prompt:
 - a. Choose a compute type
 - b. Choose a VM size. Learn more about [VM sizes](#).
 - c. Provide a name for your compute.

View compute configuration

1. Expand the subscription node that contains your workspace.
2. Expand the **Compute clusters** node inside your workspace.

3. Right-click the compute you want to view and select **View Compute Properties**.

Edit compute scale settings

1. Expand the subscription node that contains your workspace.
2. Expand the **Compute clusters** node inside your workspace.
3. Right-click the compute you want to edit and select **Edit Compute**.
4. A configuration file for your compute opens in the editor. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

Delete compute

1. Expand the subscription node that contains your workspace.
2. Expand the **Compute clusters** node inside your workspace.
3. Right-click the compute you want to delete and select **Delete Compute**.

Create run configuration

To create a run configuration in the extension:

1. Expand the subscription node that contains your workspace.
2. Expand the **Compute clusters** node inside your workspace.
3. Right-click the compute target you want to create the run configuration under and select **Create Run Configuration**.
4. In the prompt:
 - a. Provide a name for your compute target
 - b. Choose or create a new environment.
 - c. Type the name of the script you want to run or press **Enter** to browser for the script on your local computer.
 - d. (Optional) Chose whether you want to create a data reference for your training run. Doing so will prompt you to define a dataset in your run configuration.
 - a. Select from one of your registered datasets to link to the run configuration A configuration file for your dataset opens in the editor. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.
 - e. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

Edit run configuration

1. Expand the subscription node that contains your workspace.
2. Expand your compute cluster node in the **Compute clusters** node of your workspace.
3. Right-click the run configuration you want to edit and select **Edit Run Configuration**.
4. A configuration file for your run configuration opens in the editor. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

Delete run configuration

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Expand the compute cluster node of interest inside the **Compute clusters** node.
4. Right-click the run configuration you want to edit and select **Delete Run Configuration**.

Models

For more information, see [models](#)

Register model

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Right-click the **Models** node and select **Register Model**.
4. In the prompt:
 - a. Provide a name for your model
 - b. Choose whether your model is a file or folder.
 - c. Find the model in your local PC.
 - d. A configuration file for your model in the editor. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

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.

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**.

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**.

Endpoints

The VS Code extension supports the following deployment targets:

- Azure Container Instances
- Azure Kubernetes Service

For more information, see [web service endpoints](#).

Create deployments

NOTE

Deployment creation currently only works with Conda environments.

1. Expand the subscription node that contains your workspace.
2. Expand your workspace node.
3. Right-click the **Endpoints** node and select **Deploy Service**.
4. In the prompt:
 - a. Choose whether you want to use an already registered model or a local model file.
 - b. Select your model
 - c. Choose the deployment target you want to deploy your model to.

- d. Provide a name for your model.
- e. Provide the script to run when scoring the model.
- f. Provide a Conda dependencies file.
- g. A configuration file for your deployment appears in the editor. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.

NOTE

Alternatively, you can right-click a registered model in the *Models* node and select **Deploy Service From Registered Model**.

Delete deployments

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**.

Manage deployments

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 edit settings, select **Edit service**.
 - A configuration file for your deployment appears in the editor. If you're satisfied with your configuration, select **Save and continue** or open the VS Code command palette (**View > Command Palette**) and type **Azure ML: Save and Continue**.
 - To view deployment configuration settings, select **View service properties**.

Next steps

[Train an image classification model](#) with the VS Code extension.

Interactive debugging with Visual Studio Code

12/23/2020 • 16 minutes to read • [Edit Online](#)

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](#).
- [Docker](#)
 - Docker Desktop for Mac and Windows
 - Docker Engine for Linux.
- [Python 3](#)

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](#).

IMPORTANT

Before running your experiment locally, make sure that Docker is running.

Debug experiment locally

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 to run your experiment, 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 are 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 AML 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 AML 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.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 are 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 is 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/step"` 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 cannot be determined by additional logging. By using VS Code and the debugpy, you can attach to the code running inside the Docker container.

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](#). Note that 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', 'debugpy'])

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())
```

2. To start debugpy and wait for a connection when the service starts, add the following to the top of your `score.py` file:

```
import debugpy
# Allows other computers to attach to debugpy on this IP address and port.
debugpy.listen(('0.0.0.0', 5678))
# Wait 30 seconds for a debugger to attach. If none attaches, the script continues as normal.
debugpy.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/base:intelmpi2018.3-ubuntu16.04"
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, so please wait patiently), 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.

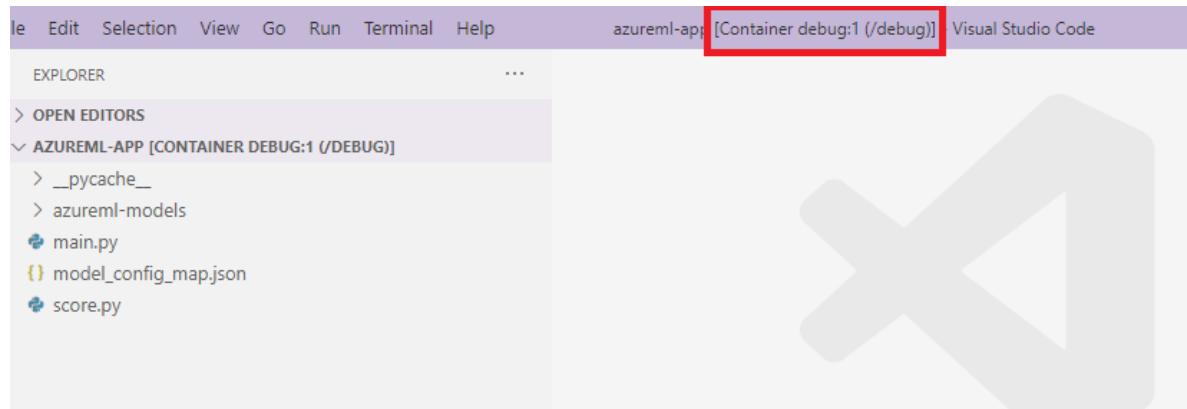
For more information on debugging and setting breakpoints, see [Debugging](#).

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 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 it's `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.



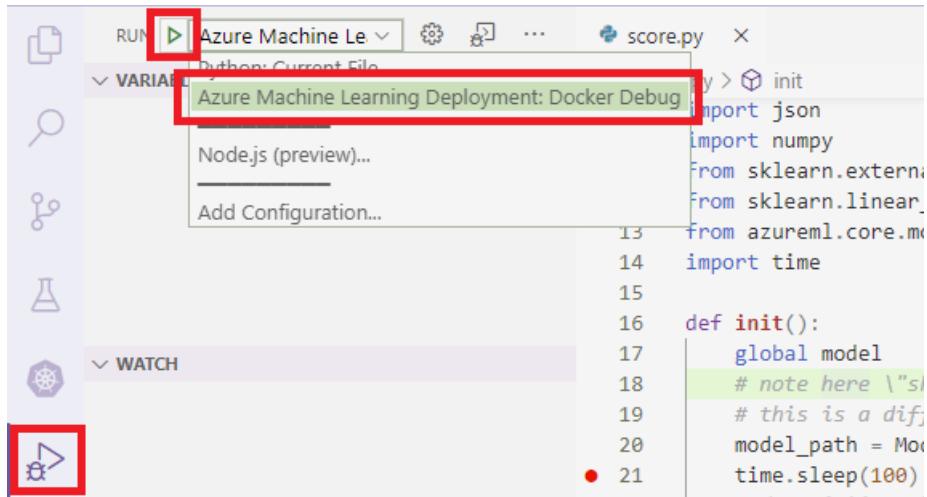
3. Inside the container, run the following command in the shell

```
runsvdir /var/run/it
```

Then you can see the following output in the shell inside your container:

```
PROBLEMS TERMINAL ... 1: runsvdir + - ×  
/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.
```

4. To attach VS Code to debug your 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** extention 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.

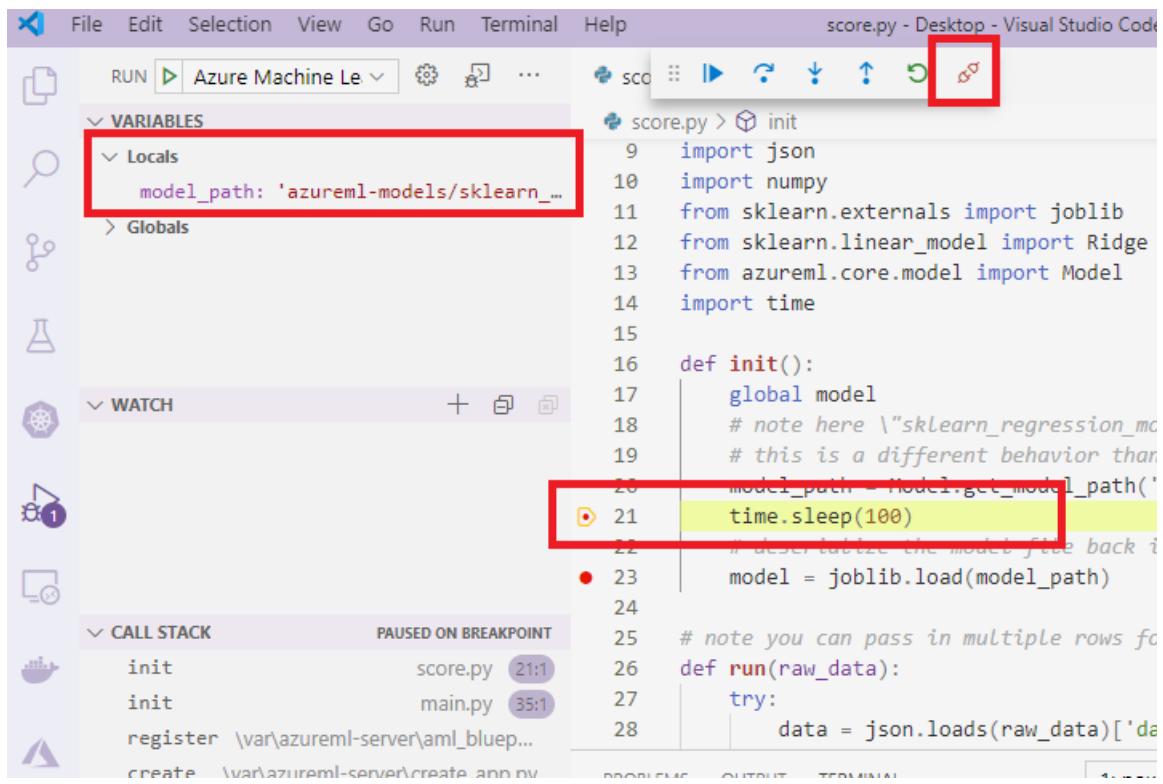


After clicking the green arrow and attaching the debugger, in the container VS Code interface you can see some new information:

The terminal window shows the following log output:

```
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...  
initializing logger  
Starting up app insights client  
Starting up request id generator  
Starting up app insight hooks  
Invoking user's init function
```

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](#)

Export or delete your Machine Learning service workspace data

12/23/2020 • 2 minutes to read • [Edit Online](#)

In Azure Machine Learning, you can export or delete your workspace data using either the portal's graphical interface or the Python SDK. This article describes both options.

NOTE

For information about viewing or deleting personal data, see [Azure Data Subject Requests for the GDPR](#). For more information about GDPR, see the [GDPR section of the Service Trust portal](#).

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Control your workspace data

In-product data stored by Azure Machine Learning is available for export and deletion. You can export and delete using Azure Machine Learning studio, CLI, and SDK. Telemetry data can be accessed through the Azure Privacy portal.

In Azure Machine Learning, personal data consists of user information in run history documents.

Delete high-level resources using the portal

When you create a workspace, Azure creates a number of resources within the resource group:

- The workspace itself
- A storage account
- A container registry
- An Applications Insights instance
- A key vault

These resources can be deleted by selecting them from the list and choosing **Delete**

Subscription (change) : mysubscription

Subscription ID : abcdeabcd-0f67-45d2-b838-b8f373d61234

Tags (change) : Click here to add tags

Name	Type	Location
designerpython	Machine Learning	East US
designerpytha32138ef	Container registry	East US
designerpython3584489984	Application Insights	East US
designerpython4720032234	Storage account	East US
designerpython6289077319	Key vault	East US

Run history documents, which may contain personal user information, are stored in the storage account in blob storage, in subfolders of `/azureml`. You can download and delete the data from the portal.

NAME	ACCESS TIER	ACCESS TIER LAST MODIFIED	LAST MODIFIED	BLOB TYPE
ComputeRecord				
Dataset				
ExperimentRun				
Labeling				

Export and delete machine learning resources using Azure Machine Learning studio

Azure Machine Learning studio provides a unified view of your machine learning resources, such as notebooks, datasets, models, and experiments. Azure Machine Learning studio emphasizes preserving a record of your data and experiments. Computational resources such as pipelines and compute resources can be deleted using the browser. For these resources, navigate to the resource in question and choose **Delete**.

Datasets can be unregistered and Experiments can be archived, but these operations don't delete the data. To entirely remove the data, datasets and run data must be deleted at the storage level. Deleting at the storage level is done using the portal, as described previously.

You can download training artifacts from experimental runs using the Studio. Choose the **Experiment** and **Run** in which you are interested. Choose **Output + logs** and navigate to the specific artifacts you wish to download. Choose ... and **Download**.

You can download a registered model by navigating to the desired **Model** and choosing **Download**.

my-workspace > Models > test:1

test:1

⟳ Refresh ➤ Deploy ⬇ Download

Export and delete resources using the Python SDK

You can download the outputs of a particular run using:

```
# Retrieved from Azure Machine Learning web UI
run_id = 'aaaaaaaa-bbbb-cccc-dddd-0123456789AB'
experiment = ws.experiments['my-experiment']
run = next(run for run in ex.get_runs() if run.id == run_id)
metrics_output_port = run.get_pipeline_output('metrics_output')
model_output_port = run.get_pipeline_output('model_output')

metrics_output_port.download('.', show_progress=True)
model_output_port.download('.', show_progress=True)
```

The following machine learning resources can be deleted using the Python SDK:

TYPE	FUNCTION CALL	NOTES
Workspace	delete	Use <code>delete-dependent-resources</code> to cascade the delete
Model	delete	
ComputeTarget	delete	
WebService	delete	

Trigger applications, processes, or CI/CD workflows based on Azure Machine Learning events (preview)

12/23/2020 • 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

NOTE

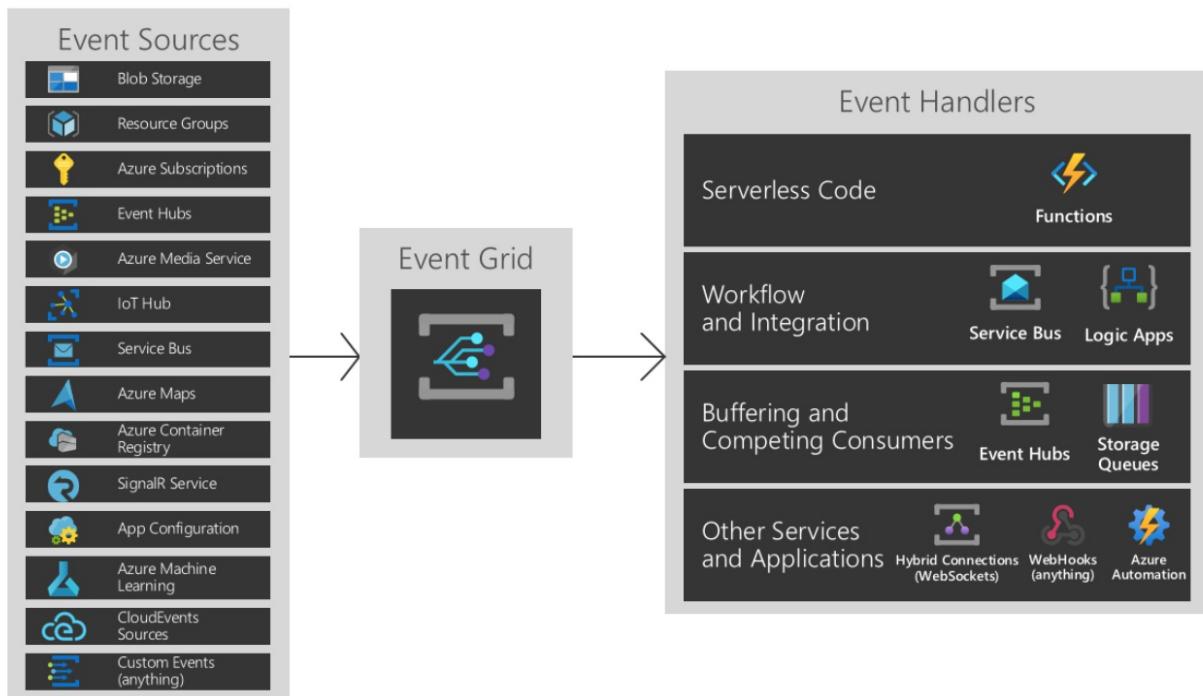
Currently, runStatusChanged events only trigger when the run status is **failed**

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 changed, currently only raised when a run status is 'failed'

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 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.

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 5 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 azureml.StepRun
<input type="button" value="Add new value (up to 5)"/>		
Add new filter		

- **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.Run	experiments/{ExperimentId}/runs/{RunId}	experiments/b1d7966c-f73a-4c68-b846-992ace89551f/runs/my_expl_1554835758_38dbaa94
Microsoft.MachineLearningServices.Model	models/{modelName}: {modelVersion}	models/sklearn_regression_model:3
Microsoft.MachineLearningServices.Model	endpoints/{serviceId}	endpoints/my_sklearn_aks
Microsoft.MachineLearningServices.Data	datadrift/{data.DataDriftId}/run/{dataRunId}	datadrift/4e694bf5-712e-4e40-b06a-d2a2755212d4/run/my_driftrun1_1550564444_fbdc
Microsoft.MachineLearningServices.Run	experiments/{ExperimentId}/runs/{RunId}	experiments/b1d7966c-f73a-4c68-b846-992ace89551f/runs/my_expl_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:

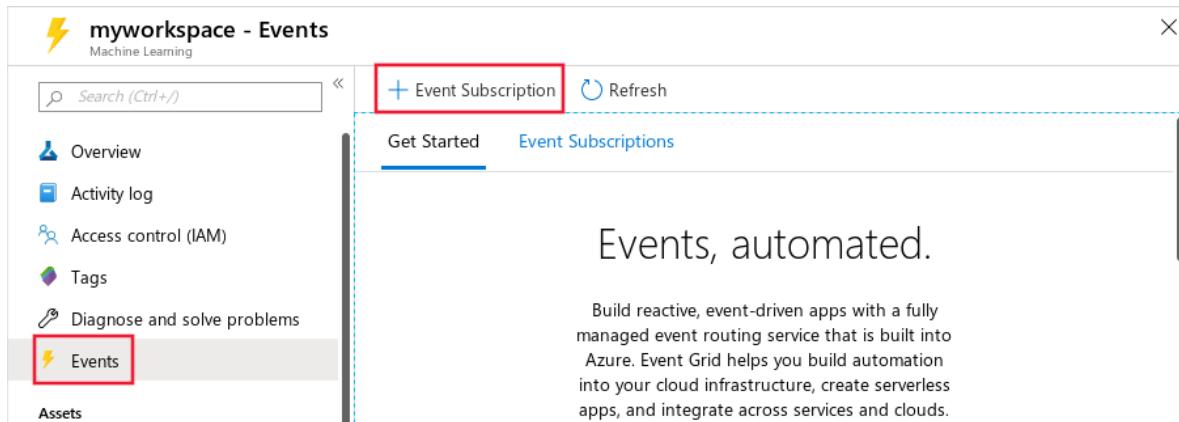
- As multiple subscriptions can be configured to route events to the same event handler, it is important not to assume events are from a particular source, but to check the topic of the message to ensure that it comes from the machine learning workspace you are expecting.
- Similarly, check that the 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.



The screenshot shows the Azure portal interface for a workspace named "myworkspace - Events". The left sidebar has a "Events" menu item highlighted with a red box. The main content area shows a "Get Started" section with a "Event Subscriptions" tab selected. A large callout text "Events, automated." is displayed. Below it is a descriptive paragraph: "Build reactive, event-driven apps with a fully managed event routing service that is built into Azure. Event Grid helps you build automation into your cloud infrastructure, create serverless apps, and integrate across services and clouds." At the top right of the main area, there is a "Event Subscription" button also highlighted with a red box.

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

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



Machine Learning

Topic Resource

shipatel-test

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:

The screenshot shows the Azure portal interface for creating an event subscription. On the left, the main pane displays 'Create Event Subscription' under 'Event Grid'. It includes tabs for 'Basic', 'Filters', and 'Additional Features'. Under 'EVENT SUBSCRIPTION DETAILS', there are fields for 'Name' (mylearning) and 'Event Schema' (Event Grid). In 'TOPIC DETAILS', it says 'Pick a topic resource for which events should be pushed'. Below that, 'Topic Type' is set to 'Machine Learning' and 'Topic Resource' is 'myworkspace'. Under 'EVENT TYPES', there's a 'Filter to Event Types' button showing '4 selected'. In 'ENDPOINT DETAILS', 'Endpoint Type' is 'Event Hubs' and 'Endpoint' is 'Select an endpoint'. At the bottom are 'Create' and 'Confirm Selection' buttons.

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 extension add --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**.

The screenshot shows the 'Events' tab in the Azure Machine Learning workspace. On the left, there's a sidebar with links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Events. The 'Events' link is selected and highlighted. To the right, there are two main sections: 'Logic Apps' and 'Functions'. The 'Logic Apps' section is highlighted with a red box. It contains the text: 'Use events as a trigger for executing a Logic App, starting Azure-wide workflows and automation.' Below it is another section for 'Functions'.

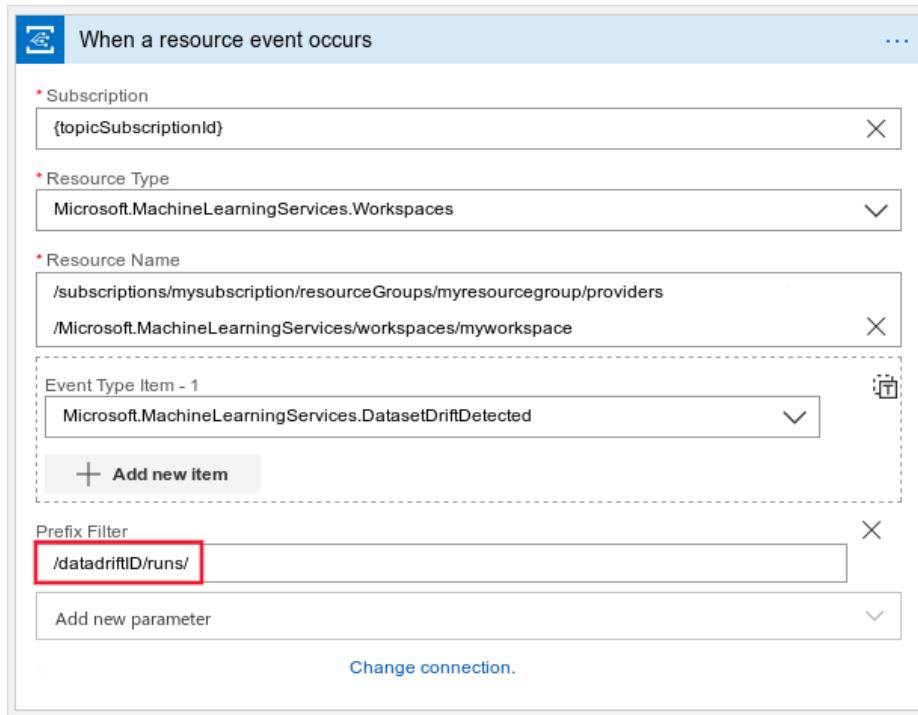
2. Sign into the Logic App UI and select Machine Learning service as the topic type.

The screenshot shows the 'When a resource event occurs' configuration screen in the Logic App UI. The 'Subscription' field is set to '{topicSubscriptionId}'. The 'Resource Type' dropdown is set to 'Microsoft.MachineLearningServices.Workspaces'. A list of other resource types is shown below, with 'Microsoft.MachineLearningServices.Workspaces' highlighted with a red box. The 'Event Type Item - 1' section is visible at the bottom.

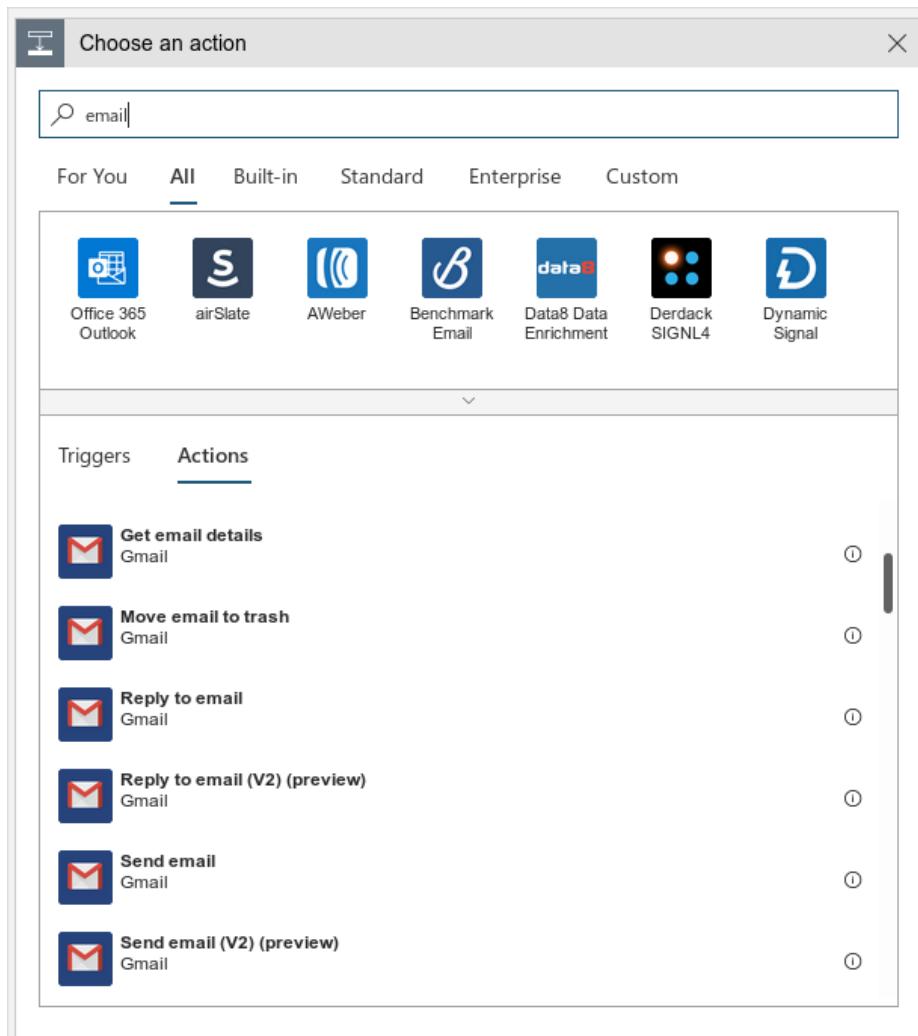
3. Select which event(s) to be notified for. For example, the following screenshot RunCompleted.

The screenshot shows the 'When a resource event occurs' configuration screen in the Logic App UI. The 'Subscription' field is set to '{topicSubscriptionId}', 'Resource Type' is set to 'Microsoft.MachineLearningServices.Workspaces', and the 'Resource Name' field contains '/subscriptions/mysubscription/resourceGroups/myresourcegroup/providers/Microsoft.MachineLearningServices/workspaces/myworkspace'. The 'Event Type Item - 1' dropdown is open, showing several event types: Microsoft.MachineLearningServices.DatasetDriftDetected, Microsoft.MachineLearningServices.ModelDeployed, Microsoft.MachineLearningServices.ModelRegistered, and Microsoft.MachineLearningServices.RunCompleted. 'Microsoft.MachineLearningServices.RunCompleted' is highlighted with a red box.

4. You can use the filtering method in the section above or add filters to only trigger the logic app on a subset of event types. In the following screenshot, a prefix filter of /datadriftID/runs/ is used.



5. 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.



6. 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 'Event Type' parameter is selected and highlighted with a red box. To the right, a sidebar titled 'Dynamic content' lists several options: 'Event Time', 'Event Type', 'ID', 'Subject', and 'Topic'. The 'Event Type' option is also highlighted with a red box in this sidebar.

7. 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 'Create' blade for a new Logic App. The 'Name' field is set to 'newLogicApp'. The 'Subscription' field is set to 'documentationteam'. Under 'Resource group', the 'Use existing' radio button is selected, and 'larrygroup1029' is chosen. The 'Location' is set to 'Central US'. The 'Logic App Status' is set to 'Enabled'. Under 'Log Analytics', the 'On' button is selected. At the bottom right, the 'Create' button is highlighted with a red box.

Example: Data drift triggers retraining

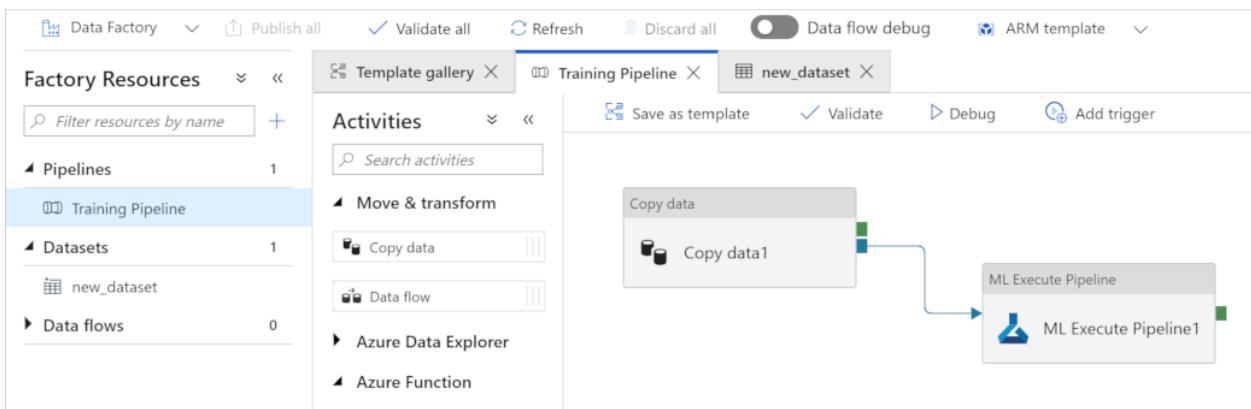
Models go stale over time, and not remain useful in the context it is running in. One way to tell if it's time to retrain the model is detecting data drift.

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. The fields filled in are:

- Name ***: triggerADF-demo
- Subscription ***: documentationteam
- Resource group * ⓘ**: Create new (radio button selected)
- Location ***: Central US
- Log Analytics ⓘ**: Off (button selected)

A note at the bottom states: "You can add triggers and actions to your Logic App after creation." At the bottom right are 'Create' and 'Automation options' buttons.

3. Once you have created the logic app, select **When an Event Grid resource event occurs**.

Logic Apps Designer

>

Start with a common trigger

Pick from one of the most commonly used triggers, then orchestrate any number of actions using the rich collection of connectors



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

4. Login and fill in the details for the event. Set the Resource Name to the workspace name. Set the Event Type to DatasetDriftDetected.

When a resource event occurs ...

* Subscription
documentationteam

* Resource Type
Microsoft.MachineLearningServices.Workspaces

* Resource Name
myworkspace

Event Type Item - 1
Microsoft.MachineLearningServices.DatasetDriftDetected

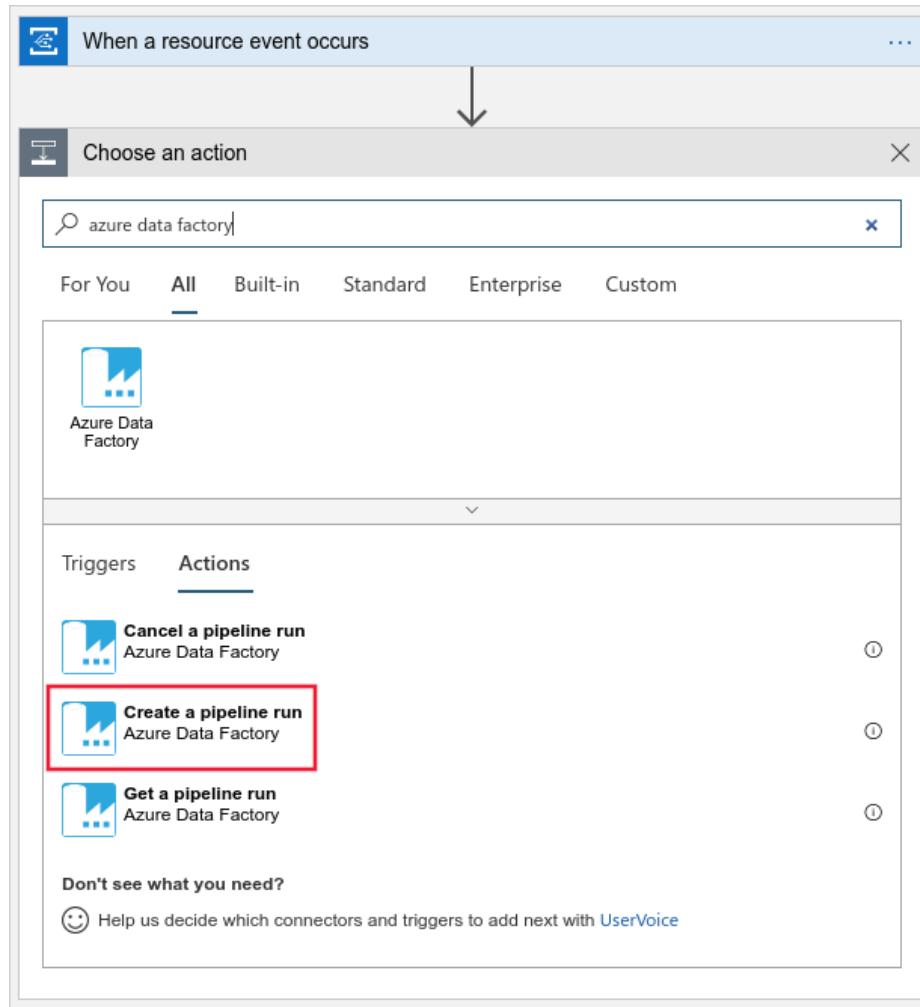
+ Add new item

Add new parameter

Connected to larryfr@microsoft.com. [Change connection.](#)

+ New step

5. 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.

The screenshot shows the configuration of the 'Create a pipeline run' step. It has several dropdown fields:

- * Subscription: documentationteam
- * Resource Group: myresourcegroup
- * Data Factory Name: mydatafactory
- * Data Factory Pipeline Name: mypipeline

Below these fields, a note says 'Connected to larryfr@microsoft.com. [Change connection.](#)' At the bottom right of the step configuration area, there's a button '+ New step'.

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 on the [new workspace portal](#).

Endpoints							
Real-time endpoints		Pipeline endpoints					
<input type="button" value="Refresh"/>		<input type="checkbox"/> Disable <input checked="" type="checkbox"/> Enable <input type="checkbox"/> View disabled					
Name ↓	Description	Modified on	Modified by	Last run submit time	Last run status	Status	
My_New_Pipeline	My Published Pipeline D...	Oct 31st, 2019 12:51 AM		Oct 31st, 2019 2:19 AM	● Running	Active	
DataDriftPipeline-68d4e40f	Pipeline for run_invoker.py	Oct 30th, 2019 11:03 PM	:	Oct 31st, 2019 1:24 AM	● Finished	Active	

Example: Deploy a model based on tags

An Azure Machine Learning model object contains parameters you can pivot deployments on such as model name, version, tag, and property. The model registration event can trigger an endpoint and you can use an Azure Function to deploy a model based on the value of those parameters.

For an example, see the <https://github.com/Azure-Samples/MachineLearningSamples-NoCodeDeploymentTriggeredByEventGrid> repository and follow the steps in the `readme` file.

Next steps

Learn more about Event Grid and give Azure Machine Learning events a try:

- [About Event Grid](#)
- [Event schema for Azure Machine Learning](#)

Install & use the CLI extension for Azure Machine Learning

12/23/2020 • 17 minutes to read • [Edit Online](#)

The Azure Machine Learning CLI is an extension to the [Azure CLI](#), a cross-platform command-line interface for the Azure platform. This extension provides commands for working with Azure Machine Learning. It allows you to automate your machine learning activities. The following list provides some example actions that you can do with the CLI extension:

- Run experiments to create machine learning models
- Register machine learning models for customer usage
- Package, deploy, and track the lifecycle of your machine learning models

The CLI is not a replacement for the Azure Machine Learning SDK. It is a complementary tool that is optimized to handle highly parameterized tasks which suit themselves well to automation.

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](#).

If you use the [Azure Cloud Shell](#), the CLI is accessed through the browser and lives in the cloud.

Full reference docs

Find the [full reference docs for the azure-cli-ml extension of Azure 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 basic 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](#).

Install the extension

To install the Machine Learning CLI extension, use the following command:

```
az extension add -n azure-cli-ml
```

TIP

Example files you can use with the commands below can be found [here](#).

When prompted, select `y` to install the extension.

To verify that the extension has been installed, use the following command to display a list of ML-specific subcommands:

```
az ml -h
```

Update the extension

To update the Machine Learning CLI extension, use the following command:

```
az extension update -n azure-cli-ml
```

Remove the extension

To remove the CLI extension, use the following command:

```
az extension remove -n azure-cli-ml
```

Resource management

The following commands demonstrate how to use the CLI to manage resources used by Azure Machine Learning.

- If you do not already have one, create a resource group:

```
az group create -n myresourcegroup -l westus2
```

- Create an Azure Machine Learning workspace:

```
az ml workspace create -w myworkspace -g myresourcegroup
```

For more information, see [az ml workspace create](#).

- Attach a workspace configuration to a folder to enable CLI contextual awareness.

```
az ml folder attach -w myworkspace -g myresourcegroup
```

This command creates a `.azureml` subdirectory that contains example runconfig and conda environment files. It also contains a `config.json` file that is used to communicate with your Azure Machine Learning workspace.

For more information, see [az ml folder attach](#).

- Attach an Azure blob container as a Datastore.

```
az ml datastore attach-blob -n datastorename -a accountname -c containername
```

For more information, see [az ml datastore attach-blob](#).

- Upload files to a Datastore.

```
az ml datastore upload -n datastorename -p sourcepath
```

For more information, see [az ml datastore upload](#).

- Attach an AKS cluster as a Compute Target.

```
az ml computetarget attach aks -n myaks -i myaksresourceid -g myresourcegroup -w myworkspace
```

For more information, see [az ml computetarget attach aks](#)

Compute clusters

- Create a new managed compute cluster.

```
az ml computetarget create amlcompute -n cpu --min-nodes 1 --max-nodes 1 -s STANDARD_D3_V2
```

- Create a new managed compute cluster with managed identity

- User-assigned managed identity

```
az ml computetarget create amlcompute --name cpu-cluster --vm-size Standard_NC6 --max-nodes 5 --assign-identity '/subscriptions/<subscription_id>/resourcegroups/<resource_group>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<user_assigned_identity>'
```

- System-assigned managed identity

```
az ml computetarget create amlcompute --name cpu-cluster --vm-size Standard_NC6 --max-nodes 5 --assign-identity '[system]'
```

- Add a managed identity to an existing cluster:
 - User-assigned managed identity

```
az ml computetarget amlcompute identity assign --name cpu-cluster '/subscriptions/<subscription_id>/resourcegroups/<resource_group>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<user_assigned_identity>'
```

- System-assigned managed identity

```
az ml computetarget amlcompute identity assign --name cpu-cluster '[system]'
```

For more information, see [az ml computetarget create amlcompute](#).

NOTE

Azure Machine Learning compute clusters support only **one system-assigned identity or multiple user-assigned identities**, not both concurrently.

Compute instance

Manage compute instances. In all the examples below, the name of the compute instance is **cpu**

- Create a new computeinstance.

```
az ml computetarget create computeinstance -n cpu -s "STANDARD_D3_V2" -v
```

For more information, see [az ml computetarget create computeinstance](#).

- Stop a computeinstance.

```
az ml computetarget stop computeinstance -n cpu -v
```

For more information, see [az ml computetarget stop computeinstance](#).

- Start a computeinstance.

```
az ml computetarget start computeinstance -n cpu -v
```

For more information, see [az ml computetarget start computeinstance](#).

- Restart a computeinstance.

```
az ml computetarget restart computeinstance -n cpu -v
```

For more information, see [az ml computetarget restart computeinstance](#).

- Delete a computeinstance.

```
az ml computetarget delete -n cpu -v
```

For more information, see [az ml computetarget delete computeinstance](#).

Run experiments

- Start a run of your experiment. When using this command, specify the name of the runconfig file (the text before *.runconfig if you are looking at your file system) against the -c parameter.

```
az ml run submit-script -c sklearn -e testexperiment train.py
```

TIP

The `az ml folder attach` command creates a `.azureml` subdirectory, which contains two example runconfig files.

If you have a Python script that creates a run configuration object programmatically, you can use `RunConfig.save()` to save it as a runconfig file.

The full runconfig schema can be found in this [JSON file](#). The schema is self-documenting through the `description` key of each object. Additionally, there are enums for possible values, and a template snippet at the end.

For more information, see [az ml run submit-script](#).

- View a list of experiments:

```
az ml experiment list
```

For more information, see [az ml experiment list](#).

HyperDrive run

You can use HyperDrive with Azure CLI to perform parameter tuning runs. First, create a HyperDrive configuration file in the following format. See [Tune hyperparameters for your model](#) article for details on hyperparameter tuning parameters.

```
# hdconfig.yml
sampling:
    type: random # Supported options: Random, Grid, Bayesian
    parameter_space: # specify a name|expression|values tuple for each parameter.
        - name: --penalty # The name of a script parameter to generate values for.
            expression: choice # supported options: choice, randint, uniform, quniform, loguniform, qloguniform, normal, qnormal, lognormal, qlognormal
            values: [0.5, 1, 1.5] # The list of values, the number of values is dependent on the expression specified.
    policy:
        type: BanditPolicy # Supported options: BanditPolicy, MedianStoppingPolicy, TruncationSelectionPolicy, NoTerminationPolicy
        evaluation_interval: 1 # Policy properties are policy specific. See the above link for policy specific parameter details.
        slack_factor: 0.2
    primary_metric_name: Accuracy # The metric used when evaluating the policy
    primary_metric_goal: Maximize # Maximize|Minimize
    max_total_runs: 8 # The maximum number of runs to generate
    max_concurrent_runs: 2 # The number of runs that can run concurrently.
    max_duration_minutes: 100 # The maximum length of time to run the experiment before cancelling.
```

Add this file alongside the run configuration files. Then submit a HyperDrive run using:

```
az ml run submit-hyperdrive -e <experiment> -c <runconfig> --hyperdrive-configuration-name <hdconfig>  
my_train.py
```

Note the *arguments* section in runconfig and *parameter space* in HyperDrive config. They contain the command-line arguments to be passed to training script. The value in runconfig stays the same for each iteration, while the range in HyperDrive config is iterated over. Do not specify the same argument in both files.

Dataset management

The following commands demonstrate how to work with datasets in Azure Machine Learning:

- Register a dataset:

```
az ml dataset register -f mydataset.json
```

For information on the format of the JSON file used to define the dataset, use

```
az ml dataset register --show-template .
```

For more information, see [az ml dataset register](#).

- List all datasets in a workspace:

```
az ml dataset list
```

For more information, see [az ml dataset list](#).

- Get details of a dataset:

```
az ml dataset show -n dataset-name
```

For more information, see [az ml dataset show](#).

- Unregister a dataset:

```
az ml dataset unregister -n dataset-name
```

For more information, see [az ml dataset unregister](#).

Environment management

The following commands demonstrate how to create, register, and list Azure Machine Learning [environments](#) for your workspace:

- Create scaffolding files for an environment:

```
az ml environment scaffold -n myenv -d myenvdirectory
```

For more information, see [az ml environment scaffold](#).

- Register an environment:

```
az ml environment register -d myenvdirectory
```

For more information, see [az ml environment register](#).

- List registered environments:

```
az ml environment list
```

For more information, see [az ml environment list](#).

- Download a registered environment:

```
az ml environment download -n myenv -d downloaddirectory
```

For more information, see [az ml environment download](#).

Environment configuration schema

If you used the `az ml environment scaffold` command, it generates a template `azureml_environment.json` file that can be modified and used to create custom environment configurations with the CLI. The top level object loosely maps to the [Environment](#) class in the Python SDK.

```
{
    "name": "testenv",
    "version": null,
    "environmentVariables": {
        "EXAMPLE_ENV_VAR": "EXAMPLE_VALUE"
    },
    "python": {
        "userManagedDependencies": false,
        "interpreterPath": "python",
        "condaDependenciesFile": null,
        "baseCondaEnvironment": null
    },
    "docker": {
        "enabled": false,
        "baseImage": "mcr.microsoft.com/azureml/base:intelmpi2018.3-ubuntu16.04",
        "baseDockerfile": null,
        "sharedVolumes": true,
        "shmSize": "2g",
        "arguments": [],
        "baseImageRegistry": {
            "address": null,
            "username": null,
            "password": null
        }
    },
    "spark": {
        "repositories": [],
        "packages": [],
        "precachePackages": true
    },
    "databricks": {
        "mavenLibraries": [],
        "pypiLibraries": [],
        "rcranLibraries": [],
        "jarLibraries": [],
        "eggLibraries": []
    },
    "inferencingStackVersion": null
}
```

The following table details each top-level field in the JSON file, its type, and a description. If an object type is linked to a class from the Python SDK, there is a loose 1:1 match between each JSON field and the public variable name in the Python class. In some cases the field may map to a constructor argument rather than a class variable. For example, the `environmentVariables` field maps to the `environment_variables` variable in the `Environment` class.

JSON FIELD	TYPE	DESCRIPTION
<code>name</code>	<code>string</code>	Name of the environment. Do not start name with Microsoft or AzureML .
<code>version</code>	<code>string</code>	Version of the environment.
<code>environmentVariables</code>	<code>{string: string}</code>	A hash-map of environment variable names and values.
<code>python</code>	<code>PythonSection</code>	hat defines the Python environment and interpreter to use on target compute resource.

JSON FIELD	TYPE	DESCRIPTION
docker	DockerSection	Defines settings to customize the Docker image built to the environment's specifications.
spark	SparkSection	The section configures Spark settings. It is only used when framework is set to PySpark.
databricks	DatabricksSection	Configures Databricks library dependencies.
inferencingStackVersion	string	Specifies the inferencing stack version added to the image. To avoid adding an inferencing stack, leave this field <code>null</code> . Valid value: "latest".

ML pipeline management

The following commands demonstrate how to work with machine learning pipelines:

- Create a machine learning pipeline:

```
az ml pipeline create -n mypipeline -y mypipeline.yml
```

For more information, see [az ml pipeline create](#).

For more information on the pipeline YAML file, see [Define machine learning pipelines in YAML](#).

- Run a pipeline:

```
az ml run submit-pipeline -n myexperiment -y mypipeline.yml
```

For more information, see [az ml run submit-pipeline](#).

For more information on the pipeline YAML file, see [Define machine learning pipelines in YAML](#).

- Schedule a pipeline:

```
az ml pipeline create-schedule -n myschedule -e myexpereiment -i mypipelineid -y
myschedule.yml
```

For more information, see [az ml pipeline create-schedule](#).

For more information on the pipeline schedule YAML file, see [Define machine learning pipelines in YAML](#).

Model registration, profiling, deployment

The following commands demonstrate how to register a trained model, and then deploy it as a production service:

- Register a model with Azure Machine Learning:

```
az ml model register -n mymodel -p sklearn_regression_model.pkl
```

For more information, see [az ml model register](#).

- **OPTIONAL** Profile your model to get optimal CPU and memory values for deployment.

```
az ml model profile -n myprofile -m mymodel:1 --ic inferenceconfig.json -d "{\"data\": [[1,2,3,4,5,6,7,8,9,10],[10,9,8,7,6,5,4,3,2,1]]}" -t myprofileresult.json
```

For more information, see [az ml model profile](#).

- Deploy your model to AKS

```
az ml model deploy -n myservice -m mymodel:1 --ic inferenceconfig.json --dc deploymentconfig.json --ct akscomputetarget
```

For more information on the inference configuration file schema, see [Inference configuration schema](#).

For more information on the deployment configuration file schema, see [Deployment configuration schema](#).

For more information, see [az ml model deploy](#).

Inference configuration schema

The entries in the `inferenceconfig.json` document map to the parameters for the [InferenceConfig](#) class. The following table describes the mapping between entities in the JSON document and the parameters for the method:

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
<code>entryScript</code>	<code>entry_script</code>	Path to a local file that contains the code to run for the image.
<code>sourceDirectory</code>	<code>source_directory</code>	Optional. Path to folders that contain all files to create the image, which makes it easy to access any files within this folder or subfolder. You can upload an entire folder from your local machine as dependencies for the Webservice. Note: your <code>entry_script</code> , <code>conda_file</code> , and <code>extra_docker_file_steps</code> paths are relative paths to the <code>source_directory</code> path.
<code>environment</code>	<code>environment</code>	Optional. Azure Machine Learning environment .

You can include full specifications of an Azure Machine Learning [environment](#) in the inference configuration file. If this environment doesn't exist in your workspace, Azure Machine Learning will create it. Otherwise, Azure Machine Learning will update the environment if necessary. The following JSON is an example:

```
{
    "entryScript": "score.py",
    "environment": {
        "docker": {
            "arguments": [],
            "baseDockerfile": null,
            "baseImage": "mcr.microsoft.com/azureml/base:intelmpi2018.3-ubuntu16.04",
            "enabled": false,
            "sharedVolumes": true,
            "shmSize": null
        },
        "environmentVariables": {
            "EXAMPLE_ENV_VAR": "EXAMPLE_VALUE"
        },
        "name": "my-deploy-env",
        "python": {
            "baseCondaEnvironment": null,
            "condaDependencies": {
                "channels": [
                    "conda-forge"
                ],
                "dependencies": [
                    "python=3.6.2",
                    {
                        "pip": [
                            "azureml-defaults",
                            "azureml-telemetry",
                            "scikit-learn==0.22.1",
                            "inference-schema[numpy-support]"
                        ]
                    }
                ],
                "name": "project_environment"
            },
            "condaDependenciesFile": null,
            "interpreterPath": "python",
            "userManagedDependencies": false
        },
        "version": "1"
    }
}
```

You can also use an existing Azure Machine Learning [environment](#) in separated CLI parameters and remove the "environment" key from the inference configuration file. Use -e for the environment name, and --ev for the environment version. If you don't specify --ev, the latest version will be used. Here is an example of an inference configuration file:

```
{
    "entryScript": "score.py",
    "sourceDirectory": null
}
```

The following command demonstrates how to deploy a model using the previous inference configuration file (named myInferenceConfig.json).

It also uses the latest version of an existing Azure Machine Learning [environment](#) (named AzureML-Minimal).

```
az ml model deploy -m mymodel:1 --ic myInferenceConfig.json -e AzureML-Minimal --dc deploymentconfig.json
```

Deployment configuration schema

Local deployment configuration schema

The entries in the `deploymentconfig.json` document map to the parameters for [LocalWebservice.deploy_configuration](#). The following table describes the mapping between the entities in the JSON document and the parameters for the method:

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
<code>computeType</code>	NA	The compute target. For local targets, the value must be <code>local</code> .
<code>port</code>	<code>port</code>	The local port on which to expose the service's HTTP endpoint.

This JSON is an example deployment configuration for use with the CLI:

```
{  
  "computeType": "local",  
  "port": 32267  
}
```

Azure Container Instance deployment configuration schema

The entries in the `deploymentconfig.json` document map to the parameters for [AciWebservice.deploy_configuration](#). The following table describes the mapping between the entities in the JSON document and the parameters for the method:

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
<code>computeType</code>	NA	The compute target. For ACI, the value must be <code>ACI</code> .
<code>containerResourceRequirements</code>	NA	Container for the CPU and memory entities.
<code>cpu</code>	<code>cpu_cores</code>	The number of CPU cores to allocate. Defaults, <code>0.1</code>
<code>memoryInGB</code>	<code>memory_gb</code>	The amount of memory (in GB) to allocate for this web service. Default, <code>0.5</code>
<code>location</code>	<code>location</code>	The Azure region to deploy this Webservice to. If not specified the Workspace location will be used. More details on available regions can be found here: ACI Regions
<code>authEnabled</code>	<code>auth_enabled</code>	Whether to enable auth for this Webservice. Defaults to False
<code>sslEnabled</code>	<code>ssl_enabled</code>	Whether to enable SSL for this Webservice. Defaults to False.

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
appInsightsEnabled	enable_app_insights	Whether to enable AppInsights for this Webservice. Defaults to False
sslCertificate	ssl_cert_pem_file	The cert file needed if SSL is enabled
sslKey	ssl_key_pem_file	The key file needed if SSL is enabled
cname	ssl_cname	The cname for if SSL is enabled
dnsNameLabel	dns_name_label	The dns name label for the scoring endpoint. If not specified a unique dns name label will be generated for the scoring endpoint.

The following JSON is an example deployment configuration for use with the CLI:

```
{
  "computeType": "aci",
  "containerResourceRequirements":
  {
    "cpu": 0.5,
    "memoryInGB": 1.0
  },
  "authEnabled": true,
  "sslEnabled": false,
  "appInsightsEnabled": false
}
```

Azure Kubernetes Service deployment configuration schema

The entries in the `deploymentconfig.json` document map to the parameters for [AksWebservice.deploy_configuration](#). The following table describes the mapping between the entities in the JSON document and the parameters for the method:

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
computeType	NA	The compute target. For AKS, the value must be <code>aks</code> .
autoScaler	NA	Contains configuration elements for autoscale. See the autoscaler table.
autoscaleEnabled	autoscale_enabled	Whether to enable autoscaling for the web service. If <code>numReplicas</code> = <code>0</code> , <code>True</code> ; otherwise, <code>False</code> .
minReplicas	autoscale_min_replicas	The minimum number of containers to use when autoscaling this web service. Default, <code>1</code> .
maxReplicas	autoscale_max_replicas	The maximum number of containers to use when autoscaling this web service. Default, <code>10</code> .

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
<code>refreshPeriodInSeconds</code>	<code>autoscale_refresh_seconds</code>	How often the autoscaler attempts to scale this web service. Default, <code>1</code> .
<code>targetUtilization</code>	<code>autoscale_target_utilization</code>	The target utilization (in percent out of 100) that the autoscaler should attempt to maintain for this web service. Default, <code>70</code> .
<code>dataCollection</code>	NA	Contains configuration elements for data collection.
<code>storageEnabled</code>	<code>collect_model_data</code>	Whether to enable model data collection for the web service. Default, <code>False</code> .
<code>authEnabled</code>	<code>auth_enabled</code>	Whether or not to enable key authentication for the web service. Both <code>tokenAuthEnabled</code> and <code>authEnabled</code> cannot be <code>True</code> . Default, <code>True</code> .
<code>tokenAuthEnabled</code>	<code>token_auth_enabled</code>	Whether or not to enable token authentication for the web service. Both <code>tokenAuthEnabled</code> and <code>authEnabled</code> cannot be <code>True</code> . Default, <code>False</code> .
<code>containerResourceRequirements</code>	NA	Container for the CPU and memory entities.
<code>cpu</code>	<code>cpu_cores</code>	The number of CPU cores to allocate for this web service. Defaults, <code>0.1</code>
<code>memoryInGB</code>	<code>memory_gb</code>	The amount of memory (in GB) to allocate for this web service. Default, <code>0.5</code>
<code>appInsightsEnabled</code>	<code>enable_app_insights</code>	Whether to enable Application Insights logging for the web service. Default, <code>False</code> .
<code>scoringTimeoutMs</code>	<code>scoring_timeout_ms</code>	A timeout to enforce for scoring calls to the web service. Default, <code>60000</code> .
<code>maxConcurrentRequestsPerContainer</code>	<code>replica_max_concurrent_requests</code>	The maximum concurrent requests per node for this web service. Default, <code>1</code> .

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
maxQueueWaitMs	max_request_wait_time	The maximum time a request will stay in the queue (in milliseconds) before a 503 error is returned. Default, 500.
numReplicas	num_replicas	The number of containers to allocate for this web service. No default value. If this parameter is not set, the autoscaler is enabled by default.
keys	NA	Contains configuration elements for keys.
primaryKey	primary_key	A primary auth key to use for this Webservice
secondaryKey	secondary_key	A secondary auth key to use for this Webservice
gpuCores	gpu_cores	The number of GPU cores (per-container replica) to allocate for this Webservice. Default is 1. Only supports whole number values.
livenessProbeRequirements	NA	Contains configuration elements for liveness probe requirements.
periodSeconds	period_seconds	How often (in seconds) to perform the liveness probe. Default to 10 seconds. Minimum value is 1.
initialDelaySeconds	initial_delay_seconds	Number of seconds after the container has started before liveness probes are initiated. Defaults to 310
timeoutSeconds	timeout_seconds	Number of seconds after which the liveness probe times out. Defaults to 2 seconds. Minimum value is 1
successThreshold	success_threshold	Minimum consecutive successes for the liveness probe to be considered successful after having failed. Defaults to 1. Minimum value is 1.
failureThreshold	failure_threshold	When a Pod starts and the liveness probe fails, Kubernetes will try failureThreshold times before giving up. Defaults to 3. Minimum value is 1.

JSON ENTITY	METHOD PARAMETER	DESCRIPTION
namespace	namespace	The Kubernetes namespace that the webservice is deployed into. Up to 63 lowercase alphanumeric ('a'-'z', '0'-'9') and hyphen ('-') characters. The first and last characters can't be hyphens.

The following JSON is an example deployment configuration for use with the CLI:

```
{
  "computeType": "aks",
  "autoScaler":
  {
    "autoscaleEnabled": true,
    "minReplicas": 1,
    "maxReplicas": 3,
    "refreshPeriodInSeconds": 1,
    "targetUtilization": 70
  },
  "dataCollection":
  {
    "storageEnabled": true
  },
  "authEnabled": true,
  "containerResourceRequirements":
  {
    "cpu": 0.5,
    "memoryInGB": 1.0
  }
}
```

Next steps

- [Command reference for the Machine Learning CLI extension.](#)
- [Train and deploy machine learning models using Azure Pipelines](#)

Algorithm & module reference for Azure Machine Learning designer

12/23/2020 • 3 minutes to read • [Edit Online](#)

This reference content provides the technical background on each of the machine learning algorithms and modules available in Azure Machine Learning designer.

Each module represents a set of code that can run independently and perform a machine learning task, given the required inputs. A module might contain a particular algorithm, or perform a task that is important in machine learning, such as missing value replacement, or statistical analysis.

For help with choosing algorithms, see

- [How to select algorithms](#)
- [Azure Machine Learning Algorithm Cheat Sheet](#)

TIP

In any pipeline in the designer, you can get information about a specific module. Select the **Learn more** link in the module card when hovering on the module in the module list, or in the right pane of the module.

Data preparation modules

FUNCTIONALITY	DESCRIPTION	MODULE
Data Input and Output	Move data from cloud sources into your pipeline. Write your results or intermediate data to Azure Storage, SQL Database, or Hive, while running a pipeline, or use cloud storage to exchange data between pipelines.	Enter Data Manually Export Data Import Data
Data Transformation	Operations on data that are unique to machine learning, such as normalizing or binning data, dimensionality reduction, and converting data among various file formats.	Add Columns Add Rows Apply Math Operation Apply SQL Transformation Clean Missing Data Clip Values Convert to CSV Convert to Dataset Convert to Indicator Values Edit Metadata Group Data into Bins Join Data Normalize Data Partition and Sample Remove Duplicate Rows SMOTE Select Columns Transform Select Columns in Dataset Split Data

FUNCTIONALITY	DESCRIPTION	MODULE
Feature Selection	Select a subset of relevant, useful features to use in building an analytical model.	Filter Based Feature Selection Permutation Feature Importance
Statistical Functions	Provide a wide variety of statistical methods related to data science.	Summarize Data

Machine learning algorithms

FUNCTIONALITY	DESCRIPTION	MODULE
Regression	Predict a value.	Boosted Decision Tree Regression Decision Forest Regression Fast Forest Quantile Regression Linear Regression Neural Network Regression Poisson Regression
Clustering	Group data together.	K-Means Clustering
Classification	Predict a class. Choose from binary (two-class) or multiclass algorithms.	Multiclass Boosted Decision Tree Multiclass Decision Forest Multiclass Logistic Regression Multiclass Neural Network One vs. All Multiclass One vs. One Multiclass Two-Class Averaged Perceptron Two-Class Boosted Decision Tree Two-Class Decision Forest Two-Class Logistic Regression Two-Class Neural Network Two Class Support Vector Machine

Modules for building and evaluating models

FUNCTIONALITY	DESCRIPTION	MODULE
Model Training	Run data through the algorithm.	Train Clustering Model Train Model Train Pytorch Model Tune Model Hyperparameters
Model Scoring and Evaluation	Measure the accuracy of the trained model.	Apply Transformation Assign Data to Clusters Cross Validate Model Evaluate Model Score Image Model Score Model
Python Language	Write code and embed it in a module to integrate Python with your pipeline.	Create Python Model Execute Python Script
R Language	Write code and embed it in a module to integrate R with your pipeline.	Execute R Script

FUNCTIONALITY	DESCRIPTION	MODULE
Text Analytics	Provide specialized computational tools for working with both structured and unstructured text.	Convert Word to Vector Extract N Gram Features from Text Feature Hashing Preprocess Text Latent Dirichlet Allocation Score Vowpal Wabbit Model Train Vowpal Wabbit Model
Computer Vision	Image data preprocessing and Image recognition related modules.	Apply Image Transformation Convert to Image Directory Init Image Transformation Split Image Directory DenseNet ResNet
Recommendation	Build recommendation models.	Evaluate Recommender Score SVD Recommender Score Wide and Deep Recommender Train SVD Recommender Train Wide and Deep Recommender
Anomaly Detection	Build anomaly detection models.	PCA-Based Anomaly Detection Train Anomaly Detection Model

Web service

Learn about the [web service modules](#) which are necessary for real-time inference in Azure Machine Learning designer.

Error messages

Learn about the [error messages and exception codes](#) you might encounter using modules in Azure Machine Learning designer.

Next steps

- [Tutorial: Build a model in designer to predict auto prices](#)

Monitoring Azure machine learning data reference

12/23/2020 • 6 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 deploy failed	Count	The number of model deployments that failed.
Model deploy started	Count	The number of model deployments started.
Model deploy succeeded	Count	The number of model deployments that succeeded.
Model register failed	Count	The number of model registrations that failed.
Model register succeeded	Count	The number of model registrations that succeeded.

Quota

Quota information is for Azure Machine Learning compute only.

METRIC	UNIT	DESCRIPTION
Active cores	Count	The number of active compute cores.
Active nodes	Count	The number of active nodes.
Idle cores	Count	The number of idle compute cores.
Idle nodes	Count	The number of idle compute nodes.
Leaving cores	Count	The number of leaving cores.
Leaving nodes	Count	The number of leaving nodes.
Preempted cores	Count	The number of preempted cores.
Preempted nodes	Count	The number of preempted nodes.

METRIC	UNIT	DESCRIPTION
Quota utilization percentage	Percent	The percentage of quota used.
Total cores	Count	The total cores.
Total nodes	Count	The total nodes.
Unusable cores	Count	The number of unusable cores.
Unusable nodes	Count	The number of unusable nodes.

Resource

METRIC	UNIT	DESCRIPTION
CpuUtilization	Percent	How much percent of CPU was utilized for a given node during a run/job. This metric is published only when a job is running on a node. One job may use one or more nodes. This metric is published per node.
GpuUtilization	Percent	How much percentage of GPU was utilized for a given node during a run/job. One node can have one or more GPUs. This metric is published per GPU per node.

Run

Information on training runs.

METRIC	UNIT	DESCRIPTION
Completed runs	Count	The number of completed runs.
Failed runs	Count	The number of failed runs.
Started runs	Count	The number of started runs.

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.

DIMENSION	DESCRIPTION
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.
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

OPERATION	DESCRIPTION
List secrets	On 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	AmlComputeClusterNodeEvent
AmlComputeCpuGpuUtilization	AmlComputeCpuGpuUtilization
AmlComputeJobEvent	AmlComputeJobEvent
AmlRunStatusChangedEvent	AmlRunStatusChangedEvent

Schemas

The following schemas are in use by Azure Machine Learning

AmlComputeJobEvents 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, AmlComputeClusterNodeEvent
JobId	ID of the Job submitted
ExperimentId	ID of the Experiment
ExperimentName	Name of the Experiment
CustomerSubscriptionId	SubscriptionId where Experiment and Job as 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

PROPERTY	DESCRIPTION
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

AmlComputeClusterEvents 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, AmlComputeClusterNodeEvent
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

PROPERTY	DESCRIPTION
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
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

PROPERTY	DESCRIPTION
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

AmlComputeClusterNodeEvents 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, AmlComputeClusterNodeEvent
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
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

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.

Define machine learning pipelines in YAML

12/23/2020 • 10 minutes to read • [Edit Online](#)

Learn how to define your machine learning pipelines in [YAML](#). When using the machine learning extension for the Azure CLI, many of the pipeline-related commands expect a YAML file that defines the pipeline.

The following table lists what is and is not currently supported when defining a pipeline in YAML:

STEP TYPE	SUPPORTED?
PythonScriptStep	Yes
ParallelRunStep	Yes
AdlaStep	Yes
AzureBatchStep	Yes
DatabricksStep	Yes
DataTransferStep	Yes
AutoMLStep	No
HyperDriveStep	No
ModuleStep	Yes
MPIStep	No
EstimatorStep	No

Pipeline definition

A pipeline definition uses the following keys, which correspond to the [Pipelines](#) class:

YAML KEY	DESCRIPTION
<code>name</code>	The description of the pipeline.
<code>parameters</code>	Parameter(s) to the pipeline.
<code>data_reference</code>	Defines how and where data should be made available in a run.
<code>default_compute</code>	Default compute target where all steps in the pipeline run.
<code>steps</code>	The steps used in the pipeline.

Parameters

The `parameters` section uses the following keys, which correspond to the [PipelineParameter](#) class:

YAML KEY	DESCRIPTION
<code>type</code>	The value type of the parameter. Valid types are <code>string</code> , <code>int</code> , <code>float</code> , <code>bool</code> , or <code>datapath</code> .
<code>default</code>	The default value.

Each parameter is named. For example, the following YAML snippet defines three parameters named

`NumIterationsParameter`, `DataPathParameter`, and `NodeCountParameter`:

```
pipeline:  
  name: SamplePipelineFromYaml  
  parameters:  
    NumIterationsParameter:  
      type: int  
      default: 40  
    DataPathParameter:  
      type: datapath  
      default:  
        datastore: workspaceblobstore  
        path_on_datastore: sample2.txt  
    NodeCountParameter:  
      type: int  
      default: 4
```

Data reference

The `data_references` section uses the following keys, which correspond to the [DataReference](#):

YAML KEY	DESCRIPTION
<code>datastore</code>	The datastore to reference.
<code>path_on_datastore</code>	The relative path in the backing storage for the data reference.

Each data reference is contained in a key. For example, the following YAML snippet defines a data reference stored in the key named `employee_data`:

```
pipeline:  
  name: SamplePipelineFromYaml  
  parameters:  
    PipelineParam1:  
      type: int  
      default: 3  
  data_references:  
    employee_data:  
      datastore: adftestadla  
      path_on_datastore: "adla_sample/sample_input.csv"
```

Steps

Steps define a computational environment, along with the files to run on the environment. To define the type of a

step, use the `type` key:

STEP TYPE	DESCRIPTION
<code>AdlaStep</code>	Runs a U-SQL script with Azure Data Lake Analytics. Corresponds to the AdlaStep class.
<code>AzureBatchStep</code>	Runs jobs using Azure Batch. Corresponds to the AzureBatchStep class.
<code>DatabricksStep</code>	Adds a Databricks notebook, Python script, or JAR. Corresponds to the DatabricksStep class.
<code>DataTransferStep</code>	Transfers data between storage options. Corresponds to the DataTransferStep class.
<code>PythonScriptStep</code>	Runs a Python script. Corresponds to the PythonScriptStep class.
<code>ParallelRunStep</code>	Runs a Python script to process large amounts of data asynchronously and in parallel. Corresponds to the ParallelRunStep class.

ADLA step

YAML KEY	DESCRIPTION
<code>script_name</code>	The name of the U-SQL script (relative to the source_directory).
<code>compute_target</code>	The Azure Data Lake compute target to use for this step.
<code>parameters</code>	Parameters to the pipeline.
<code>inputs</code>	Inputs can be InputPortBinding , DataReference , PortDataReference , PipelineData , Dataset , DatasetDefinition , or PipelineDataset .
<code>outputs</code>	Outputs can be either PipelineData or OutputPortBinding .
<code>source_directory</code>	Directory that contains the script, assemblies, etc.
<code>priority</code>	The priority value to use for the current job.
<code>params</code>	Dictionary of name-value pairs.
<code>degree_of_parallelism</code>	The degree of parallelism to use for this job.
<code>runtime_version</code>	The runtime version of the Data Lake Analytics engine.
<code>allow_reuse</code>	Determines whether the step should reuse previous results when run again with the same settings.

The following example contains an ADLA Step definition:

```

pipeline:
  name: SamplePipelineFromYaml
  parameters:
    PipelineParam1:
      type: int
      default: 3
  data_references:
    employee_data:
      datastore: adftestadla
      path_on_datastore: "adla_sample/sample_input.csv"
  default_compute: adlacomp
  steps:
    Step1:
      runconfig: "D:\\Yaml\\default_runconfig.yml"
      parameters:
        NUM_ITERATIONS_2:
          source: PipelineParam1
        NUM_ITERATIONS_1: 7
      type: "AdlaStep"
      name: "MyAdlaStep"
      script_name: "sample_script.usql"
      source_directory: "D:\\scripts\\Adla"
      inputs:
        employee_data:
          source: employee_data
      outputs:
        OutputData:
          destination: Output4
          datastore: adftestadla
          bind_mode: mount

```

Azure Batch step

YAML KEY	DESCRIPTION
<code>compute_target</code>	The Azure Batch compute target to use for this step.
<code>inputs</code>	Inputs can be InputPortBinding , DataReference , PortDataReference , PipelineData , Dataset , DatasetDefinition , or PipelineDataset .
<code>outputs</code>	Outputs can be either PipelineData or OutputPortBinding .
<code>source_directory</code>	Directory that contains the module binaries, executable, assemblies, etc.
<code>executable</code>	Name of the command/executable that will be ran as part of this job.
<code>create_pool</code>	Boolean flag to indicate whether to create the pool before running the job.
<code>delete_batch_job_after_finish</code>	Boolean flag to indicate whether to delete the job from the Batch account after it's finished.
<code>delete_batch_pool_after_finish</code>	Boolean flag to indicate whether to delete the pool after the job finishes.
<code>is_positive_exit_code_failure</code>	Boolean flag to indicate if the job fails if the task exits with a positive code.

YAML KEY	DESCRIPTION
vm_image_urn	If <code>create_pool</code> is <code>True</code> , and VM uses <code>VirtualMachineConfiguration</code> .
pool_id	The ID of the pool where the job will run.
allow_reuse	Determines whether the step should reuse previous results when run again with the same settings.

The following example contains an Azure Batch step definition:

```

pipeline:
  name: SamplePipelineFromYaml
  parameters:
    PipelineParam1:
      type: int
      default: 3
  data_references:
    input:
      datastore: workspaceblobstore
      path_on_datastore: "input.txt"
  default_compute: testbatch
  steps:
    Step1:
      runconfig: "D:\\Yaml\\default_runconfig.yml"
      parameters:
        NUM_ITERATIONS_2:
          source: PipelineParam1
        NUM_ITERATIONS_1: 7
      type: "AzureBatchStep"
      name: "MyAzureBatchStep"
      pool_id: "MyPoolName"
      create_pool: true
      executable: "azurebatch.cmd"
      source_directory: "D:\\scripts\\AureBatch"
      allow_reuse: false
      inputs:
        input:
          source: input
      outputs:
        output:
          destination: output
          datastore: workspaceblobstore

```

Databricks step

YAML KEY	DESCRIPTION
compute_target	The Azure Databricks compute target to use for this step.
inputs	Inputs can be InputPortBinding , DataReference , PortDataReference , PipelineData , Dataset , DatasetDefinition , or PipelineDataset .
outputs	Outputs can be either PipelineData or OutputPortBinding .
run_name	The name in Databricks for this run.

YAML KEY	DESCRIPTION
source_directory	Directory that contains the script and other files.
num_workers	The static number of workers for the Databricks run cluster.
runconfig	The path to a <code>.runconfig</code> file. This file is a YAML representation of the RunConfiguration class. For more information on the structure of this file, see runconfigschema.json .
allow_reuse	Determines whether the step should reuse previous results when run again with the same settings.

The following example contains a Databricks step:

```

pipeline:
  name: SamplePipelineFromYaml
  parameters:
    PipelineParam1:
      type: int
      default: 3
  data_references:
    adls_test_data:
      datastore: adftestadla
      path_on_datastore: "testdata"
    blob_test_data:
      datastore: workspaceblobstore
      path_on_datastore: "dbtest"
  default_compute: mydatabricks
  steps:
    Step1:
      runconfig: "D:\\Yaml\\default_runconfig.yml"
      parameters:
        NUM_ITERATIONS_2:
          source: PipelineParam1
        NUM_ITERATIONS_1: 7
      type: "DatabricksStep"
      name: "MyDatabrickStep"
      run_name: "DatabricksRun"
      python_script_name: "train-db-local.py"
      source_directory: "D:\\scripts\\Databricks"
      num_workers: 1
      allow_reuse: true
      inputs:
        blob_test_data:
          source: blob_test_data
      outputs:
        OutputData:
          destination: Output4
          datastore: workspaceblobstore
          bind_mode: mount

```

Data transfer step

YAML KEY	DESCRIPTION
compute_target	The Azure Data Factory compute target to use for this step.

YAML KEY	DESCRIPTION
<code>source_data_reference</code>	Input connection that serves as the source of data transfer operations. Supported values are InputPortBinding , DataReference , PortDataReference , PipelineData , Dataset , DatasetDefinition , or PipelineDataset .
<code>destination_data_reference</code>	Input connection that serves as the destination of data transfer operations. Supported values are PipelineData and OutputPortBinding .
<code>allow_reuse</code>	Determines whether the step should reuse previous results when run again with the same settings.

The following example contains a data transfer step:

```
pipeline:
  name: SamplePipelineFromYaml
  parameters:
    PipelineParam1:
      type: int
      default: 3
  data_references:
    adls_test_data:
      datastore: adftestadla
      path_on_datastore: "testdata"
    blob_test_data:
      datastore: workspaceblobstore
      path_on_datastore: "testdata"
  default_compute: adftest
  steps:
    Step1:
      runconfig: "D:\\Yaml\\default_runconfig.yml"
      parameters:
        NUM_ITERATIONS_2:
          source: PipelineParam1
        NUM_ITERATIONS_1: 7
        type: "DataTransferStep"
        name: "MyDataTransferStep"
        adla_compute_name: adftest
        source_data_reference:
          adls_test_data:
            source: adls_test_data
        destination_data_reference:
          blob_test_data:
            source: blob_test_data
```

Python script step

YAML KEY	DESCRIPTION
<code>inputs</code>	Inputs can be InputPortBinding , DataReference , PortDataReference , PipelineData , Dataset , DatasetDefinition , or PipelineDataset .
<code>outputs</code>	Outputs can be either PipelineData or OutputPortBinding .
<code>script_name</code>	The name of the Python script (relative to <code>source_directory</code>).

YAML KEY	DESCRIPTION
<code>source_directory</code>	Directory that contains the script, Conda environment, etc.
<code>runconfig</code>	The path to a <code>.runconfig</code> file. This file is a YAML representation of the RunConfiguration class. For more information on the structure of this file, see runconfig.json .
<code>allow_reuse</code>	Determines whether the step should reuse previous results when run again with the same settings.

The following example contains a Python script step:

```

pipeline:
  name: SamplePipelineFromYaml
  parameters:
    PipelineParam1:
      type: int
      default: 3
  data_references:
    DataReference1:
      datastore: workspaceblobstore
      path_on_datastore: testfolder/sample.txt
  default_compute: cpu-cluster
  steps:
    Step1:
      runconfig: "D:\\\\Yaml\\\\default_runconfig.yml"
      parameters:
        NUM_ITERATIONS_2:
          source: PipelineParam1
        NUM_ITERATIONS_1: 7
      type: "PythonScriptStep"
      name: "MyPythonScriptStep"
      script_name: "train.py"
      allow_reuse: True
      source_directory: "D:\\\\scripts\\\\PythonScript"
      inputs:
        InputData:
          source: DataReference1
      outputs:
        OutputData:
          destination: Output4
          datastore: workspaceblobstore
          bind_mode: mount

```

Parallel run step

YAML KEY	DESCRIPTION
<code>inputs</code>	Inputs can be Dataset , DatasetDefinition , or PipelineDataset .
<code>outputs</code>	Outputs can be either PipelineData or OutputPortBinding .
<code>script_name</code>	The name of the Python script (relative to <code>source_directory</code>).
<code>source_directory</code>	Directory that contains the script, Conda environment, etc.

YAML KEY	DESCRIPTION
<code>parallel_run_config</code>	The path to a <code>parallel_run_config.yml</code> file. This file is a YAML representation of the ParallelRunConfig class.
<code>allow_reuse</code>	Determines whether the step should reuse previous results when run again with the same settings.

The following example contains a Parallel run step:

```

pipeline:
  description: SamplePipelineFromYaml
  default_compute: cpu-cluster
  data_references:
    MyMinistInput:
      dataset_name: mnist_sample_data
  parameters:
    PipelineParamTimeout:
      type: int
      default: 600
  steps:
    Step1:
      parallel_run_config: "yaml/parallel_run_config.yml"
      type: "ParallelRunStep"
      name: "parallel-run-step-1"
      allow_reuse: True
      arguments:
        - "--progress_update_timeout"
        - parameter:timeout_parameter
        - "--side_input"
        - side_input:SideInputData
      parameters:
        timeout_parameter:
          source: PipelineParamTimeout
      inputs:
        InputData:
          source: MyMinistInput
      side_inputs:
        SideInputData:
          source: Output4
          bind_mode: mount
      outputs:
        OutputDataStep2:
          destination: Output5
          datastore: workspaceblobstore
          bind_mode: mount

```

Pipeline with multiple steps

YAML KEY	DESCRIPTION
<code>steps</code>	Sequence of one or more PipelineStep definitions. Note that the <code>destination</code> keys of one step's <code>outputs</code> become the <code>source</code> keys to the <code>inputs</code> of the next step.

```

pipeline:
  name: SamplePipelineFromYAML
  description: Sample multistep YAML pipeline
  data_references:
    TitanicDS:
      dataset_name: 'titanic_ds'
      bind_mode: download
  default_compute: cpu-cluster
  steps:
    Dataprep:
      type: "PythonScriptStep"
      name: "DataPrep Step"
      compute: cpu-cluster
      runconfig: ".\\default_runconfig.yml"
      script_name: "prep.py"
      arguments:
        - '--train_path'
        - output:train_path
        - '--test_path'
        - output:test_path
      allow_reuse: True
      inputs:
        titanic_ds:
          source: TitanicDS
          bind_mode: download
      outputs:
        train_path:
          destination: train_csv
          datastore: workspaceblobstore
        test_path:
          destination: test_csv
    Training:
      type: "PythonScriptStep"
      name: "Training Step"
      compute: cpu-cluster
      runconfig: ".\\default_runconfig.yml"
      script_name: "train.py"
      arguments:
        - "--train_path"
        - input:train_path
        - "--test_path"
        - input:test_path
      inputs:
        train_path:
          source: train_csv
          bind_mode: download
        test_path:
          source: test_csv
          bind_mode: download

```

Schedules

When defining the schedule for a pipeline, it can be either datastore-triggered or recurring based on a time interval. The following are the keys used to define a schedule:

YAML KEY	DESCRIPTION
<code>description</code>	A description of the schedule.
<code>recurrence</code>	Contains recurrence settings, if the schedule is recurring.

YAML KEY	DESCRIPTION
<code>pipeline_parameters</code>	Any parameters that are required by the pipeline.
<code>wait_for_provisioning</code>	Whether to wait for provisioning of the schedule to complete.
<code>wait_timeout</code>	The number of seconds to wait before timing out.
<code>datastore_name</code>	The datastore to monitor for modified/added blobs.
<code>polling_interval</code>	How long, in minutes, between polling for modified/added blobs. Default value: 5 minutes. Only supported for datastore schedules.
<code>data_path_parameter_name</code>	The name of the data path pipeline parameter to set with the changed blob path. Only supported for datastore schedules.
<code>continue_on_step_failure</code>	Whether to continue execution of other steps in the submitted PipelineRun if a step fails. If provided, will override the <code>continue_on_step_failure</code> setting of the pipeline.
<code>path_on_datastore</code>	Optional. The path on the datastore to monitor for modified/added blobs. The path is under the container for the datastore, so the actual path the schedule monitors is <code>container/ path_on_datastore</code> . If none, the datastore container is monitored. Additions/modifications made in a subfolder of the <code>path_on_datastore</code> are not monitored. Only supported for datastore schedules.

The following example contains the definition for a datastore-triggered schedule:

```
Schedule:
  description: "Test create with datastore"
  recurrence: ~
  pipeline_parameters: {}
  wait_for_provisioning: True
  wait_timeout: 3600
  datastore_name: "workspaceblobstore"
  polling_interval: 5
  data_path_parameter_name: "input_data"
  continue_on_step_failure: None
  path_on_datastore: "file/path"
```

When defining a **recurring schedule**, use the following keys under `recurrence`:

YAML KEY	DESCRIPTION
<code>frequency</code>	How often the schedule recurs. Valid values are <code>"Minute"</code> , <code>"Hour"</code> , <code>"Day"</code> , <code>"Week"</code> , or <code>"Month"</code> .
<code>interval</code>	How often the schedule fires. The integer value is the number of time units to wait until the schedule fires again.

YAML KEY	DESCRIPTION
<code>start_time</code>	The start time for the schedule. The string format of the value is <code>YYYY-MM-DDThh:mm:ss</code> . If no start time is provided, the first workload is run instantly and future workloads are run based on the schedule. If the start time is in the past, the first workload is run at the next calculated run time.
<code>time_zone</code>	The time zone for the start time. If no time zone is provided, UTC is used.
<code>hours</code>	If <code>frequency</code> is "Day" or "Week", you can specify one or more integers from 0 to 23, separated by commas, as the hours of the day when the pipeline should run. Only <code>time_of_day</code> or <code>hours</code> and <code>minutes</code> can be used.
<code>minutes</code>	If <code>frequency</code> is "Day" or "Week", you can specify one or more integers from 0 to 59, separated by commas, as the minutes of the hour when the pipeline should run. Only <code>time_of_day</code> or <code>hours</code> and <code>minutes</code> can be used.
<code>time_of_day</code>	If <code>frequency</code> is "Day" or "Week", you can specify a time of day for the schedule to run. The string format of the value is <code>hh:mm</code> . Only <code>time_of_day</code> or <code>hours</code> and <code>minutes</code> can be used.
<code>week_days</code>	If <code>frequency</code> is "Week", you can specify one or more days, separated by commas, when the schedule should run. Valid values are "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", and "Sunday".

The following example contains the definition for a recurring schedule:

```

Schedule:
  description: "Test create with recurrence"
  recurrence:
    frequency: Week # Can be "Minute", "Hour", "Day", "Week", or "Month".
    interval: 1 # how often fires
    start_time: 2019-06-07T10:50:00
    time_zone: UTC
    hours:
      - 1
    minutes:
      - 0
    time_of_day: null
    week_days:
      - Friday
  pipeline_parameters:
    'a': 1
  wait_for_provisioning: True
  wait_timeout: 3600
  datastore_name: ~
  polling_interval: ~
  data_path_parameter_name: ~
  continue_on_step_failure: None
  path_on_datastore: ~

```

Next steps

Learn how to [use the CLI extension for Azure Machine Learning](#).

Azure Machine Learning sovereign cloud parity

12/23/2020 • 8 minutes to read • [Edit Online](#)

Learn what Azure Machine Learning features are available in sovereign cloud regions.

In the list of global Azure regions, there are several 'sovereign' regions that serve specific markets. For example, the Azure Government and the Azure China 21Vianet regions. Currently Azure Machine Learning is deployed into the following sovereign cloud regions:

- Azure Government regions **US-Arizona** and **US-Virginia**.
- Azure China 21Vianet region **China-East-2**.

TIP

To differentiate between sovereign and non-sovereign regions, this article will use the term **public cloud** to refer to non-sovereign regions.

We aim to provide maximum parity between our public cloud and sovereign regions. All Azure Machine Learning features will be available in these regions within **30 days of GA** (general availability) in our public cloud. We also enable a select number of preview features in these regions. Below display the current parity differences between our sovereign and public clouds.

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, CosmosDB, and HDInsight integrations	GA	YES	YES

FEATURE	PUBLIC CLOUD STATUS	US-VIRGINIA	US-ARIZONA
Machine Learning pipelines			
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	Public Preview	NO	NO
Compute instance			
Managed compute Instances for integrated Notebooks	GA	YES	YES
Jupyter, JupyterLab Integration	GA	YES	YES
Virtual Network (VNet) support	Public Preview	YES	YES
SDK support			
R SDK support	Public Preview	YES	YES
Python SDK support	GA	YES	YES

Feature	Public Cloud Status	US-Virginia	US-Arizona
Security			
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 link	Public Preview	NO	NO
ACI behind VNet	Public Preview	NO	NO
ACR behind VNet	Public Preview	NO	NO
Private IP of AKS cluster	Public Preview	NO	NO
Compute			
quota management across workspaces	GA	YES	YES
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 DevOps extension for Machine Learning & the Azure ML CLI	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			
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	GA	YES	YES
.NET integration ML.NET 1.0	GA	YES	YES
Inference			

FEATURE	PUBLIC CLOUD STATUS	US-VIRGINIA	US-ARIZONA
Batch inferencing	GA	YES	YES
Data Box Edge with FPGA	Public Preview	NO	NO
Other			
Open Datasets	Public Preview	YES	YES
Custom Cognitive Search	Public Preview	YES	YES
Many Models	Public Preview	NO	NO

Azure Government scenarios

SCENARIO	US-VIRGINIA	US-ARIZONA	LIMITATIONS
General security setup			
Private network communication between services	NO	NO	No Private Link currently
Disable/control internet access (inbound and outbound) and specific VNet	PARTIAL	PARTIAL	ACR behind VNet is not available in Azure Government - double checking on ACI
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 and private endpoint currently not available
Control (disable/limit/restrict) the use of ACI/AKS integration	PARTIAL	PARTIAL	ACI behind VNet and private endpoint 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	ACR behind private endpoint and VNet not supported in Azure Government

SCENARIO	US-VIRGINIA	US-ARIZONA	LIMITATIONS
General Machine Learning Service Usage			
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	

Additional 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 [VNet and forced tunneling](#) 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 additional 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 additional 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	Public Preview	YES	N/A
Industry-leading forecasting capabilities	GA	YES	N/A

Feature	Public Cloud Status	CH-East-2	CH-North-3
Support for deep learning and other advanced learners	GA	YES	N/A
Large data support (up to 100 GB)	Public Preview	YES	N/A
Azure Databricks Integration	GA	NO	N/A
SQL, CosmosDB, 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	NO	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	Public Preview	NO	N/A
Compute instance			
Managed compute Instances for integrated Notebooks	GA	NO	N/A

Feature	Public Cloud Status	CH-East-2	CH-North-3
Jupyter, JupyterLab Integration	GA	YES	N/A
Virtual Network (VNet) support	Public Preview	YES	N/A
SDK support			
R SDK support	Public Preview	YES	N/A
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	Public Preview	YES	N/A
Workplace Private link	Public Preview	NO	N/A
ACI behind VNet	Public Preview	NO	N/A
ACR behind VNet	Public Preview	NO	N/A
Private IP of AKS cluster	Public Preview	NO	N/A
Compute			
quota management across workspaces	GA	YES	N/A
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	Public Preview	YES	N/A
View, edit, or delete dataset drift monitors from the UI	Public Preview	YES	N/A

Feature	Public Cloud Status	CH-East-2	CH-North-3
Machine learning lifecycle			
Model profiling	GA	PARTIAL	N/A
The Azure DevOps extension for Machine Learning & the Azure ML CLI	GA	YES	N/A
FPGA-based Hardware Accelerated Models	GA	NO	N/A
Visual Studio Code integration	Public Preview	NO	N/A
Event Grid integration	Public Preview	YES	N/A
Integrate Azure Stream Analytics with Azure Machine Learning	Public 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)	Public Preview	YES	N/A
Responsible ML			
Explainability in UI	Public Preview	NO	N/A
Differential privacy SmartNoise toolkit	OSS	NO	N/A
custom tags in Azure Machine Learning to implement datasheets	GA	NO	N/A
Fairness AzureML Integration	Public 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	Public Preview	NO	N/A
Experimentation UI	GA	YES	N/A
.NET integration ML.NET 1.0	GA	YES	N/A
Inference			
Batch inferencing	GA	YES	N/A
Data Box Edge with FPGA	Public Preview	NO	N/A
Other			
Open Datasets	Public Preview	YES	N/A
Custom Cognitive Search	Public Preview	YES	N/A
Many Models	Public Preview	NO	N/A

Additional 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.azure 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 [VNet forced tunneling](#) 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 Link is enabled for now, but CI will be supported in the next deployment for the service expansion to all AML regions.

Next steps

To learn more about the regions that Azure Machine learning is available in, see [Products by region](#).

Azure Policy built-in policy definitions for Azure Machine Learning

12/23/2020 • 2 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)
Azure Machine Learning workspaces should be encrypted with a customer-managed key (CMK)	Evaluate Azure Machine Learning workspaces that do not have encryption enabled with customer-managed keys (CMK). Customer-managed keys add an additional layer of security for workspaces. For more information, visit https://aka.ms/azureml-workspaces-cmk .	Audit, Deny, Disabled	1.0.1
Azure Machine Learning workspaces should use private link	Evaluate Azure Machine Learning workspaces that do not have at least one approved private endpoint connection. Clients in a virtual network can securely access resources that have private endpoint connections through private links. For more information, visit: https://aka.ms/azureml-workspaces-privatelink .	Audit, Disabled	1.0.0
Configure allowed module authors for specified Azure Machine Learning computes	This policy helps provide allowed module authors in specified Azure Machine Learning computes and can be assigned at the workspace. For more information, visit https://aka.ms/amlpolicydoc .	enforceSetting, disabled	1.0.1-preview

NAME	DESCRIPTION	EFFECT(S)	VERSION
Configure allowed Python packages for specified Azure Machine Learning computes	This policy helps provide allowed Python packages in specified Azure Machine Learning computes and can be assigned at the workspace. For more information, visit https://aka.ms/amlpolicydoc .	enforceSetting, disabled	1.0.0-preview
Configure allowed registries for specified Azure Machine Learning computes	This policy helps provide registries that are allowed in specified Azure Machine Learning computes and can be assigned at the workspace. For more information, visit https://aka.ms/amlpolicydoc .	enforceSetting, disabled	1.0.0-preview
Configure an approval endpoint called prior to jobs running for specified Azure Machine Learning computes	This policy helps configure an approval endpoint called prior to jobs running for specified Azure Machine Learning computes and can be assigned at the workspace. For more information. For more information, visit https://aka.ms/amlpolicydoc .	enforceSetting, disabled	1.0.0-preview
Configure code signing for training code for specified Azure Machine Learning computes	This policy helps provide code signing for training code in specified Azure Machine Learning computes and can be assigned at the workspace. For more information, visit https://aka.ms/amlpolicydoc .	enforceSetting, disabled	1.0.0-preview
Configure log filter expressions and datastore to be used for full logs for specified Azure Machine Learning computes	This policy helps provide log filter expression and datastore to be used for full logs in specified Azure Machine Learning computes and can be assigned at the workspace. For more information, visit https://aka.ms/amlpolicydoc .	enforceSetting, disabled	1.0.0-preview

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

12/23/2020 • 2 minutes to read • [Edit Online](#)

This article lists the curated environments 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

This list is updated as of December 2020. Use the Python SDK to get the most updated list of environments and their dependencies. For more information, see the [environments article](#).

AutoML

- AzureML-AutoML
- AzureML-AutoML-DNN
- AzureML-AutoML-DNN-GPU
- AzureML-AutoML-DNN-Vision-GPU
- AzureML-AutoML-GPU

Chainer

- AzureML-Chainer-5.1.0-CPU
- AzureML-Chainer-5.1.0-GPU

Dask

- AzureML-Dask-CPU
- AzureML-Dask-GPU

DeepSpeed

- AzureML-DeepSpeed-0.3-GPU

Hyperdrive

- AzureML-Hyperdrive-ForecastDNN

Minimal

- AzureML-Minimal

PySpark

- AzureML-PySpark-MmlSpark-0.15

PyTorch

- AzureML-PyTorch-1.0-CPU
- AzureML-PyTorch-1.0-GPU
- AzureML-PyTorch-1.1-CPU
- AzureML-PyTorch-1.1-GPU
- AzureML-PyTorch-1.2-CPU
- AzureML-PyTorch-1.2-GPU
- AzureML-PyTorch-1.3-CPU
- AzureML-PyTorch-1.3-GPU
- AzureML-PyTorch-1.4-CPU
- AzureML-PyTorch-1.4-GPU
- AzureML-PyTorch-1.5-CPU
- AzureML-PyTorch-1.5-GPU
- AzureML-PyTorch-1.6-CPU
- AzureML-PyTorch-1.6-GPU

Scikit

- AzureML-Scikit-learn-0.20.3

TensorFlow

- AzureML-TensorFlow-1.10-CPU
- AzureML-TensorFlow-1.10-GPU
- AzureML-TensorFlow-1.12-CPU
- AzureML-TensorFlow-1.12-GPU
- AzureML-TensorFlow-1.13-CPU
- AzureML-TensorFlow-1.13-GPU
- AzureML-TensorFlow-2.0-CPU
- AzureML-TensorFlow-2.0-GPU
- AzureML-TensorFlow-2.1-CPU
- AzureML-TensorFlow-2.1-GPU
- AzureML-TensorFlow-2.2-CPU
- AzureML-TensorFlow-2.2-GPU
- AzureML-TensorFlow-2.3-CPU
- AzureML-TensorFlow-2.3-GPU

Triton

- AzureML-Triton

Tutorial

- AzureML-Tutorial

VowpalWabbit

- AzureML-VowpalWabbit-8.8.0

Azure Machine Learning release notes

12/23/2020 • 101 minutes to read • [Edit Online](#)

In this article, learn about Azure Machine Learning releases. For the full SDK reference content, visit the Azure Machine Learning's [main SDK for Python](#) reference page.

See [the list of known issues](#) to learn about known bugs and workarounds.

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

- Fixed a bug in setting a default pipeline for pipeline snapshot created via UI.
- o **azureml-pipeline-steps**
 - o Added experimental support for test data to AutoMLStep.
- o **azureml-tensorboard**
 - o Fixing the xref warnings for documentation in azureml-core package
- o **azureml-train-automl-client**
 - o Added experimental support for test data to AutoMLStep.
 - o Added the initial core implementation of test set ingestion feature.
 - o introduce a new AutoML task type of "image-instance-segmentation".
- o **azureml-train-automl-runtime**
 - o Added the initial core implementation of test set ingestion feature.
 - o Fix the computation of the raw explanations for the best AutoML model if the AutoML models are trained using validation_size setting.
 - o Moved references to sklearn.externals.joblib to depend directly on joblib.
- o **azureml-train-core**
 - o HyperDriveRun.get_children_sorted_by_primary_metric() should complete faster now
 - o Improved error handling in HyperDrive SDK.
 - o Deprecated all estimator classes in favor of using ScriptRunConfig to configure experiment runs. Deprecated classes include:
 - o MMLBaseEstimator
 - o Estimator
 - o PyTorch
 - o TensorFlow
 - o Chainer
 - o SKLearn
 - o Deprecated the use of Nccl and Gloo as valid input types for Estimator classes in favor of using PyTorchConfiguration with ScriptRunConfig.
 - o Deprecated the use of Mpi as a valid input type for Estimator classes in favor of using MpiConfiguration with ScriptRunConfig.
 - o Adding command property to runconfiguration. The feature enables users to run an actual command or executables on the compute through AzureML SDK.
 - o Deprecated all estimator classes in favor of using ScriptRunConfig to configure experiment runs. Deprecated classes include: + MMLBaseEstimator + Estimator + PyTorch + TensorFlow + Chainer + SKLearn
 - o Deprecated the use of Nccl and Gloo as a valid type of input for Estimator classes in favor of using PyTorchConfiguration with ScriptRunConfig.
 - o Deprecated the use of Mpi as a valid type of input for Estimator classes in favor of using MpiConfiguration with ScriptRunConfig.

2020-11-09

Azure Machine Learning SDK for Python v1.18.0

- Bug fixes and improvements
 - o **azureml-automl-core**
 - o 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-datatprep**
 - Better error message when using currently unsupported Spark with Scala 2.12.
 - **azureml-explain-model**
 - The azureml-explain-model package is officially deprecated
 - **azureml-mlflow**
 - Resolved a bug in mlflow.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 ValueException 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 enables 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 enables 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 nbconvert dependency to < 6 so that papermill 1.x continues to work.
 - **azureml-core**
 - Added parameters to the TensorflowConfiguration and MpiConfiguration 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-databricks**
 - 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-datatprep**
 - 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 "backlist_models" input parameter to "blocked_models".
 - Renamed "whitelist_models" input parameter to "allowed_models".
 - 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-itp**
 - CmAks compute type is supported. You can attach your own AKS cluster to the workspace for training job.
- **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-datatprep**
 - 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 OS errors 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-dataprep 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-datatrade[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 ph 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-databricks**
 - 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 ph 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 ph 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 Get started [here](#)

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
- Autosave
- 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. [Read more here.](#)
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-datatprep` 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. [Read more here](#). 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. [Read more here](#).

- 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-datatprep` 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-databriff](#) 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.
 - [azureml-contrib-dataset](#)
 - 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

- [azure-cli-ml](#)
 - CLI now supports model packaging.
 - Added dataset CLI. For more information: `az ml dataset --help`
 - 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.
- **azureml-automl-core**
 - Training an iteration would run in a child process only when runtime constraints are being set.
 - 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.
 - Added support for complex frequencies like two years and one month. -Added comprehensible error message if frequency cannot be determined.
 - Add azureml-defaults to auto generated conda env to solve the model deployment failure
 - Allow intermediate data in Azure Machine Learning Pipeline to be converted to tabular dataset and used in `AutoMLStep`.
 - Implemented column purpose update for streaming.
 - Implemented transformer parameter update for Imputer and HashOneHotEncoder for streaming.
 - Added the current data size and the minimum required data size to the validation error messages.
 - Updated the minimum required data size for Cross-validation to guarantee a minimum of two samples in each validation fold.
- **azureml-cli-common**
 - 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 DataDriftDector 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-dataadrift/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 `Ncc1` 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 thrown 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.

| Run Number | Experiment   | Creation Time         | Status   |
|------------|--------------|-----------------------|----------|
| 98         | spawnruns    | 9/9/2019, 9:30:14 ... | NotSt... |
| 583        | Schedule_Run | 9/9/2019, 9:07:48 ... | Compl... |
| 581        | Schedule_Run | 9/9/2019, 8:07:48 ... | Compl... |
| 579        | Schedule_Run | 9/9/2019, 7:07:49 ... | Compl... |
| 577        | Schedule_Run | 9/9/2019, 6:07:48 ... | Compl... |

| Name           | Type            | Provis... |
|----------------|-----------------|-----------|
| greg10         | Virtual Machine | Succ...   |
| greg11         | Virtual Machine | Succ...   |
| wb             | Virtual Machine | Succ...   |
| sindhub-weNBVM | Virtual Machine | Succ...   |
| sindhub-VM     | Virtual Machine | Succ...   |

### 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 v1.1.14

- 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 `Webservice` 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/" instead of ""
- **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 regards 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 regards 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](#).

# Service limits in Azure Machine Learning

12/23/2020 • 2 minutes to read • [Edit Online](#)

This section lists basic quotas and throttling thresholds in Azure Machine Learning.

## Workspaces

| LIMIT          | VALUE           |
|----------------|-----------------|
| Workspace name | 2-32 characters |

## Runs

| LIMIT                                    | VALUE            |
|------------------------------------------|------------------|
| Runs per workspace                       | 10 million       |
| RunId/ParentRunId                        | 256 characters   |
| DataContainerId                          | 261 characters   |
| DisplayName                              | 256 characters   |
| Description                              | 5,000 characters |
| Number of properties                     | 50               |
| Length of property key                   | 100 characters   |
| Length of property value                 | 1,000 characters |
| Number of tags                           | 50               |
| Length of tag key                        | 100              |
| Length of tag value                      | 1,000 characters |
| CancelUri / CompleteUri / DiagnosticsUri | 1,000 characters |
| Error message length                     | 3,000 characters |
| Warning message length                   | 300 characters   |
| Number of input datasets                 | 200              |
| Number of output datasets                | 20               |

## Metrics

| LIMIT                          | VALUE          |
|--------------------------------|----------------|
| Metric names per run           | 50             |
| Metric rows per metric name    | 10 million     |
| Columns per metric row         | 15             |
| Metric column name length      | 255 characters |
| Metric column value length     | 255 characters |
| Metric rows per batch uploaded | 250            |

### NOTE

If you are hitting the limit of metric names per run because you are formatting variables into the metric name, consider instead to use a row metric where one column is the variable value and the second column is the metric value.

## Artifacts

| LIMIT                       | VALUE            |
|-----------------------------|------------------|
| Number of artifacts per run | 10 million       |
| Max length of artifact path | 5,000 characters |

## Limit increases

Some limits can be increased for individual workspaces by [contacting support](#).

## Next steps

- [Configure your Azure Machine Learning environment](#)

# What happened to Azure Machine Learning Workbench?

12/23/2020 • 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](#), R 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.

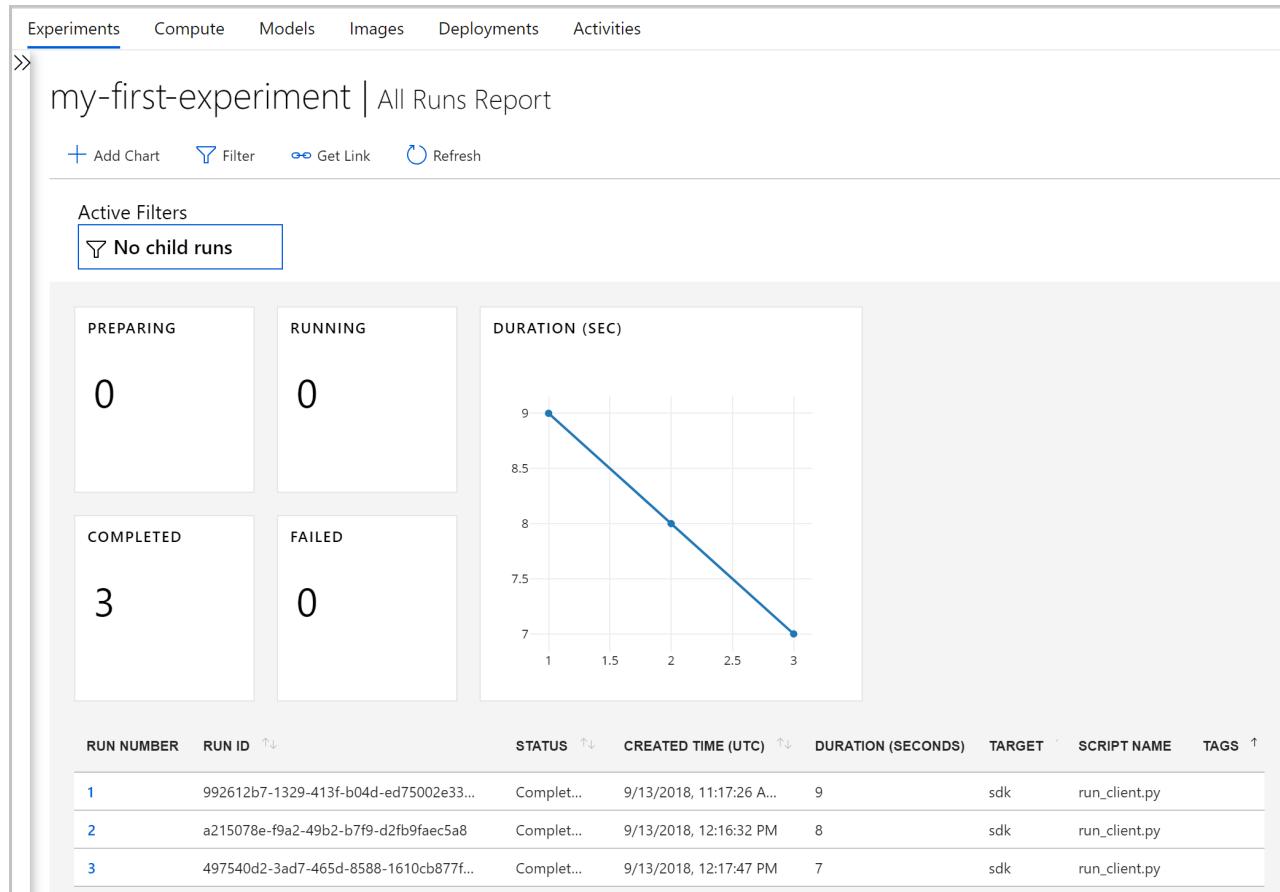
All the latest capabilities are available by using this [SDK](#), the [CLI](#), and the [portal](#).

## 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 portal's workspace dashboard is supported on Microsoft Edge, Chrome, and Firefox browsers only:



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:

```
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 and to Azure IoT Edge.

Learn more in these articles:

- [Where and how to deploy models](#)
- [Tutorial: Deploy models with Azure Machine Learning](#)

## Next steps

Learn about the [latest architecture for Azure Machine Learning](#).

For an overview of the service, read [What is Azure Machine Learning?](#).

Create your first experiment with your preferred method:

- [Use your own environment](#)
- [Use Python notebooks](#)
- [Use R Markdown](#)
- [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

11/2/2020 • 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 module list, which describes all the modules in the pipeline graph. The inner list is a connection list, which describes all the connections of a specific module.

1. In the module list, use the arrow key to switch modules.
2. Use tab to open the connection list for the target module.
3. Use arrow key to switch between the connection ports for the module.
4. Use "G" to go to the target module.

## Edit the pipeline graph

### Add a module to the graph

1. Use Ctrl+F6 to switch focus from the canvas to the module tree.
2. Find the desired module in the module tree using standard treeview control.

### Edit a module

To connect a module to another module:

1. Use Ctrl + Shift + H when targeting a module in the module list to open the connection helper.
2. Edit the connection ports for the module.

To adjust module properties:

1. Use Ctrl + Shift + E when targeting a module to open the module properties.
2. Edit the module properties.

## Navigation shortcuts

| KEYSTROKE        | DESCRIPTION                                                      |
|------------------|------------------------------------------------------------------|
| Ctrl + F6        | Toggle focus between canvas and module tree                      |
| Ctrl + F1        | Open the information card when focusing on a node in module tree |
| Ctrl + Shift + H | Open the connection helper when focus is on a node               |
| Ctrl + Shift + E | Open module properties when focus is on a node                   |

| KEYSTROKE | DESCRIPTION                                                |
|-----------|------------------------------------------------------------|
| 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](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                   |

## Next steps

- [Turn on high contrast or change theme](#)
- [Accessibility related tools at Microsoft](#)