

# Contents

## Azure Machine Learning service Documentation

### Overview

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

[What happened to Workbench](#)

### Quickstarts

[Get started - Portal](#)

[Get started - Python](#)

### Tutorials

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

1. Train a model

2. Deploy a model

[Regression \(NYC Taxi data\)](#)

1. Prepare data for modeling

2. Auto-train an ML model

### Samples

### Concepts

[How the service works](#)

[Automated machine learning](#)

[Model management](#)

[Accelerate with FPGAs](#)

[Machine learning pipelines](#)

[Enterprise security](#)

### How-to guides

[Manage workspaces](#)

Use portal for workspaces

Use a Resource Manager template

Manage users and roles

[Set up development environment](#)

Configure dev environments

## Visual Studio Code extension

[Get started](#)

[Train and deploy](#)

[Prepare data](#)

[Load data](#)

[Transform data](#)

[Write data](#)

[Access data](#)

[Train models](#)

[Set up training environments](#)

[Create estimators in training](#)

[Use PyTorch](#)

[Use TensorFlow & Keras](#)

[Tune hyperparameters](#)

[Track experiments and metrics](#)

[Automate machine learning](#)

[Configure auto training](#)

[Use remote compute targets](#)

[ONNX models](#)

[Deploy models](#)

[Where and how to deploy](#)

[FPGAs](#)

[Troubleshoot](#)

[Consume web services](#)

[Consume in real-time](#)

[Run batch predictions](#)

[Monitor web services](#)

[Collect & evaluate model data](#)

[Monitor with Application Insights](#)

[Security](#)

[Use virtual networks](#)

[Secure web services with SSL](#)

[Create your first pipeline](#)

[Enable logging](#)

[Manage resource quotas](#)

[Use the Machine Learning CLI](#)

[Export and delete data](#)

## Reference

[Machine learning SDK](#)

[Data prep SDK](#)

[Monitoring SDK](#)

## Resources

[Release notes](#)

[Azure roadmap](#)

[Pricing](#)

[Regional availability](#)

[Known issues](#)

[Get support](#)

[Compare our ML products](#)

# What is Azure Machine Learning service?

2/26/2019 • 4 minutes to read

Azure Machine Learning service is a cloud service that you use to train, deploy, automate, and manage machine learning models, all at the broad scale that the cloud provides.

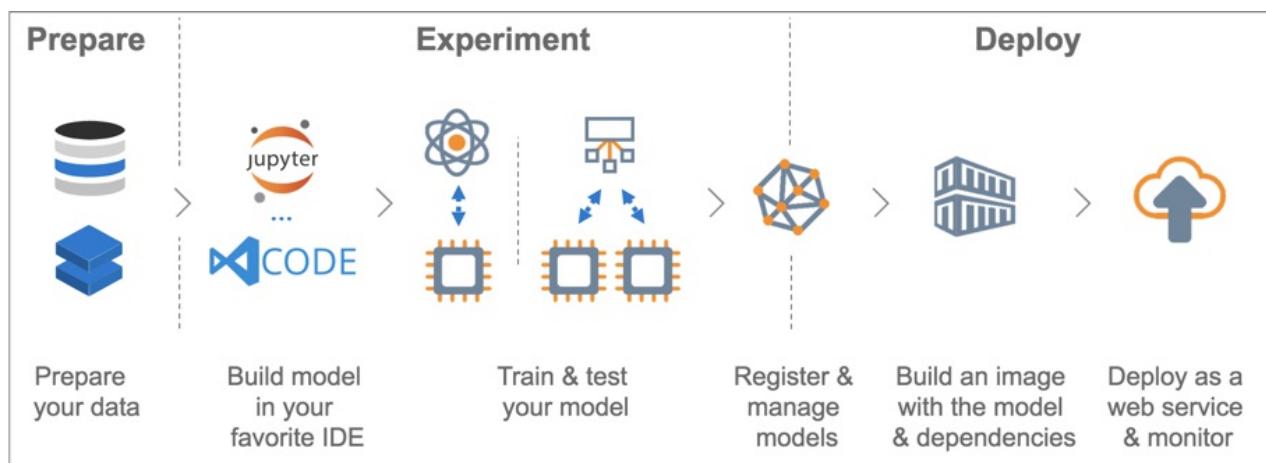
## What is machine learning?

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.

## What is Azure Machine Learning service?

Azure Machine Learning service provides a cloud-based environment you can use to prep data, train, test, deploy, manage, and track machine learning models.



Azure Machine Learning service fully supports open-source technologies. So you can use tens of thousands of open-source Python packages with machine learning components. Examples are PyTorch, TensorFlow, and scikit-learn. Support for rich tools makes it easy to interactively explore and prepare data and then develop and test models. Examples are [Jupyter notebooks](#) or the [Azure Machine Learning for Visual Studio Code](#) extension. Azure Machine Learning service also includes features that [automate model generation and tuning](#) to help you create models with ease, efficiency, and accuracy.

By using Azure Machine Learning service, you can start training on your local machine and then 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.

When you have the right model, you can easily deploy it in a container such as Docker. So it's simple to deploy to Azure Container Instances or Azure Kubernetes Service. Or you can use the container in your own deployments, either on-premises or in the cloud. For more information, see the article on [how to deploy and where](#).

You can manage the deployed models and track multiple runs as you experiment to find the best solution. After it's deployed, your model can return predictions in [real time](#) or [asynchronously](#) on large quantities of data.

And with advanced [machine learning pipelines](#), you can collaborate on all the steps of data preparation, model training and evaluation, and deployment.

## What can I do with Azure Machine Learning service?

Using the [main Python SDK](#) and the [Data Prep SDK](#) for Azure Machine Learning as well as open-source Python packages, you can build and train highly accurate machine learning and deep-learning models yourself in an Azure Machine Learning service Workspace. You can choose from many machine learning components available in open-source Python packages, such as the following examples:

- [Scikit-learn](#)
- [Tensorflow](#)
- [PyTorch](#)
- [CNTK](#)
- [MXNet](#)

Azure Machine Learning service can also autotrain a model and autotune it for you. For an example, see [Train a regression model with automated machine learning](#).

After you have a model, you use it to create a container, such as Docker, that can be deployed locally for testing. After testing is done, you can deploy the model as a production web service in either Azure Container Instances or Azure Kubernetes Service. 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](#) or the [Azure portal](#). You can evaluate model metrics, retrain, and redeploy new versions of the model, all while tracking the model's experiments.

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

## How is Azure Machine Learning service different from Machine Learning Studio?

[Azure Machine Learning Studio](#) is a collaborative, drag-and-drop visual workspace where you can build, test, and deploy machine learning solutions without needing to write code. It uses prebuilt and preconfigured machine learning algorithms and data-handling modules.

Use Machine Learning Studio when you want to experiment with machine learning models quickly and easily, and the built-in machine learning algorithms are sufficient for your solutions.

Use Machine Learning service if you work in a Python environment, you want more control over your machine learning algorithms, or you want to use open-source machine learning libraries.

### NOTE

Models created in Azure Machine Learning Studio can't be deployed or managed by Azure Machine Learning service.

## 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 service](#) 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. Or [activate MSDN subscriber benefits](#), which give you credits every month that you can use for paid Azure services.

## Next steps

- Create a Machine Learning service workspace to get started [by using the Azure portal](#) (no-install approach) or [in Python](#) (SDK install approach).
- Follow the full-length tutorials:
  - [Train an image classification model with Azure Machine Learning service](#)
  - [Prepare data and use automated machine learning to auto-train a regression model](#)
- Use the [Azure Machine Learning Data Prep SDK](#) to prepare your data.
- Learn about [machine learning pipelines](#) to build, optimize, and manage your machine learning scenarios.
- Read the in-depth [Azure Machine Learning service architecture and concepts](#) article.
- For more information, see [other machine learning products from Microsoft](#).

# What happened to Azure Machine Learning Workbench?

2/26/2019 • 5 minutes to read

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 [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 service 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 service 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 service 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 service 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 portal](#). 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 has ended.

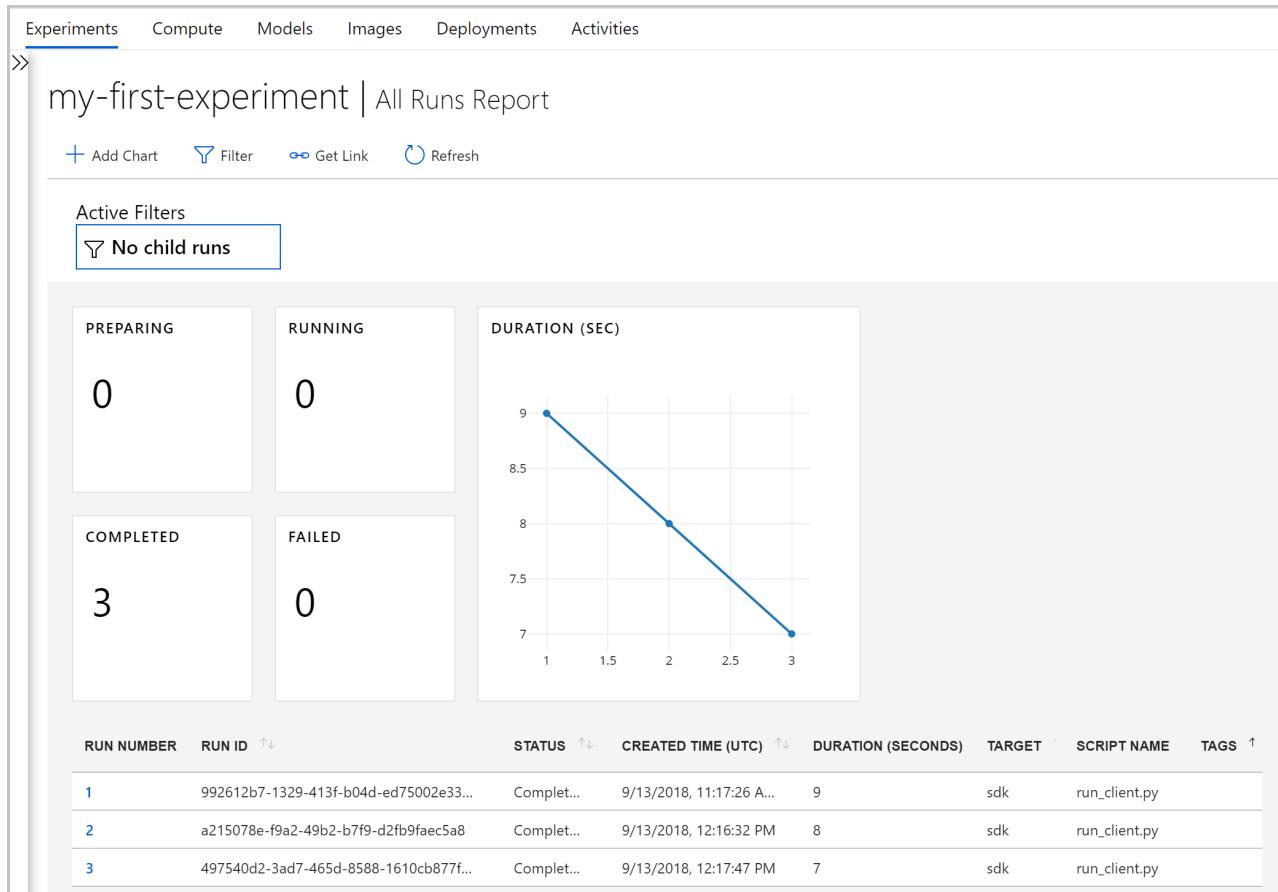
All the latest capabilities are available by using this [SDK](#), the [CLI](#), and the [portal](#).

## 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 portal.

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 service](#).

## Can I still prep data?

Your pre-existing data preparation files aren't portable to the latest release because we don't have Machine Learning Workbench anymore. But you can still prepare any size data set for modeling.

With data sets of any size, you can use the [Azure Machine Learning Data Prep SDK](#) to quickly prepare your data prior to modeling by writing Python code.

You can follow [this tutorial](#) to learn more about how to use Azure Machine Learning Data Prep SDK.

## 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 service 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)
```

Learn how to get started in [Python with the main SDK](#) or using [Azure portal](#).

## 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 must be re-created in the new workspace to continue to use them. You can re-create these images by following the [Configure and create image](#) sections.

## 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 service](#)

## What about the old SDK and CLI?

Yes, they'll continue to work until January. See the preceding [timeline](#). We recommend that you start creating your new experiments and models with the latest SDK or CLI.

By using the new Python SDK in the latest release, you can interact with Azure Machine Learning service in any Python environment. Learn how to install the latest [SDK](#). You can also use the updated [Azure Machine Learning CLI extension](#) with the rich set of `az ml` commands to interact with the service in any command-line environment, including Azure Cloud Shell.

## What about Visual Studio Code Tools for AI?

In this latest release, the extension was renamed to Azure Machine Learning for Visual Studio Code and has been expanded and improved to work with the preceding new features.

```

train.py - MNIST - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
AZURE M... train.py score.py ...
MyTeamWorkspace
Experiments
MNIST
DiabetesDete...
Compute
traincluster
Models
MNIST:1
Images
mnist-svc1
Deployments
mnist-svc
Project MNIST
Experiments: MNIST
Run Configs
docker
gpu_cluster
local
1 from azureml.core import Run
2 import numpy as np
3 import argparse
4 import os
5 import tensorflow as tf
6 import sys
7 import os
8 import utils
9 from utils import load_data, one_hot_encode
10
11 # Get training data to recognize handwritten digits
12
13 os.makedirs('./data/mnist', exist_ok = True)
14
15 def prepare_data():
16
17     n_inputs = 28 * 28
18     n_h1 = 300
19     n_h2 = 100
20     n_outputs = 10
21     learning_rate = 0.01
22     n_epochs = 20
23     batch_size = 50
24
25     with tf.name_scope('network'):
26
27         # construct the DNN
28         X = tf.placeholder(tf.float32, shape = (None, n_inputs))
29         y = tf.placeholder(tf.int64, shape = (None))
30         h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu)
31         h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu)
32         output = tf.layers.dense(h2, n_outputs, name = 'output')
33
34         # define loss and optimizer
35         loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = output))
36         optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)
37
38         # evaluate accuracy
39         correct_predictions = tf.equal(tf.argmax(output, 1), y)
40         accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32))
41
42     return accuracy, loss, X, y, optimizer
43
44
45
46
47
48
49
50

```

Ln 1, Col 10 Spaces: 4 UTF-8 CRLF Python 3

## What about domain packages?

The domain packages for computer vision, text analytics, and forecasting can't be used with the latest version of Azure Machine Learning. However, you can still build and train computer vision, text, and forecasting models with the latest Azure Machine Learning Python [SDK](#). To learn how to migrate pre-existing models built by using the computer vision, text analytics, and forecasting packages, contact [AML-Packages@microsoft.com](mailto:AML-Packages@microsoft.com).

## Next steps

Learn about the [latest architecture for Azure Machine Learning service](#).

For an overview of the service, read [What is Azure Machine Learning service?](#)

For a quickstart showing you how to create a workspace, create a project, run a script, and explore the run history of the script with the latest version of Azure Machine Learning service, try [get started with Azure Machine Learning service](#).

For a more in-depth experience of this workflow, follow the [full-length tutorial](#) that contains detailed steps for training and deploying models with Azure Machine Learning service.

# Quickstart: Use the Azure portal to get started with Azure Machine Learning

2/22/2019 • 5 minutes to read

Use the Azure portal to create an Azure Machine Learning workspace. This workspace is the foundational block in the cloud that you use to experiment, train, and deploy machine learning models with Machine Learning. This quickstart uses cloud resources and requires no installation. To configure your own Jupyter Notebooks server instead, see [Quickstart: Use Python to get started with Azure Machine Learning](#).

In this quickstart, you take the following actions:

- Create a workspace in your Azure subscription.
- Try it out with Python in a Jupyter notebook and log values across multiple iterations.
- View the logged values in your workspace.

The following Azure resources are added automatically to your workspace when they're regionally available:

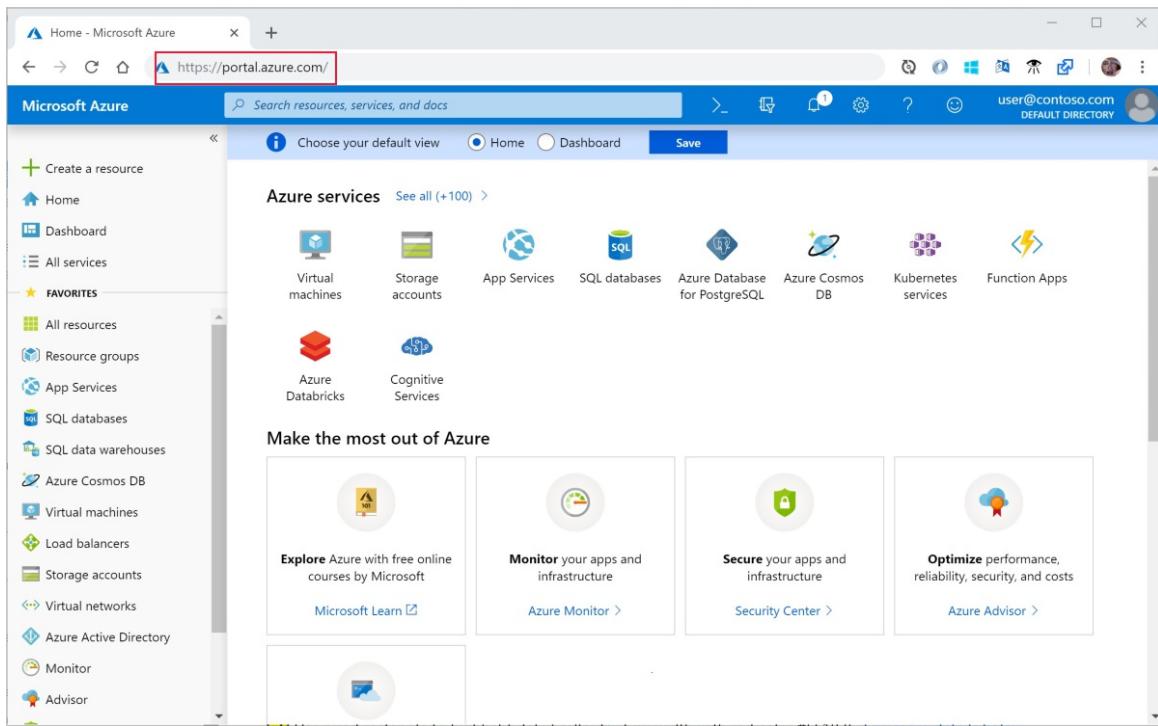
- [Azure Container Registry](#)
- [Azure Storage](#)
- [Azure Application Insights](#)
- [Azure Key Vault](#)

The resources you create can be used as prerequisites to other Machine Learning service tutorials and how-to articles. As with other Azure services, there are limits on certain resources associated with Machine Learning. An example is compute cluster size. Learn more about the [default limits and how to increase your quota](#).

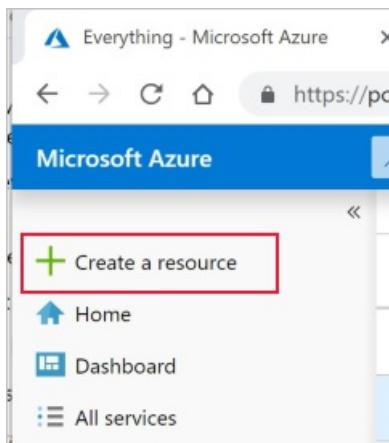
If you don't have an Azure subscription, create a free account before you begin. Try the [free or paid version of Azure Machine Learning service](#) today.

## Create a workspace

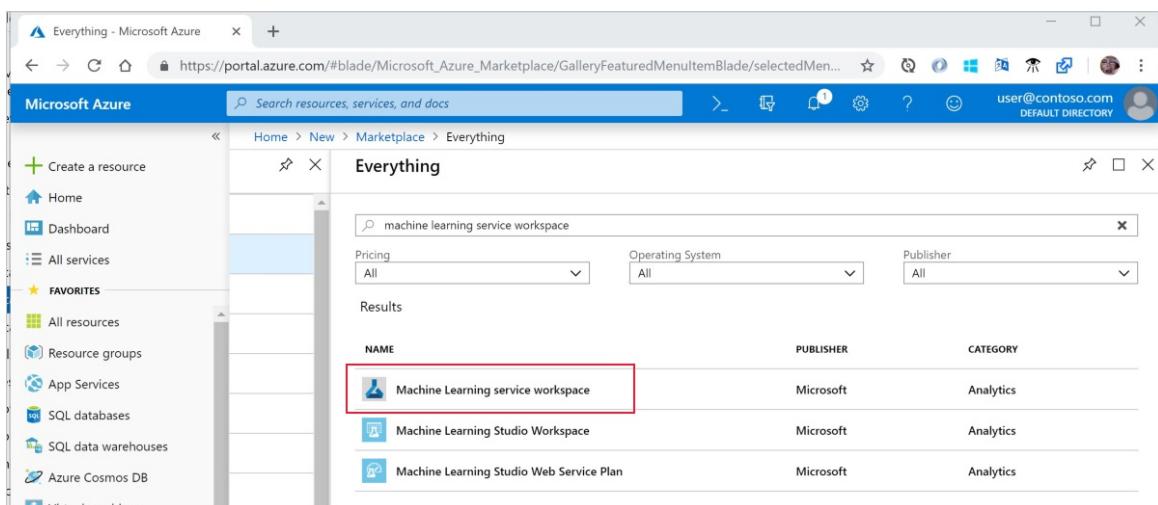
1. Sign in to the [Azure portal](#) by using the credentials for the Azure subscription you use.



2. In the upper-left corner of the portal, select **Create a resource**.



3. In the search bar, enter **Machine Learning**. Select the **Machine Learning service workspace** search result.



4. In the **ML service workspace** pane, scroll to the bottom and select **Create** to begin.

The screenshot shows the Azure Machine Learning service workspace creation interface. At the top, there are navigation icons and a user sign-in area. Below the header, the title "Machine Learning service workspace" is displayed, followed by a Microsoft logo. A descriptive text block explains that Azure Machine Learning is a secure and powerful cloud-based offering for rapidly building, deploying, and monitoring advanced machine learning and analytics solutions. It notes that this workspace template is different from the Machine Learning Studio Workspace. A "Save for later" button is visible. The main content area displays a preview of the workspace configuration, including sections for "Overview", "Activity log", "Access control (IAM)", "Tags", "Diagnostic and error problems", "Metrics", "Logs", "Automation script", "Properties", and "SUPPORT - FEEDBACK". A "Create" button at the bottom of this section is highlighted with a red border.

5. In the **ML service workspace** pane, configure your workspace.

FIELD	DESCRIPTION
Workspace name	Enter a unique name that identifies your workspace. In this example, we use <b>docs-ws</b> . Names must be unique across the resource group. Use a name that's easy to recall and 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 is a container that holds related resources for an Azure solution. In this example, we use <b>docs-aml</b> .
Location	Select the location closest to your users and the data resources. This location is where the workspace is created.

**ML service workspace**

Machine Learning service workspace

\* Workspace name

\* Subscription

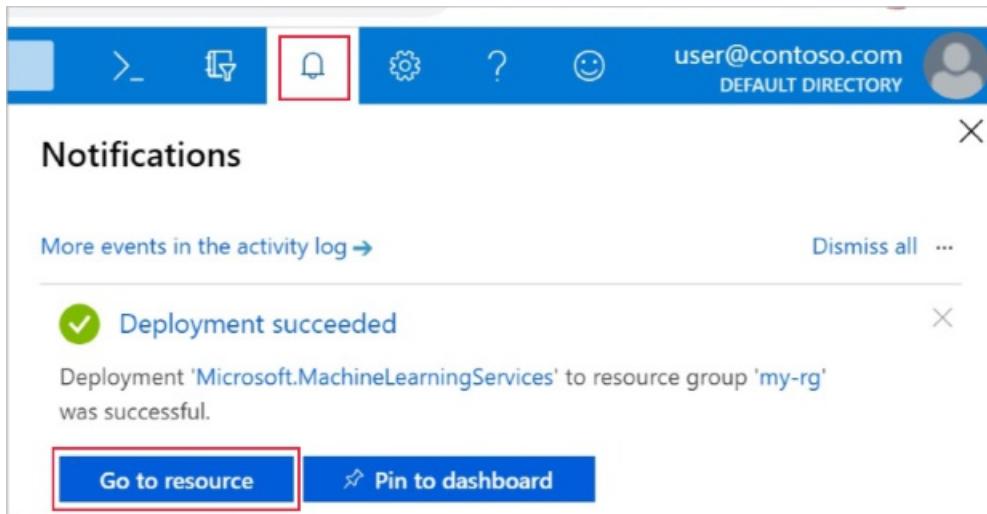
\* Resource group

\* Location

**Info** For your convenience, these resources are added automatically to the workspace, if regionally available: Azure Container Registry, Azure storage, Azure Application Insights and Azure Key Vault.

**Create** **Automation options**

6. To start the creation process, select **Create**. It can take a few moments to create the workspace.
7. To check on the status of the deployment, select the Notifications icon, **bell**, on the toolbar.
8. When the process is finished, a deployment success message appears. It's also present in the notifications section. To view the new workspace, select **Go to resource**.



## Use the workspace

Now learn how a workspace helps you manage your machine learning scripts. In this section, you take the following steps:

- Open a notebook in Azure Notebooks.
- Run code that creates some logged values.
- View the logged values in your workspace.

This example shows how the workspace can help you keep track of information generated in a script.

## Open a notebook

Azure Notebooks provides a free cloud platform for Jupyter notebooks that is preconfigured with everything you need to run Machine Learning. From your workspace you can launch this platform to get started using your Azure Machine Learning service workspace.

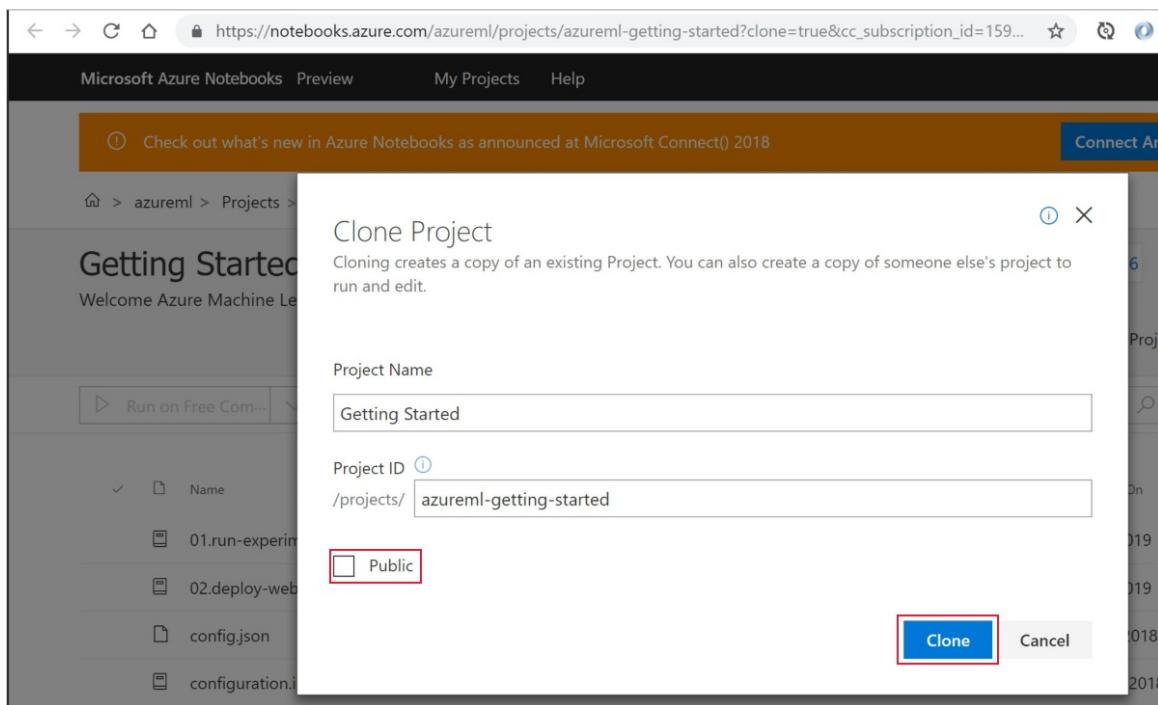
1. On the workspace page, select **Explore your Azure Machine Learning service Workspace**.

The screenshot shows the 'myworkspace' machine learning service workspace. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Locks, Automation script, Properties), Application (Experiments, Pipelines, Compute, Models), and a search bar. The main content area displays workspace details: Resource group (my-rg), Location (East US), Subscription (Visual Studio Ultimate with MSDN), Subscription ID (xxxxxx-xxxx-xxxx-xxxx-xxxxxxxx), Storage (myworkspace1716152667), Registry (myworkspace0777891341), Key Vault (myworkspace3472481689), and Application Insights (myworkspace9913803213). Below this is a 'Getting Started' section with a callout box titled 'Explore your Azure Machine Learning service workspace' containing the text 'Explore your Machine Learning service workspace to run and track experiments, compare model performance, and deploy models.' There are also 'View Documentation' and 'View Forum' buttons.

2. Select **Open Azure Notebooks** to try your first experiment in Azure Notebooks. Azure Notebooks is a separate service that lets you run Jupyter notebooks for free in the cloud. When you use this link to the service, information about how to connect to your workspace will be added to the library you create in Azure Notebooks.

The screenshot shows the Microsoft Azure portal with the 'myworkspace' machine learning service workspace selected. The left sidebar lists 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES' (Resource groups, App Services, SQL databases, SQL data warehouses, Azure Cosmos DB), and 'All resources'. The main content area has tabs for Experiments, Pipelines, Compute, Models, Images, Deployments, and Activities. A 'Welcome to your new Workspace' message is displayed. Below it are two sections: '1. Getting started' (Create your first experiment in Azure Notebooks to be able to view and track metrics) with a 'Open Azure Notebooks' button, and '2. Done getting started?' (Once you run the Azure Notebook, you will be able to view the data from the experiment in the Experiments page) with a 'View Experiments' button.

3. Sign into Azure Notebooks. Make sure you sign in with the same account you used to sign into the Azure portal. Your organization might require **administrator consent** before you can sign in.
4. After you sign in, a new tab opens and a **Clone Library** prompt appears. Cloning this library will load a set of notebooks and other files into your Azure Notebooks account. These files help you explore the capabilities of Azure Machine Learning.
5. Uncheck **Public** so that you don't share your workspace information with others.
6. Select **Clone**.



7. If you see that the project status is stopped, click on **Run on Free Computer** to use the free notebook server.

**Getting Started**

Welcome Azure Machine Learning service through Azure Notebooks

Cloned from [azureml/azureml-getting-started](#)

Status: **Stopped**

**Run on Free Co...**

### Run the notebook

In the list of files for this project, you see a `config.json` file. This config file contains information about the workspace you created in the Azure portal. This file allows your code to connect to and add information into your workspace.

1. Select **01.run-experiment.ipynb** to open the notebook.
2. The status area tells you to wait until the kernel has started. The message disappears once the kernel is ready.

3. After the kernel has started, run the cells one at a time using **Shift+Enter**. Or select **Cells > Run All** to run the entire notebook. When you see an asterisk, \*, next to a cell, the cell is still running. After the code for that cell finishes, a number appears.
4. Follow instructions in the notebook to authenticate your Azure subscription.

After you've finished running all of the cells in the notebook, you can view the logged values in your workspace.

## View logged values

1. The output from the `run` cell contains a link back to the Azure portal to view the experiment results in your workspace.

In [5]:	run												
Out[5]:	<table border="1"> <thead> <tr> <th>Experiment</th> <th>Id</th> <th>Type</th> <th>Status</th> <th>Details Page</th> <th>Docs Page</th> </tr> </thead> <tbody> <tr> <td>my-first-experiment</td> <td>57eba271-865f-4d37-aca1-0d86408b9dd7</td> <td></td> <td>Completed</td> <td><a href="#">Link to Azure Portal</a></td> <td><a href="#">Link to Documentation</a></td> </tr> </tbody> </table>	Experiment	Id	Type	Status	Details Page	Docs Page	my-first-experiment	57eba271-865f-4d37-aca1-0d86408b9dd7		Completed	<a href="#">Link to Azure Portal</a>	<a href="#">Link to Documentation</a>
Experiment	Id	Type	Status	Details Page	Docs Page								
my-first-experiment	57eba271-865f-4d37-aca1-0d86408b9dd7		Completed	<a href="#">Link to Azure Portal</a>	<a href="#">Link to Documentation</a>								

2. Click the **Link to Azure Portal** to view information about the run in your workspace. This link opens your workspace in the Azure portal.
3. The plots of logged values you see were automatically created in the workspace. Whenever you log multiple values with the same name parameter, a plot is automatically generated for you.

ATTRIBUTES	
Status	Completed
Created Time	Jan 17, 2019 3...
Duration	4.956
Target	sdk
Run ID	57eba271-865...
Run Number	1

CHARTS	
<b>Pi estimate</b> 	<b>Error</b> 

Because the code to approximate pi uses random values, your plots will show different values.

## Clean up resources

## IMPORTANT

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

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.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a red box around the 'Resource groups' option under the 'New' section. The main content area is titled 'newacct' and shows the 'Overview' tab selected. At the top right of this area, there is a red box around the 'Delete resource group' button. Below the overview, there is some subscription information and a table listing one item: 'newacct' (Azure Cosmos DB account) located in South Central US.

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

You created the necessary resources to experiment with and deploy models. You also ran some code in a notebook. And you explored the run history from that code in your workspace in the cloud.

For an in-depth workflow experience, follow Machine Learning tutorials to train and deploy a model:

[Tutorial: Train an image classification model](#)

# Quickstart: Use the Python SDK to get started with Azure Machine Learning

3/5/2019 • 5 minutes to read

In this article, you use the Azure Machine Learning SDK for Python 3 to create and then use an Azure Machine Learning service [workspace](#). The workspace is the foundational block in the cloud that you use to experiment, train, and deploy machine learning models with Machine Learning.

You begin by configuring your own Python environment and Jupyter Notebook Server. To run it with no installation, see [Quickstart: Use the Azure portal to get started with Azure Machine Learning](#).

View a video version of this quickstart:

In this quickstart, you:

- Install the Python SDK.
- Create a workspace in your Azure subscription.
- Create a configuration file for that workspace to use later in other notebooks and scripts.
- Write code that logs values inside the workspace.
- View the logged values in your workspace.

You create a workspace and a configuration file to use as prerequisites to other Machine Learning tutorials and how-to articles. As with other Azure services, certain limits and quotas are associated with Machine Learning. [Learn about quotas and how to request more](#).

The following Azure resources are added automatically to your workspace when they're regionally available:

- [Azure Container Registry](#)
- [Azure Storage](#)
- [Azure Application Insights](#)
- [Azure Key Vault](#)

## NOTE

Code in this article requires Azure Machine Learning SDK version 1.0.2 or later and was tested with version 1.0.8.

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 service](#) today.

## Install the SDK

### IMPORTANT

Skip this section if you use an Azure Data Science Virtual Machine or Azure Databricks.

- Azure Data Science Virtual Machines created after September 27, 2018 come with the Python SDK preinstalled.
- In the Azure Databricks environment, use the [Databricks installation steps](#) instead.

Before you install the SDK, we recommend that you create an isolated Python environment. Although this article uses [Miniconda](#), you can also use full [Anaconda](#) installed or [Python virtualenv](#).

The instructions in this quickstart will install all the packages you need to run the quickstart and tutorial notebooks. Other sample notebooks may require installation of additional components. For more information about these components, see [Install the Azure Machine Learning SDK for Python](#).

## Install Miniconda

[Download and install Miniconda](#). Select the Python 3.7 version to install. Don't select the Python 2.x version.

## Create an isolated Python environment

1. Open a command-line window, then create a new conda environment named *myenv* and install Python 3.6.5. Azure Machine Learning SDK will work with Python 3.5.2 or later, but the automated machine learning components are not fully functional on Python 3.7. It will take several minutes to create the environment while components and packages are downloaded.

```
conda create -n myenv python=3.6.5
```

2. Activate the environment.

```
conda activate myenv
```

3. Enable environment-specific ipython kernels:

```
conda install notebook ipykernel
```

Then create the kernel:

```
ipython kernel install --user
```

## Install the SDK

1. In the activated conda environment, install the core components of the Machine Learning SDK with Jupyter notebook capabilities. The installation takes a few minutes to finish based on the configuration of your machine.

```
pip install --upgrade azureml-sdk[notebooks]
```

2. To use this environment for the Azure Machine Learning tutorials, install these packages.

```
conda install -y cython matplotlib pandas
```

3. To use this environment for the Azure Machine Learning tutorials, install the automated machine learning components.

```
pip install --upgrade azureml-sdk[automl]
```

## IMPORTANT

In some command-line tools, you might need to add quotation marks as follows:

- 'azureml-sdk[notebooks]'
- 'azureml-sdk[automl]'

## Create a workspace

Create your workspace in a Jupyter Notebook using the Python SDK.

1. Create and/or cd to the directory you want to use for the quickstart and tutorials.
2. To launch Jupyter Notebook, enter this command:

```
jupyter notebook
```

3. In the browser window, create a new notebook by using the default `Python 3` kernel.
4. To display the SDK version, enter and then execute the following Python code in a notebook cell:

```
import azureml.core  
print(azureml.core.VERSION)
```

5. Find a value for the `<azure-subscription-id>` parameter in the [subscriptions list in the Azure portal](#). Use any subscription in which your role is owner or contributor.

```
from azureml.core import Workspace  
ws = Workspace.create(name='myworkspace',  
                      subscription_id='<azure-subscription-id>',  
                      resource_group='myresourcegroup',  
                      create_resource_group=True,  
                      location='eastus2'  
)
```

When you execute the code, you might be prompted to sign into your Azure account. After you sign in, the authentication token is cached locally.

6. To view the workspace details, such as associated storage, container registry, and key vault, enter the following code:

```
ws.get_details()
```

## Write a configuration file

Save the details of your workspace in a configuration file to the current directory. This file is called `aml_config\config.json`.

This workspace configuration file makes it easy to load the same workspace later. You can load it with other notebooks and scripts in the same directory or a subdirectory.

```
# Create the configuration file.  
ws.write_config()  
  
# Use this code to load the workspace from  
# other scripts and notebooks in this directory.  
# ws = Workspace.from_config()
```

This `write_config()` API call creates the configuration file in the current directory. The `config.json` file contains the following:

```
{  
    "subscription_id": "<azure-subscription-id>",  
    "resource_group": "myresourcegroup",  
    "workspace_name": "myworkspace"  
}
```

## Use the workspace

Run some code that uses the basic APIs of the SDK to track experiment runs:

1. Create an experiment in the workspace.
2. Log a single value into the experiment.
3. Log a list of values into the experiment.

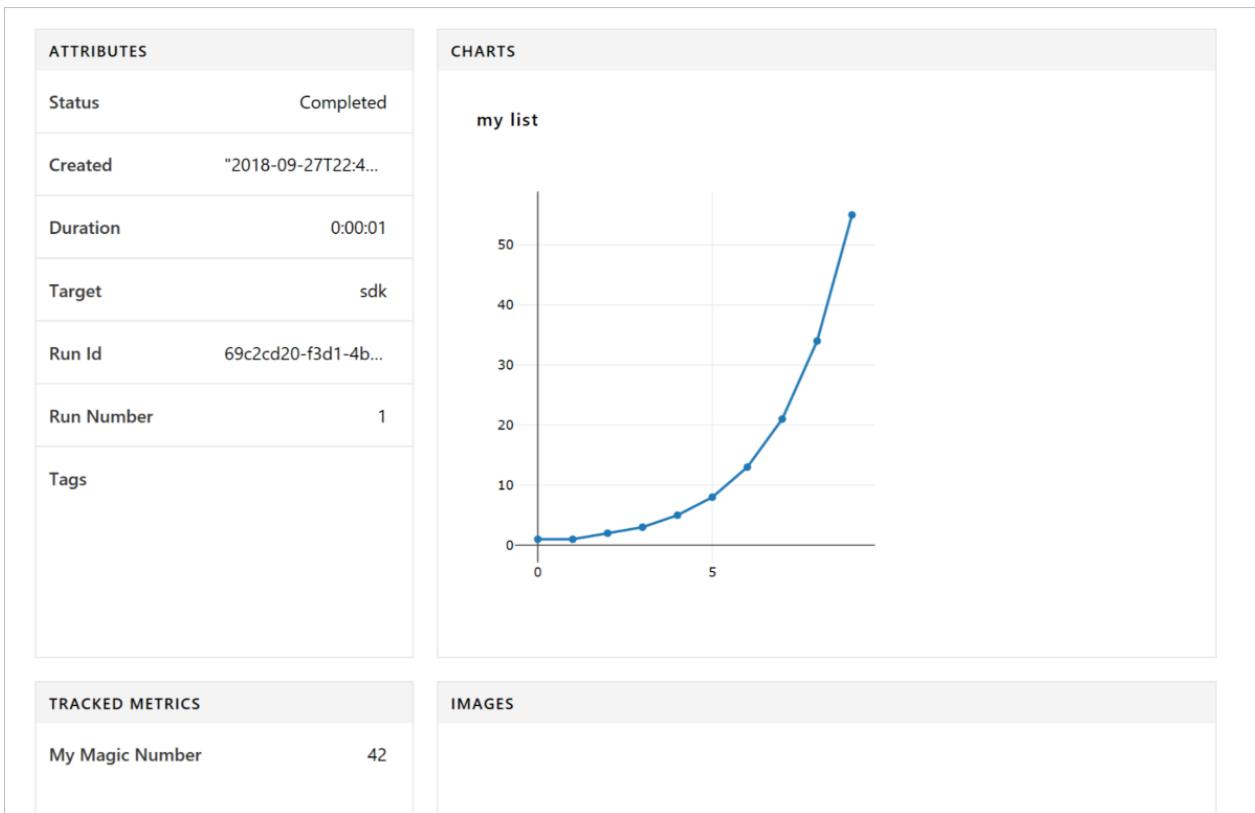
```
from azureml.core import Experiment  
  
# Create a new experiment in your workspace.  
exp = Experiment(workspace=ws, name='myexp')  
  
# Start a run and start the logging service.  
run = exp.start_logging()  
  
# Log a single number.  
run.log('my magic number', 42)  
  
# Log a list (Fibonacci numbers).  
run.log_list('my list', [1, 1, 2, 3, 5, 8, 13, 21, 34, 55])  
  
# Finish the run.  
run.complete()
```

## View logged results

When the run finishes, you can view the experiment run in the Azure portal. To print a URL that navigates to the results for the last run, use the following code:

```
print(run.get_portal_url())
```

Use the link to view the logged values in the Azure portal in your browser.



## Clean up resources

### IMPORTANT

You can use the resources you've created here as prerequisites to other Machine Learning tutorials and how-to articles.

If you don't plan to use the resources that you created in this article, delete them to avoid incurring any charges.

```
ws.delete(delete_dependent_resources=True)
```

## Next steps

In this article, you created the resources you need to experiment with and deploy models. You ran code in a notebook, and you explored the run history for the code in your workspace in the cloud.

[Tutorial: Train an image classification model](#)

You can also explore [more advanced examples on GitHub](#).

# Tutorial: Train an image classification model with Azure Machine Learning service

2/15/2019 • 13 minutes to read

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 service (preview) 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 service. 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 multiclass 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 locally by using the popular scikit-learn machine learning library.
- Train multiple models 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 service](#) today.

## NOTE

Code in this article was tested with Azure Machine Learning SDK version 1.0.8.

## Prerequisites

Skip to [Set up your development environment](#) to read through the notebook steps, or use the instructions below to get the notebook and run it on Azure Notebooks or your own notebook server. To run the notebook you will need:

- A Python 3.6 notebook server with the following installed:
  - The Azure Machine Learning SDK for Python
  - `matplotlib` and `scikit-learn`
- The tutorial notebook and the file `utils.py`
- A machine learning workspace
- The configuration file for the workspace in the same directory as the notebook

Get all these prerequisites from either of the sections below.

- Use [Azure Notebooks](#)
- Use [your own notebook server](#)

**Use Azure Notebooks: Free Jupyter notebooks in the cloud**

It's easy to get started with Azure Notebooks! The [Azure Machine Learning SDK for Python](#) is already installed and configured for you on [Azure Notebooks](#). The installation and future updates are automatically managed via Azure services.

After you complete the steps below, run the **tutorials/img-classification-part1-training.ipynb** notebook in your **Getting Started** project.

1. Complete the [Azure Machine Learning portal quickstart](#) to create a workspace and launch Azure Notebooks. Feel free to skip the **Use the notebook** section if you wish.
2. If you've already completed the [quickstart](#), sign back into [Azure Notebooks](#) and open the **Getting Started** project.
3. Remember to start the project if its status is stopped.

The screenshot shows the Azure Notebooks interface. At the top, there is a navigation bar with a home icon, followed by 'My Projects' and 'Getting Started'. Below this, the title 'Getting Started' is displayed in large letters. A sub-header says 'Welcome Azure Machine Learning service through Azure Notebooks'. It indicates that the project was 'Cloned from azureml/azureml-getting-started' and shows a 'Status: Stopped'. At the bottom, there are two buttons: a red-bordered 'Run on Free Co...' button and a standard blue 'dropdown' button.

### Use your own Jupyter notebook server

Use these steps to create a local Jupyter Notebook server on your computer. After you complete the steps, run the **tutorials/img-classification-part1-training.ipynb** notebook.

1. Complete the [Azure Machine Learning Python quickstart](#) to install the SDK and create a workspace. Feel free to skip the **Use the notebook** section if you wish.
2. Clone [the GitHub repository](#).

```
git clone https://github.com/Azure/MachineLearningNotebooks.git
```

3. Add a workspace configuration file using either of these methods:

- Copy the **aml\_config\config.json** file you created using the prerequisite quickstart into the cloned directory.
- Create a new workspace using code in the [configuration.ipynb](#).

4. Start the notebook server from your cloned directory.

```
jupyter notebook
```

## 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, ws.location, sep = '\t')
```

## Create an experiment

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

```
experiment_name = 'sklearn-mnist'

from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)
```

## Create or attach an existing compute resource

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

**Creation of the compute takes about five minutes.** If the compute 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", "cpucluster")
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 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. You also need to copy the data into the cloud. Then it can be accessed by your cloud training environment. In this section, you learn how to take the following actions:

- Download the MNIST dataset.
- Display some sample images.
- Upload data to the cloud.

### Download the MNIST dataset

Download the MNIST dataset and save the files into a `data` directory locally. Images and labels for both training and testing are downloaded:

```

import urllib.request
import os

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

urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz',
filename=os.path.join(data_folder, 'train-images.gz'))
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz',
filename=os.path.join(data_folder, 'train-labels.gz'))
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz',
filename=os.path.join(data_folder, 'test-images.gz'))
urllib.request.urlretrieve('http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz',
filename=os.path.join(data_folder, 'test-labels.gz'))

```

You will see output similar to this: ('./data/test-labels.gz', <http.client.HTTPMessage at 0x7f40864c77b8>)

### 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 `util.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

# note we also shrink the intensity values (X) from 0-255 to 0-1. This helps the model converge faster.
X_train = load_data(os.path.join(data_folder, 'train-images.gz'), False) / 255.0
X_test = load_data(os.path.join(data_folder, 'test-images.gz'), False) / 255.0
y_train = load_data(os.path.join(data_folder, 'train-labels.gz'), True).reshape(-1)
y_test = load_data(os.path.join(data_folder, 'test-labels.gz'), 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:

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

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

### Upload data to the cloud

Now make the data accessible remotely by uploading that data from your local machine into Azure. Then it can be accessed for remote training. The datastore is a convenient construct associated with your workspace for you to upload or download data. You can also interact with it from your remote compute targets. It's backed by an Azure Blob storage account.

The MNIST files are uploaded into a directory named `mnist` at the root of the datastore:

```
ds = ws.get_default_datastore()
print(ds.datastore_type, ds.account_name, ds.container_name)

ds.upload(src_dir=data_folder, target_path='mnist', overwrite=True, show_progress=True)
```

You now have everything you need to start training a model.

## Train on a remote cluster

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

- Create a directory
- Create a training script
- Create an estimator object
- 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

from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib

from azureml.core import Run
from utils import load_data

# let user feed in 2 parameters, the location of the data files (from datastore), 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(os.path.join(data_folder, 'train-images.gz'), False) / 255.0
X_test = load_data(os.path.join(data_folder, 'test-images.gz'), False) / 255.0
y_train = load_data(os.path.join(data_folder, 'train-labels.gz'), True).reshape(-1)
y_test = load_data(os.path.join(data_folder, 'test-labels.gz'), 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, 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**:

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

Anything written in this directory is automatically uploaded into your workspace. You access your model from this directory later in the tutorial. The file `utils.py` is referenced from the training script to load

the dataset correctly. Copy this script into the script folder, so that it can be accessed along with the training script on the remote resource.

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

## Create an estimator

An estimator object is used to submit the run. Create your estimator by running the following code to define these items:

- The name of the estimator object, `est`.
- 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`.
- Parameters required from the training script.
- Python packages needed for training.

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 datastore, `ds.path('mnist').as_mount()`:

```
from azureml.train.estimator import Estimator  
  
script_params = {  
    '--data-folder': ds.path('mnist').as_mount(),  
    '--regularization': 0.8  
}  
  
est = Estimator(source_directory=script_folder,  
                 script_params=script_params,  
                 compute_target=compute_target,  
                 entry_script='train.py',  
                 conda_packages=['scikit-learn'])
```

## Submit the job to the cluster

Run the experiment by submitting the estimator object:

```
run = exp.submit(config=est)  
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 estimator. 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()
```

This still snapshot is the widget shown 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 you created can be used as prerequisites to other Azure Machine Learning service tutorials and how-to articles.

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.

NAME	TYPE	LOCATION
newacct	Azure Cosmos DB account	South Central US

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 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 service 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

2/1/2019 • 8 minutes to read

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 in [Azure Container Instances](#). 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 service for the following tasks:

- Set up your testing environment.
- Retrieve the model from your workspace.
- Test the model locally.
- 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 1.0.8.

## Prerequisites

Skip to [Set the development environment](#) to read through the notebook steps.

To run the notebook, first complete the model training in [Tutorial \(part 1\): Train an image classification model with Azure Machine Learning service](#). Then run the **tutorials/img-classification-part2-deploy.ipynb** notebook using the same notebook server.

## 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
import matplotlib.pyplot as plt

import azureml
from azureml.core import Workspace, Run

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

## Retrieve the model

You registered a model in your workspace in the previous tutorial. Now load this workspace and download the model to your local directory:

```
from azureml.core import Workspace
from azureml.core.model import Model
import os
ws = Workspace.from_config()
model=Model(ws, 'sklearn_mnist')

model.download(target_dir=os.getcwd(), exist_ok=True)

# verify the downloaded model file
file_path = os.path.join(os.getcwd(), "sklearn_mnist_model.pkl")

os.stat(file_path)
```

## Test the model locally

Before you deploy, make sure your model is working locally:

- Load test data.
- Predict test data.
- Examine the confusion matrix.

### Load test data

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

```
from utils import load_data
import os

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(os.path.join(data_folder, 'test-images.gz'), False) / 255.0
y_test = load_data(os.path.join(data_folder, 'test-labels.gz'), True).reshape(-1)
```

### Predict test data

To get predictions, feed the test dataset to the model:

```
import pickle
from sklearn.externals import joblib

clf = joblib.load( os.path.join(os.getcwd(), 'sklearn_mnist_model.pkl'))
y_hat = clf.predict(X_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:

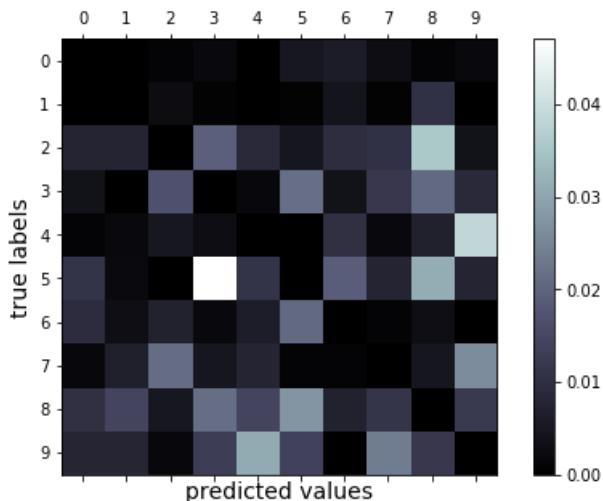
```
[[ 960     0     1     2     1     5     6     3     1     1]
 [   0 1112     3     1     0     1     5     1    12     0]
 [   9     8 920     20    10     4    10    11    37     3]
 [   4     0    17 921     2    21     4    12    20     9]
 [   1     2     5     3 915     0    10     2     6    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 shows the actual values, and the y-axis shows the predicted values. The color in each grid shows the error rate. The lighter the color, the higher the error rate is. For example, many 5's are misclassified 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()
```



## Deploy as a web service

After you tested the model and you're satisfied with the results, deploy the model as a web service hosted in Container Instances.

To build the correct environment for Container Instances, provide the following components:

- A scoring script to show how to use the model.

- An environment file to show what packages need to be installed.
- A configuration file to build the container instance.
- The model you trained previously.

### Create scoring script

Create the scoring script, called **score.py**. The web service call uses this script to show how to use the model.

Include these two required functions in 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
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression

from azureml.core.model import Model

def init():
    global model
    # retrieve the path to the model file using the model name
    model_path = Model.get_model_path('sklearn_mnist')
    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 environment file

Next create an environment file, called **myenv.yml**, that specifies all of the script's package dependencies. This file is used to make sure that all of those dependencies are installed in the Docker image. This model needs

`scikit-learn` and `azureml-sdk`:

```
from azureml.core.conda_dependencies import CondaDependencies

myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())
```

Review the content of the `myenv.yml` file:

```
with open("myenv.yml","r") as f:
    print(f.read())
```

### Create a configuration file

Create a deployment configuration file. Specify the number of CPUs and gigabytes of RAM needed for your Container Instances container. Although it depends on your model, the default of one core and 1 gigabyte of RAM is sufficient for many models. If you need more later, you have to re-create 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 Container Instances

The estimated time to finish deployment is **about seven to eight minutes**.

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

1. Build an image by using these files:
  - The scoring file, `score.py`.
  - The environment file, `myenv.yml`.
  - The model file.
2. Register the image under the workspace.
3. Send the image to the Container Instances container.
4. Start up a container in Container Instances by using the image.
5. Get the web service HTTP endpoint.

```
%%time
from azureml.core.webservice import Webservice
from azureml.core.image import ContainerImage

# configure the image
image_config = ContainerImage.image_configuration(execution_script="score.py",
                                                   runtime="python",
                                                   conda_file="myenv.yml")

service = Webservice.deploy_from_model(workspace=ws,
                                       name='sklearn-mnist-svc',
                                       deployment_config=aciconfig,
                                       models=[model],
                                       image_config=image_config)

service.wait_for_deployment(show_output=True)
```

Get the scoring 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:

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

## Test the deployed service

Earlier, you scored all the test data with the local version of the model. Now you can test the deployed model with a random sample of 30 images from the test data.

The following code goes through these steps:

1. Send the data as a JSON array to the web service hosted in Container Instances.

2. Use the SDK's `run` API to invoke the service. You can also make raw calls by using any HTTP tool such as `curl`.

3. 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.

Because 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))

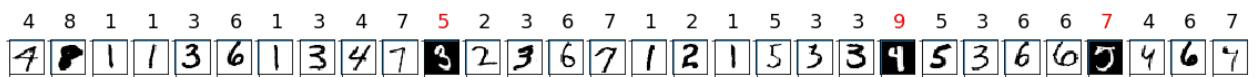
for s in sample_indices:
    plt.subplot(1, n, i + 1)
    plt.axhline('')
    plt.axvline('')

    # 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()
```

This result is from one random sample of test images:

4 8 1 1 3 6 1 3 4 7 5 2 3 6 7 1 2 1 5 3 3 9 5 3 6 6 7 4 6 7  


You can also send a 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 add 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:

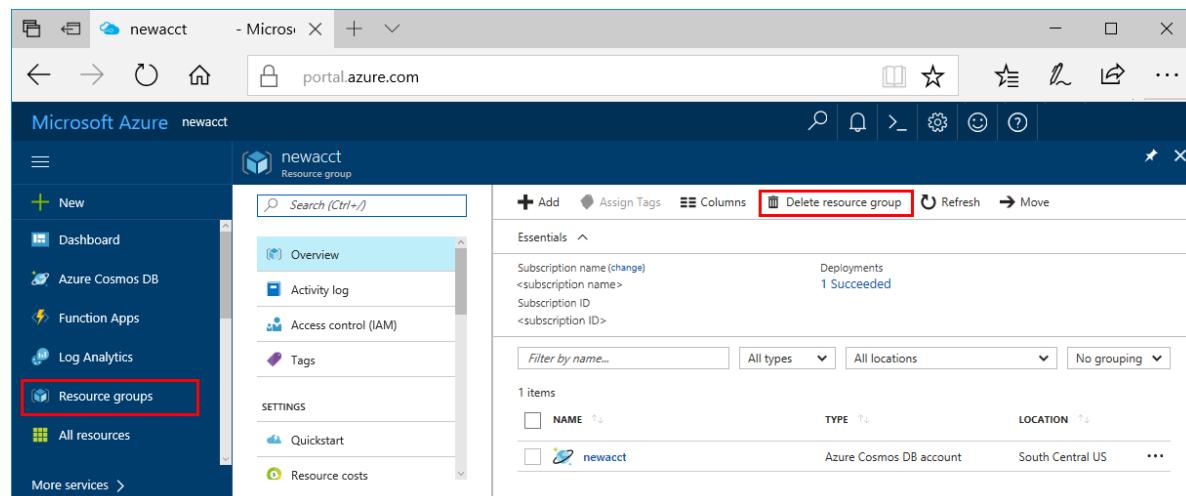
```
service.delete()
```

### IMPORTANT

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

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**.

## Next steps

- Learn about all of the [deployment options for Azure Machine Learning service](#).
- 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: Prepare data for regression modeling

2/27/2019 • 12 minutes to read

In this tutorial, you learn how to prepare data for regression modeling by using the [Azure Machine Learning Data Prep SDK for Python](#). You run various transformations to filter and combine two different NYC taxi data sets.

This tutorial is **part one of a two-part tutorial series**. After you complete the tutorial series, you can predict the cost of a taxi trip by training a model on data features. These features include the pickup day and time, the number of passengers, and the pickup location.

In this tutorial, you:

- Set up a Python environment and import packages.
- Load two datasets with different field names.
- Cleanse data to remove anomalies.
- Transform data by using intelligent transforms to create new features.
- Save your dataflow object to use in a regression model.

## Prerequisites

Skip to [Set up your development environment](#) to read through the notebook steps, or use the instructions below to get the notebook and run it on Azure Notebooks or your own notebook server. To run the notebook you will need:

- A Python 3.6 notebook server with the following installed:
  - The Azure Machine Learning Data Prep SDK for Python
- The tutorial notebook

Get all these prerequisites from either of the sections below.

- Use [Azure Notebooks](#)
- Use [your own notebook server](#)

### Use Azure Notebooks: Free Jupyter notebooks in the cloud

It's easy to get started with Azure Notebooks! The Azure Machine Learning Data Prep SDK is already installed and configured for you on [Azure Notebooks](#). The installation and future updates are automatically managed via Azure services.

After you complete the steps below, run the **tutorials/regression-part1-data-prep.ipynb** notebook in your **Getting Started** project.

1. Complete the [Azure Machine Learning portal quickstart](#) to create a workspace and launch Azure Notebooks.  
Feel free to skip the **Use the notebook** section if you wish.
2. If you've already completed the [quickstart](#), sign back into [Azure Notebooks](#) and open the **Getting Started** project.
3. Remember to start the project if its status is stopped.

## Getting Started

Welcome Azure Machine Learning service through Azure Notebooks

Cloned from [azureml/azureml-getting-started](#)

Status: **Stopped**

Run on Free Co...



### Use your own Jupyter notebook server

Use these steps to create a local Jupyter Notebook server on your computer. After you complete the steps, run the **tutorials/regression-part1-data-prep.ipynb** notebook.

1. Complete the [Azure Machine Learning Python quickstart](#) to create a Miniconda environment. Feel free to skip the **Create a workspace** section if you wish, but you will need it for [part 2](#) of this tutorial series.
2. Install the Data Prep SDK in your environment using `pip install azureml-dataprep`.
3. Clone [the GitHub repository](#).

```
git clone https://github.com/Azure/MachineLearningNotebooks.git
```

4. Start the notebook server from your cloned directory.

```
jupyter notebook
```

## Set up your development environment

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

- Install the SDK
- Import Python packages

### Install and import packages

Use the following to install necessary packages if you don't already have them.

```
pip install azureml-dataprep
```

Import the SDK.

```
import azureml.dataprep as dprep
```

## Load data

Download two different NYC taxi data sets into dataflow objects. The datasets have slightly different fields. The `auto_read_file()` method automatically recognizes the input file type.

```

from IPython.display import display
dataset_root = "https://dprepdata.blob.core.windows.net/demo"

green_path = "/".join([dataset_root, "green-small/*"])
yellow_path = "/".join([dataset_root, "yellow-small/*"])

green_df_raw = dprep.read_csv(path=green_path, header=dprep.PromoteHeadersMode.GROUPED)
# auto_read_file automatically identifies and parses the file type, which is useful when you don't know the
file type.
yellow_df_raw = dprep.auto_read_file(path=yellow_path)

display(green_df_raw.head(5))
display(yellow_df_raw.head(5))

```

A `Dataflow` object is similar to a dataframe, and represents a series of lazily-evaluated, immutable operations on data. Operations can be added by invoking the different transformation and filtering methods available. The result of adding an operation to a `Dataflow` is always a new `Dataflow` object.

## Cleanse data

Now you populate some variables with shortcut transforms to apply to all dataflows. The `drop_if_all_null` variable is used to delete records where all fields are null. The `useful_columns` variable holds an array of column descriptions that are kept in each dataflow.

```

all_columns = dprep.ColumnSelector(term=".*", use_regex=True)
drop_if_all_null = [all_columns, dprep.ColumnRelationship(dprep.ColumnRelationship.ALL)]
useful_columns = [
    "cost", "distance", "dropoff_datetime", "dropoff_latitude", "dropoff_longitude",
    "passengers", "pickup_datetime", "pickup_latitude", "pickup_longitude", "store_forward", "vendor"
]

```

You first work with the green taxi data to get it into a valid shape that can be combined with the yellow taxi data. Call the `replace_na()`, `drop_nulls()`, and `keep_columns()` functions by using the shortcut transform variables you created. Additionally, rename all the columns in the dataframe to match the names in the `useful_columns` variable.

```

green_df = (green_df_raw
    .replace_na(columns=all_columns)
    .drop_nulls(*drop_if_all_null)
    .rename_columns(column_pairs={
        "VendorID": "vendor",
        "lpep_pickup_datetime": "pickup_datetime",
        "Lpep_dropoff_datetime": "dropoff_datetime",
        "lpep_dropoff_datetime": "dropoff_datetime",
        "Store_and_fwd_flag": "store_forward",
        "store_and_fwd_flag": "store_forward",
        "Pickup_longitude": "pickup_longitude",
        "Pickup_latitude": "pickup_latitude",
        "Dropoff_longitude": "dropoff_longitude",
        "Dropoff_latitude": "dropoff_latitude",
        "Passenger_count": "passengers",
        "Fare_amount": "cost",
        "Trip_distance": "distance"
    })
    .keep_columns(columns=useful_columns)
green_df.head(5)

```

	VENDOR	PICKUP_DATETIME	DROP_OFF_DATETIME	STORE_FORWARD	PICKUP_LONGITUDE	PICKUP_LATITUDE	DROP_OFF_LONGITUDE	DROP_OFF_LATITUDE	PASSENGERS	DISTANCE	COST
0	2	2013-08-01 08:14:37	2013-08-01 09:09:06	N	0	0	0	0	1	.00	21.25
1	2	2013-08-01 09:13:00	2013-08-01 11:38:00	N	0	0	0	0	2	.00	74.5
2	2	2013-08-01 09:48:00	2013-08-01 09:49:00	N	0	0	0	0	1	.00	1
3	2	2013-08-01 10:38:35	2013-08-01 10:38:51	N	0	0	0	0	1	.00	3.25
4	2	2013-08-01 11:51:45	2013-08-01 12:03:52	N	0	0	0	0	1	.00	8.5

Run the same transformation steps on the yellow taxi data. These functions ensure that null data is removed from the data set, which will help increase machine learning model accuracy.

```
yellow_df = (yellow_df_raw
    .replace_na(columns=all_columns)
    .drop_nulls(*drop_if_all_null)
    .rename_columns(column_pairs={
        "vendor_name": "vendor",
        "VendorID": "vendor",
        "vendor_id": "vendor",
        "Trip_Pickup_DateTime": "pickup_datetime",
        "tpep_pickup_datetime": "pickup_datetime",
        "Trip_Dropoff_DateTime": "dropoff_datetime",
        "tpep_dropoff_datetime": "dropoff_datetime",
        "store_and_forward": "store_forward",
        "store_and_fwd_flag": "store_forward",
        "Start_Lon": "pickup_longitude",
        "Start_Lat": "pickup_latitude",
        "End_Lon": "dropoff_longitude",
        "End_Lat": "dropoff_latitude",
        "Passenger_Count": "passengers",
        "passenger_count": "passengers",
        "Fare_Amt": "cost",
        "fare_amount": "cost",
        "Trip_Distance": "distance",
        "trip_distance": "distance"
    })
    .keep_columns(columns=useful_columns))
yellow_df.head(5)
```

Call the `append_rows()` function on the green taxi data to append the yellow taxi data. A new combined dataframe

is created.

```
combined_df = green_df.append_rows([yellow_df])
```

## Convert types and filter

Examine the pickup and drop-off coordinates summary statistics to see how the data is distributed. First, define a `TypeConverter` object to change the latitude and longitude fields to decimal type. Next, call the `keep_columns()` function to restrict output to only the latitude and longitude fields, and then call the `get_profile()` function. These function calls create a condensed view of the dataflow to just show the lat/long fields, which makes it easier to evaluate missing or out-of-scope coordinates.

```
decimal_type = dprep.TypeConverter(data_type=dprep.FieldType.DECIMAL)
combined_df = combined_df.set_column_types(type_conversions={
    "pickup_longitude": decimal_type,
    "pickup_latitude": decimal_type,
    "dropoff_longitude": decimal_type,
    "dropoff_latitude": decimal_type
})
combined_df.keep_columns(columns=[
    "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude"
]).get_profile()
```

PICKUP_TYPE	MISSING_COUNT	NOT_MISSING_COUNT	PERCENT_IS_MISSING	ERROR_COUNT	EMPTY_TYPE_COUNT	0.1_QUANTILE	1_QUANTILE	5_QUANTILE	25_QUANTILE	50_QUANTILE	75_QUANTILE	95_QUANTILE	99_QUANTILE	99.9_QUANTILE	STANDARD_DEVIATION	MEAN			
PICKUP_TYPE	Field	0	4	7	0	7	0.	0.	0.	4	4	4	4	4	4	1	3		
	Field	.	0	7	.	7	0	0	0	0.	0.	0.	0.	0.	0.	0.	0.		
	Field	0	0	2	0	2	2.	0	0	6	6	7	7	8	8	0.	7.		
	Type	0	9	2	.	2	0	0	0	8	7	2	5	0	4	4	3		
	Type	0	1	.	0	0	0	0	0	2	5	1	6	3	9	0	5		
	Type	0	9	0	0	0	0	0	0	8	5	0	1	9	4	6	7		
	Type	0	0	1	0	0	0	0	0	8	4	7	5	0	0	2	4		
	Type	0	0	2	1	0	0	0	0	9	1	5	9	9	6	1	4		
	Type	0	0	2	1	0	0	0	0	9	1	5	9	9	6	1	4		
	Type	0	0	2	1	0	0	0	0	9	1	5	9	9	6	1	4		
DROPOUT_TYPE	Field	-	0	7	0	7	0.	0.	0.	-	-	-	-	-	0.	0.	1	-	
	Field	1	.	7	.	7	0	0	0	8	7	7	7	7	0	0	0	8.	
	Field	1	0	2	0	2	2.	0	0	7.	3.	3.	3.	3.	0	0	0	6.	
	Type	1	0	2	.	2	0	0	0	6	9	9	9	9	0	0	0	8.	
	Type	1	0	2	0	2	2.	0	0	9	8	8	5	2	0	0	0	6.	
	Type	1	0	2	0	2	2.	0	0	9	4	5	6	8	6	0	0	5.	
	Type	1	0	2	0	2	2.	0	0	6	7	7	2	9	2	0	0	2.	
	Type	1	0	2	0	2	2.	0	0	1	3	7	5	4	0	0	0	6.	
	Type	1	0	2	0	2	2.	0	0	1	4	7	0	8	8	0	0	7.	
	Type	1	0	2	0	2	2.	0	0	1	4	7	0	8	8	0	0	8.	
DROPOUT_TYPE	Field	0	4	7	0	7	0.	0.	0.	4	4	4	4	4	4	4	4	1	3
	Field	.	1	7	.	7	0	0	0	0.	0.	0.	0.	0.	0.	0.	0.	0.	7.
	Field	0	0	2	0	2	2.	0	0	6	6	7	7	7	8	8	9	2.	
	Type	0	0	2	.	2	0	0	0	6	5	1	5	8	5	7	3.	9.	
	Type	0	0	2	0	2	2.	0	0	2	4	7	6	4	2	9	7.	0.	
	Type	0	0	2	0	2	2.	0	0	7	8	8	5	6	4	2	7.	7.	
	Type	0	0	2	0	2	2.	0	0	6	5	2	3	3	8	3	8.	7.	
	Type	0	0	2	0	2	2.	0	0	3	1	1	4	8	7	9	1.	4.	
	Type	0	0	2	0	2	2.	0	0	3	1	1	4	8	7	9	1.	4.	
	Type	0	0	2	0	2	2.	0	0	3	1	1	4	8	7	9	1.	4.	

From the summary statistics output, you see there are missing coordinates and coordinates that aren't in New York City (this is determined from subjective analysis). Filter out coordinates for locations that are outside the city border. Chain the column filter commands within the `filter()` function and define the minimum and maximum

bounds for each field. Then call the `get_profile()` function again to verify the transformation.

```

latlong_filtered_df = (combined_df
    .drop_nulls(
        columns=["pickup_longitude", "pickup_latitude", "dropoff_longitude", "dropoff_latitude"],
        column_relationship=dprep.ColumnRelationship(dprep.ColumnRelationship.ANY)
    )
    .filter(dprep.f_and(
        dprep.col("pickup_longitude") <= -73.72,
        dprep.col("pickup_longitude") >= -74.09,
        dprep.col("pickup_latitude") <= 40.88,
        dprep.col("pickup_latitude") >= 40.53,
        dprep.col("dropoff_longitude") <= -73.72,
        dprep.col("dropoff_longitude") >= -74.09,
        dprep.col("dropoff_latitude") <= 40.88,
        dprep.col("dropoff_latitude") >= 40.53
    )))
latlong_filtered_df.keep_columns(columns=[
    "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude"
]).get_profile()

```

	Type	Min	Max	Count	Missing Count	Not Missing Count	Percentage	Error Count	Empirical Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Standard Deviation	Mean	
PICKUP_LOCATION	FILED	-74.	-70.	79.	0.0	70.	0.0	0.0	-7.0	-7.0	-7.0	-7.0	-7.0	-7.0	-7.0	-7.0	-7.0	-7.0	0.0	-7.0	
TIME	TYPE	0707	0700	0700	0500	0900	0900	0000	4.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	4.0	3.0
DE	DATE	0808	0606	0606	0404	0808	0808	0404	3.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	1.0	2.0
	TIME	0506	0801	0801	0801	0801	0801	0801	4.0	2.0	3.0	5.0	7.0	5.0	6.0	6.0	6.0	6.0	6.0	5.0	6.0

## Split and rename columns

Look at the data profile for the `store_forward` column. This field is a boolean flag that is `y` when the taxi did not have a connection to the server after the trip, and thus had to store the trip data in memory, and later forward it to

the server when connected.

```
latlong_filtered_df.keep_columns(columns='store_forward').get_profile()
```

					M	O	P													S	T	A	N	D	A	R	D	D	E	V	I	M	
					S	M	R	E	E	0.	1	1	5	2	5	7	9	9	9	9.	9.	9	9	9	9	9	9	9	9	9	9		
					I	SI	E	R	P	%	%	%	%	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q		
					N	N	N	O	T	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q		
					G	G	T	R	Y	U	U	U	U	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A		
					C	C	C	M	C	C	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A		
					T	M	M	O	O	IS	O	O	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N		
					Y	M	M	U	U	SI	U	U	TI	ME																			
					P	I	A	N	N	N	N	N	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	O	A		
					E	N	X	T	T	G	T	T	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	N	A	N		
ST	O	RE	_F	O	R	R	W	A	R	D	Fi	N	Y	7	9	6	0.	0.	0.														
											el	0	9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
											d	5	.	6	1																		
											T	9	0	0.	4																		
											RE	.	0	0	0																		
											F	0	0	2	5																		
											O	e.	.	0																			
											R	S	T	R	I	N	G																
											W	A	R	R	I	N	G																
											A	R	R	I	N	G																	
											R	D	I	N	G																		

Notice that the data profile output in the `store_forward` column shows that the data is inconsistent and there are missing or null values. Use the `replace()` and `fill_nulls()` functions to replace these values with the string "N":

```
replaced_stfor_vals_df = latlong_filtered_df.replace(columns="store_forward", find="0",
replace_with="N").fill_nulls("store_forward", "N")
```

Execute the `replace` function on the `distance` field. The function reformats distance values that are incorrectly labeled as `.00`, and fills any nulls with zeros. Convert the `distance` field to numerical format. These incorrect data points are likely anomalies in the data collection system on the taxi cabs.

```
replaced_distance_vals_df = replaced_stfor_vals_df.replace(columns="distance", find=".00",
replace_with=0).fill_nulls("distance", 0)
replaced_distance_vals_df = replaced_distance_vals_df.to_number(["distance"])
```

Split the pickup and dropoff datetime values into the respective date and time columns. Use the `split_column_by_example()` function to make the split. In this case, the optional `example` parameter of the `split_column_by_example()` function is omitted. Therefore, the function automatically determines where to split based on the data.

```
time_split_df = (replaced_distance_vals_df
    .split_column_by_example(source_column="pickup_datetime")
    .split_column_by_example(source_column="dropoff_datetime"))
time_split_df.head(5)
```

		VE ND OR	PIC KU P_D ATE TIM E	PIC KU P_D ATE TIM E_1	PIC KU P_D ATE TIM E_2	DR OP OFF _DA TET IME	DR OP OFF _DA TET IME	DR OP OFF _DA TET IME	STO RE_ FOR WA RD	PIC KU P_L ON GIT UD E	PIC KU P_L ATI TU DE	DR OP OFF _LO NGI TU DE	DR OP OFF _LA TIT UD E	PAS SEN GER S	DIS TA NCE	COS T
0	2	20 13- 08- 01 17: 22: 00	20 13- 08- 01 01 17: 22: 00	17: 22: 00	20 13- 08- 01 01 17: 22: 00	20 13- 08- 01 01 17: 25: 00	17: 25: 00	N	- 73. 93 77 67	40. 75 84 80 77	- 73. 78 43 27	40. 75 84 80 67	1	0.0	2.5	
1	2	20 13- 08- 01 17: 24: 00	20 13- 08- 01 01 17: 25: 00	17: 24: 00	20 13- 08- 01 01 17: 25: 00	20 13- 08- 01 01 17: 25: 00	17: 25: 00	N	- 73. 93 79 27	40. 75 84 79 27	- 73. 78 43 27	40. 75 78 43	1	0.0	2.5	
2	2	20 13- 08- 06 06: 51: 19	20 13- 08- 06 06: 51: 19	06: 51: 19	20 13- 08- 06 06: 51: 36	20 13- 08- 06 06: 51: 36	06: 51: 36	N	- 73. 93 77 21	40. 75 84 04 21	- 73. 93 93 77	40. 75 83 69	1	0.0	3.3	
3	2	20 13- 08- 06 13: 26: 34	20 13- 08- 06 13: 26: 34	13: 26: 34	20 13- 08- 06 13: 26: 57	20 13- 08- 06 13: 26: 57	13: 26: 57	N	- 73. 93 76 91	40. 75 84 19 90	- 73. 93 77 90	40. 75 83 58	1	0.0	3.3	
4	2	20 13- 08- 06 13: 27: 53	20 13- 08- 06 13: 27: 53	13: 27: 53	20 13- 08- 06 13: 28: 08	20 13- 08- 06 13: 28: 08	13: 28: 08	N	- 73. 93 78 05	40. 75 83 96 75	- 73. 93 96 77	40. 75 84 50	1	0.0	3.3	

Rename the columns generated by the `split_column_by_example()` function to use meaningful names.

```
renamed_col_df = (time_split_df
    .rename_columns(column_pairs={
        "pickup_datetime_1": "pickup_date",
        "pickup_datetime_2": "pickup_time",
        "dropoff_datetime_1": "dropoff_date",
        "dropoff_datetime_2": "dropoff_time"
    }))
renamed_col_df.head(5)
```

Call the `get_profile()` function to see the full summary statistics after all cleansing steps.

```
renamed_col_df.get_profile()
```

## Transform data

Split the pickup and dropoff date further into the day of the week, day of the month, and month values. To get the day of the week value, use the `derive_column_by_example()` function. The function takes an array parameter of example objects that define the input data, and the preferred output. The function automatically determines your preferred transformation. For the pickup and dropoff time columns, split the time into the hour, minute, and second by using the `split_column_by_example()` function with no example parameter.

After you generate the new features, use the `drop_columns()` function to delete the original fields as the newly generated features are preferred. Rename the rest of the fields to use meaningful descriptions.

Transforming the data in this way to create new time-based features will improve machine learning model accuracy. For example, generating a new feature for the weekday will help establish a relationship between the day of the week and the taxi fare price, which is often more expensive on certain days of the week due to high demand.

```
transformed_features_df = (renamed_col_df
    .derive_column_by_example(
        source_columns="pickup_date",
        new_column_name="pickup_weekday",
        example_data=[("2009-01-04", "Sunday"), ("2013-08-22", "Thursday")]
    )
    .derive_column_by_example(
        source_columns="dropoff_date",
        new_column_name="dropoff_weekday",
        example_data=[("2013-08-22", "Thursday"), ("2013-11-03", "Sunday")]
    )

    .split_column_by_example(source_column="pickup_time")
    .split_column_by_example(source_column="dropoff_time")
    # The following two calls to split_column_by_example reference the column names generated from the
    # previous two calls.
    .split_column_by_example(source_column="pickup_time_1")
    .split_column_by_example(source_column="dropoff_time_1")
    .drop_columns(columns=[
        "pickup_date", "pickup_time", "dropoff_date", "dropoff_time",
        "pickup_date_1", "dropoff_date_1", "pickup_time_1", "dropoff_time_1"
    ])
    .rename_columns(column_pairs={
        "pickup_date_2": "pickup_month",
        "pickup_date_3": "pickup_monthday",
        "pickup_time_1_1": "pickup_hour",
        "pickup_time_1_2": "pickup_minute",
        "pickup_time_2": "pickup_second",
        "dropoff_date_2": "dropoff_month",
        "dropoff_date_3": "dropoff_monthday",
        "dropoff_time_1_1": "dropoff_hour",
        "dropoff_time_1_2": "dropoff_minute",
        "dropoff_time_2": "dropoff_second"
    }))
    .drop_columns(["pickup_time_1", "dropoff_time_1"])

transformed_features_df.head(5)
```

	VENDOR	PICKUP_DATE	PICKUP_WEEKDAY	PICKUP_HOUR	PICKUP_MINUTE	PICKUP_SECOND	DROPOFF_DAY	DROPOFF_HOUR	DROPOFF_MINUTE	DROPOFF_SECOND	STORE_FORWARD	PICKUP_LOCATION	PICKUP_LAUNCHTIME	DROPOFF_LAUNCHTIME	DROPOFF_LAUNCHTIME	PASSENGERS	DISTANCE	COST	
0	2013-08-00	2013-08-00	Thursday	17:22	00:00	20:00	Thursday	17:22	00:00	N	-7	40.	-70.	40.	1	0.0	2.5		
	11:00	11:00					11:00				3.9	75	39	55					
	11:17	11:17					11:17				3.9	78	33	88					
	11:22	11:22					11:22				3.9	74	37	44					
	11:22	11:22					11:22				3.9	78	37	88					
	11:22	11:22					11:22				3.9	70	36	70					
	11:22	11:22					11:22				3.9	77	36	70					
	11:22	11:22					11:22				3.9	78	37	88					
	11:22	11:22					11:22				3.9	74	37	44					
	11:22	11:22					11:22				3.9	78	37	88					
1	2013-08-00	2013-08-00	Thursday	17:24	00:00	20:00	Thursday	17:25	00:00	N	-7	40.	-70.	40.	1	0.0	2.5		
	11:00	11:00					11:00				3.9	75	39	55					
	11:17	11:17					11:17				3.9	78	33	77					
	11:22	11:22					11:22				3.9	74	39	44					
	11:22	11:22					11:22				3.9	78	37	88					
	11:22	11:22					11:22				3.9	70	32	77					
	11:22	11:22					11:22				3.9	77	32	77					
	11:22	11:22					11:22				3.9	78	37	88					
	11:22	11:22					11:22				3.9	74	39	44					
	11:22	11:22					11:22				3.9	78	37	88					

VENDOR	PI C K U P_	PI C K U P_	PI C K U P_	PI C O FF	PASSENGERS	DISTANCE	COST													
	TIME	DAY	HOUR	MINUTE	COND	TIME	DAY	HOUR	COND	MINUTE	COND	FORWARD	STORE	UP_LONGITUDE	UP_LATITUDE	OFF_LONGITUDE	OFF_LATITUDE			
2	2011-08-00:19	Tuesday	06:19	51	19	2011-08-00:19	Tuesday	06:19	51	36	N	-73.0821	40.7759	-4.0735	0.7359	4.238	1	0.0	3.3	
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0821	40.7759	-4.0735	0.7359	4.238			
3	2011-08-00:19	Tuesday	13:08	26	34	2011-08-00:19	Tuesday	13:08	26	57	N	-73.0916	40.7759	-4.0735	0.7359	4.238	1	0.0	3.3	
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			
	2011-08-00:19					2011-08-00:19							-73.0916	40.7759	-4.0735	0.7359	4.238			

		PI C K U P D A E N D O M R	PI C K U P K P M I N O U T E	PI C K U P P S E C O N D	D R O P O FF -D W EE TE K H N U O M A Y	R O P O FF -W FF EE K H M O A Y	D R O P O FF -D W FF EE K H M O A R	D R O P O FF -D W FF EE K H M O A D	S T O R E F O R W T U D	PI C K U P L O N G I T T U D	PI C K U P L O N G I T T U D	D R O P O FF -L O N G I T T U D	P A S S E N G E R S	D R O P O FF -L O N G I T T U D	DI S T A N C E	C O S T
4	2	T 0 1 e 3 s - d 0 8 -	1 u 3 7 3 d -	2 0 0 1 3 0 0 1 3: 2 7: 5 3	5 3 3 1 8 8 0 6 1 3: 2 8: 0 8	2 0 1 es 3 d - a 0 y 8 -	0 8 8 D D	0 8 8 D D	N -	4 0. 7 3. 7 9 3 7 8 9 0 6 5	- 7 0. 7 3. 7 9 3 7 9 7 7 0 5	4 0. 7 3. 7 9 3 8 7 7 5 7 0	1 0. 0.	0. 3		
	0															
	6															
	1															
	3:															
	2															
	7:															
	5															
	3															

Notice that the data shows that the pickup and dropoff date and time components produced from the derived transformations are correct. Drop the `pickup_datetime` and `dropoff_datetime` columns because they're no longer needed (granular time features like hour, minute and second are more useful for model training).

```
processed_df = transformed_features_df.drop_columns(columns=["pickup_datetime", "dropoff_datetime"])
```

Use the type inference functionality to automatically check the data type of each field, and display the inference results.

```
type_infer = processed_df.builders.set_column_types()
type_infer.learn()
type_infer
```

The resulting output of `type_infer` is as follows.

```
Column types conversion candidates:  
'pickup_weekday': [FieldType.STRING],  
'pickup_hour': [FieldType.DECIMAL],  
'pickup_minute': [FieldType.DECIMAL],  
'pickup_second': [FieldType.DECIMAL],  
'dropoff_hour': [FieldType.DECIMAL],  
'dropoff_minute': [FieldType.DECIMAL],  
'dropoff_second': [FieldType.DECIMAL],  
'store_forward': [FieldType.STRING],  
'pickup_longitude': [FieldType.DECIMAL],  
'dropoff_longitude': [FieldType.DECIMAL],  
'passengers': [FieldType.DECIMAL],  
'distance': [FieldType.DECIMAL],  
'vendor': [FieldType.STRING],  
'dropoff_weekday': [FieldType.STRING],  
'pickup_latitude': [FieldType.DECIMAL],  
'dropoff_latitude': [FieldType.DECIMAL],  
'cost': [FieldType.DECIMAL]
```

The inference results look correct based on the data. Now apply the type conversions to the dataflow.

```
type_converted_df = type_infer.to_dataflow()  
type_converted_df.get_profile()
```

Before you package the dataflow, run two final filters on the data set. To eliminate incorrectly captured data points, filter the dataflow on records where both the `cost` and `distance` variable values are greater than zero. This step will significantly improve machine learning model accuracy, because data points with a zero cost or distance represent major outliers that throw off prediction accuracy.

```
final_df = type_converted_df.filter(dprep.col("distance") > 0)  
final_df = final_df.filter(dprep.col("cost") > 0)
```

You now have a fully transformed and prepared dataflow object to use in a machine learning model. The SDK includes object serialization functionality, which is used as shown in the following code.

```
import os  
file_path = os.path.join(os.getcwd(), "dfflows.dprep")  
  
package = dprep.Package([final_df])  
package.save(file_path)
```

## Clean up resources

To continue with part two of the tutorial, you need the **dfflows.dprep** file in the current directory.

If you don't plan to continue to part two, delete the **dfflows.dprep** file in your current directory. Delete this file whether you're running the execution locally or in [Azure Notebooks](#).

## Next steps

In part one of this tutorial, you:

- Set up your development environment.
- Loaded and cleansed data sets.
- Used smart transforms to predict your logic based on an example.
- Merged and packaged datasets for machine learning training.

You're ready to use the training data in part two of the tutorial:

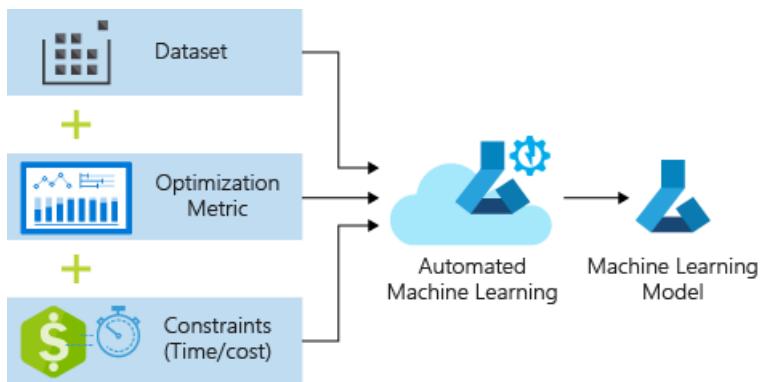
[Tutorial \(part two\): Train the regression model](#)

# Tutorial: Use automated machine learning to build your regression model

3/4/2019 • 15 minutes to read

This tutorial is **part two of a two-part tutorial series**. In the previous tutorial, you [prepared the NYC taxi data for regression modeling](#).

Now you're ready to start building your model with Azure Machine Learning service. In this part of the tutorial, you use the prepared data and automatically generate a regression model to predict taxi fare prices. By using the automated machine learning capabilities of the service, you define your machine learning goals and constraints. You launch the automated machine learning process. Then allow the algorithm selection and hyperparameter tuning to happen for you. The automated machine learning technique iterates over many combinations of algorithms and hyperparameters until it finds the best model based on your criterion.



In this tutorial, you learn the following tasks:

- Set up a Python environment and import the SDK packages.
- Configure an Azure Machine Learning service workspace.
- Autotrain a regression model.
- Run the model locally with custom parameters.
- Explore the results.

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 service](#) today.

## NOTE

Code in this article was tested with Azure Machine Learning SDK version 1.0.0.

## Prerequisites

Skip to [Set up your development environment](#) to read through the notebook steps, or use the instructions below to get the notebook and run it on Azure Notebooks or your own notebook server. To run the notebook you will need:

- [Run the data preparation tutorial](#).
- A Python 3.6 notebook server with the following installed:
  - The Azure Machine Learning SDK for Python with `automl` and `notebooks` extras

- `matplotlib`
- The tutorial notebook
- A machine learning workspace
- The configuration file for the workspace in the same directory as the notebook

Get all these prerequisites from either of the sections below.

- Use [Azure Notebooks](#)
- Use [your own notebook server](#)

### Use Azure Notebooks: Free Jupyter notebooks in the cloud

It's easy to get started with Azure Notebooks! The [Azure Machine Learning SDK for Python](#) is already installed and configured for you on [Azure Notebooks](#). The installation and future updates are automatically managed via Azure services.

After you complete the steps below, run the **tutorials/regression-part2-automated-ml.ipynb** notebook in your **Getting Started** project.

1. Complete the [Azure Machine Learning portal quickstart](#) to create a workspace and launch Azure Notebooks.  
Feel free to skip the **Use the notebook** section if you wish.
2. If you've already completed the [quickstart](#), sign back into [Azure Notebooks](#) and open the **Getting Started** project.
3. Remember to start the project if its status is stopped.

The screenshot shows the Azure Notebooks interface. At the top, there is a breadcrumb navigation: Home > My Projects > Getting Started. Below this, the title "Getting Started" is displayed in large font. Underneath the title, it says "Welcome Azure Machine Learning service through Azure Notebooks". It indicates that the project was "Cloned from [azureml/azureml-getting-started](#)". The current "Status" is shown as "Stopped". At the bottom of the card, there are two buttons: "Run on Free Co..." and a dropdown arrow icon.

### Use your own Jupyter notebook server

Use these steps to create a local Jupyter Notebook server on your computer. After you complete the steps, run the **tutorials/regression-part2-automated-ml.ipynb** notebook.

1. Complete the [Azure Machine Learning Python quickstart](#) to create a Miniconda environment and create a workspace.
2. Install the `automl` and `notebooks` extras in your environment using  
`pip install azureml-sdk[automl,notebooks]`.
3. Install `matplotlib` using `pip install matplotlib`.
4. Clone [the GitHub repository](#).

```
git clone https://github.com/Azure/MachineLearningNotebooks.git
```

5. Start the notebook server from your cloned directory.

```
jupyter notebook
```

# Set up your development environment

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

- Install the SDK
- Import Python packages
- Configure your workspace

## Install and import packages

If you are following the tutorial in your own Python environment, use the following to install necessary packages.

```
pip install azureml-sdk[automl,notebooks] matplotlib
```

Import the Python packages you need in this tutorial:

```
import azureml.core
import pandas as pd
from azureml.core.workspace import Workspace
import logging
import os
```

## 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 **aml\_config/config.json** and loads the details into an object named `ws`. `ws` is used throughout the rest of the code in this tutorial.

After you have a workspace object, specify a name for the experiment. Create and register a local directory with the workspace. The history of all runs is recorded under the specified experiment and in the [Azure portal](#).

```
ws = Workspace.from_config()
# choose a name for the run history container in the workspace
experiment_name = 'automated-ml-regression'
# project folder
project_folder = './automated-ml-regression'

output = {}
output['SDK version'] = azureml.core.VERSION
output['Subscription ID'] = ws.subscription_id
output['Workspace'] = ws.name
output['Resource Group'] = ws.resource_group
output['Location'] = ws.location
output['Project Directory'] = project_folder
pd.set_option('display.max_colwidth', -1)
pd.DataFrame(data=output, index=['']).T
```

# Explore data

Use the data flow object created in the previous tutorial. To summarize, part 1 of this tutorial cleaned the NYC Taxi data so it could be used in a machine learning model. Now, you use various features from the data set and allow an automated model to build relationships between the features and the price of a taxi trip. Open and run the data flow and review the results:

```
import azureml.dataprep as dprep

file_path = os.path.join(os.getcwd(), "dfflows.dprep")

package_saved = dprep.Package.open(file_path)
dflow_prepared = package_saved.dataflows[0]
dflow_prepared.get_profile()
```

Type	Min	Max	Count	Missing Count	Percentage Missing	Error Count	Empty Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
PICKUP-HOUR	Field Type	0.0	2.3	6.1	0.0	6.1	0.0	0.0	0.0	0.0	0.9	1.6	1.9	2.2	2.3	1.4	6.5	4.9	-0.4	-0.6	-0.5
PICKUP-MINUTE	Field Type	0.0	5.9	6.1	0.0	6.1	0.0	0.0	0.0	0.0	0.9	0.9	1.1	1.3	1.6	1.9	2.1	1.7	3.0	0.0	-1.1

		Data Summary																		
		Missing Data					Percents					Quantiles					Standard Deviation		Skewness	
Type	Min	Max	Count	Count	Count	Count	Error	Empty	Percent	Quantile	Quantile	Quantile	Quantile	Quantile	Quantile	Mean	Variance	SKEWNESS	KURTOSIS	
PICKUP-SECOND	Field Type	0.0	59.0	61.0	0.0	61.0	0.0	0.0	0.0	5.0	5.0	14.0	29.0	44.0	56.0	5.9	2.9	1.7	3.0	-0.1
DROPOFF-WEEKDAY	Field Type	Wednesday	61.0	0.0	61.0	0.0	0.0	0.0	0.0	6.0	6.0	14.0	28.0	43.0	55.0	4.4	2.4	2.5	2.5	-1.1

Type	Min	Max	Count	Missing Count	Percent Missing	Error Count	Empty Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
DROPOff-HOUR	FieldType	0.0	2.3	6.1	0.4	6.0	1.4	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.4	6.7	4.5	-0.0	-0.0
DROPOff-HOUR	Decimal	0.0	5.9	6.1	0.4	6.0	1.4	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.1	1.0	0.9	0.6	0.1
DROPOff-MINUTE	FieldType	0.0	5.9	6.1	0.4	6.0	1.4	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.9	1.7	3.0	0.0	-1.1
DROPOff-MINUTE	Decimal	0.0	5.9	6.1	0.4	6.0	1.4	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.9	1.0	0.8	0.6	0.1

Type	Min	Max	Count	Missing Count	Percentage Missing	Error Count	Empty Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis
DROP_OF_F_SECOND	FieldType.Decimal	0.0	5.9	6.14	0.48	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.9	1.7	3.0	-0.0	-1.1
STORE_FORWARD	FieldType.String	N	Y	6.14	0.0	6.14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.7	1.7	2.5	2.7	2.5

PICKUP-LONGITUDE	TYPE	MIN	MAX	COUNT	MISSING	MISSING	PERCENT	NOTMISSING	COUNT	ERRORTYPE	EMPTYCOUNT	QUANTILE	MEAN	STANDARDDEVIATION	VARIANCE	SKEWNESS	KURTOSIS										
PICKUP-LONGITUDE	Field Type	-7	-7	6140	0480	0140	0000	-740	730	730	730	730	730	730	730	730	730	730	730	730	730	730	730	730	0000	0000	
PICKUP-LATITUDE	Field Type	405755	409759	6140	0480	0140	0000	402	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	417	4000	4000	4000

	Type	Min	Max	Count	Missing Count	Percentage Missing	Error Count	Empty Count	0 % Quantile	1 % Quantile	5 % Quantile	25 % Quantile	50 % Quantile	75 % Quantile	95 % Quantile	99 % Quantile	Mean	Standard Deviation			Skewness	Kurtosis	
																		Mean	Variance	SKEWNESS			
DROPOFF_LONGITUDE	File	-74	-74	61	00	61	00	00	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	-7	0.0	0.0	-0.0	
DROPOFF_LONGITUDE	Type	0857	0200	79	00	80	00	00	00	00	00	00	00	00	00	00	00	00	00	0.6	0.0	2.2	0.2
DROPOFF_LATITUDE	File	40.	40.	61	00	61	00	00	40	40	40	40	40	40	40	40	40	40	40	0.0	0.0	-0.0	
DROPOFF_LATITUDE	Type	1583	0873	89	00	80	00	00	00	00	00	00	00	00	00	00	00	00	00	0.3	0.0	2.0	0.3

Type	Min	Max	Count	Missing Count	Percent Missing	Error Count	Empty Count	0.1% Quantile	1% Quantile	5% Quantile	25% Quantile	50% Quantile	75% Quantile	95% Quantile	99% Quantile	99.9% Quantile	Mean	Standard Deviation	Variance	Skewness	Kurtosis	
PASSENGERS	Field Type	1.0	6.4	6.8	0.0	6.14	0.0	0.0	1.0	1.1	1.1	1.1	1.5	1.5	1.6	1.6	2.1	2.39249	3.149	0.749	-1.144	2.347
DISTANCE	Field Type	0.01	3.21	6.14	0.0	6.14	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.2	0.4	1.0	1.421	1.547	1.294	1.556	2.988

You prepare the data for the experiment by adding columns to `dflow_x` to be features for our model creation.

You define `dflow_y` to be our prediction value, **cost**:

```
dflow_X = dfflow_prepared.keep_columns(['pickup_weekday','pickup_hour', 'distance','passengers', 'vendor'])  
dflow_y = dfflow_prepared.keep_columns('cost')
```

## Split the data into train and test sets

Now you split the data into training and test sets by using the `train_test_split` function in the `sklearn` library. This function segregates the data into the `x, features`, dataset for model training and the `y, values to predict`, dataset 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 always deterministic:

```
from sklearn.model_selection import train_test_split

x_df = df_X.to_pandas_dataframe()
y_df = df_y.to_pandas_dataframe()

x_train, x_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_state=223)
# flatten y_train to 1d array
y_train.values.flatten()
```

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 the necessary packages and data ready for

autotraining your 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 settings for autogeneration and tuning

Define the experiment parameter and model settings for autogeneration and tuning. View the full list of [settings](#).

Submitting the experiment with these default settings will take approximately 10-15 min, but if you want a shorter run time, reduce either `iterations` OR `iteration_timeout_minutes`.

PROPERTY	VALUE IN THIS TUTORIAL	DESCRIPTION
<code>iteration_timeout_minutes</code>	10	Time limit in minutes for each iteration. Reduce this value to decrease total runtime.
<code>iterations</code>	30	Number of iterations. In each iteration, a new machine learning model is trained with your data. This is the primary value that affects total run time.
<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>preprocess</code>	True	By using <code>True</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.

```
automl_settings = {
    "iteration_timeout_minutes" : 10,
    "iterations" : 30,
    "primary_metric" : 'spearman_correlation',
    "preprocess" : True,
    "verbosity" : logging.INFO,
    "n_cross_validations": 5
}
```

Use your defined training settings as a 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

# local compute
automated_ml_config = AutoMLConfig(task = 'regression',
                                     debug_log = 'automated_ml_errors.log',
                                     path = project_folder,
                                     X = x_train.values,
                                     y = y_train.values.flatten(),
                                     **automl_settings)
```

## Train the automatic regression model

Start the experiment to run locally. Pass the defined `automated_ml_config` object to the experiment. Set the output to `True` to view progress during the experiment:

```
from azureml.core.experiment import Experiment
experiment=Experiment(ws, experiment_name)
local_run = experiment.submit(automated_ml_config, show_output=True)
```

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.

```

Parent Run ID: AutoML_02778de3-3696-46e9-a71b-521c8fcfa0651
*****
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	MaxAbsScaler ExtremeRandomTrees	0:00:08	0.9447	0.9447
1	StandardScalerWrapper GradientBoosting	0:00:09	0.9536	0.9536
2	StandardScalerWrapper ExtremeRandomTrees	0:00:09	0.8580	0.9536
3	StandardScalerWrapper RandomForest	0:00:08	0.9147	0.9536
4	StandardScalerWrapper ExtremeRandomTrees	0:00:45	0.9398	0.9536
5	MaxAbsScaler LightGBM	0:00:08	0.9562	0.9562
6	StandardScalerWrapper ExtremeRandomTrees	0:00:27	0.8282	0.9562
7	StandardScalerWrapper LightGBM	0:00:07	0.9421	0.9562
8	MaxAbsScaler DecisionTree	0:00:08	0.9526	0.9562
9	MaxAbsScaler RandomForest	0:00:09	0.9355	0.9562
10	MaxAbsScaler SGD	0:00:09	0.9602	0.9602
11	MaxAbsScaler LightGBM	0:00:09	0.9553	0.9602
12	MaxAbsScaler DecisionTree	0:00:07	0.9484	0.9602
13	MaxAbsScaler LightGBM	0:00:08	0.9540	0.9602
14	MaxAbsScaler RandomForest	0:00:10	0.9365	0.9602
15	MaxAbsScaler SGD	0:00:09	0.9602	0.9602
16	StandardScalerWrapper ExtremeRandomTrees	0:00:49	0.9171	0.9602
17	SparseNormalizer LightGBM	0:00:08	0.9191	0.9602
18	MaxAbsScaler DecisionTree	0:00:08	0.9402	0.9602
19	StandardScalerWrapper ElasticNet	0:00:08	0.9603	0.9603
20	MaxAbsScaler DecisionTree	0:00:08	0.9513	0.9603
21	MaxAbsScaler SGD	0:00:08	0.9603	0.9603
22	MaxAbsScaler SGD	0:00:10	0.9602	0.9603
23	StandardScalerWrapper ElasticNet	0:00:09	0.9603	0.9603
24	StandardScalerWrapper ElasticNet	0:00:09	0.9603	0.9603
25	MaxAbsScaler SGD	0:00:09	0.9603	0.9603
26	TruncatedSVDWrapper ElasticNet	0:00:09	0.9602	0.9603
27	MaxAbsScaler SGD	0:00:12	0.9413	0.9603
28	StandardScalerWrapper ElasticNet	0:00:07	0.9603	0.9603
29	Ensemble	0:00:38	0.9622	0.9622

## Explore the results

Explore the results of automatic training with a Jupyter widget or by examining the experiment history.

### Option 1: Add a Jupyter widget to see results

If you use a Jupyter notebook, use this Jupyter notebook widget to see a graph and a table of all results:

```

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

```

AutoML\_797ff8a7-a369-4986-8180-e9bbcd938259:

Status: Completed



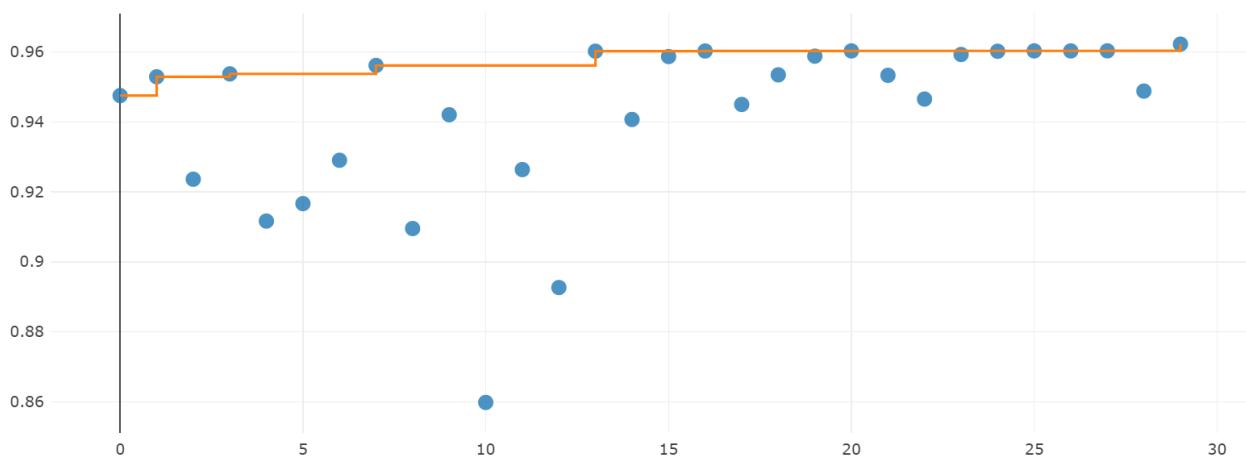
Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started	Ran
29	Ensemble	0.96225654	0.96225654	Completed	0:01:10	Dec 6, 2018 6:12 PM	26
27	MaxAbsScaler, SGD	0.96033526	0.96033526	Completed	0:00:15	Dec 6, 2018 6:11 PM	25
25	StandardScalerWrapper, ElasticNet	0.96031892	0.96031892	Completed	0:00:11	Dec 6, 2018 6:11 PM	24
20	MaxAbsScaler, SGD	0.96031661	0.96031661	Completed	0:00:20	Dec 6, 2018 6:09 PM	23
26	StandardScalerWrapper, ElasticNet	0.96031391	0.96031892	Completed	0:00:12	Dec 6, 2018 6:11 PM	22

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](#)

## Option 2: Get and examine all run iterations in Python

You can also retrieve the history of each experiment and explore the individual metrics for each iteration run. By examining RMSE (root\_mean\_squared\_error) for each individual model run, you see that most iterations are predicting the taxi fair cost within a reasonable margin (\$3-4).

```
children = list(local_run.get_children())
metricslist = []
for run in children:
    properties = run.get_properties()
    metrics = {k: v for k, v in run.get_metrics().items() if isinstance(v, float)}
    metricslist[int(properties['iteration'])] = metrics

rundata = pd.DataFrame(metricslist).sort_index(1)
rundata
```

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
EXPLAI NED_VA RIANCE	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	.	.	.	.	.	.	.	.	.	.	.	8	8	8	8	8	8	8	5	8	8
	8	8	3	7	6	8	1	5	8	7	.	5	8	8	8	8	8	8	8	8	8
	1	8	9	7	6	7	1	8	5	9	.	0	3	3	0	1	3	1	5	3	6
	1	0	8	6	3	5	5	6	1	3	.	0	6	7	7	5	7	8	3	1	8
	0	5	5	0	8	9	6	9	9	9	.	0	6	0	9	6	0	2	7	2	1
	3	5	8	4	6	1	3	0	1	6	.	2	0	0	7	6	0	2	7	2	1
	7	3	2	0	9	1	2	5	1	4	.	3	3	4	7	4	8	6	7	3	7
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	2	1	5	2	2	1	6	4	1	2	...	1	1	1	1	1	1	1	4	1	1
MEAN_AB SOLUTE_E RROR	.	.	.	.	.	.	.	.	.	.	.	7	4	4	5	4	5	0	5	4	.
	1	5	4	6	9	5	3	4	7	2	.	9	1	1	7	5	1	5	6	0	3
	8	0	8	2	7	5	8	1	4	9	.	7	5	8	8	9	3	1	9	5	0
	9	0	0	6	3	0	3	4	3	4	.	7	5	8	9	3	1	9	5	0	0
	4	4	5	3	0	1	8	2	3	6	.	4	8	1	6	4	0	6	1	7	9
	4	1	3	1	2	9	6	4	2	0	.	0	1	6	1	2	4	9	9	9	5
	4	2	1	6	6	9	8	1	8	1	.	2	5	7	7	7	2	8	6	5	7
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	0	4	1	1	0	4	3	0	1	...	0	0	0	1	1	0	1	2	1	0
	.	.	.	.	.	.	.	.	.	.	.	9	7	7	1	1	7	0	7	0	8
MEDIAN_AB SOLUTE_E RROR	4	8	5	7	5	8	2	6	9	3	.	7	7	9	4	1	8	9	0	0	5
	3	5	7	6	9	6	6	2	5	6	.	3	4	7	7	6	3	8	9	3	1
	8	0	9	5	4	9	6	7	4	1	.	6	8	2	2	4	9	4	0	7	7
	4	8	6	2	6	8	4	3	9	0	.	3	1	6	3	2	5	6	2	2	2
	1	9	6	1	0	8	5	5	9	1	.	3	4	9	4	8	4	2	7	8	4
	7	9	2	0	0	3	0	5	2	4	.	4	4	9	4	4	8	4	2	7	4
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	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	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	.	2	1	1	1	1	1	1	4	1	1
NORMALIZE _MEAN_AB SOLUTE_E RROR	2	1	6	2	3	1	7	5	1	2	.	0	6	6	7	7	6	7	6	7	6
	4	7	2	9	3	7	2	0	9	6	.	4	1	1	9	7	0	6	2	1	2
	9	0	3	8	8	6	6	2	8	1	.	4	0	3	5	4	7	5	9	3	7
	0	7	5	7	2	3	2	1	3	0	.	4	7	4	9	1	6	3	3	1	9
	8	0	0	8	3	6	6	9	3	5	.	8	7	4	9	1	6	3	3	1	9
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	0	4	1	1	0	4	3	0	1	...	0	0	0	1	1	0	1	2	1	0
	.	.	.	.	.	.	.	.	.	.	.	9	7	7	1	1	7	0	7	0	8
	4	8	5	7	5	8	2	6	9	3	.	7	5	8	9	3	1	8	9	0	5
	3	5	7	6	9	6	6	2	5	6	.	3	4	7	7	6	3	8	9	3	1

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
NORMA LIZE D_MEDI AN_ABSS OLUTE _ERROR	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	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	1	0	5	2	1	0	4	4	1	1	.	1	0	0	1	1	0	1	3	1	0
	6	9	2	0	8	9	8	1	0	5	.	1	8	9	3	2	8	2	0	1	9
	3	6	1	0	1	8	5	2	8	4	.	0	8	0	0	7	9	4	8	4	6
	6	8	0	8	4	9	3	6	6	8	.	7	1	7	5	0	1	9	1	1	9
	4	0	1	2	1	6	8	7	5	4	.	7	5	0	2	1	9	7	9	9	0
NORMA LIZE D_MEAN_SQ UARE _ERRR	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	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	4	3	8	5	6	3	0	7	4	4	.	4	3	3	3	3	3	3	7	3	3
	7	7	5	2	5	8	9	1	2	9	.	2	7	7	7	7	7	7	2	7	6
	9	8	5	2	8	6	4	1	2	9	.	5	6	5	6	5	5	4	0	2	7
	R	6	8	7	8	0	6	0	0	9	.	6	8	5	4	1	6	6	7	4	1
	O	8	2	2	2	9	4	1	4	4	.	5	5	7	3	3	0	5	7	9	6

	0	1	2	3	4	5	6	7	8	9	...	2	0	2	1	2	2	3	2	4	2	5	2	6	2	7	2	8	2	9
NORMALIZED_ROOT_MEAN_SQUARED_LOG_ERROR	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	0	0	0	
R2_SCORE	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	0	0	0	
ROOT_MEAN_SQUARED_ERROR	4	3	7	4	5	3	9	6	3	4	...	3	3	3	3	3	3	3	3	3	3	3	6	3	3	3	3	3	3	
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
	2	3	5	5	7	3	6	2	7	3	7	8	8	8	8	8	8	8	8	8	8	8	5	8	8	8	8	8	8	
	1	8	9	7	4	7	2	8	5	9	7	4	8	8	8	8	8	8	8	8	8	8	8	4	8	8	8	8	8	
	0	0	8	5	2	5	1	6	1	3	9	9	0	0	0	0	0	0	1	0	1	0	1	8	2	6	1	8	2	
	9	3	0	9	8	7	6	5	7	6	8	8	1	9	5	3	8	6	1	8	6	1	8	2	3	1	8	2	3	
	0	2	7	5	1	1	0	1	6	7	7	0	4	5	8	4	8	8	1	2	8	1	2	8	2	3	1	3	1	
	0	8	6	7	2	9	3	4	7	1	9	9	2	2	6	7	7	7	3	1	3	1	3	1	3	1	3	1	3	

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27	28	29
R O O T_ M E A N S Q U A R E D _L O G E R R O R	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	2	1	4	2	2	1	5	4	2	2	.	2	1	1	1	1	1	1	3	1	1
	4	9	8	8	7	9	4	0	0	4	.	0	8	8	9	9	8	9	4	8	8
	3	7	4	8	9	5	2	5	2	2	.	4	3	3	8	6	2	5	9	8	2
	1	7	2	3	3	1	2	5	6	7	.	4	6	5	4	0	8	0	9	0	4
	8	0	2	4	6	1	8	5	6	0	.	6	5	1	6	6	3	8	3	3	5
	4	2	7	9	7	6	1	9	6	2	.	4	8	4	8	7	6	7	5	1	5
S P E A R M A N C O R R E L A T I O N	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	9	9	8	9	9	9	8	9	9	9	.	9	9	9	9	9	9	9	9	9	9
	4	5	5	1	3	5	2	4	5	3	.	5	6	6	6	6	6	6	4	6	6
	4	3	7	4	9	6	8	2	2	5	.	1	0	0	0	0	0	0	1	0	2
	7	6	9	7	8	1	1	0	5	4	.	2	3	1	2	2	3	1	2	2	1
	4	1	6	0	4	5	8	6	8	7	.	8	3	9	7	8	2	6	5	9	5
	3	8	5	3	6	9	7	9	1	7	.	7	5	5	9	8	3	1	4	3	8
											.										
											.										
											.										
											.										
S P E A R M A N C O R R E L A T I O N - M A X	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	9	9	9	9	9	9	9	9	9	9	.	9	9	9	9	9	9	9	9	9	9
	4	5	5	5	5	5	5	5	5	5	.	6	6	6	6	6	6	6	6	6	6
	4	3	3	3	3	6	6	6	6	6	.	0	0	0	0	0	0	0	0	0	2
	7	6	6	6	6	1	1	1	1	1	.	3	3	3	3	3	3	3	3	3	1
	4	1	1	1	1	5	5	5	5	5	.	0	3	3	3	3	3	3	3	3	5
	3	8	8	8	8	9	9	9	9	9	.	3	5	5	5	5	5	5	5	5	8
											.										
											.										
											.										
											.										
											.										

12 rows × 30 columns

## Retrieve the best model

Select the best pipeline from our iterations. The `get_output` method on `automl_classifier` 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 dataset to predict taxi fares. The function `predict` uses the best model and predicts the values of y, **trip cost**, from the `x_test` dataset. Print the first 10 predicted cost values from `y_predict`:

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

Create a scatter plot to visualize the predicted cost values compared to the actual cost values. The following code uses the `distance` feature as the x-axis and trip `cost` as the y-axis. To compare the variance of predicted cost at each trip distance value, the first 100 predicted and actual cost values are created as separate series. Examining the plot shows that the distance/cost relationship is nearly linear, and the predicted cost values are in most cases very close to the actual cost values for the same trip distance.

```
import matplotlib.pyplot as plt

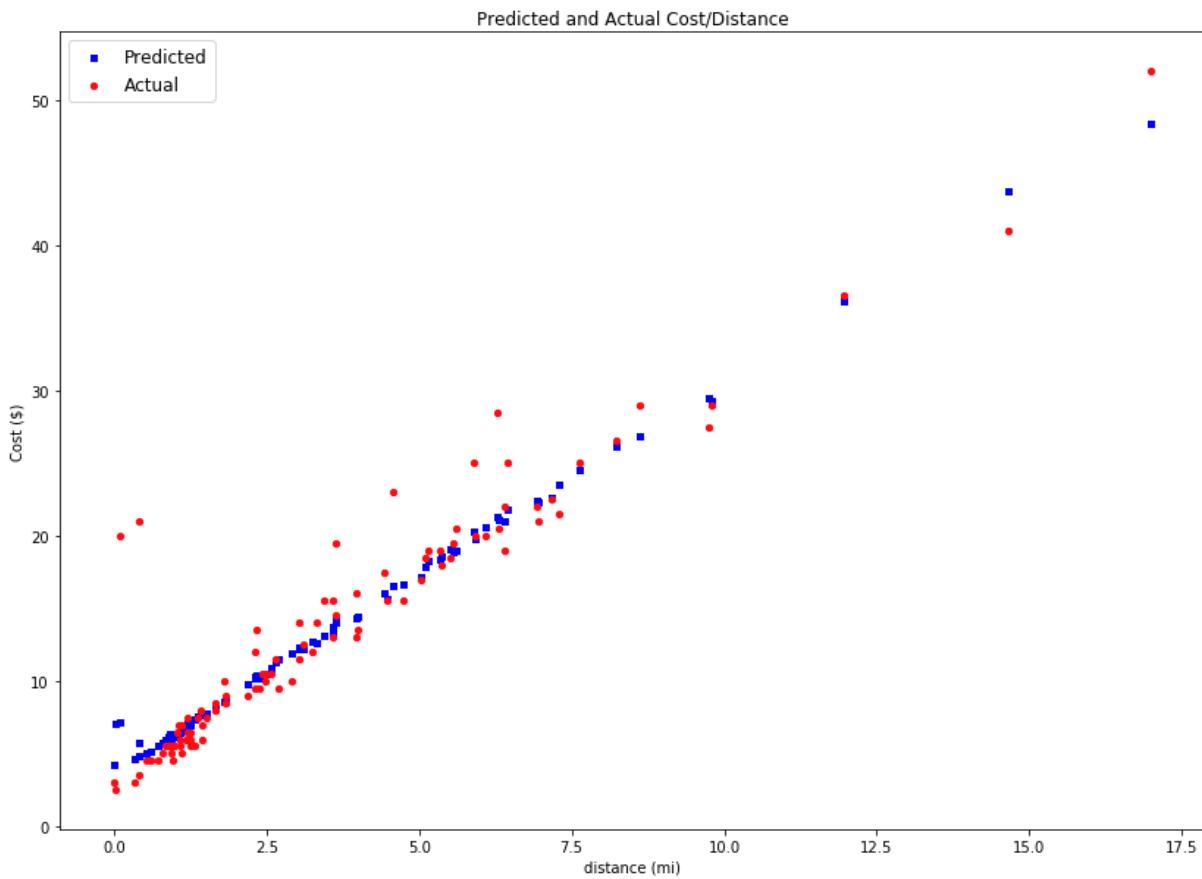
fig = plt.figure(figsize=(14, 10))
ax1 = fig.add_subplot(111)

distance_vals = [x[4] for x in x_test.values]
y_actual = y_test.values.flatten().tolist()

ax1.scatter(distance_vals[:100], y_predict[:100], s=18, c='b', marker="s", label='Predicted')
ax1.scatter(distance_vals[:100], y_actual[:100], s=18, c='r', marker="o", label='Actual')

ax1.set_xlabel('distance (mi)')
ax1.set_title('Predicted and Actual Cost/Distance')
ax1.set_ylabel('Cost ($)')

plt.legend(loc='upper left', prop={'size': 12})
plt.rcParams.update({'font.size': 14})
plt.show()
```



Calculate the `root mean squared error` of the results. Use the `y_test` dataframe. Convert it 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

rmse = sqrt(mean_squared_error(y_actual, y_predict))
rmse
```

```
3.2204936862688798
```

Run the following code to calculate mean absolute percent error (MAPE) by using the full `y_actual` and `y_predict` datasets. 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_actualls = 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_actualls = sum_actualls + actual_val

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

```

Model MAPE:  
0.10545153869569586

Model Accuracy:  
0.8945484613043041

From the final prediction accuracy metrics, you see that the model is fairly good at predicting taxi fares from the data set's features, typically within +/- \$3.00. 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.

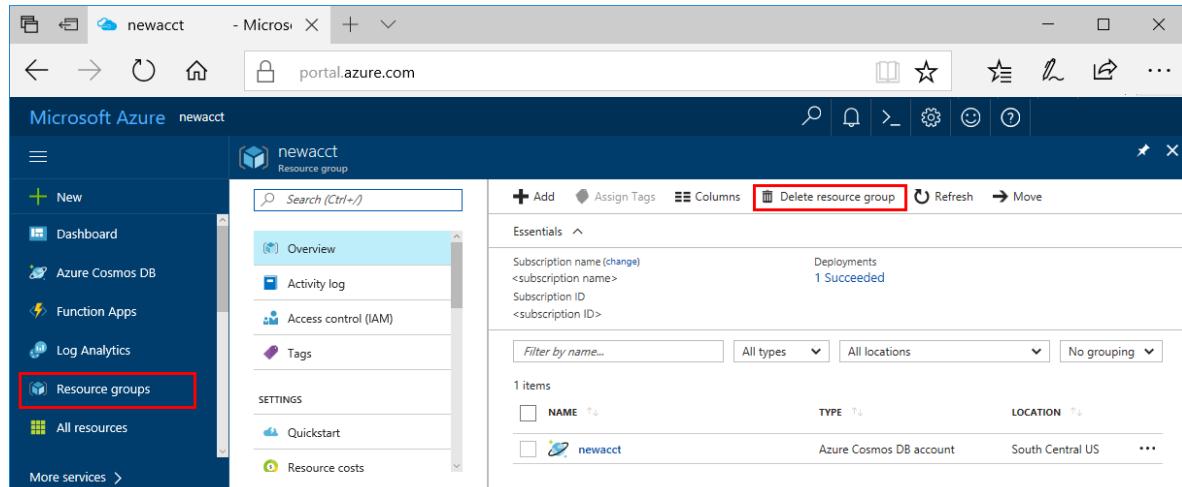
## Clean up resources

### IMPORTANT

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

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**.

## 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.

# Use Jupyter notebooks to explore Azure Machine Learning service

3/5/2019 • 3 minutes to read

For your convenience, we have developed a series of Jupyter Python notebooks you can use to explore the Azure Machine Learning service.

Learn how to use the service with the documentation on this site and use these notebooks to customize them to your situation.

Use one of the paths below to run a notebook server with these sample notebooks. Once the server is running, find tutorial notebooks in **tutorials** folder, or explore different features in **how-to-use-azureml** folder.

## Try Azure Notebooks: Free Jupyter notebooks in the cloud

It's easy to get started with Azure Notebooks! The [Azure Machine Learning SDK for Python](#) is already installed and configured for you on [Azure Notebooks](#). The installation and future updates are automatically managed via Azure services.

1. Complete the [Azure Machine Learning portal quickstart](#) to create a workspace and launch Azure Notebooks.  
Feel free to skip the **Use the notebook** section if you wish.
2. If you've already completed the [quickstart](#), sign back into [Azure Notebooks](#) and open the **Getting Started** project.
3. Remember to start the project if its status is stopped.

The screenshot shows the 'Getting Started' project page in Azure Notebooks. At the top, there is a breadcrumb navigation: Home > My Projects > Getting Started. The main title is 'Getting Started'. Below it, the text says 'Welcome Azure Machine Learning service through Azure Notebooks'. It also indicates that the project was 'Cloned from [azureml/azureml-getting-started](#)' and has a 'Status: Stopped'. At the bottom, there are two buttons: a red-bordered 'Run on Free Co...' button and a standard '▼' button.

## Use a Data Science Virtual Machine (DSVM)

The [Azure Machine Learning SDK for Python](#) and notebook server are already installed and configured for you on a DSVM.

After you [create a DSVM](#), use these steps on the DSVM to run the notebooks.

1. Complete the [Azure Machine Learning Python quickstart](#) to create a workspace. Feel free to skip the **Use the notebook** section if you wish.
2. Clone [the GitHub repository](#).

```
git clone https://github.com/Azure/MachineLearningNotebooks.git
```

3. Add a workspace configuration file using either of these methods:
  - Copy the **aml\_config\config.json** file you created using the prerequisite quickstart into the cloned directory.
  - 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
```

## Use your own Jupyter notebook server

Use these steps to create a local Jupyter Notebook server on your computer.

1. Complete the [Azure Machine Learning Python quickstart](#) to install the SDK and create a workspace. Feel free to skip the **Use the notebook** section if you wish.
2. Clone [the GitHub repository](#).

```
git clone https://github.com/Azure/MachineLearningNotebooks.git
```

3. Add a workspace configuration file using either of these methods:

- Copy the **aml\_config\config.json** file you created using the prerequisite quickstart into the cloned directory.
- Create a new workspace using code in the [configuration.ipynb](#).

4. Start the notebook server from your cloned directory.

```
jupyter notebook
```

The quickstart instructions will install the packages you need to run the quickstart and tutorial notebooks. Other sample notebooks may require installation of additional components. For more information about these components, see [Install the Azure Machine Learning SDK for Python](#).

## Automated machine learning setup

*These steps apply only to the notebooks in the **how-to-use-azureml/automated-machine-learning** folder.*

While you can use any of the above options, you can also install the environment and create a workspace at the same time with the following instructions.

1. Install [Mini-conda](#). Choose 3.7 or higher. Follow prompts to install.

### NOTE

You can use an existing conda as long as it is version 4.4.10 or later. Use `conda -v` to display the version. You can update a conda version with the command: `conda update conda`. There's no need to install mini-conda specifically.

2. Download the sample notebooks from [GitHub](#) as a zip and extract the contents to a local directory. The Automated machine learning notebooks are in the `how-to-use-azureml/automated-machine-learning` folder.
3. Set up a new Conda environment.
  - a. Open a Conda prompt on your local machine.

- b. Navigate to the files you extracted to your local machine.
- c. Open the **automl-machine-learning** folder.
- d. Execute `automl_setup.cmd` in the conda prompt for Windows, or the `.sh` file for your operating system. It can take about 10 minutes to execute.

The setup script:

- Creates a new conda environment
- Installs the necessary packages
- Configures the widget
- Starts a jupyter notebook

#### NOTE

The script takes the conda environment name as an optional parameter. The default conda environment name is `azure_automl`. The exact command depends on the operating system. This is useful if you are creating a new environment or upgrading to a new version. For example you can use 'automl\_setup.cmd azure\_automl\_sandbox' to create an environment name `azure_automl_sandbox`.

4. Once the script has completed, you will see a Jupyter notebook home page in your browser.
5. Navigate to the path where you saved the notebooks.
6. Open the automated-machine-learning folder, then open the **configuration.ipynb** notebook.
7. Execute the cells in the notebook to register Machine Learning Services Resource Provider and create a workspace.

You are now ready to open and run the notebooks saved on your local machine.

## Next steps

Explore the [GitHub notebooks repository for Azure Machine Learning service](#)

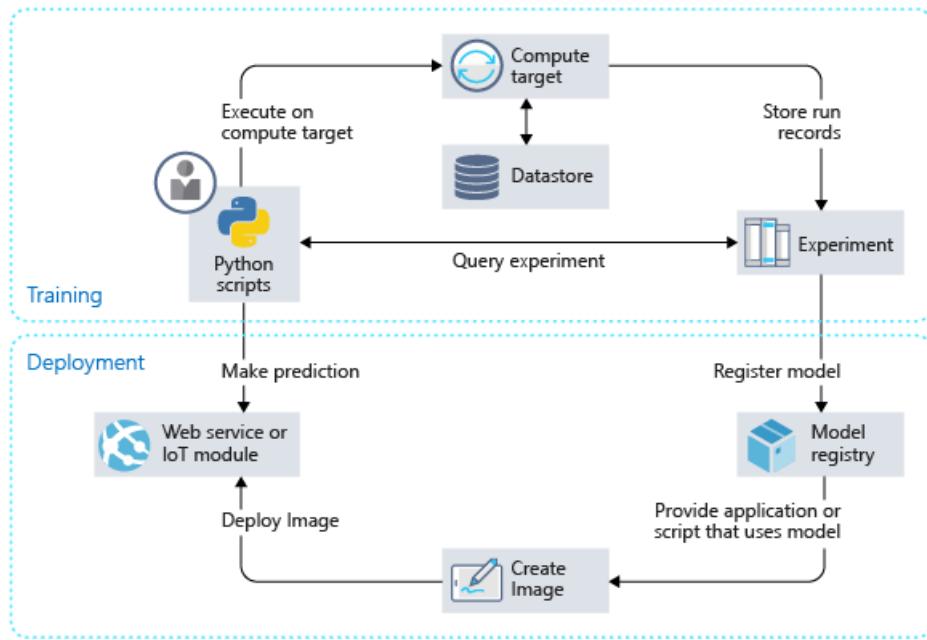
Try these tutorials:

- [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](#)

# How Azure Machine Learning service works: Architecture and concepts

2/12/2019 • 10 minutes to read

This article describes the architecture and concepts for Azure Machine Learning service. The major components of the service and the general workflow for using the service are shown in the following diagram:



The workflow generally follows this sequence:

1. Develop machine learning training scripts in **Python**.
2. Create and configure a **compute target**.
3. **Submit the scripts** to the configured compute target to run in that environment. During training, the scripts can read from or write to **datastore**. And the records of execution are saved as **runs** in the **workspace** and grouped under **experiments**.
4. **Query the experiment** for logged metrics from the current and past runs. If the metrics don't indicate a desired outcome, loop back to step 1 and iterate on your scripts.
5. After a satisfactory run is found, register the persisted model in the **model registry**.
6. Develop a scoring script.
7. **Create an image** and register it in the **image registry**.
8. **Deploy the image** as a **web service** in Azure.

## NOTE

Although this article defines terms and concepts used by Azure Machine Learning service, it does not define terms and concepts for the Azure platform. For more information about Azure platform terminology, see the [Microsoft Azure glossary](#).

## Workspace

The workspace is the top-level resource for Azure Machine Learning service. It provides a centralized place to work with all the artifacts you create when you use Azure Machine Learning service.

The workspace keeps a list of compute targets that you can use to train your model. It also keeps a history of the 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.

You register models with the workspace. You use a registered model and scoring scripts to create an image. You can then deploy the image 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 image to an Azure IoT Edge device as a module.

You can create multiple workspaces, and each workspace can be shared by multiple people. When you share a workspace, you can control access to it by assigning the following roles to users:

- Owner
- Contributor
- Reader

When you create a new workspace, it automatically creates several Azure resources that are used by the workspace:

- [Azure Container Registry](#): Registers docker containers that you use during training and when you deploy a model.
- [Azure storage account](#): Is used as the default datastore for the workspace.
- [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

In addition to creating new versions, you can also use existing Azure services.

A taxonomy of the workspace is illustrated in the following diagram:

## Experiment

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 you submit a run and specify an experiment name that doesn't exist, a new experiment with that newly specified name is automatically created.

For an example of using an experiment, see [Quickstart: Get started with Azure Machine Learning service](#).

## Model

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.

A model is produced by a run in Azure Machine Learning. You can also use a model that's trained outside of Azure Machine Learning. You can register a model in an Azure Machine Learning service workspace.

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

Cognitive Toolkit (formerly known as CNTK).

For an example of training a model, see [Quickstart: Create a Machine Learning service workspace](#).

## Model registry

The model registry keeps track of all the models in your Azure Machine Learning service 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.

You can't delete models that are being used by an image.

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

## Run configuration

A run configuration is a set of instructions that defines how a script should be run in a specified compute target. The configuration includes a wide set of behavior definitions, such as whether to use an existing Python environment or to use a Conda environment that's built from a specification.

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 [Select and use a compute target to train your model](#).

## Datastore

A datastore is a storage abstraction over an Azure storage account. The datastore can use either an Azure blob container or an Azure file share as the back-end storage. Each workspace has a default datastore, and you can register additional datastores.

Use the Python SDK API or the Azure Machine Learning CLI to store and retrieve files from the datastore.

## Compute target

A compute target is the compute resource that you use to run your training script or host your service deployment. The supported compute targets are:

COMPUTE TARGET	TRAINING	DEPLOYMENT
Your local computer	✓	
Azure Machine Learning compute	✓	
A Linux VM in Azure (such as the Data Science Virtual Machine)	✓	
Azure Databricks	✓	
Azure Data Lake Analytics	✓	
Apache Spark for HDInsight	✓	

COMPUTE TARGET	TRAINING	DEPLOYMENT
Azure Container Instances		✓
Azure Kubernetes Service		✓
Azure IoT Edge		✓
Project Brainwave (Field-programmable gate array)		✓

Compute targets are attached to a workspace. Compute targets other than the local machine are shared by users of the workspace.

### Managed and unmanaged compute targets

- **Managed:** Compute targets that are created and managed by Azure Machine Learning service. These compute targets are optimized for machine learning workloads. Azure Machine Learning compute is the only managed compute target as of December 4, 2018. Additional managed compute targets may be added in the future.

You can create machine learning compute instances directly through the workspace by using the Azure portal, the Azure Machine Learning SDK, or the Azure CLI. All other compute targets must be created outside the workspace and then attached to it.

- **Unmanaged:** Compute targets that are *not* managed by Azure Machine Learning service. You might need to create them outside Azure Machine Learning and then attach them to your workspace before use. Unmanaged compute targets can require additional steps for you to maintain or to improve performance for machine learning workloads.

For information about selecting a compute target for training, see [Select and use a compute target to train your model](#).

For information about selecting a compute target for deployment, see the [Deploy models with Azure Machine Learning service](#).

## Training script

To train a model, you specify the directory that contains the training script and associated files. You also specify an experiment name, which is used to store information that's gathered during training. During training, the entire directory is copied to the training environment (compute target), and the script that's specified by the run configuration is started. A snapshot of the directory is also stored under the experiment in the workspace.

For an example, see [Create a workspace with Python](#).

## Run

A run is a record that contains the following information:

- 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.

For an example of viewing runs that are produced by training a model, see [Quickstart: Get started with Azure Machine Learning service](#).

## 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.

## Activity

An activity represents a long running operation. The following operations are examples of activities:

- Creating or deleting a compute target
- Running a script on a compute target

Activities can provide notifications through the SDK or the web UI so that you can easily monitor the progress of these operations.

## Image

Images provide a way to reliably deploy a model, along with all components you need to use the model. An image contains the following items:

- A model.
- A scoring script or application. You use the script to pass input to the model and return the output of the model.
- The dependencies that are needed by the model or scoring script or application. For example, you might include a Conda environment file that lists Python package dependencies.

Azure Machine Learning can create two types of images:

- **FPGA image:** Used when you deploy to a field-programmable gate array in Azure.
- **Docker image:** Used when you deploy to compute targets other than FPGA. Examples are Azure Container Instances and Azure Kubernetes Service.

For an example of creating an image, see [Deploy an image classification model in Azure Container Instances](#).

### Image registry

The image registry keeps track of images that are created from your models. You can provide additional metadata tags when you create the image. Metadata tags are stored by the image registry, and you can query them to find your image.

## Deployment

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

### Web service

A deployed web service can use Azure Container Instances, Azure Kubernetes Service, or FPGAs. You create the service from an image that encapsulates your model, script, and associated files. The image has a load-balanced, HTTP endpoint that receives scoring requests that are sent to the web service.

Azure helps you monitor your web service deployment by collecting Application Insights telemetry or model telemetry, if you've chosen to enable this feature. The telemetry data is accessible only to you, and it's stored in

your Application Insights and storage account instances.

If you've enabled automatic scaling, Azure automatically scales your deployment.

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

### IoT module

A deployed IoT module 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.

## Pipeline

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 inferencing phases. Each phase can encompass multiple steps, each of which can run unattended in various compute targets.

For more information about machine learning pipelines with this service, see [Pipelines and Azure Machine Learning](#).

## Logging

When you develop your solution, use the Azure Machine Learning Python SDK in your Python script to log arbitrary metrics. After the run, query the metrics to determine whether the run has produced the model you want to deploy.

## Next steps

To get started with Azure Machine Learning service, see:

- [What is Azure Machine Learning service?](#)
- [Quickstart: Create a workspace with Python](#)
- [Tutorial: Train a model](#)
- [Create a workspace with a resource manager template](#)

# What is automated machine learning?

2/26/2019 • 3 minutes to read

Automated machine learning is the process of taking training data with a defined target feature, and iterating through combinations of algorithms and feature selections to automatically select the best model for your data based on the training scores. 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. Automated machine learning simplifies this process by generating models tuned from the goals and constraints you defined for your experiment, such as the time for the experiment to run or which models to blacklist.

## How it works

1. You configure the type of machine learning problem you are trying to solve. Categories of supervised learning are supported:

- Classification
- Regression
- Forecasting

While automated machine learning is generally available, **the forecasting feature is still in public preview.**

See the [list of models](#) Azure Machine Learning can try when training.

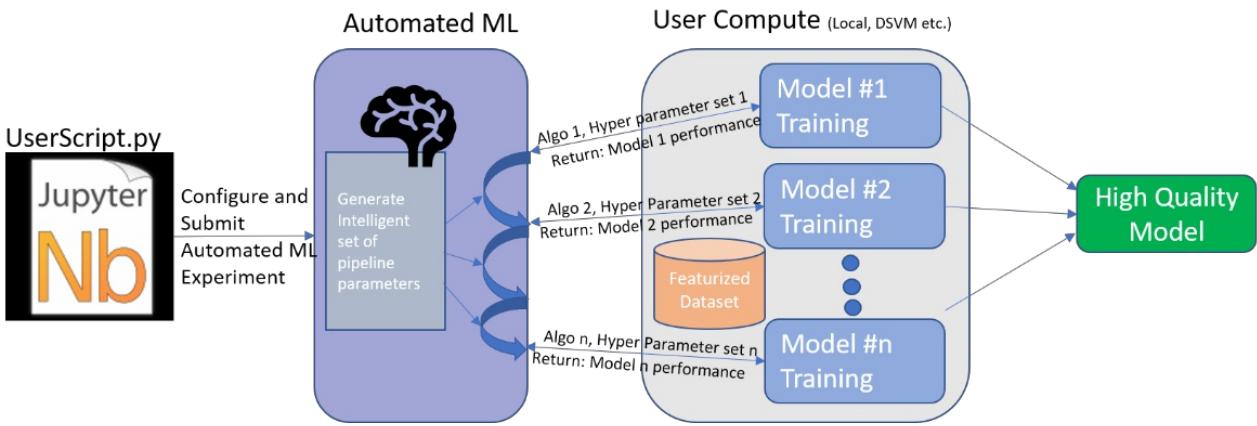
2. You specify the source and format for the training data. The data must be labeled, and can be stored on your development environment or in Azure Blob Storage. If the data is stored on your development environment, it must be in the same directory as your training scripts. This directory is copied to the compute target you select for training.

In your training script, the data can be read into Numpy arrays or a Pandas dataframe.

You can configure split options for selecting training and validation data, or you can specify separate training and validation data sets.

3. Configure the [compute target](#) that is used to train the model.
4. Configure the automated machine learning configuration. This controls the parameters used as Azure Machine Learning iterates over different models, hyperparameter settings, and what metrics to look at when determining the best model
5. Submit a training run.

During training, the Azure Machine Learning service creates a number of pipelines that try different algorithms and parameters. It will stop once it hits the iteration limit you provide, or when it reaches the target value for the metric you specify.



You can inspect the logged run information, which contains metrics gathered during the run. The training run also produces a Python serialized object (`.pk1` file) that contains the model and data preprocessing.

## Model explainability

A common pitfall of automated machine learning is an inability to see the end-to-end process. Azure Machine Learning allows you to view detailed information about the models to increase transparency into what's running on the back-end. Some models, like linear regression, are considered to be fairly straightforward and therefore easy to understand. But as we add more features and use more complicated machine learning models, understanding them gets more and more difficult. There are two key aspects to transparency in machine learning:

1. Awareness of the machine learning pipeline and all the steps involved including data preprocessing/featurization, and hyperparameter values.
2. Understanding the relationship between input variables (also known as "features") and model output. Knowing both the magnitude and direction of the impact of each feature on the predicted value helps better understand and explain the model. This is known as feature importance.

You can enable global feature importance on-demand post training for the pipeline of your choice, or enable it for all pipelines as part of automated machine learning training. In heavily regulated industries like healthcare and banking, this is critical to comply with regulations and best practices. Here are a few real-world scenarios to illustrate:

1. A manufacturing company using machine learning to predict future instrument failure, so they can proactively perform maintenance activity. Once you know an instrument is about to fail, what's the most likely cause going to be so preventive maintenance can be performed quickly?
2. A financial institution using machine learning to process loan or credit card applications. How do you know if the model is doing the right thing, and if a customer asks for more details on why their application was rejected, how will you respond to them?
3. An online retailer or an independent software provider using machine learning to predict customer churn. What are the key contributors to customer churn, and how can you prevent customers from churning?

This is a preview feature and we will continue to invest in providing richer information to help you better understand your machine learning models. Follow this [sample notebook](#) to experiment with model explanations in Azure Machine Learning.

## Next steps

See examples and learn how to build models using Automated Machine Learning:

- [Samples: Use Jupyter notebooks to explore Azure Machine Learning service](#)
- [Tutorial: Automatically train a classification model with Azure Automated Machine Learning](#)

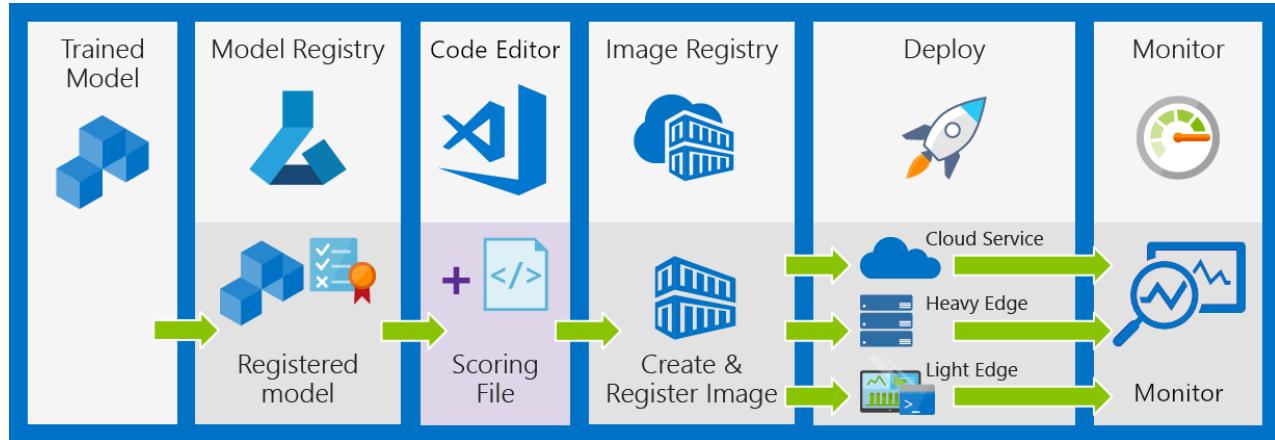
- Use automatic training on a remote resource
- Configure settings for automatic training

# Manage, deploy, and monitor models with Azure Machine Learning Service

3/5/2019 • 4 minutes to read

In this article, you can learn how to use Azure Machine Learning Service to deploy, manage, and monitor your models to continuously improve them. You can deploy the models you trained with Azure Machine Learning, on your local machine, or from other sources.

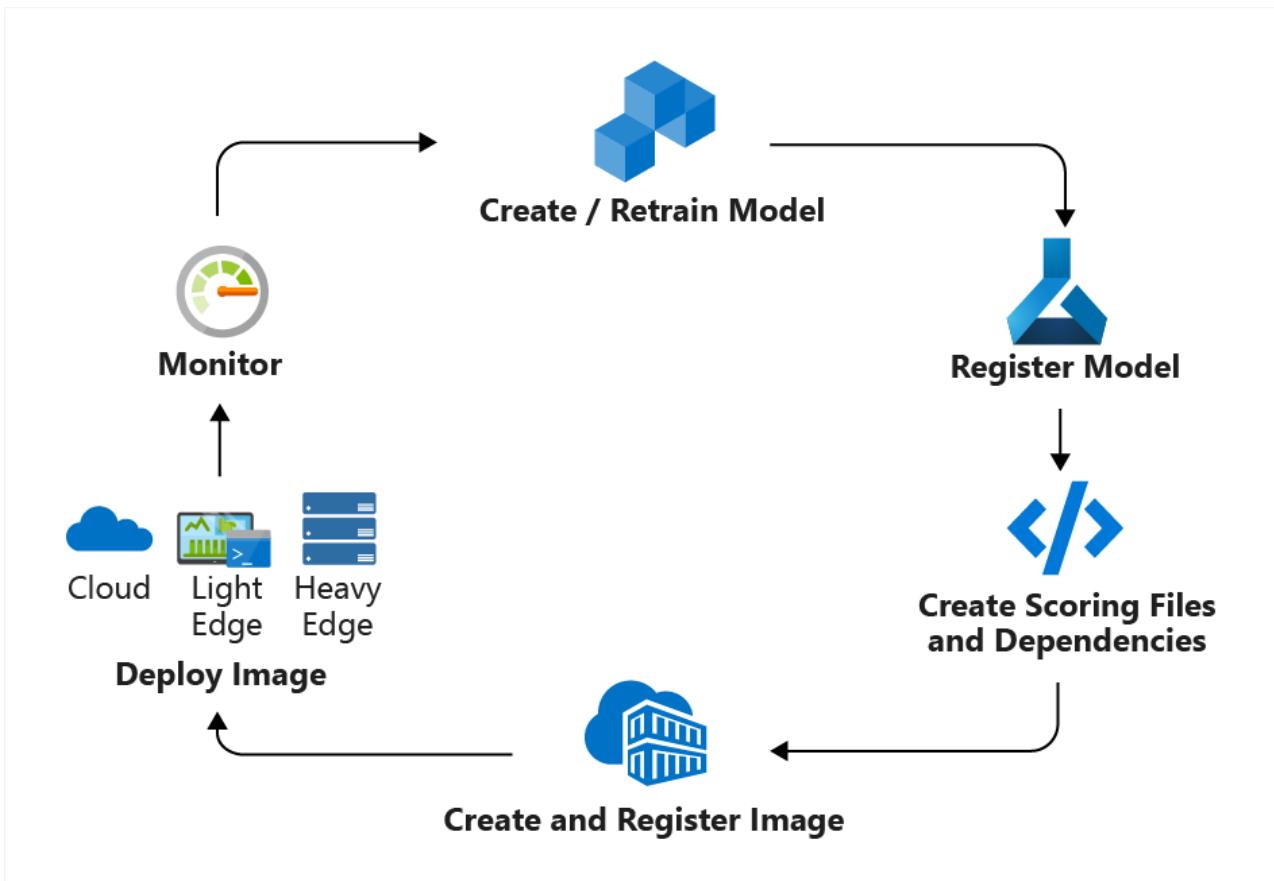
The following diagram illustrates the complete deployment workflow:



The deployment workflow includes the following steps:

1. **Register the model** in a registry hosted in your Azure Machine Learning Service workspace
2. **Register an image** that pairs a model with a scoring script and dependencies in a portable container
3. **Deploy** the image as a web service in the cloud or to edge devices
4. **Monitor and collect data**
5. **Update** a deployment to use a new image.

Each step can be performed independently or as part of a single deployment command. Additionally, you can integrate deployment into a **CI/CD workflow** as illustrated in this graphic.



## Step 1: Register model

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.

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. You can also provide additional metadata tags during registration that can be used when searching for models. The Azure Machine Learning service supports any model that can be loaded using Python 3.

You can't delete models that are being used by an image.

For more information, see the register model section of [Deploy models](#).

For an example of registering a model stored in pickle format, see [Tutorial: Train an image classification model](#).

For information on using ONNX models, see the [ONNX and Azure Machine Learning](#) document.

## Step 2: Register image

Images allow for reliable model deployment, along with all components needed to use the model. An image contains the following items:

- The model
- The scoring engine
- The scoring file or application
- Any dependencies needed to score the model

The image can also include SDK components for logging and monitoring. The SDK logs data can be used to fine-tune or retrain your model, including the input and output of the model.

Azure Machine Learning supports the most popular frameworks, but in general any framework that can be pip

installed can work.

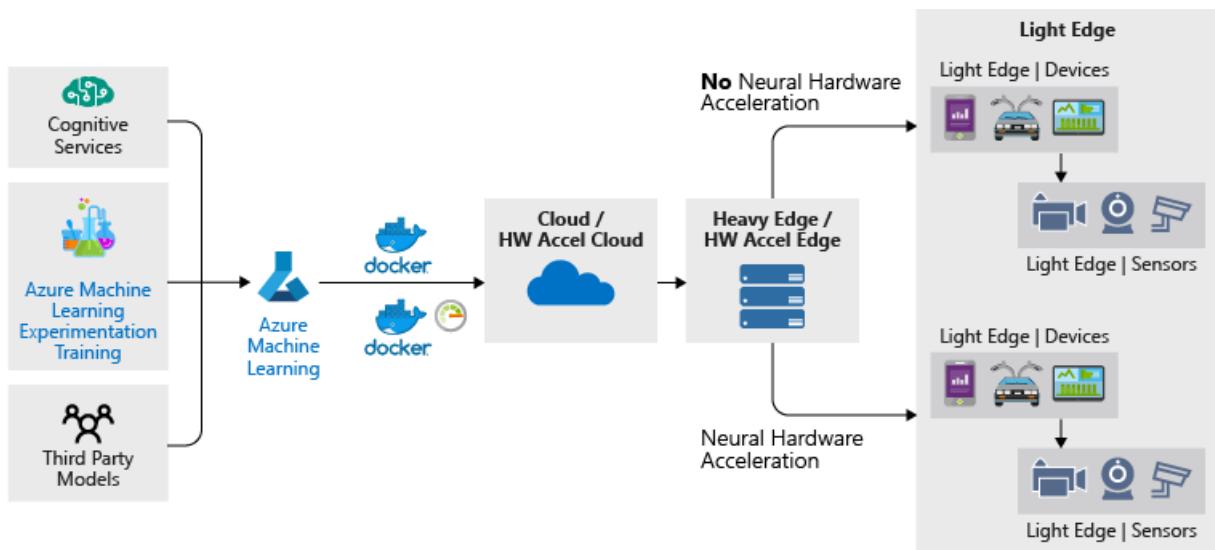
When your workspace was created, so were other several other Azure resources used by that workspace. All the objects used to create the image are stored in the Azure storage account in your workspace. You can provide additional metadata tags when creating the image. The metadata tags are also stored by the image registry, and can be queried to find your image.

For more information, see the configure and register image section of [Deploy models](#).

## Step 3: Deploy image

You can deploy registered images into the cloud or to edge devices. The deployment process creates all the resources needed to monitor, load-balance, and autoscale your model. Access to the deployed services can be secured with certificate-based authentication by providing the security assets during deployment. You can also upgrade an existing deployment to use a newer image.

Web service deployments are also searchable. For example, you can search for all deployments of a specific model or image.



You can deploy your images to the following deployment targets in the cloud:

- Azure Container Instance
- Azure Kubernetes Service
- Azure FPGA machines
- Azure IoT Edge devices

As your service is deployed, the inferencing request is automatically load-balanced and the cluster is scaled to satisfy any spikes on demand. [Telemetry about your service can be captured](#) into the Azure Application Insights service associated with your Workspace.

For more information, see the deploy section of [Deploy models](#).

## Step 4: Monitor models and collect data

An SDK for model logging and data capture is available so you can monitor input, output, and other relevant data from your model. The data is stored as a blob in the Azure Storage account for your workspace.

To use the SDK with your model, you import the SDK into your scoring script or application. You can then use the SDK to log data such as parameters, results, or input details.

If you decide to enable model data collection every time you deploy the image, the details needed to capture the

data, such as the credentials to your personal blob store, are provisioned automatically.

**IMPORTANT**

Microsoft does not see the data you collect from your model. The data is sent directly to the Azure storage account for your workspace.

For more information, see [How to enable model data collection](#).

## Step 5: Update the deployment

Updates to your model are not automatically registered. Similarly, registering a new image does not automatically update deployments that were created from a previous version of the image. Instead, you must manually register the model, register the image, and then update the model. For more information, see update section of [Deploy models](#).

## Next steps

Learn more about [how and where you can deploy models](#) with the Azure Machine Learning service. For an example of deployment, see [Tutorial: Deploy an image classification model in Azure Container Instances](#).

Learn how to create client applications and services that [Consume a model deployed as a web service](#).

# What are FPGAs and Project Brainwave?

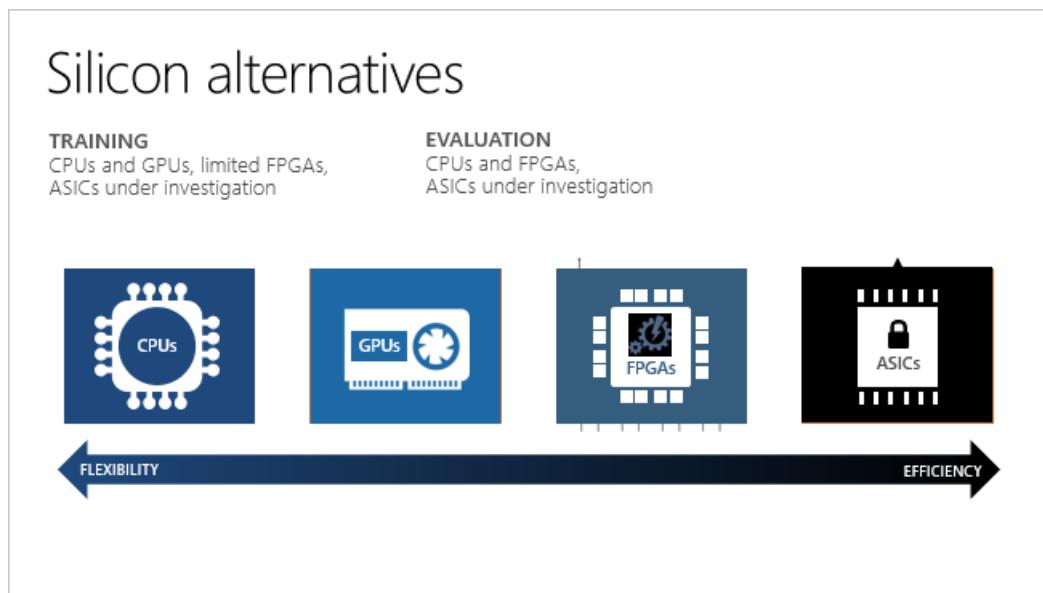
2/25/2019 • 3 minutes to read

This article provides an introduction to field-programmable gate arrays (FPGA), and how the Azure Machine Learning service provides real-time artificial intelligence (AI) when you deploy your model to an Azure FPGA.

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 vs. CPU, GPU, and ASIC

The following diagram and table show how FPGAs compare to other processors.



PROCESSOR		DESCRIPTION
Application-specific integrated circuits	ASICs	Custom circuits, such as Google's TensorFlow 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.
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.

# Project Brainwave on Azure

[Project Brainwave](#) is a hardware architecture from Microsoft. It's 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.

FPGAs make it possible to achieve low latency for real-time inferencing requests. Asynchronous requests (batching) aren't needed. Batching can cause latency, because more data needs to be processed. Project Brainwave implementations of neural processing units don't require batching; therefore the latency can be many times lower, compared to CPU and GPU processors.

## Reconfigurable power

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.

## What's supported on Azure

Microsoft Azure is the world's largest cloud investment in FPGAs. You can run Project Brainwave on Azure's scale infrastructure.

Today, Project Brainwave supports:

- Image classification and recognition scenarios
- TensorFlow deployment
- DNNs: ResNet 50, ResNet 152, VGG-16, SSD-VGG, and DenseNet-121
- Intel FPGA hardware

Using this FPGA-enabled hardware architecture, trained neural networks run quickly and with lower latency.

Project Brainwave can parallelize pre-trained deep neural networks (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 and applications

Project Brainwave is integrated with Azure Machine Learning. Microsoft uses FPGAs for DNN evaluation, Bing search ranking, and software defined networking (SDN) acceleration to reduce latency, while freeing CPUs for other tasks.

The following scenarios use FPGA on Project Brainwave architecture:

- [Automated optical inspection system](#)
- [Land cover mapping](#)

## Deploy to FPGAs on Azure

To create an image recognition service in Azure, you can use supported DNNs as a featurizer for deployment on Azure FPGAs:

1. Use 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 automatically compresses the definition and graphs into a ZIP file, and uploads the ZIP to Azure Blob storage. The DNN is already deployed on Project Brainwave to run on the FPGA.
2. Register the model by using the SDK with the ZIP file in Azure Blob storage.
3. Deploy the service with the registered model by using the SDK.

To get started deploying trained DNN models to FPGAs in the Azure cloud, see [Deploy a model as a web service](#)

on an FPGA.

## Next steps

Check out these videos and blogs:

- [Hyperscale hardware: ML at scale on top of Azure + FPGA : Build 2018 \(video\)](#)
- [Inside the Microsoft FPGA-based configurable cloud \(video\)](#)
- [Project Brainwave for real-time AI: project home page](#)

# Build machine learning pipelines with the Azure Machine Learning service

3/1/2019 • 3 minutes to read

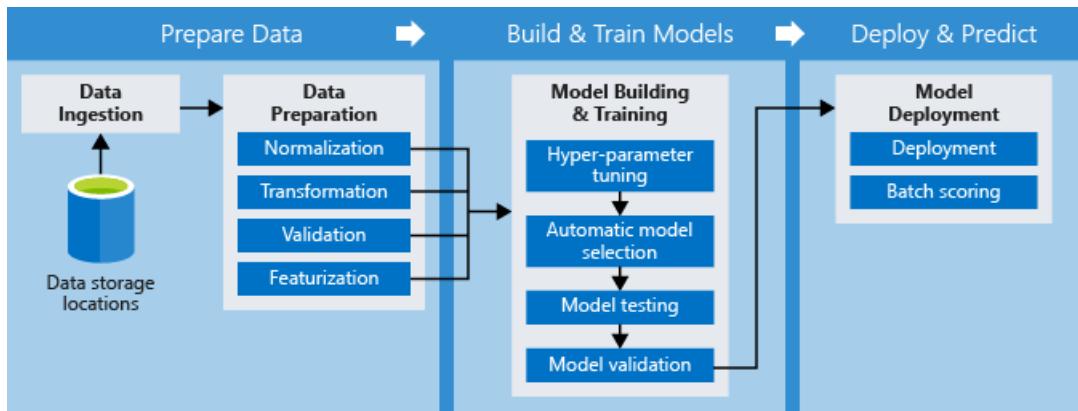
In this article, learn about the machine learning pipelines you can build with the Azure Machine Learning SDK for Python, and the advantages to using pipelines.

## What are machine learning pipelines?

Using machine learning (ML) pipelines, data scientists, data engineers, and IT professionals can collaborate on the steps involved in:

- Data preparation, such as normalizations and transformations
- Model training
- Model evaluation
- Deployment

The following diagram shows an example pipeline:



## 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:

Pipeline	What it does	Canonical pipe
Azure Machine Learning pipelines	Defines reusable machine learning workflows that can be used as a template for your machine learning scenarios.	Data -> model
Azure Data Factory pipelines	Groups data movement, transformation, and control activities needed to perform a task.	Data -> data
Azure pipelines	Continuous integration and delivery of your application to any platform/any cloud	Code -> app/service

# Why build pipelines with Azure Machine Learning?

You can use the [Azure Machine Learning SDK for Python](#) to create ML pipelines, as well as to submit and track individual pipeline runs.

With pipelines, you can optimize your workflow with simplicity, speed, portability, and reuse. When building pipelines with Azure Machine Learning, you can focus on your expertise, machine learning, rather than on infrastructure.

Using distinct steps makes it possible to rerun only the steps you need, as you tweak and test your workflow. A step is a computational unit in the pipeline. As shown in the preceding diagram, the task of preparing data can involve many steps. These steps include, but aren't limited to, normalization, transformation, validation, and featurization. Data sources and intermediate data are reused across the pipeline, which saves compute time and resources.

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, and skips what hasn't changed. The same paradigm applies to unchanged scripts used for the execution of the step.

With Azure Machine Learning, you can use various toolkits and frameworks, such as PyTorch or TensorFlow, for each step in your pipeline. Azure coordinates between the various [compute targets](#) you use, so that your intermediate data can be shared with the downstream compute targets easily.

You can [track the metrics for your pipeline experiments](#) directly in the Azure portal.

## Key advantages

The key advantages to building pipelines for your machine learning workflows are:

KEY ADVANTAGE	DESCRIPTION
<b>Unattended runs</b>	Schedule a few steps to run in parallel or in sequence in a reliable and unattended manner. Because data prep and modeling can last days or weeks, you can now focus on other tasks while your pipeline is running.
<b>Mixed and diverse compute</b>	Use multiple pipelines that are reliably coordinated across heterogeneous and scalable computes and storages. You can run individual pipeline steps on different compute targets, such as HDInsight, GPU Data Science VMs, and Databricks. This makes efficient use of available compute options.
<b>Reusability</b>	You can template pipelines for specific scenarios, such as retraining and batch scoring. Trigger them from external systems via simple REST calls.
<b>Tracking and versioning</b>	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.

## The Python SDK for pipelines

Use Python to create your ML pipelines. The Azure Machine Learning SDK offers imperative constructs for sequencing and parallelizing the steps in your pipelines when no data dependency is present. You can interact with it in Jupyter notebooks, or in another preferred integrated development environment.

Using declarative data dependencies, you can optimize your tasks. The SDK includes a framework of pre-built modules for common tasks, such as data transfer and model publishing. You can extend the framework to model your own conventions, by implementing custom steps that are reusable across pipelines. You can also manage compute targets and storage resources directly from the SDK.

You can save pipelines as templates, and deploy them to a REST endpoint so you can schedule batch-scoring or retraining jobs.

To see how to build your own, see the [Python SDK reference docs for pipelines](#) and the notebook in the next section.

## Example notebooks

The following notebooks demonstrate pipelines with Azure Machine Learning: [how-to-use-azureml/machine-learning-pipelines](#).

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

## Next steps

Learn how to [create your first pipeline](#).

# Enterprise security for the Azure Machine Learning service

2/28/2019 • 9 minutes to read

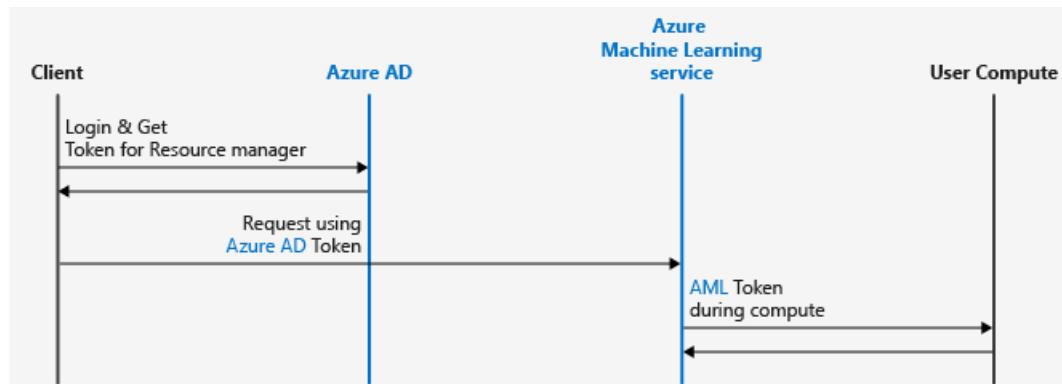
In this article, you will learn about security features available with the Azure Machine learning service.

When using a cloud service, it is a best practice to restrict access only to the users who need it. This starts by understanding the authentication and authorization model used by the service. You may also want to restrict network access, or securely join resources in your on-premises network with those in the cloud. Data encryption is also vital, both at rest and while the data moves between services. Finally, you need to be able to monitor the service and produce an audit log of all activity.

## Authentication

Multi Factor authentication is supported if Azure Active Directory (Azure AD) is configured for the same.

- Client logs into Azure AD and gets Azure Resource Manager token. Users and Service Principals are fully supported.
- Client presents token to Azure Resource Manager & all Azure Machine Learning services
- Azure Machine Learning service provides an Azure Machine Learning token to the user compute. For example, Machine Learning Compute. This Azure Machine Learning token is used by user compute to call back into Azure Machine Learning service (limits scope to workspace) after the run is complete.



### Authentication keys for Web service deployment

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](#)

## Authorization

You can create multiple workspaces, and each workspace can be shared by multiple people. When you share a workspace, you can control access to it by assigning the following roles to users:

- Owner
- Contributor
- Reader

The following table lists some of the major Azure Machine Learning service operations and the roles that can perform them:

AZURE MACHINE LEARNING SERVICE OPERATION	OWNER	CONTRIBUTOR	READER
Create Workspace	✓	✓	
Share Workspace	✓		
Create Compute	✓	✓	
Attach Compute	✓	✓	
Attach Datastores	✓	✓	
Run an Experiment	✓	✓	
View runs/metrics	✓	✓	✓
Register model	✓	✓	
Create image	✓	✓	
Deploy web service	✓	✓	
View models/images	✓	✓	✓
Call web service	✓	✓	✓

If the built-in roles are insufficient for your needs, you can also create custom roles. Note that the only custom roles we support are for operations on the workspace and Machine Learning Compute. The custom roles may have read, write, or delete permissions on the workspace and the compute resource in that workspace. You can make the role available at a specific workspace level, a specific resource group level, or a specific subscription level. For more information, see [Manage users and roles in an Azure Machine Learning workspace](#)

### Securing compute and data

Owners and contributors can use all compute targets and data stores that are attached to the workspace. Each workspace also has an associated system-assigned Managed Identity (with the same name as the workspace) with the following permissions on attached resources used in the workspace:

For more information on managed identities, see [Managed identities for Azure resources](#)

RESOURCE	PERMISSIONS
Workspace	Contributor
Storage Account	Storage Blob Data Contributor
Key Vault	Access to all Keys, Secrets, Certificates
Azure Container Registry	Contributor
Resource Group that contains the workspace	Contributor
Resource Group that contains the Key Vault (if different than the one containing the workspace)	Contributor

It is recommended that administrators do not revoke the access of the managed identity to the resources mentioned above. Access can be restored with the Resync Keys operation.

Azure Machine Learning service creates an additional application (name starts with aml-) with the contributor level access in your subscription for every workspace region. For ex. if you have a workspace in East US and another workspace in North Europe in the same subscription you will see 2 such applications. This is needed so that Azure Machine Learning service can help manage compute resources.

## Network security

The Azure Machine Learning service relies on other Azure services for compute resources. Compute resources (compute targets) are used to train and deploy models. These compute targets can be created inside a virtual network. For example, you can use the Microsoft Data Science Virtual Machine to train a model and then deploy the model to Azure Kubernetes Service (AKS).

For more information, see [How to run experiments and inferencing in a virtual network](#).

## Data encryption

### Encryption at rest

#### Azure Blob Storage

Azure Machine Learning service stores snapshots, outputs, and logs in the Azure Blob Storage account that is tied to the Azure Machine Learning service workspace and lives in user's subscription. All the data stored in Azure Blob Storage is encrypted at rest using Microsoft-Managed Keys.

For more information on how to bring your own keys for the data stored in Azure Blob Storage, see [Storage Service Encryption using customer-managed keys in Azure Key Vault](#).

Training data is typically also stored in Azure Blob storage so that it is accessible to training compute. This storage is not managed by Azure Machine Learning but mounted to compute as a remote file system.

#### Cosmos DB

Azure Machine Learning service stores metrics and metadata to the Cosmos DB that lives in a Microsoft subscription managed by Azure Machine Learning service. All the data stored in Cosmos DB is encrypted at rest using Microsoft Managed Keys.

#### Azure Container Registry (ACR)

All container images in your registry (ACR) are encrypted at rest. Azure automatically encrypts an image before storing it and decrypts it on-the-fly when Azure Machine Learning service pulls the image.

## Machine Learning Compute

The OS disk for each compute node is stored in Azure Storage is encrypted using Microsoft Managed Keys in Azure Machine Learning service storage accounts. This compute is ephemeral, and clusters are typically scaled down when there are no runs queued. The underlying virtual machine is de-provisioned and OS disk deleted. Azure disk encryption is not supported for the OS disk. Each virtual machine also has a local temporary disk for OS operations. This disk can also be optionally used to stage training data. This disk is not encrypted. For more information on how encryption at rest works in Azure, see [Azure Data Encryption-at-Rest](#).

## Encryption in transit

Both internal communication between various Azure Machine Learning micro services and external communication of calling the scoring endpoint are supported using SSL. All Azure Storage access is also over a secure channel. For more information, see [Secure Azure Machine Learning web services using SSL](#).

## Using Azure Key Vault

Key Vault instance associated with the workspace is used by Azure Machine Learning service to store credentials of various kinds:

- The associated storage account connection string
- Passwords to Azure Container Repository instances
- Connection Strings to data stores.

SSH passwords and keys to compute targets such as HDI HDInsight and VM are stored in a separate Key Vault that is associated with Microsoft subscription. Azure Machine Learning service does store any passwords or keys provided by the user instead it generates, authorizes, and stores its own SSH keys in order to connect to VM/HDInsight to run the experiments. Each workspace has an associated system-assigned Managed Identity (with the same name as the workspace) that has access to all keys, secrets, and certificates in the Key Vault.

## Monitoring

Users can see the activity log under the workspace to see various operations performed on the workspace and get the basic information like the operation name, event initiated by, timestamp etc.

The following screenshot shows the activity log for a workspace:

OPERATION NAME	STATUS	TIME	TIME STAMP	SUBSCRIPTION	EVENT INITIATED BY
List secrets for compute resources in Mac	Succeeded	2 wk ago	Tue Feb 12 ...	Aashish's Subscription	aashishb@microsoft.com
List secrets for compute resources in Mac	Succeeded	2 wk ago	Tue Feb 12 ...	Aashish's Subscription	aashishb@microsoft.com
List nodes for compute resource in Machir	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
Creates or updates the compute resource	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
List nodes for compute resource in Machir	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
List nodes for compute resource in Machir	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
Creates or updates the compute resource	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
Creates or updates the compute resource	Failed	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
Deletes the compute resources in Machir	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
Deletes the compute resources in Machir	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
Deletes the compute resources in Machir	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com
Deletes the compute resources in Machir	Succeeded	3 wk ago	Wed Feb 06 ...	Aashish's Subscription	aashishb@microsoft.com

Scoring request details are stored in the AppInsights, which is created in user's subscription while creating the

workspace. This includes fields like HttpMethod, UserAgent, ComputeType, RequestUrl, StatusCode, RequestId, Duration etc.

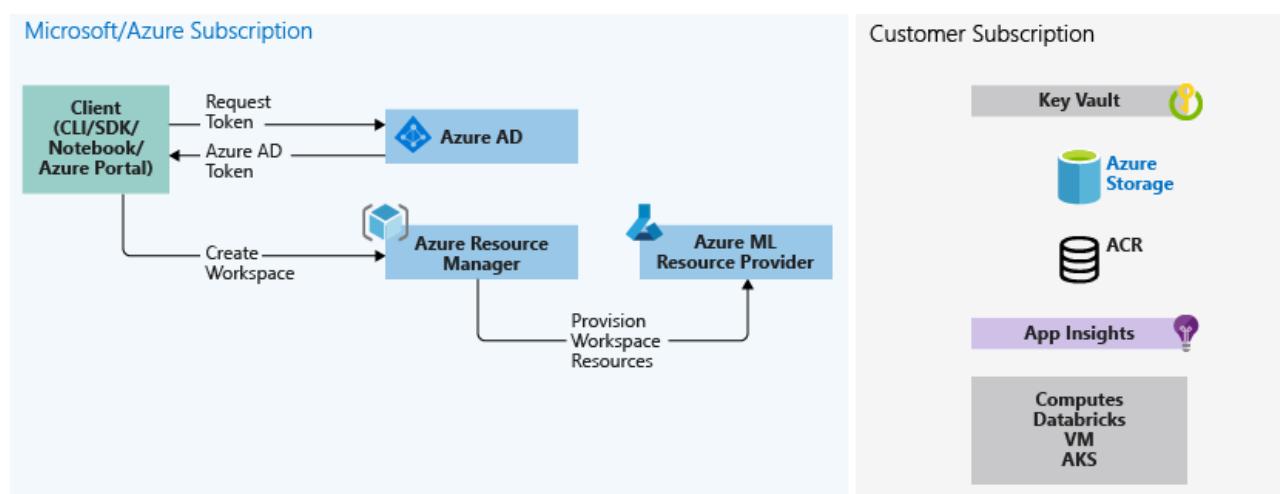
## Data Flow Diagram

### Create Workspace

The following diagram shows the create workspace workflow. User logs into Azure AD from any of the supported Azure Machine Learning service clients (CLI, Python SDK, Azure portal) and requests the appropriate Azure Resource Manager token. User then calls Azure Resource Manager to create the workspace. Azure Resource Manager contacts the Azure Machine Learning service Resource Provider to provision the workspace. Additional resources are created in the customer's subscription during workspace creation:

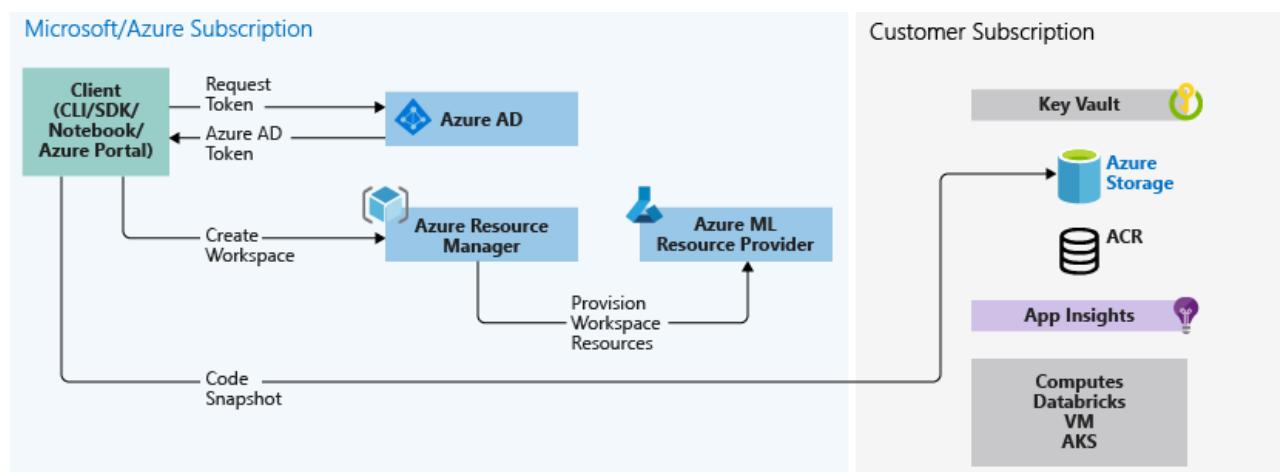
- KeyVault (to store secrets)
- An Azure Storage account (including Blob & FileShare)
- Azure Container Registry (to store docker images for inferencing and experimentation)
- Application Insights (to store telemetry)

Other computes attached to a workspace (Azure Kubernetes Service, VM etc.) can also be provisioned by customers as needed.



### Save source code (training scripts)

The following diagram shows the code snapshot workflow. Associated with an Azure Machine Learning service workspace are directories (experiments), which contain the source code (training scripts). These are stored on the customer's local machine and in the cloud (in the Azure Blob Storage under customer's subscription). These code snapshots are used for execution or inspection for historical auditing.



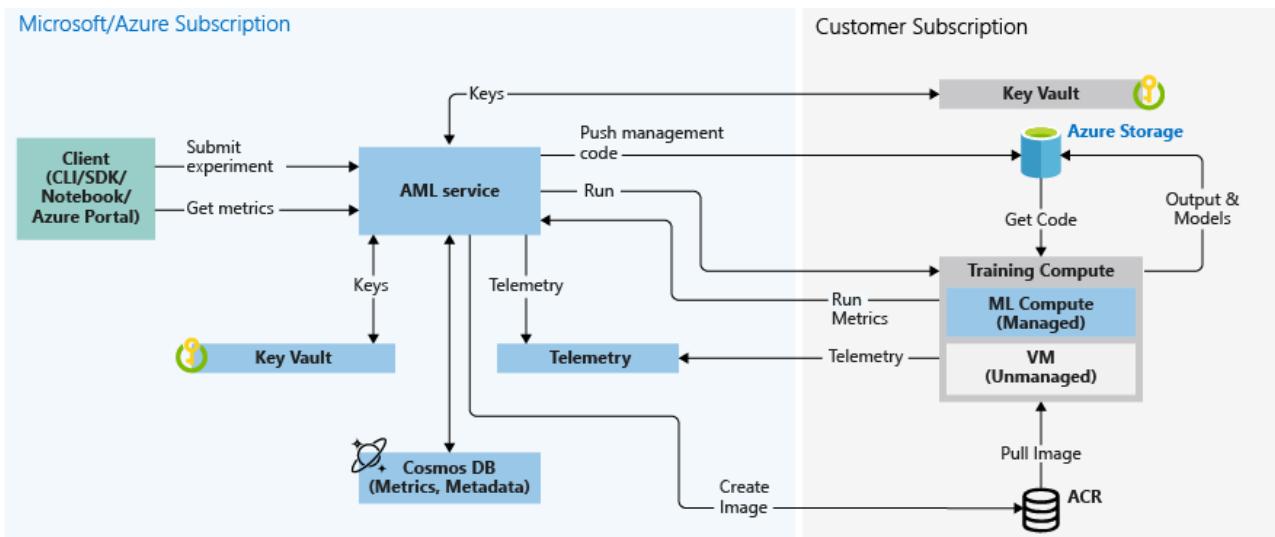
### Training

The following diagram shows the training workflow.

- Azure Machine Learning service is called with the snapshot ID for the code snapshot saved above
- Azure Machine Learning service creates run ID (optional) & Azure Machine Learning service token, which is later used by the compute targets like Machine Learning Compute/VM to talk back to Azure Machine Learning service
- You can choose either a managed compute (ex. Machine Learning Compute) or unmanaged compute (ex. VM) to run your training jobs. Data flow is explained for both the scenarios below:
- (VM/HDIInsight/Local – accessed using SSH creds in Key Vault in Microsoft subscription) Azure Machine Learning service runs management code on compute target that:
  1. Prepares the environment (note: Docker is an option for VM/Local as well. See steps for Machine Learning Compute below to understand how running experiment on docker container works)
  2. Downloads the code
  3. Sets up environment variables/configs
  4. Runs user script (code snapshot mentioned above)
- (Machine Learning Compute – accessed using workspace managed identity) Note that since Machine Learning Compute is a managed compute that is, it is managed by Microsoft, as a result it runs under the Microsoft subscription.
  1. Remote Docker construction is kicked off if needed
  2. Writes management code to user Azure FileShare
  3. Starts container with initial command that is, management code in the above step

#### Querying runs & metrics

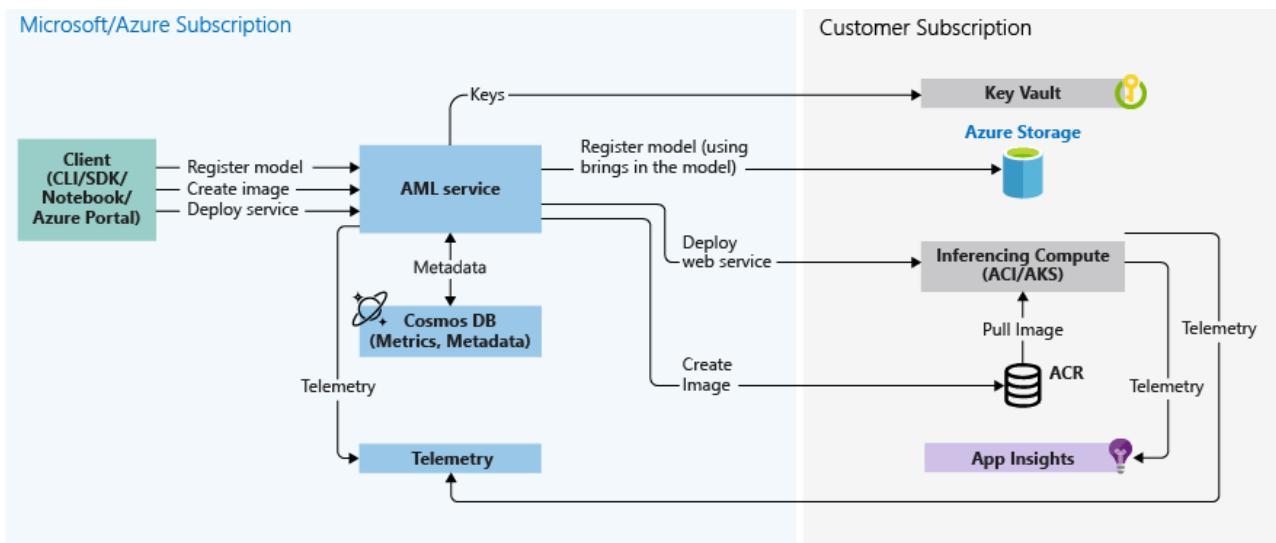
This step is shown in the flow where training compute writes the *Run Metrics* back to the Azure Machine Learning service from where it gets stored in the Cosmos DB. Clients can call Azure Machine Learning service that will in turn pull metrics from the Cosmos DB and return it back to the client.



#### Creating web services

The following diagram shows the inferencing workflow in which model is deployed as a web service. See details below:

- User registers a model using a client like Azure ML SDK
- User creates image using model, score file, and other model dependencies
- The Docker Image is created and stored in ACR
- Webservice is deployed to the compute target (ACI/AKS) using the image created above
- Scoring request details are stored in the AppInsights, which is in user's subscription
- Telemetry is also pushed to Microsoft/Azure subscription



## Next steps

- [Secure Azure Machine Learning web services with SSL](#)
- [Consume a ML Model deployed as a web service](#)
- [How to run batch predictions](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Azure Machine Learning service SDK](#)
- [Use Azure Machine Learning service with Azure Virtual Networks](#)
- [Best practices for building recommendation systems](#)
- [Build a real-time recommendation API on Azure](#)

# Create and manage Azure Machine Learning service workspaces

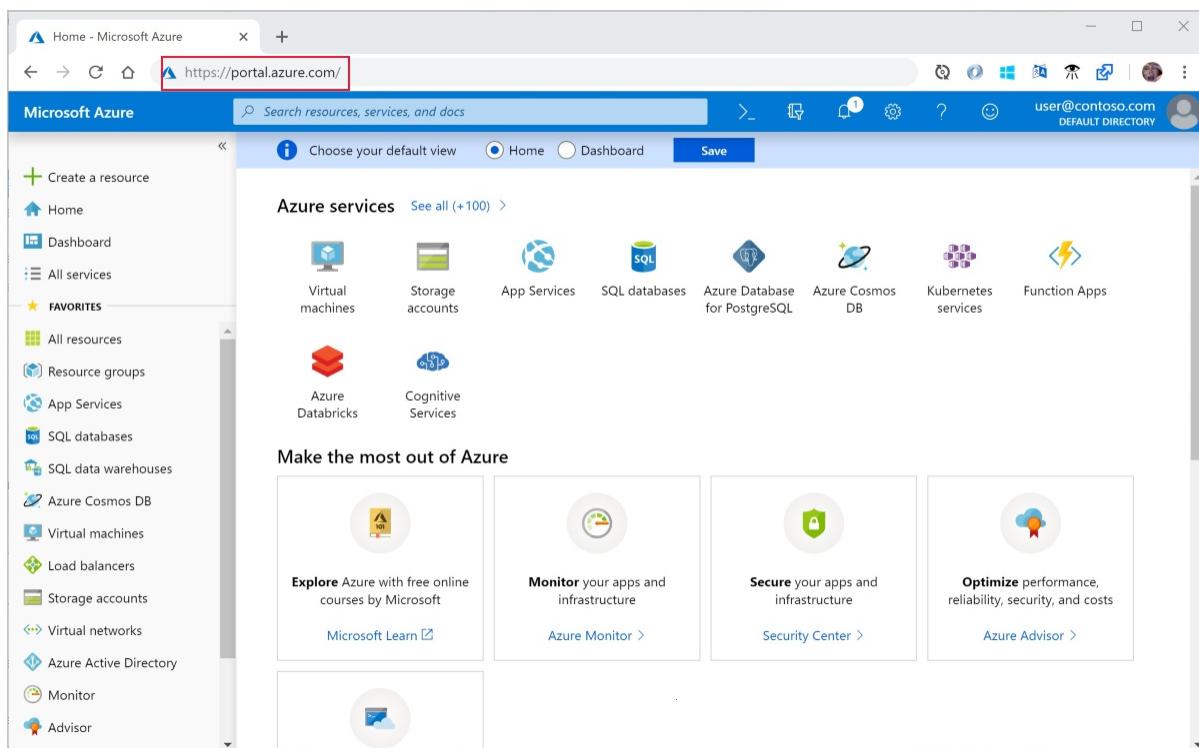
1/29/2019 • 2 minutes to read

In this article, you'll create, view, and delete **Azure Machine Learning service workspaces** in the Azure portal for [Azure Machine Learning service](#). You can also create and delete workspaces [using the CLI](#) or [with Python code](#).

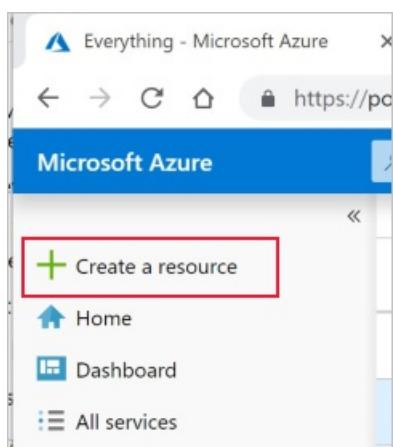
## Create a workspace

To create a workspace, 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 service](#) today.

1. Sign in to the [Azure portal](#) by using the credentials for the Azure subscription you use.



2. In the upper-left corner of the portal, select **Create a resource**.



3. In the search bar, enter **Machine Learning**. Select the **Machine Learning service workspace** search result.

4. In the **ML service workspace** pane, scroll to the bottom and select **Create** to begin.

5. In the **ML service workspace** pane, configure your workspace.

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

FIELD	DESCRIPTION
Subscription	Select the Azure subscription that you want to use.
Resource group	Use an existing resource group in your subscription, or enter a name to create a new resource group. A resource group is a container that holds related resources for an Azure solution. In this example, we use <b>docs-aml</b> .
Location	Select the location closest to your users and the data resources. This location is where the workspace is created.

**ML service workspace** □ X

Machine Learning service workspace

\* Workspace name

\* Subscription

\* Resource group  
 ▼  
[Create new](#)

\* Location  
 ▼

**i** For your convenience, these resources are added automatically to the workspace, if regionally available: [Azure Container Registry](#), [Azure storage](#), [Azure Application Insights](#) and [Azure Key Vault](#).

**Create** [Automation options](#)

6. To start the creation process, select **Create**. It can take a few moments to create the workspace.
7. To check on the status of the deployment, select the Notifications icon, **bell**, on the toolbar.
8. When the process is finished, a deployment success message appears. It's also present in the notifications section. To view the new workspace, select **Go to resource**.

The screenshot shows the Azure portal's Notifications blade. At the top, there are icons for back, forward, search, settings, help, and a user profile. The user is signed in as [user@contoso.com](#) in the Default Directory. A red box highlights the bell icon. Below the header, the title "Notifications" is displayed. A message is shown: "More events in the activity log → Deployment succeeded Deployment 'Microsoft.MachineLearningServices' to resource group 'my-rg' was successful." Two buttons at the bottom are "Go to resource" (highlighted with a red box) and "Pin to dashboard".

## View a workspace

1. In top left corner of the portal, select **All services**.
2. In the **All services** filter field, type **Machine Learning service workspace**.

The screenshot shows the Azure portal's "All services" blade. On the left is a sidebar with various service icons and links: Create a resource, All services, Favorites (Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, Cost Management + Billing, Help + support). The main area has a search bar with "All services" and a "Filter" button (highlighted with a red box). Below the search bar are buttons for "Collapse all" and "Expand all". A list of services is shown under the "GENERAL (14)" category. The "Machine Learning service workspace" service is listed, highlighted with a blue background, and has a star icon next to it.

Service	Status	Action
Dashboard		★
All resources		★
Recent		★
Management groups		★
Subscriptions		★
Resource groups		★
Cost Management + Billing	PREVIEW	★
Reservations		★
Marketplace		★
Help + support		★
Service Health		★
Templates	PREVIEW	★
Tags		★
What's new		★

3. In the filter results, select **Machine Learning service workspace** to display a list of your workspaces.

## Everything



Filter

machine learning workspace



### Results

NAME	PUBLISHER	CATEGORY
Machine Learning Workspace (preview)	Microsoft	Analytics
Machine Learning Studio Workspace	Microsoft	Analytics
Machine Learning Studio Web Service Plan	Microsoft	Analytics

4. Look through the list of workspaces found. You can filter based on subscription, resource groups, and locations.

The screenshot shows the 'Machine Learning Workspaces' page in the Azure portal. At the top, there's a breadcrumb navigation: Home > Machine Learning Workspaces. Below it, the title 'Machine Learning Workspaces' and a subtitle 'Microsoft - PREVIEW'. There are buttons for 'Add', 'Edit columns', 'Refresh', 'Assign tags', and 'Delete'. A filter bar below shows 'Subscriptions: All 30 selected' and dropdowns for 'All subscriptions', 'All resource gro...', and 'All locations'. The main table lists one item: 'doc-ws' under 'NAME', 'docs-aml' under 'RESOURCE GROUP', and 'East US 2' under 'LOCATION'. The entire row for 'doc-ws' is also highlighted with a red box.

5. Select the workspace you just created to display its properties.

The screenshot shows the 'doc-ws' workspace properties page. On the left, a sidebar lists 'Create a resource', 'All services', 'FAVORITES' (Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts), and 'OVERVIEW' (Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems). The main content area shows the workspace details: 'Machine Learning Workspace - PREVIEW', 'Resource group: docs-aml', 'Location: East US 2', 'Subscription: Visual Studio Ultimate with MSDN', 'Subscription ID: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX', 'Storage: docwsXXXXXXXXXX', 'Registry: docwsXXXXXXXXXX', 'Key Vault: docwsXXXXXXXXXX', and 'Application Insights: docwsXXXXXXXXXX'. Below this is a 'Getting Started' section with a button 'Explore your Azure Machine Learning Workspace'.

## Delete a workspace

Use the Delete button at the top of the workspace you wish to delete.

The screenshot shows the Azure Machine Learning Workspace 'doc-ws'. At the top right, there is a 'Delete' button with a trash icon, which is highlighted with a red box. Below the workspace name, it says 'Machine Learning Workspace - PREVIEW'. A search bar is on the left, and a navigation bar at the bottom includes 'Overview'.

## Clean up resources

### IMPORTANT

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

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.

The screenshot shows the Azure portal with the URL 'portal.azure.com'. On the left, the 'Resource groups' blade is open, with the 'newacct' resource group selected. The 'Delete resource group' button is highlighted with a red box. The main pane shows details for the 'newacct' resource group, including its subscription name, deployment status, and a list of items.

NAME	TYPE	LOCATION
newacct	Azure Cosmos DB account	South Central US

2. From the list, select the resource group you created.
3. Select **Delete resource group**.
4. Enter the resource group name. Then select **Delete**.

## Next steps

Follow the full-length tutorial to learn how to use a workspace to build, train, and deploy models with Azure Machine Learning service.

[Tutorial: Train models](#)

# Create an Azure Machine Learning service workspace by using a template

2/22/2019 • 3 minutes to read

In this article, you learn several ways to create an Azure Machine Learning service workspace using Azure Resource Manager templates.

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 service](#).
- To use a template from a CLI, you need either [Azure PowerShell](#) or the [Azure CLI](#).

## Resource Manager template

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.

The following template can be used to create an Azure Machine Learning service workspace and associated Azure resources:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "workspaceName": {
            "type": "string",
            "metadata": {
                "description": "The name of the Azure Machine Learning service workspace."
            }
        },
        "location": {
            "type": "string",
            "allowedValues": [
                "eastus",
                "eastus2",
                "southcentralus",
                "southeastasia",
                "westcentralus",
                "westeurope",
                "westus2"
            ],
            "metadata": {
                "description": "The location where the workspace will be created."
            }
        }
    },
    "variables": {
        "storageAccount": {
            "name": "[concat('sa',uniqueString(resourceGroup().id))]",
            "type": "Standard_LRS"
        },
        "keyVault": {

```

```

        "name": "[concat('kv',uniqueString(resourceGroup().id))]",
        "tenantId": "[subscription().tenantId]"
    },
    "applicationInsightsName": "[concat('ai',uniqueString(resourceGroup().id))]",
    "containerRegistryName": "[concat('cr',uniqueString(resourceGroup().id))]"
},
"resources": [
{
    "name": "[variables('storageAccount').name]",
    "type": "Microsoft.Storage/storageAccounts",
    "location": "[parameters('location')]",
    "apiVersion": "2018-07-01",
    "sku": {
        "name": "[variables('storageAccount').type]"
    },
    "kind": "StorageV2",
    "properties": {
        "encryption": {
            "services": {
                "blob": {
                    "enabled": true
                },
                "file": {
                    "enabled": true
                }
            },
            "keySource": "Microsoft.Storage"
        },
        "supportHttpsTrafficOnly": true
    }
},
{
    "name": "[variables('keyVault').name]",
    "type": "Microsoft.KeyVault/vaults",
    "apiVersion": "2018-02-14",
    "location": "[parameters('location')]",
    "properties": {
        "tenantId": "[variables('keyVault').tenantId]",
        "sku": {
            "name": "standard",
            "family": "A"
        },
        "accessPolicies": []
    }
},
{
    "name": "[variables('applicationInsightsName')]",
    "type": "Microsoft.Insights/components",
    "apiVersion": "2015-05-01",
    "location": "
[if(or>equals(parameters('location'),'eastus2'),equals(parameters('location'),'westcentralus'),'southcentralus',parameters('location'))]",
        "kind": "web",
        "properties": {
            "Application_Type": "web"
        }
},
{
    "name": "[variables('containerRegistryName')]",
    "type": "Microsoft.ContainerRegistry/registries",
    "apiVersion": "2017-10-01",
    "location": "[parameters('location')]",
    "sku": {
        "name": "Standard"
    },
    "properties": {
        "adminUserEnabled": true
    }
},
{

```

```

{
  "name": "[parameters('workspaceName')]",
  "type": "Microsoft.MachineLearningServices/workspaces",
  "apiVersion": "2018-11-19",
  "location": "[parameters('location')]",
  "dependsOn": [
    "[variables('storageAccount').name]",
    "[variables('keyVault').name]",
    "[variables('applicationInsightsName')]",
    "[variables('containerRegistryName')]"
  ],
  "identity": {
    "type": "systemAssigned"
  },
  "properties": {
    "friendlyName": "[parameters('workspaceName')]",
    "keyVault": "[resourceId('Microsoft.KeyVault/vaults',variables('keyVault').name)]",
    "applicationInsights": "[resourceId('Microsoft.Insights/components',variables('applicationInsightsName'))]",
    "containerRegistry": "[resourceId('Microsoft.ContainerRegistry/registries',variables('containerRegistryName'))]",
    "storageAccount": "[resourceId('Microsoft.Storage/storageAccounts/',variables('storageAccount').name)]"
  }
}
}

```

This template creates the following Azure services:

- Azure Resource Group
- 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 parameters:

- The **location** where the resource group and services 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 **workspace name**, which is the friendly name of the Azure Machine Learning workspace.

The names of the other services are generated randomly.

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](#)

## Use the Azure portal

1. Follow the steps in [Deploy resources from custom template](#). When you arrive at the **Edit template** screen,

paste in the template from this document.

2. Select **Save** to use the template. Provide the following information and agree to the listed terms and conditions:

- Subscription: Select the Azure subscription to use for these resources.
- Resource group: Select or create a resource group to contain the services.
- 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.

The screenshot shows the 'Custom deployment' blade in the Azure portal. At the top, it displays a 'Customized template' with 5 resources, and links to 'Edit template', 'Edit parameters', and 'Learn more'. Below this, the 'BASICS' section is shown with fields for Subscription (documentationteam), Resource group ((New) myresourcegroup), and Location (East US 2). In the 'SETTINGS' section, the Workspace Name is set to myworkspace and the Location is set to eastus2. Under 'TERMS AND CONDITIONS', there is a link to 'Azure Marketplace Terms' and a note about agreeing to terms. A checkbox labeled 'I agree to the terms and conditions stated above' is checked. At the bottom, a blue 'Purchase' button is visible.

For more information, see [Deploy resources from custom template](#).

## Use Azure PowerShell

This example assumes that you have saved the template to a file named `azuredeploy.json` in the current directory:

```
New-AzResourceGroup -Name examplegroup -Location "East US"
New-AzResourceGroupDeployment -Name exampledeployment ` 
    -ResourceGroupName examplegroup -Location "East US" ` 
    -TemplateFile .\azuredploy.json -WorkspaceName "exampleworkspace"
```

For more information, see [Deploy resources with Resource Manager templates and Azure PowerShell](#) and [Deploy private Resource Manager template with SAS token and Azure PowerShell](#).

## Use Azure CLI

This example assumes that you have saved the template to a file named `azuredploy.json` in the current directory:

```
az group create --name examplegroup --location "East US"
az group deployment create \
    --name exampledeployment \
    --resource-group examplegroup \
    --template-file azuredploy.json \
    --parameters workspaceName=exampleworkspace
```

For more information, see [Deploy resources with Resource Manager templates and Azure CLI](#) and [Deploy private Resource Manager template with SAS token and Azure CLI](#).

## Next steps

- [Deploy resources with Resource Manager templates and Resource Manager REST API](#).
- [Creating and deploying Azure resource groups through Visual Studio](#).

# Manage users and roles in an Azure Machine Learning workspace

3/4/2019 • 3 minutes to read

In this article, you learn how to add users to an Azure Machine Learning workspace. You also learn how to assign users to different roles and create custom roles.

## Built-in 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.

- **Reader**

This role allows read-only actions in the workspace. Readers can list and view assets in a workspace, but can't create or update these assets.

- **Contributor**

This role allows users to 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**

This role gives users full access to the workspace, including the ability to view, create, edit, or delete (where applicable) assets in a workspace. Additionally, you can add or remove users, and change role assignments.

## Add or remove users

If you're an owner of a workspace, you can add and remove users from the workspace by choosing one of the following methods:

- [Azure portal UI](#)
- [PowerShell](#)
- [Azure CLI](#)
- [REST API](#)
- [Azure Resource Manager templates](#)

If you have installed the [Azure Machine Learning CLI](#), you can also use a CLI command to add users to the workspace.

```
az ml workspace share -n <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 -n my_workspace -g my_resource_group --role Contributor --user jdoe@contoson.com
```

## 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 which specifies the permission and scope you want for the role. For example, here is a role definition file for a custom role named "Data Scientist" scoped at a specific workspace level:

`data_scientist_role.json :`

```
{
  "Name": "Data Scientist",
  "IsCustom": true,
  "Description": "Can run experiment but can't create or delete compute.",
  "Actions": ["*"],
  "NotActions": [
    "Microsoft.MachineLearningServices/workspaces/*/delete",
    "Microsoft.MachineLearningServices/workspaces/computes/*/write",
    "Microsoft.MachineLearningServices/workspaces/computes/*/delete",
    "Microsoft.Authorization/*/write"
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>/resourceGroups/<resource_group_name>/providers/Microsoft.MachineLearningServices/workspaces/<workspace_name>"
  ]
}
```

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.

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 add a user with this role by using the `az ml workspace share` CLI command:

```
az ml workspace share -n my_workspace -g my_resource_group --role "Data Scientist" --user jdoe@contoson.com
```

For more information about custom roles in Azure, see [this document](#).

## Next steps

Follow the full-length tutorial to learn how to use a workspace to build, train, and deploy models with the Azure Machine Learning service.

[Tutorial: Train models](#)

# Configure a development environment for Azure Machine Learning

3/4/2019 • 12 minutes to read

In this article, you learn how to configure a development environment to work with Azure Machine Learning service. Machine Learning service is platform agnostic.

The only requirements for your development environment are Python 3, Anaconda (for isolated environments), and a configuration file that contains your Azure Machine Learning workspace information.

This article focuses on the following environments and tools:

- **Azure Notebooks:** A Jupyter Notebooks service that's hosted in the Azure cloud. It's the easiest way to get started, because the Azure Machine Learning SDK is already installed.
- **The Data Science Virtual Machine (DSVM):** A pre-configured development or experimentation environment in the Azure cloud that's designed for data science work and can be deployed to either CPU only VM instances or GPU-based instances. Python 3, Conda, Jupyter Notebooks, and the Azure Machine Learning SDK are already installed. The VM comes with popular machine learning and deep learning frameworks, tools, and editors for developing machine learning solutions. It's probably the most complete development environment for machine learning on the Azure platform.
- **The Jupyter Notebook:** If you're already using the Jupyter Notebook, the SDK has some extras that you should install.
- **Visual Studio Code:** If you use Visual Studio Code, it has some useful extensions that you can install.
- **Azure Databricks:** A popular data analytics platform that's based on Apache Spark. Learn how to get the Azure Machine Learning SDK onto your cluster so that you can deploy models.

If you already have a Python 3 environment, or just want the basic steps for installing the SDK, see the [Local computer](#) section.

## Prerequisites

- An Azure Machine Learning service workspace. To create the workspace, see [Get started with Azure Machine Learning service](#).
- Either the [Anaconda](#) or [Miniconda](#) package manager.

### IMPORTANT

Anaconda and Miniconda are not required when you're using Azure Notebooks.

- On Linux or macOS, you need the bash shell.

### TIP

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.

- On Windows, you need the command prompt or Anaconda prompt (installed by Anaconda and Miniconda).

## Azure Notebooks

[Azure Notebooks](#) (preview) is an interactive development environment in the Azure cloud. It's the easiest way to get started with Azure Machine Learning development.

- The Azure Machine Learning SDK is already installed.
- After you create an Azure Machine Learning service workspace in the Azure portal, you can click a button to automatically configure your Azure Notebook environment to work with the workspace.

To get started developing with Azure Notebooks, see [Get started with Azure Machine Learning service](#).

By default, Azure Notebooks uses a free service tier that is limited to 4GB of memory and 1GB of data. You can, however, remove these limits by attaching a Data Science Virtual Machine instance to the Azure Notebooks project. For more information, see [Manage and configure Azure Notebooks projects - Compute tier](#).

## Data Science Virtual Machine

The DSVM is a customized virtual machine (VM) image. It's designed for data science work that's pre-configured with:

- 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

The Azure Machine Learning SDK works on either the Ubuntu or Windows version of the DSVM. But if you plan to use the DSVM as a compute target as well, only Ubuntu is supported.

To use the DSVM as a development environment, do the following:

1. Create a DSVM in either of the following environments:

- The Azure portal:
  - [Create an Ubuntu Data Science Virtual Machine](#)
  - [Create a Windows Data Science Virtual Machine](#)
- The Azure CLI:

### IMPORTANT

- When you use the Azure CLI, you must first sign in to your Azure subscription by using the `az login` command.
- When you use the commands in this step, you must provide a resource group name, a name for the VM, a username, and a password.

- To create an Ubuntu Data Science Virtual Machine, use the following command:

```
# create a Ubuntu 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
# 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 Data Science Virtual Machine, 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. The Azure Machine Learning SDK is already installed on the DSVM. To use the Conda environment that contains the SDK, use one of the following commands:

- For Ubuntu DSVM:

```
conda activate py36
```

- For Windows DSVM:

```
conda activate AzureML
```

3. To verify that you can access the SDK and check the version, use the following Python code:

```
import azureml.core
print(azureml.core.VERSION)
```

4. To configure the DSVM to use your Azure Machine Learning service workspace, see the [Create a workspace configuration file](#) section.

For more information, see [Data Science Virtual Machines](#).

## Local computer

When you're using a local computer (which might also be a remote virtual machine), create an Anaconda environment and install the SDK by doing the following:

1. Download and install [Anaconda](#) (Python 3.7 version) if you don't already have it.
2. Open an Anaconda prompt and create an environment with the following commands:

Run the following command to create the environment.

```
conda create -n myenv python=3.6.5
```

Then activate the environment.

```
conda activate myenv
```

This example creates an environment using python 3.6.5, but any specific subversions can be chosen. SDK compatibility may not be guaranteed with certain major versions (3.5+ is recommended), and it's recommended to try a different version/subversion in your Anaconda environment if you run into errors. It will take several minutes to create the environment while components and packages are downloaded.

3. Run the following commands in your new environment to enable environment-specific ipython kernels.

This will ensure expected kernel and package import behavior when working with Jupyter Notebooks within Anaconda environments:

```
conda install notebook ipykernel
```

Then run the following command to create the kernel:

```
ipython kernel install --user
```

4. Use the following commands to install packages:

This command installs the base Azure Machine Learning SDK with notebook and automl extras. The `automl` extra is a large install, and can be removed from the brackets if you don't intend to run automated machine learning experiments. The `automl` extra also includes the Azure Machine Learning Data Prep SDK by default as a dependency.

```
pip install azureml-sdk[notebooks,automl]
```

Use this command to install the Azure Machine Learning Data Prep SDK on its own:

```
pip install azureml-dataprep
```

#### NOTE

If you get a message that PyYAML can't be uninstalled, use the following command instead:

```
pip install --upgrade azureml-sdk[notebooks,automl] azureml-dataprep --ignore-installed PyYAML
```

It will take several minutes to install the SDK. See the [install guide](#) for more information on installation options.

5. Install other packages for your machine learning experimentation.

Use either of the following commands and replace `<new package>` with the package you want to install. Installing packages via `conda install` requires that the package is part of the current channels (new channels can be added in Anaconda Cloud).

```
conda install <new package>
```

Alternatively, you can install packages via `pip`.

```
pip install <new package>
```

## Jupyter Notebooks

Jupyter Notebooks are part of the [Jupyter Project](#). They provide an interactive coding experience where you create documents that mix live code with narrative text and graphics. Jupyter Notebooks are also a great way to share your results with others, because you can save the output of your code sections in the document. You can install Jupyter Notebooks on a variety of platforms.

The procedure in the [Local computer](#) section installs necessary components for running Jupyter Notebooks in an Anaconda environment. To enable these components in your Jupyter Notebook environment, do the following:

1. Open an Anaconda prompt and activate your environment.

```
conda activate myenv
```

2. Launch the Jupyter Notebook server with the following command:

```
jupyter notebook
```

3. To verify that Jupyter Notebook can use the SDK, create a **New** notebook, select **Python 3** as your kernel, and then run the following command in a notebook cell:

```
import azureml.core  
azureml.core.VERSION
```

4. If you encounter issues importing modules and receive a `ModuleNotFoundError`, ensure your Jupyter kernel is connected to the correct path for your environment by running the following code in a Notebook cell.

```
import sys  
sys.path
```

5. To configure the Jupyter Notebook to use your Azure Machine Learning service workspace, go to the [Create a workspace configuration file](#) section.

## Visual Studio Code

Visual Studio Code is a cross platform code editor. It relies on a local Python 3 and Conda installation for Python support, but it provides additional tools for working with AI. It also provides support for selecting the Conda environment from within the code editor.

To use Visual Studio Code for development, do the following:

1. To learn how to use Visual Studio Code for Python development, see [Get started with Python in VSCode](#).
2. To select the Conda environment, open VS Code, and then select Ctrl+Shift+P (Linux and Windows) or Command+Shift+P (Mac). The **Command Pallet** opens.
3. Enter **Python: Select Interpreter**, and then select the Conda environment.
4. To validate that you can use the SDK, create and then run a new Python file (.py) that contains the following code:

```
import azureml.core  
azureml.core.VERSION
```

5. To install the Azure Machine Learning extension for Visual Studio Code, see [Tools for AI](#).

For more information, see [Use Azure Machine Learning for Visual Studio Code](#).

## Azure Databricks

Azure Databricks is an Apache Spark-based environment in the Azure cloud. It provides a collaborative Notebook based environment with CPU or GPU based compute cluster.

How Azure Databricks works with Azure Machine Learning service:

- You can train a model using Spark MLlib and deploy the model to ACI/AKS from within Azure Databricks.
- You can also use [automated machine learning](#) capabilities in a special Azure ML SDK with Azure Databricks.
- You can use Azure Databricks as a compute target from an [Azure Machine Learning pipeline](#).

### Set up your Databricks cluster

Create a [Databricks cluster](#). Some settings apply only if you install the SDK for automated machine learning on Databricks. **It will take few minutes to create the cluster.**

Use these settings:

SETTING	APPLIES TO	VALUE
Cluster name	always	yourclustername
Databricks Runtime	always	Any non ML runtime (non ML 4.x, 5.x)
Python version	always	3
Workers	always	2 or higher
Worker node VM types (determines max # of concurrent iterations)	Automated ML only	Memory optimized VM preferred
Enable Autoscaling	Automated ML only	Uncheck

Wait until the cluster is running before proceeding further.

### Install the correct SDK into a Databricks library

Once the cluster is running, [create a library](#) to attach the appropriate Azure Machine Learning SDK package to your cluster.

1. Choose **only one** option (no other SDK installation are supported)

SDK PACKAGE EXTRAS	SOURCE	PYPI NAME
For Databricks	Upload Python Egg or PyPI	azureml-sdk[databricks]
For Databricks -with- automated ML capabilities	Upload Python Egg or PyPI	azureml-sdk[automl_databricks]

#### WARNING

No other SDK extras can be installed. Choose only one of the preceding options [databricks] or [automl\_databricks].

- Do not select **Attach automatically to all clusters**.
  - Select **Attach** next to your cluster name.
2. Monitor for errors until status changes to **Attached**, which may take several minutes. If this step fails, check the following:

Try restarting your cluster by:

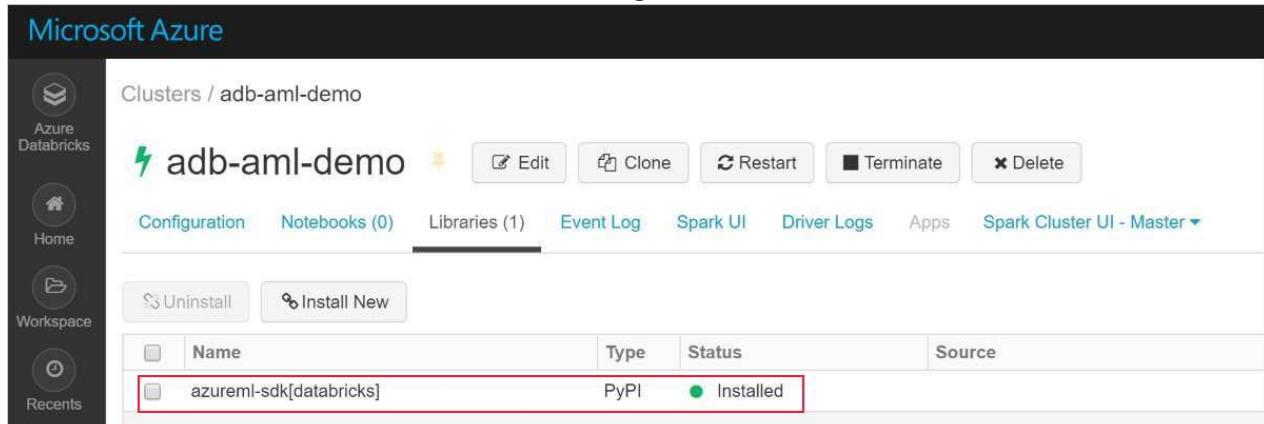
- In the left pane, select **Clusters**.
- In the table, select your cluster name.
- On the **Libraries** tab, select **Restart**.

Also consider:

- Some packages, such as `psutil`, can cause Databricks conflicts during installation. To avoid such errors, install packages by freezing lib version, such as  
`psutil cryptography==1.5 pyopenssl==16.0.0 ipython==2.2.0`.
- Or, if you have an old SDK version, deselect it from cluster's installed libs and move to trash. Install the new SDK version and restart the cluster. If there is an issue after this, detach and reattach your cluster.

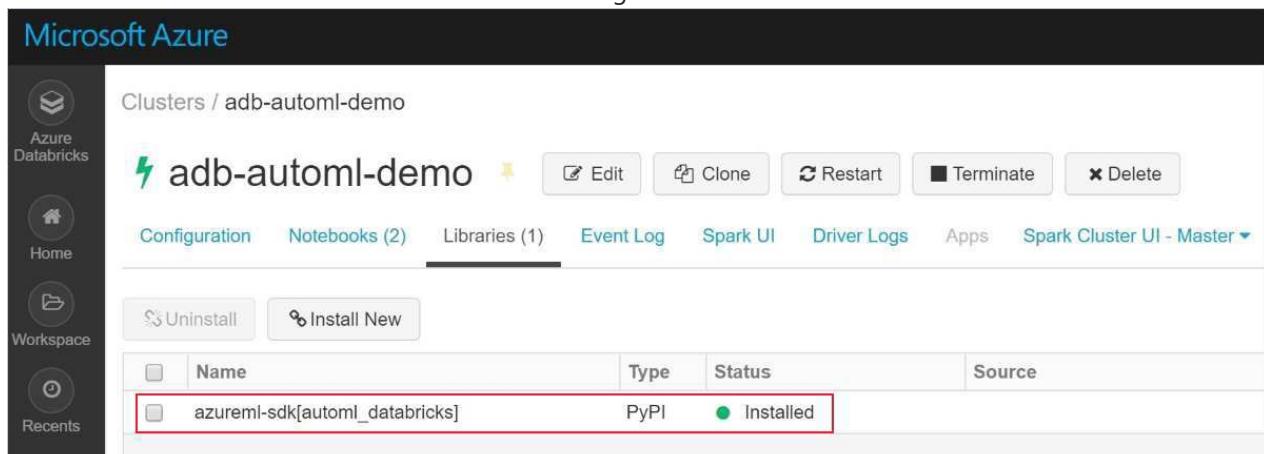
If install was successful, the imported library should look like one of these:

SDK for Databricks **without** automated machine learning



Name	Type	Status	Source
azureml-sdk[databricks]	PyPI	Installed	

SDK for Databricks **WITH** automated machine learning



Name	Type	Status	Source
azureml-sdk[automl_databricks]	PyPI	Installed	

## Start exploring

Try it out:

- Download the [notebook archive file](#) for Azure Databricks/Azure Machine Learning SDK and [import the archive file](#) into your Databricks cluster.
- While many sample notebooks are available, **only these sample notebooks work with Azure Databricks.**

- Learn how to create a pipeline with Databricks as the training compute.

## 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 service workspace. The file is named *config.json*, and it has the following format:

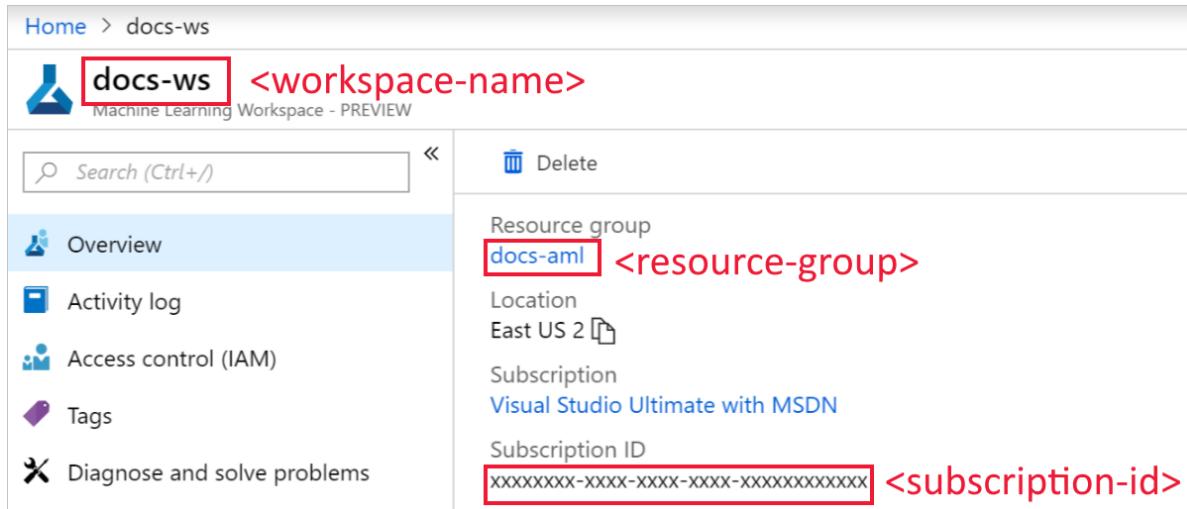
```
{
    "subscription_id": "<subscription-id>",
    "resource_group": "<resource-group>",
    "workspace_name": "<workspace-name>"
}
```

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 *aml\_config*, or in a parent directory.

To use this file from your code, use `ws=Workspace.from_config()`. This code loads the information from the file and connects to your workspace.

You can create the configuration file in three ways:

- **Follow the Azure Machine Learning quickstart:** A *config.json* file is created in your Azure Notebooks library. The file contains the configuration information for your workspace. You can download or copy the *config.json* to other development environments.
- **Create the file manually:** With this method, you use a text editor. You can find the values that go into the configuration file by visiting your workspace in the [Azure portal](#). Copy the workspace name, resource group, and subscription ID values and use them in the configuration file.



- **Create the file programmatically:** In the following code snippet, you connect to a workspace by providing the subscription ID, resource group, and workspace name. It then saves the workspace configuration to the file:

```
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')
```

This code writes the configuration file to the *aml\_config/config.json* file.

## Next steps

- [Train a model](#) on Azure Machine Learning with the MNIST dataset
- View the [Azure Machine Learning SDK for Python](#) reference
- Learn about the [Azure Machine Learning Data Prep SDK](#)

# Get started with Azure Machine Learning for Visual Studio Code

2/5/2019 • 4 minutes to read

In this article, you'll learn how to install the **Azure Machine Learning for Visual Studio Code** extension and create your first experiment with Azure Machine Learning service in Visual Studio Code (VS Code).

Use the Azure Machine Learning extension in Visual Studio code to use the Azure Machine Learning service to prep your data, train, and test machine learning models on local and remote compute targets, deploy those models and track custom metrics and experiments.

## 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 service](#) today.
- Visual Studio Code must be installed. VS Code is a lightweight but powerful source code editor that runs on your desktop. It comes with built-in support for Python and more. [Learn how to install VS Code](#).
- [Install Python 3.5 or greater](#).

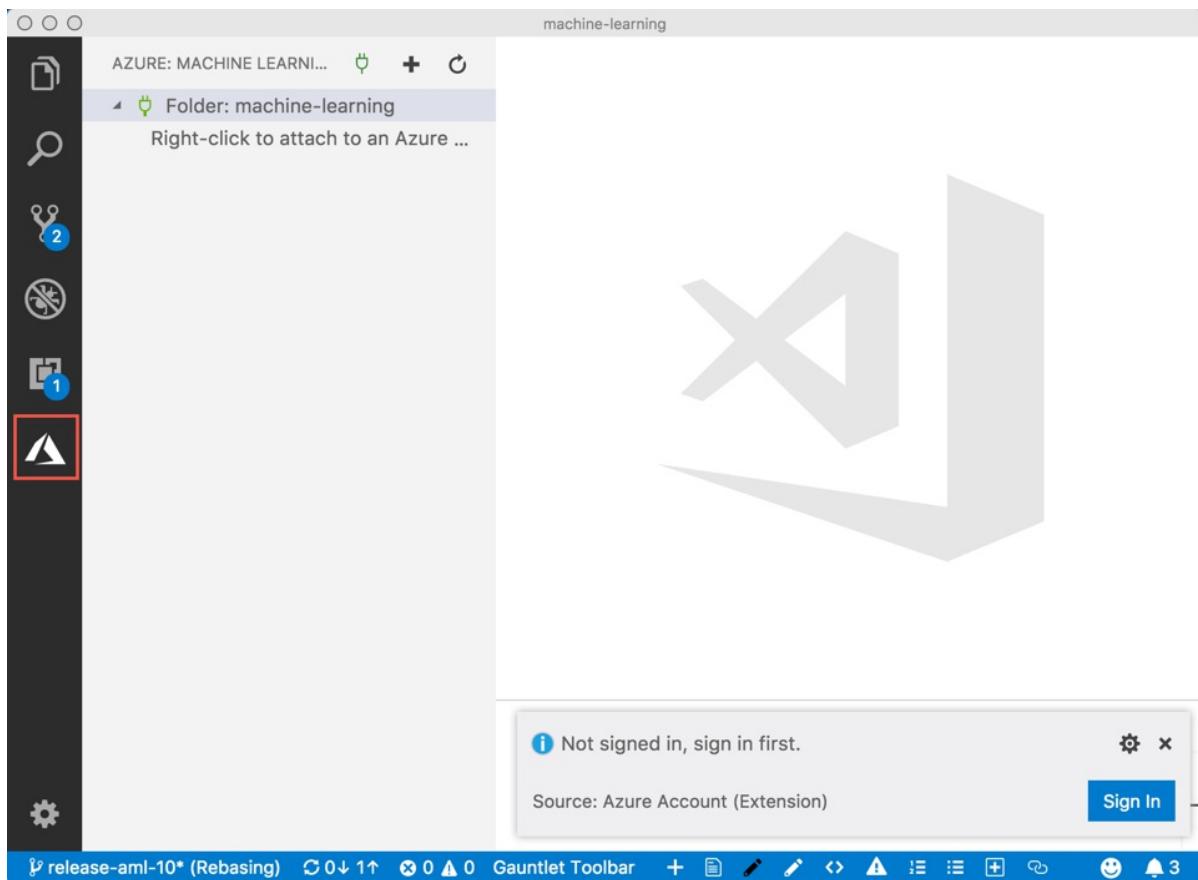
## Install the Azure Machine Learning for VS Code extension

When you install the **Azure Machine Learning** extension, two more extensions are automatically installed (if you have internet access). They are the [Azure Account](#) extension and the [Microsoft Python](#) extension

To work with Azure Machine Learning, we need to turn VS Code into a Python IDE. Working with [Python in Visual Studio Code](#), requires the Microsoft Python extension, which gets installed with the Azure Machine Learning extension automatically. The extension makes VS Code an excellent IDE, and works on any operating system with a variety of Python interpreters. It leverages all of VS Code's power to provide auto complete and IntelliSense, linting, debugging, and unit testing, along with the ability to easily switch between Python environments, including virtual and conda environments. Check out this walk-through of editing, running, and debugging Python code, see the [Python Hello World Tutorial](#)

### To install the Azure Machine Learning extension:

1. Launch VS Code.
2. In a browser, visit:[Azure Machine Learning for Visual Studio Code \(Preview\)](#) extension
3. In that web page, click **Install**.
4. In the extension tab, click **Install**.
5. A welcome tab opens in VS Code for the extension and the Azure symbol is added to activity bar.



6. In the dialog box, click **Sign In** and follow the onscreen prompt to authenticate with Azure.

The Azure Account extension, which was installed along with the Azure Machine Learning for VS Code extension, helps you authenticate with your Azure account. See the list of commands in the [Azure Account extension](#) page.

**TIP**

Check out the [IntelliCode extension for VS Code \(preview\)](#). IntelliCode provides a set of AI-assisted capabilities for IntelliSense in Python, such as inferring the most relevant auto-completions based on the current code context.

## Azure ML SDK Installation

1. Make sure that Python 3.5 or greater is installed and recognized by VS Code. If you install it now, then restart VS Code and select a Python interpreter using instructions at <https://code.visualstudio.com/docs/python/python-tutorial>.
2. In the integrated terminal window, specify the Python interpreter to use or you can hit **Enter** to use your default Python interpreter.

```
PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL 1: Azure ML Package + - ×  
bash-3.2$ bash setup_aml_env.sh 'python'  
Enter path to Python3 interpreter [default: python]:
```

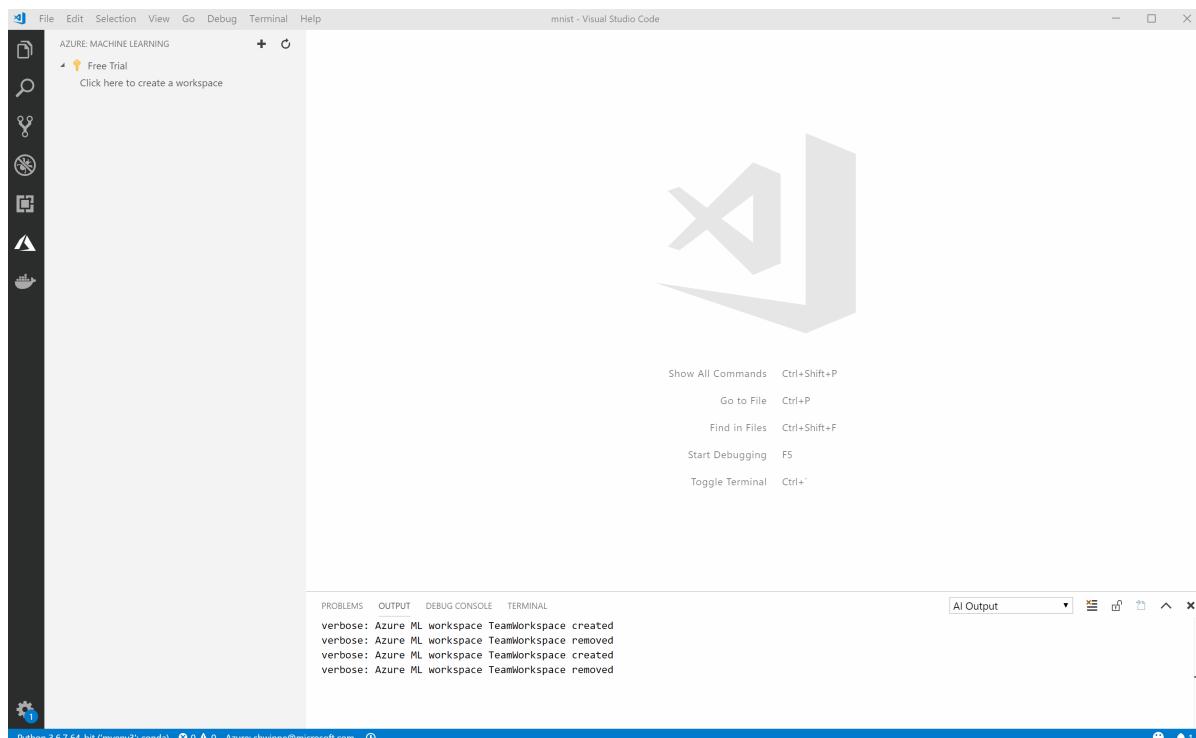
3. In the bottom-right corner of the window, a notification will appear indicating that the Azure ML SDK is being automatically installed. A local private Python environment is created that has the Visual Studio Code prerequisites for working with Azure Machine Learning.

 Azure ML extension starting runtime dependencies...

## Get started with Azure Machine Learning

Before you start training and deploying machine learning models using VS Code, you need to create an [Azure Machine Learning service workspace](#) in the cloud to contain your models and resources. Learn how to create one and create your first experiment in that workspace.

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure Machine Learning sidebar appears.



2. Right-click your Azure subscription and select **Create Workspace**. A list appears. In the animated image, the subscription name is 'Free Trial' and the workspace is 'TeamWorkspace'.
3. Select an existing resource group from the list or create a new one using the wizard in the Command Palette.
4. In the field, type a unique and clear name for your new workspace. In the screenshots, the workspace is named 'TeamWorkspace'.
5. Hit enter and the new workspace is created. It appears in the tree below the subscription name.
6. Right-click on the Experiment node and choose **Create Experiment** from the context menu. Experiments keep track of your runs using Azure Machine Learning.
7. In the field, enter a name for your experiment. In the screenshots, the experiment is named 'MNIST'.
8. Hit enter and the new experiment is created. It appears in the tree below the workspace name.
9. You can right-click on an Experiment in a Workspace and select 'Set as Active Experiment'. The '**Active**' experiment is the experiment you are currently using and your open folder in VS Code will be linked to this experiment in the cloud. This folder should contain your local Python scripts.

Now each of your experiment runs with your experiment, so all of your key metrics will be stored in the experiment history and the models you train will get automatically uploaded to Azure Machine Learning

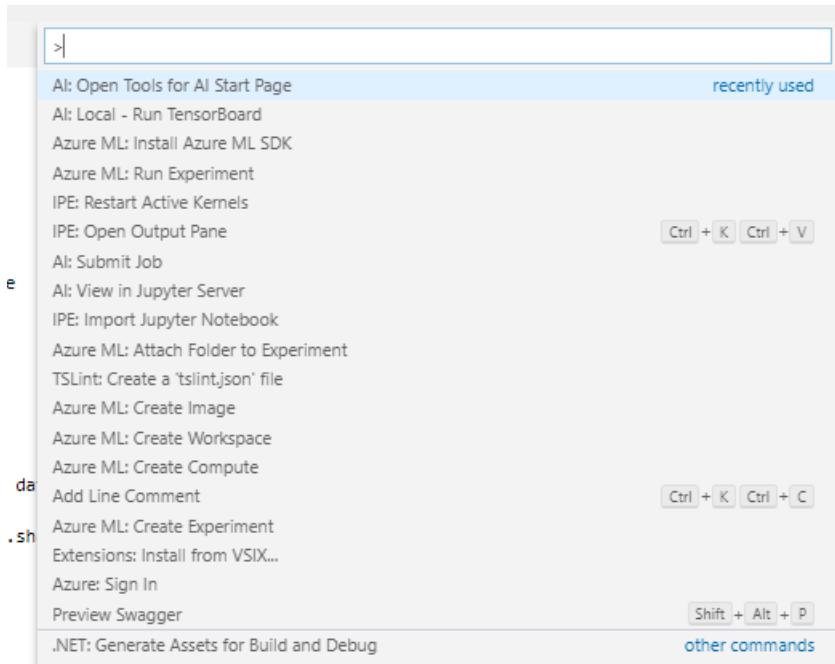
and stored with your experiment metrics and logs.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows an Azure Machine Learning workspace structure with items like Free Trial, TeamWorkspace, Experiments, Pipelines, Compute, Models, Images, and Deployments.
- Code Editor:** The active file is `train.py`, which contains Python code for training a TensorFlow model on the MNIST dataset. The code includes imports for gzip, struct, azureml, and tensorflow, along with placeholder definitions for inputs, hidden layers, and output, as well as a loss function and optimizer setup.
- Output Panel:** Shows the command line output: "Python 3.6.7 64-bit ('myenv3': conda) 0 ▲ 0 0 13 Azure: shwinne@microsoft.com python train.py".
- Status Bar:** Displays "Ln 80, Col 65 Spaces: 4 UTR-8 CRLF Python".

## Use keyboard shortcuts

Like most of VS Code, the Azure Machine Learning features in VS Code are accessible from the keyboard. The most important key combination to know is **Ctrl+Shift+P**, which brings up the Command Palette. From here, you have access to all of the functionality of VS Code, including keyboard shortcuts for the most common operations.



## Next steps

You can now use Visual Studio Code to work with Azure Machine Learning.

Learn how to [create compute targets, train, and deploy models in Visual Studio Code](#).

# Use Visual Studio Code to train and deploy machine learning models

2/5/2019 • 7 minutes to read

In this article, you will learn how to use the **Azure Machine Learning for Visual Studio Code** extension to train and deploy machine learning and deep learning models with Azure Machine Learning service in Visual Studio Code (VS Code).

Azure Machine Learning provides support for running experiments locally and on remote compute targets. For every experiment, you can keep track of multiple runs as often you need to iteratively try different techniques, hyperparameters, and more. You can use Azure Machine Learning to track custom metrics and experiment runs, enabling data science reproducibility and auditability.

And you can deploy these models for your testing and production needs.

## 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 service](#) today.
- Have the [Azure Machine Learning for VS Code](#) extension set up.
- Have the [Azure Machine Learning SDK for Python installed](#) with VS Code.

## Create and manage compute targets

With Azure Machine Learning for VS Code, you can prepare your data, train models, and deploy them both locally and on remote compute targets.

This extension supports several different remote compute targets for Azure Machine Learning. See the [full list of supported compute targets](#) for Azure Machine Learning.

### Create compute targets for Azure Machine Learning in VS Code

#### To create a compute target:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and Azure Machine Learning service workspace. In the animated image, the subscription name is 'Free Trial' and the workspace is 'TeamWorkspace'.
3. Under the workspace node, right-click the **Compute** node and choose **Create Compute**.
4. Choose the compute target type from the list.
5. In the Command Palette, select a Virtual Machine size.
6. In the Command Palette, enter a name for the compute target in the field.
7. Specify any advanced properties in the JSON config file that opens in a new tab. You can specify properties such as a maximum node count..
8. When you are done configuring your compute target, click **Submit** in the bottom-right corner of the screen.

Here is an example for creating an Azure Machine Learning Compute (AMLCompute):

```
File Edit Selection View Go Debug Terminal Help
AZURE MACHINE LEARNING
+ ⌂ train.py x
7
8 import gzip
9 import struct
10 import azureml
11 from azureml.core import Run, Workspace
12 from utils import prepare_data
13
14 # ## Download MNIST dataset
15 # In order to train on the MNIST dataset we will first need to download
16
17 X_train, X_test, y_train, y_test = prepare_data('mnist', './data')
18
19 training_set_size = X_train.shape[0]
20
21 n_inputs = 28 * 28
22 n_h1 = 300
23 n_h2 = 100
24 n_outputs = 10
25 learning_rate = 0.01
26 n_epochs = 10
27 batch_size = 50
28
29 with tf.name_scope('network'):
30     # construct the DNN
31     X = tf.placeholder(tf.float32, shape = (None, n_inputs), name = 'X')
32     y = tf.placeholder(tf.int64, shape = (None), name = 'y')
33     h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu, name = 'h1')
34     h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu, name = 'h2')
35     output = tf.layers.dense(h2, n_outputs, name = 'output')
36
37 with tf.name_scope('train'):
38     cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = output)
39     loss = tf.reduce_mean(cross_entropy, name = 'loss')
40     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
41     train_op = optimizer.minimize(loss)
42
PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL
verbose: Azure ML Experiment MNIST created
AI Output
```

### The 'run configuration' file

The VS Code extension will automatically create a local compute target and run configurations for your **local** and **docker** environments on your local computer. The run configuration files can be found under the associated compute target.

This is a snippet from the default local run configuration file. By default, `userManagedDependencies: True` so you must install all of your libraries/dependencies yourself and then local experiment runs will use your default Python environment as specified by the VS Code Python extension.

```
# user_managed_dependencies = True indicates that the environment will be user managed. False indicates that
# Azure Machine Learning service will manage the user environment.
userManagedDependencies: True
# The python interpreter path
interpreterPath: python
# Path to the conda dependencies file to use for this run. If a project
# contains multiple programs with different sets of dependencies, it may be
# convenient to manage those environments with separate files.
condaDependenciesFile: aml_config/conda_dependencies.yml
# Docker details
docker:
# Set True to perform this run inside a Docker container.
enabled: false
```

## Train and tune models

Use Azure Machine Learning for VS Code (Preview) to rapidly iterate on your code, step through and debug, and use your source code control solution of choice.

### To run your experiment locally with Azure Machine Learning:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and Azure Machine Learning service workspace.
3. Under the workspace node, expand the **Compute** node and right-click on the **Run Config** of compute you want to use.

#### 4. Select **Run Experiment**.

5. Select the script to run from the File Explorer.

6. Click **View Experiment Run** to see the integrated Azure Machine Learning portal to monitor your runs and see your trained models.

Here is an example for running an experiment locally:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the "AZURE MACHINE LEARNING" workspace with a "TeamWorkspace (Active)" folder containing "Experiments" (with "MNIST (Active)" selected), "Pipelines", "Compute" (with "local" and "mycluster" selected), "Models", "Images", and "Deployments".
- Code Editor:** The file "train.py" is open, displaying Python code for training a neural network on the MNIST dataset using TensorFlow. The code includes imports for numpy, argparse, os, tensorflow, sys, gzip, struct, and azureml. It defines constants for input dimensions, hidden layer sizes, output dimensions, learning rate, epochs, and batch size. It constructs a DNN with two hidden layers and an output layer. It also defines a "train" scope with cross-entropy loss and gradient descent optimization.
- Bottom Status Bar:** Shows "Python 3.6.7 64-bit ('myenv3': conda) 0 0 0 0 13 Azure: shwinne@microsoft.com python | train.py" and "Ln 1, Col 1 Space: 4 UTF-8 CRLF Python".

#### Use remote computes for experiments in VS Code

To use a remote compute target when training, you need to create a run configuration file. This file tells Azure Machine Learning not only where to run your experiment but also how to prepare the environment.

##### The conda dependencies file

By default, a new conda environment is created for you and your installation dependencies are managed. However, you must specify your dependencies and their versions in the `aml_config/conda_dependencies.yml` file.

This is a snippet from the default 'aml\_config/conda\_dependencies.yml'. For example, you can specify 'tensorflow=1.12.0' as seen below. If you do not specify the version of the dependency, then the latest version will be used.

You can add additional dependencies in the config file.

```

# The dependencies defined in this file will be automatically provisioned for runs with
userManagedDependencies=False.

name: project_environment
dependencies:
    # The python interpreter version.

    # Currently Azure Machine Learning service only supports 3.5.2 and later.

- python=3.6.2
- tensorflow=1.12.0

- pip:
    # Required packages for Azure Machine Learning service execution, history, and data preparation.

    - --index-url https://azuremlsdktestpypi.azureedge.net/sdk-release/Preview/E7501C02541B433786111FE8E140CAA1
    - --extra-index-url https://pypi.python.org/simple
    - azureml-defaults

```

## To run your experiment with Azure Machine Learning on a remote compute target:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and Azure Machine Learning service workspace.
3. Right-click on your python script in the editor window and select **AML: Run as Experiment in Azure**.
4. In the Command Palette, select the compute target.
5. In the Command Palette, enter the run configuration name in the field.
6. Edit the conda\_dependencies.yml file to specify the experiment's runtime dependencies, then click **Submit** in the bottom-right corner of the screen.
7. Click **View Experiment Run** to see the integrated Azure Machine Learning portal to monitor your runs and see your trained models.

## Here is an example for running an experiment on a remote compute target:

The screenshot shows the Visual Studio Code interface with the Azure Machine Learning sidebar open. The sidebar displays the Azure subscription, TeamWorkspace (Active), Experiments (with 'MNIST (Active)' selected), Pipelines, Compute (with 'local' and 'mycluster (Last Run)' listed), Models, Images, and Deployments. The main editor window contains the Python script 'train.py' which imports numpy, argparse, os, tensorflow, sys, gzip, struct, and azureml. It uses azureml.core to import Run and Workspace, and from utils import prepare\_data. The script then defines variables for dataset download, network architecture, training parameters, and batch size. It constructs a network with two hidden layers (h1 and h2) and an output layer, and performs cross-entropy loss calculation. The code is annotated with line numbers from 1 to 38. The status bar at the bottom shows the Python version (3.6.7 64-bit), the file path (c:\Users\shwinne\Desktop\mnist\mnist\aml\_config\conda\_dependencies.yml), and other development details.

```

File Edit Selection View Go Debug Terminal Help
train.py - mnist - Visual Studio Code
AZURE MACHINE LEARNING
+ ⌂ train.py ×
Azure Boot Camp (Quebec)
TeamWorkspace (Active)
Experiments
MNIST (Active)
Pipelines
Compute
local
mycluster (Last Run)
Models
Images
Deployments
train.py - mnist - Visual Studio Code
1  import numpy as np
2  import argparse
3  import os
4  import tensorflow as tf
5  import sys
6  import os
7
8  import gzip
9  import struct
10 import azureml
11 from azureml.core import Run, Workspace
12 from utils import prepare_data
13
14 # ## Download MNIST dataset
15 # In order to train on the MNIST dataset we will first need to download
16
17 X_train, X_test, y_train, y_test = prepare_data('mnist', './data')
18
19 training_set_size = X_train.shape[0]
20
21 n_inputs = 28 * 28
22 n_h1 = 300
23 n_h2 = 100
24 n_outputs = 10
25 learning_rate = 0.01
26 n_epochs = 10
27 batch_size = 50
28
29 with tf.name_scope('network'):
30     # construct the DNN
31     X = tf.placeholder(tf.float32, shape = (None, n_inputs), name = 'X')
32     y = tf.placeholder(tf.int64, shape = (None), name = 'y')
33     h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu, name = 'h1')
34     h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu, name = 'h2')
35     output = tf.layers.dense(h2, n_outputs, name = 'output')
36
37 with tf.name_scope('train'):
38     cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = output)
PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL
verbose: Editing c:\Users\shwinne\Desktop\mnist\mnist\aml_config\conda_dependencies.yml finished.
Al Output
Ln 22, Col 11  Spaces: 4  UTF-8  CRLF  Python
Python 3.6.7 64-bit ('myenv3': conda) 0 0 0 13  Azure: shwinne@microsoft.com  python | train.py

```

# Deploy and manage models

Azure Machine Learning enables deploying and managing your machine learning models in the cloud and on the edge.

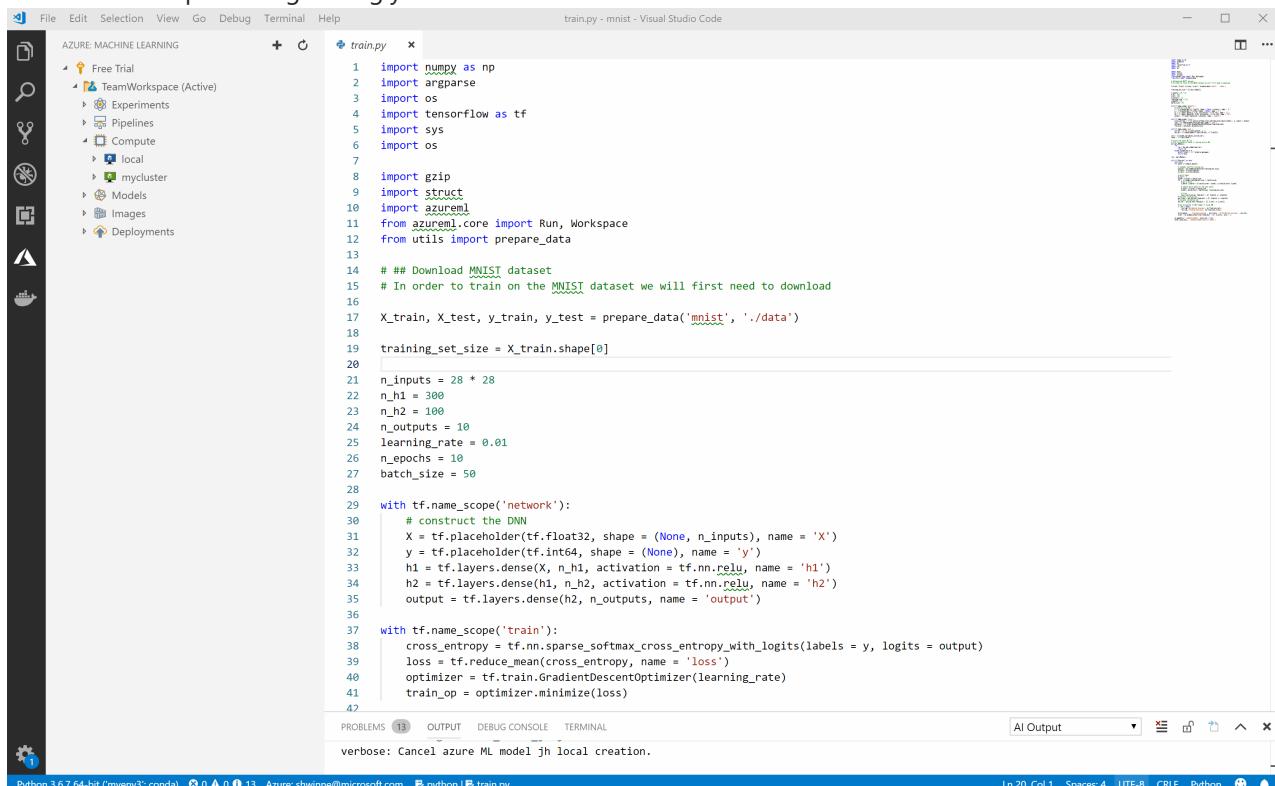
## Register your model to Azure Machine Learning from VS Code

Now that you have trained your model, you can register it in your workspace. Registered models can be tracked and deployed.

### To register your model:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and Azure Machine Learning service workspace.
3. Under the workspace node, right-click **Models** and choose **Register Model**.
4. In the Command Palette, enter a model name in the field.
5. From the list, choose whether you want to upload a **model file** (for single models) a **model folder** (for models with multiple files, such as Tensorflow).
6. Select your folder or file.
7. When you are done configuring your model properties, click **Submit** in the bottom-right corner of the screen.

Here is an example for registering your model to AML:



The screenshot shows the Visual Studio Code interface with the Azure Machine Learning sidebar open. The sidebar displays a tree view of Azure resources, including 'Free Trial', 'TeamWorkspace (Active)', 'Experiments', 'Pipelines', 'Compute' (with 'local' and 'mycluster' options), 'Models', 'Images', and 'Deployments'. The 'train.py' file is open in the editor, containing Python code for training a TensorFlow model on the MNIST dataset. The code includes imports for numpy, argparse, os, tensorflow, sys, gzip, struct, and azureml. It defines variables for input dimensions, hidden layer sizes, output size, learning rate, epochs, and batch size. It constructs a DNN with two hidden layers and an output layer. It also defines a 'train' scope with cross-entropy loss and gradient descent optimization. The code is annotated with comments explaining its purpose. The status bar at the bottom shows the file path 'train.py - mnist - Visual Studio Code', the Python version 'Python 3.6.7 64-bit (myenv3; conda)', and other development details.

```
File Edit Selection View Go Debug Terminal Help train.py - mnist - Visual Studio Code

AZURE MACHINE LEARNING
+ ⌂ Free Trial
  + TeamWorkspace (Active)
    + Experiments
    + Pipelines
    + Compute
      + local
        + mycluster
    + Models
    + Images
    + Deployments

train.py * train.py - mnist - Visual Studio Code

1 import numpy as np
2 import argparse
3 import os
4 import tensorflow as tf
5 import sys
6 import os
7
8 import gzip
9 import struct
10 import azureml
11 from azureml.core import Run, Workspace
12 from utils import prepare_data
13
14 # ## Download MNIST dataset
15 # In order to train on the MNIST dataset we will first need to download
16
17 X_train, X_test, y_train, y_test = prepare_data('mnist', './data')
18
19 training_set_size = X_train.shape[0]
20
21 n_inputs = 28 * 28
22 n_h1 = 300
23 n_h2 = 100
24 n_outputs = 10
25 learning_rate = 0.01
26 n_epochs = 10
27 batch_size = 50
28
29 with tf.name_scope('network'):
30     # construct the DNN
31     X = tf.placeholder(tf.float32, shape = (None, n_inputs), name = 'X')
32     y = tf.placeholder(tf.int64, shape = (None), name = 'y')
33     h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu, name = 'h1')
34     h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu, name = 'h2')
35     output = tf.layers.dense(h2, n_outputs, name = 'output')
36
37 with tf.name_scope('train'):
38     cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = output)
39     loss = tf.reduce_mean(cross_entropy, name = 'loss')
40     optimizer = tf.train.GradientDescentOptimizer(learning_rate)
41     train_op = optimizer.minimize(loss)
42

PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL
verbose: Cancel azure ML model jh local creation.
AI Output
Ln 20, Col 1 Spaces: 4 UTF-8 CRLF Python 🎉
```

## Deploy your service from VS Code

Using VS Code, you can deploy your web service to:

- Azure Container Instance (ACI): for testing
- Azure Kubernetes Service (AKS): for production

You do not need to create an ACI container to test in advance since they are created on the fly. However, AKS clusters do need to be configured in advance.

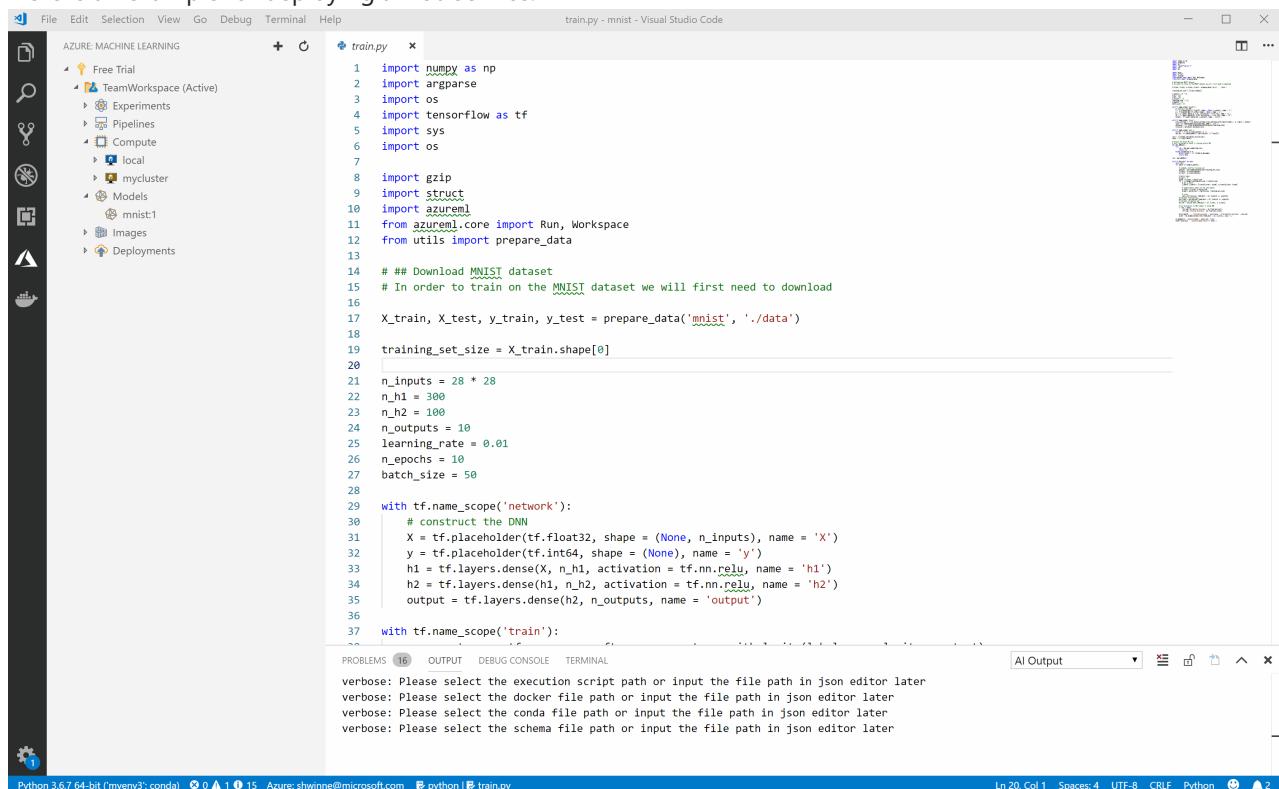
Learn more about [deployment with Azure Machine Learning](#) in general.

## To deploy a web service:

1. Click the Azure icon in the Visual Studio Code activity bar. The Azure Machine Learning sidebar appears.
2. In the tree view, expand your Azure subscription and your Azure Machine Learning service workspace.
3. Under the workspace node, expand the **Models** node.
4. Right-click the model you want to deploy and choose **Deploy Service from Registered Model** command from the context menu.
5. In the Command Palette, choose the compute target to which to deploy from the list.
6. In the Command Palette, enter a name for this service in the field.
7. In the Command Palette, press the Enter key on your keyboard to browse and select the script file.
8. In the Command Palette, press the Enter key on your keyboard to browse and select the conda dependency file.
9. When you are done configuring your service properties, click **Submit** in the bottom-right corner of the screen to deploy. In this service properties file, you can specify a local Docker file or a schema.json file that you may want to use.

The web service is now deployed.

Here is an example for deploying a web service:



A screenshot of Visual Studio Code showing the Azure Machine Learning sidebar on the left. The sidebar displays a tree view of Azure resources, including a 'TeamWorkspace (Active)' node with 'Experiments', 'Pipelines', 'Compute' (with 'local' and 'mycluster' options), 'Models' (with 'mnist1' selected), 'Images', and 'Deployments'. The main editor area shows a Python script named 'train.py'. The script imports numpy, argparse, os, tensorflow, sys, gzip, struct, and azureml. It uses the azureml.core Run and Workspace classes to handle data preparation. The code then defines a neural network architecture with two hidden layers (n\_h1=300, n\_h2=100) and one output layer (n\_outputs=10). The learning rate is set to 0.01, and training is performed over 10 epochs with a batch size of 50. The script concludes with a 'train' scope containing placeholder definitions for X and y, and dense layers h1, h2, and output. The bottom of the screen shows the terminal output, which is mostly empty except for some verbose messages about selecting execution paths and schema files. The status bar at the bottom indicates Python 3.6.7 64-bit ('myenv3:conda'), the current file is 'train.py', and the terminal shows 'Ln 20, Col 1'.

```
File Edit Selection View Go Debug Terminal Help train.py - mnist - Visual Studio Code

AZURE: MACHINE LEARNING + ⌂ train.py ×

1 import numpy as np
2 import argparse
3 import os
4 import tensorflow as tf
5 import sys
6 import os
7
8 import gzip
9 import struct
10 import azureml
11 from azureml.core import Run, Workspace
12 from utils import prepare_data
13
14 # ## Download MNIST dataset
15 # In order to train on the MNIST dataset we will first need to download
16
17 X_train, X_test, y_train, y_test = prepare_data('mnist', './data')
18
19 training_set_size = X_train.shape[0]
20
21 n_inputs = 28 * 28
22 n_h1 = 300
23 n_h2 = 100
24 n_outputs = 10
25 learning_rate = 0.01
26 n_epochs = 10
27 batch_size = 50
28
29 with tf.name_scope('network'):
30     # construct the DNN
31     X = tf.placeholder(tf.float32, shape = (None, n_inputs), name = 'X')
32     y = tf.placeholder(tf.int64, shape = (None), name = 'y')
33     h1 = tf.layers.dense(X, n_h1, activation = tf.nn.relu, name = 'h1')
34     h2 = tf.layers.dense(h1, n_h2, activation = tf.nn.relu, name = 'h2')
35     output = tf.layers.dense(h2, n_outputs, name = 'output')
36
37 with tf.name_scope('train'):

PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL
verbose: Please select the execution script path or input the file path in json editor later
verbose: Please select the docker file path or input the file path in json editor later
verbose: Please select the conda file path or input the file path in json editor later
verbose: Please select the schema file path or input the file path in json editor later

Python 3.6.7 64-bit ('myenv3:conda') 0 1 15 Azure: shwanne@microsoft.com python | train.py
Ln 20, Col 1 Spaces: 4 UFT-8 CRLF Python 2
```

## Next steps

For a walk-through of training with Machine Learning outside of VS Code, read [Tutorial: Train models with Azure Machine Learning](#).

For a walk-through of editing, running, and debugging code locally, see the [Python Hello World Tutorial](#)

# Load and read data with Azure Machine Learning

2/24/2019 • 7 minutes to read

In this article, you learn different methods of loading data using the Azure Machine Learning Data Prep SDK. To see reference documentation for the SDK, see the [overview](#). The SDK supports multiple data ingestion features, including:

- Load from many file types with parsing parameter inference (encoding, separator, headers)
- Type-conversion using inference during file loading
- Connection support for MS SQL Server and Azure Data Lake Storage

The following table shows a selection of functions used for loading data from common file types.

FILE TYPE	FUNCTION	REFERENCE LINK
Any	<code>auto_read_file()</code>	<a href="#">reference</a>
Text	<code>read_lines()</code>	<a href="#">reference</a>
CSV	<code>read_csv()</code>	<a href="#">reference</a>
Excel	<code>read_excel()</code>	<a href="#">reference</a>
Fixed-width	<code>read_fwf()</code>	<a href="#">reference</a>
JSON	<code>read_json()</code>	<a href="#">reference</a>

## Load data automatically

To load data automatically without specifying the file type, use the `auto_read_file()` function. The type of the file and the arguments required to read it are inferred automatically.

```
import azureml.dataprep as dprep

dataflow = dprep.auto_read_file(path='./data/any-file.txt')
```

This function is useful for automatically detecting file type, encoding, and other parsing arguments all from one convenient entry point. The function also automatically performs the following steps commonly performed when loading delimited data:

- Inferring and setting the delimiter
- Skipping empty records at the top of the file
- Inferring and setting the header row

Alternatively, if you know the file type ahead of time and want to explicitly control the way it is parsed, use the file-specific functions.

## Load text line data

To read simple text data into a dataflow, use the `read_lines()` without specifying optional parameters.

```
dataflow = dprep.read_lines(path='./data/text_lines.txt')
dataflow.head(5)
```

	LINE
0	Date    Minimum temperature    Maximum temperature
1	2015-07-1    -4.1    10.0
2	2015-07-2    -0.8    10.8

After the data is ingested, run the following code to convert the dataflow object into a Pandas dataframe.

```
pandas_df = dataflow.to_pandas_dataframe()
```

## Load CSV data

When reading delimited files, the underlying runtime can infer the parsing parameters (separator, encoding, whether to use headers, etc.). Run the following code to attempt to read a CSV file by specifying only its location.

```
dataflow =
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv?
st=2018-06-15T23%3A01%3A42Z&se=2019-06-16T23%3A01%3A00Z&sp=r&sv=2017-04-
17&sr=b&sig=ugQQCmeC2eBamm6ynM7wnI%2BI3TTDTM6z9RPKj4a%2FU6g%3D')
dataflow.head(5)
```

	STNAM	FIPST	LEAID	LEANM10	NCESSCH	MAM_MTH00N UMVALID_1011
0		stnam	fipst	leaid	leanm10	ncessch
1	ALABAMA	1	101710	Hale County	10171002158	
2	ALABAMA	1	101710	Hale County	10171002162	

To exclude lines during loading, define the `skip_rows` parameter. This parameter will skip loading rows descending in the CSV file (using a one-based index).

```
dataflow =
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv',
               skip_rows=1)
dataflow.head(5)
```

	STNAM	FIPST	LEAID	LEANM10	NCESSCH	MAM_MTH00N UMVALID_1011
0	ALABAMA	1	101710	Hale County	10171002158	29
1	ALABAMA	1	101710	Hale County	10171002162	40

Run the following code to display the column data types.

```
dataflow.head(1).dtypes
```

## Output:

```
stnam          object
fipst          object
leaid          object
leanm10        object
ncessch        object
schnam10       object
MAM_MTH00numvalid_1011   object
dtype: object
```

By default, the Azure Machine Learning Data Prep SDK does not change your data type. The data source you're reading from is a text file, so the SDK reads all values as strings. For this example, numeric columns should be parsed as numbers. Set the `inference_arguments` parameter to `InferenceArguments.current_culture()` to automatically infer and convert the column types during the file read.

```
dataflow =  
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv',  
                skip_rows=1,  
                inference_arguments=dprep.InferenceArguments.current_culture())  
dataflow.head(1).dtypes
```

## Output:

```
stnam          object
fipst         float64
leaid         float64
leanm10        object
ncessch       float64
schnam10      object
ALL_MTH0numvalid_1011 float64
dtype: object
```

Several of the columns were correctly detected as numeric and their type is set to `float64`.

## Use Excel data

The SDK includes a `read_excel()` function to load Excel files. By default the function will load the first sheet in the workbook. To define a specific sheet to load, define the `sheet_name` parameter with the string value of the sheet name.

```
dataflow = dprep.read_excel(path='./data/excel.xlsx', sheet_name='Sheet2')
dataflow.head(5)
```

	COLUMN1	COLUMN2	COLUMN3	COLUMN4	COLUMN5	COLUMN6	COLUMN7	COLUMN8
2	None	None	None	None	None	None	None	None
3	Rank	Title	Studio	Worldwide	Domestic / %	Column1	Overseas / %	Column2
4	1	Avatar	Fox	2788	760.5	0.273	2027.5	0.727

The output shows that the data in the second sheet had three empty rows before the headers. The `read_excel()` function contains optional parameters for skipping rows and using headers. Run the following code to skip the first three rows, and use the fourth row as the headers.

```
dataflow = dprep.read_excel(path='./data/excel.xlsx', sheet_name='Sheet2', use_column_headers=True, skip_rows=3)
```

	RANK	TITLE	STUDIO	WORLD WIDE	DOMESTIC %	COLUMN 1	OVERSEAS %	COLUMN 2	YEAR^
0	1	Avatar	Fox	2788	760.5	0.273	2027.5	0.727	2009^
1	2	Titanic	Par.	2186.8	658.7	0.301	1528.1	0.699	1997^

## Load fixed-width data files

To load fixed-width files, you specify a list of character offsets. The first column is always assumed to start at zero offset.

```
dataflow = dprep.read_fwf('./data/fixed_width_file.txt', offsets=[7, 13, 43, 46, 52, 58, 65, 73])
dataflow.head(5)
```

	010000	99999	BOGUS NORWAY	NO	NO_1	ENRS	COLUMN 7	COLUMN 8	COLUMN 9
0	010003	99999	BOGUS NORWAY	NO	NO	ENSO			
1	010010	99999	JAN MAYEN	NO	JN	ENJA	+70933	-008667	+00090

To avoid header detection and parse the correct data, pass `PromoteHeadersMode.NONE` to the `header` parameter.

COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
1	2	3	4	5	6	7	8	9

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	010000	99999	BOGUS NORWA Y	NO	NO_1	ENRS	Column 7	Column 8	Column 9
1	010003	99999	BOGUS NORWA Y	NO	NO	ENSO			

# Load SQL data

The SDK can also load data from a SQL source. Currently, only Microsoft SQL Server is supported. To read data from a SQL server, create a `MSSQLDataSource` object that contains the connection parameters. The `password` parameter of `MSSQLDataSource` accepts a `Secret` object. You can build a secret object in two ways:

- Register the secret and its value with the execution engine.
  - Create the secret with only an `id` (if the secret value is already registered in the execution environment) using  
`dprep.create_secret("[SECRET-ID]")`.

```
secret = dprep.register_secret(value="[SECRET-PASSWORD]", id="[SECRET-ID]")

ds = dprep.MSSQLDataSource(server_name="[SERVER-NAME]",
                           database_name="[DATABASE-NAME]",
                           user_name="[DATABASE-USERNAME]",
                           password=secret)
```

After you create a data source object, you can proceed to read data from query output.

```
dataflow = dprep.read_sql(ds, "SELECT top 100 * FROM [SalesLT].[Product]")
dataflow.head(5)
```

PR	O	PR	D	UC	ST				PR	OD	TC	OD	PR	SE		DI	TH	TH	U
O	TN	AN	AN	DA	LIS				UC	AT	UC	ST	LL		SC	ON	M	BN	M
D	U	DA	DA	TP				EG	TM	AR	LLE	NU		TI	BN	OT	PH	AIL	PH
UC	M	CO	RD	TP				OR	OD	TD	ND	ED		AIL	OF	OF	RO	IFI	OD
TI	NA	BE	LO	CO	RI	SIZ	WE	YI	ELI	AT	DA	DA	DA	DA	OT	NA	GU	W	ED
D	ME	R	R	ST	CE	E	HT	D	D	E	TE	TE	TE	TE	O	ME	ID	TE	DA

		PR O D UC TI D	PR O D UC TN U M M BE R	PR O D UC TN U M M BE R	ST AN DA	LIS		WE IG HT	PR O D UC TC AT EG	PR O D UC TM OD	SE LL ST AR	DI SC ON TI NU ED	TH U M BN AIL	TH U M BN AIL	TH U M BN AIL	RO W GU ID	M OD IFI ED DA TE	
		NA ME	CO LO R	RD CO ST	TP RI CE	SIZ E		WE IG HT	YI D	ELI D	TD AT E	DA TE	DA TE	DA TE	DA TE	DA TE	DA TE	
0	68	HL	FR	Bl	10	14	58	10	18	6	20	No	No	b'	no	43	20	
0	Ro ad	-	ac	59	31			16		02	-	ne	ne	Gl	_i	dd	08	
	R9	R9	k	.3	.5			.0			06			F8	m	68	-	
	Fr	2B		10	0			4			-			9a	ag	d6	03	
	a	-		0							01			P\	e\_	-	-	
	m			58							00			x0	av	14	11	
	e	-									:0			01	ail	a4		
	Bl										0:			\x	abl	-		
	ac										00			00	e\_	46		
	k,										+0			\xf	sm	1f-		
	58										0:			7\	all.	90		
											00			x0	gif	69		
														0\	-			
														x0	55			
														0\	30			
														x0	9d			
														0\	90			
														x0	ea			
														0\	7e			
														x0				
														0\				
														x8				
														0...				
1	70	HL	FR	Re	10	14	58	10	18	6	20	No	No	b'	no	95	20	
6	Ro ad	-	rd	59	31			16		02	-	ne	ne	Gl	_i	40	08	
	R9	R9		.3	.5			.0			06			F8	m	ff1	-	
	Fr	2R		10	0			4			-			9a	ag	7-	03	
	a	-		0							01			P\	e\_	27	-	
	m	-		58							00			x0	av	12	11	
	e	-									:0			01	ail	-		
	Re										0:			\x	abl	4c		
	d										00			00	e\_	90		
	58										+0			\xf	sm	-		
											0:			7\	all.	a3		
											00			x0	gif	d1		
														0\	-			
														x0	8c			
														0\	e5			
														x0	56			
														0\	8b			
														x0	24			
														0\	62			
														x0				
														0\				
														x8				
														0...				

		PR O D UC TI D	NA ME	PR O D UC TN U M BE R	ST AN DA CO LO R	LIS RD CO ST	WE IG HT	PR OD UC TC AT EG OR YI D	PR OD UC TM AR OD ELI D	SE LL ST AT TD AT E	DI SC ON TI NU ED DA TE	TH U M BN AIL PH OT O	TH U M BN AIL PH OT OF ILE NA ME	M OD IFI ED DA TE	
2	70 7	Sp or t- 10 0 He Im et, Re d	HL - U 50 9- R	Re d	13 .0 86 3	34 .9 9	No ne	No ne	35	33	20 05 - 07 - 01 00 :0 0: 00 +0 0: 00	No ne	No ne	b' Gl _i F8 m 9a ag P\ e_ x0 x0 01 \x 00 \\xf 7\ x0 0\ x0 0\ x0 0\ x0 0\ x8 0...	no 1e f4 - 1a 03 - - c0 11 - 8a abl - 4ff sm 6- all. 8a da - bd e5 8b 64 a7 12

## Use Azure Data Lake Storage

There are two ways the SDK can acquire the necessary OAuth token to access Azure Data Lake Storage:

- Retrieve the access token from a recent session of the user's Azure CLI login
- Use a service principal (SP) and a certificate as secret

### Use an access token from a recent Azure CLI session

On your local machine, run the following command.

```
az login
az account show --query tenantId
dataflow = read_csv(path = DataLakeDataSource(path='adl://dpreptestfiles.azuredatalakestore.net/farmers-markets.csv', tenant='microsoft.onmicrosoft.com')) head = dataflow.head(5) head
```

#### NOTE

If your user account is a member of more than one Azure tenant, you need to specify the tenant in the AAD URL hostname form.

### Create a service principal with the Azure CLI

Use the Azure CLI to create a service principal and the corresponding certificate. This particular service principal is configured with the `reader` role, with its scope reduced to only the Azure Data Lake Storage account 'dpreptestfiles'.

```

az account set --subscription "Data Wrangling development"
az ad sp create-for-rbac -n "SP-ADLS-dpreptestfiles" --create-cert --role reader --scopes
/subscriptions/35f16a99-532a-4a47-9e93-
00305f6c40f2/resourceGroups/dpreptestfiles/providers/Microsoft.DataLakeStore/accounts/dpreptestfiles

```

This command emits the `appId` and the path to the certificate file (usually in the home folder). The .crt file contains both the public cert and the private key in PEM format.

```
openssl x509 -in adls-dpreptestfiles.crt -noout -fingerprint
```

To configure the ACL for the Azure Data Lake Storage file system, use the `objectId` of the user. In this example, the service principal is used.

```
az ad sp show --id "8dd38f34-1fcf-4ff9-accd-7cd60b757174" --query objectId
```

To configure `Read` and `Execute` access for the Azure Data Lake Storage file system, you configure the ACL for folders and files individually. This is due to the fact that the underlying HDFS ACL model doesn't support inheritance.

```

az dls fs access set-entry --account dpreptestfiles --acl-spec "user:e37b9b1f-6a5e-4bee-9def-402b956f4e6f:r-x"
--path /
az dls fs access set-entry --account dpreptestfiles --acl-spec "user:e37b9b1f-6a5e-4bee-9def-402b956f4e6f:r--"
--path /farmers-markets.csv

```

```

certThumbprint = 'C2:08:9D:9E:D1:74:FC:EB:E9:7E:63:96:37:1C:13:88:5E:B9:2C:84'
certificate = ''
with open('./data/adls-dpreptestfiles.crt', 'rt', encoding='utf-8') as crtFile:
    certificate = crtFile.read()

servicePrincipalAppId = "8dd38f34-1fcf-4ff9-accd-7cd60b757174"

```

## Acquire an OAuth access token

Use the `adal` package (`pip install adal`) to create an authentication context on the MSFT tenant and acquire an OAuth access token. For ADLS, the resource in the token request must be for '<https://datalake.azure.net>', which is different from most other Azure resources.

```

import adal
from azureml.dataprep.api.datasources import DataLakeDataSource

ctx = adal.AuthenticationContext('https://login.microsoftonline.com/microsoft.onmicrosoft.com')
token = ctx.acquire_token_with_client_certificate('https://datalake.azure.net/', servicePrincipalAppId,
certificate, certThumbprint)
dataflow = dprep.read_csv(path = DataLakeDataSource(path='adl://dpreptestfiles.azuredatalakestore.net/farmers-
markets.csv', accessToken=token['accessToken']))
dataflow.to_pandas_dataframe().head()

```

	FMID	MARKETNAME	WEBSITE	STREET	CITY	COUNTY
0	1012063	Caledonia Farmers Market Association - Danville	<a href="https://sites.google.com/site/caledoniafarmers...">https://sites.google.com/site/caledoniafarmers...</a>		Danville	Caledonia

	<b>FMID</b>	<b>MARKETNAME</b>	<b>WEBSITE</b>	<b>STREET</b>	<b>CITY</b>	<b>COUNTY</b>
1	1011871	Stearns Homestead Farmers' Market	<a href="http://Stearns homestead.co m">http://Stearns homestead.co m</a>	6975 Ridge Road	Parma	Cuyahoga
2	1011878	100 Mile Market	<a href="http://www.pfc markets.com">http://www.pfc markets.com</a>	507 Harrison St	Kalamazoo	Kalamazoo
3	1009364	106 S. Main Street Farmers Market	<a href="http://thetow nofsixmile.wor dpress.com/">http://thetow nofsixmile.wor dpress.com/</a>	106 S. Main Street	Six Mile	
4	1010691	10th Street Community Farmers Market	<a href="https://agrimis souri.com/">https://agrimis souri.com/...</a>	10th Street and Poplar	Lamar	Barton

## Next steps

- See the SDK [overview](#) for design patterns and usage examples
- See the Azure Machine Learning Data Prep SDK [tutorial](#) for an example of solving a specific scenario

# Transform data with the Azure Machine Learning Data Prep SDK

2/26/2019 • 13 minutes to read

In this article, you learn different methods of loading data using the Azure Machine Learning Data Prep SDK. The SDK offers functions that make it simple to add columns, filter out unwanted rows or columns, and impute missing values. To see reference documentation for the SDK, see the [overview](#).

This how-to shows examples for the following tasks:

- Add column using an expression
  - Impute missing values
  - Derive column by example
  - Filtering
  - Custom Python transforms

## Add column using an expression

The Azure Machine Learning Data Prep SDK includes `substring` expressions you can use to calculate a value from existing columns, and then put that value in a new column. In this example, you load data and try to add columns to that input data.

```
import azureml.dataprep as dprep

# loading data
dataflow = dprep.read_csv(path=r'data\crime0-10.csv')
dataflow.head(3)
```

ID	CASE NUMBER	DATE	BLOCK	LOCATION	PRIMARY TYPE	DESCRIPTION	DOMESTIC	WARD	COMMUNITY AREA	X COORDINATE	Y COORDINATE	YEAR	UPDATED ON	LATITUDE	LONGITUDE	LOCATION
1	1234567890	2023-01-01	100	123 Main St, Chicago, IL 60601	Domestic	Arrest	Domestic	1	123 Main St, Chicago, IL 60601	41.8819	-87.6298	2023-01-01	2023-01-01 12:00:00	41.8819	-87.6298	123 Main St, Chicago, IL 60601

ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location	Domestic	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Update Date	Latitude	Longitude	Location
0	100134200949007	HY7/0X5N07	05/0XN07	08/0N07	THEFT AND UNDER	\$500000	STREET	false	10	0619230	1192315	2015	07/11/2015	12:42PM	41.877304	-87.649466	Wicker Park

ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location	Domestic	Community Area	Community Code	X Coordinate	Y Coordinate	Year	Update Date	Latitude	Longitude	Location
											Address	Block	Code	Year	Month	Day	Hour
1	1013297765	HY / 05/2023	01/22/2023	04X0W0	BATTERY	SIMPLE	STREET	false	true	49B	1867370	1942771	2015270	07/2022	42.700000	-87.598889	(42.700000,-87.598889)

ID	Case Number	Location										Community		Coordinate		Year		Update On		Latitude		Longitude	
		Date	Block	Unit Type	Primary Description	Secondary Description	Domestic	Arrest	...	Ward	Community Area	FBI Code	Coordinate X	Coordinate Y	...	...	...	...	...	...	...	...	...
2	101420092275203	H / 5 / 22 / 25 / 20 : 00 PM	012860	BATTERRY FRONTAGE	DOME TREESTORY	S T E E T	fals e	true	...	9	53	08B	0	0	2015	07/12/2020	15/12/2020	15/12/2020	07/12/2020	07/12/2020	07/12/2020	07/12/2020	

Use the `substring(start, length)` expression to extract the prefix from the Case Number column and put that string in a new column, `Case Category`. Passing the `substring_expression` variable to the `expression` parameter creates a new calculated column that executes the expression on each record.

```
substring_expression = dprep.col('Case Number').substring(0, 2)
case_category = dataflow.add_column(new_column_name='Case Category',
                                     prior_column='Case Number',
                                     expression=substring_expression)
case_category.head(3)
```

ID	Case Number	Category	Date	Block	IUCR	Primary Type	Description	Location	Address	...	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Update Edition	Latitude	Longitude	Location
Residential Description	Area	Ward																	
0	1001349007	HY	07/05/2019	05020XNNDUN	0820FTANDE	\$THEEET	STREET	False	False	...	41	10	06	1192323150	2015	07/12/2022	12:42PM	41:49:00PM	41:49:00PM
1	1013297765	HY	07/05/2019	1105XW20MOR	160TERY15RS	SIMPLET	STREET	False	True	...	49	18B	119467370	1946271	2015	07/12/2022	12:42PM	42:00PM	-8:00PM

Use the `substring(start)` expression to extract only the number from the Case Number column and create a new column. Convert it to a numeric data type using the `to_number()` function, and pass the string column name as a parameter.

```
substring_expression2 = dprep.col('Case Number').substring(2)
case_id = dataflow.add_column(new_column_name='Case Id',
                               prior_column='Case Number',
                               expression=substring_expression2)
case_id = case_id.to_number('Case Id')
```

# Impute missing values

The SDK can impute missing values in specified columns. In this example, you load latitude and longitude values and then try to impute missing values in the input data.

```

import azureml.dataprep as dprep

# loading input data
df = dprep.read_csv(r'data\crime0-10.csv')
df = df.keep_columns(['ID', 'Arrest', 'Latitude', 'Longitude'])
df = df.to_number(['Latitude', 'Longitude'])
df.head(3)

```

	ID	ARREST	LATITUDE	LONGITUDE
0	10140490	false	41.973309	-87.800175
1	10139776	false	42.008124	-87.659550
2	10140270	false	NaN	NaN

The third record is missing latitude and longitude values. To impute those missing values, you use

`ImputeMissingValuesBuilder` to learn a fixed expression. It can impute the columns with either a calculated `MIN`, `MAX`, `MEAN` value, or a `CUSTOM` value. When `group_by_columns` is specified, missing values will be imputed by group with `MIN`, `MAX`, and `MEAN` calculated per group.

Check the `MEAN` value of the latitude column using the `summarize()` function. This function accepts an array of columns in the `group_by_columns` parameter to specify the aggregation level. The `summary_columns` parameter accepts a `SummaryColumnsValue` call. This function call specifies the current column name, the new calculated field name, and the `SummaryFunction` to perform.

```

df_mean = df.summarize(group_by_columns=['Arrest'],
                       summary_columns=[dprep.SummaryColumnsValue(column_id='Latitude',
                                                               summary_column_name='Latitude_MEAN',
                                                               summary_function=dprep.SummaryFunction.MEAN)])
df_mean = df_mean.filter(dprep.col('Arrest') == 'false')
df_mean.head(1)

```

	ARREST	LATITUDE_MEAN
0	false	41.878961

The `MEAN` value of latitudes looks accurate, use the `ImputeColumnArguments` function to impute it. This function accepts a `column_id` string, and a `ReplaceValueFunction` to specify the impute type. For the missing longitude value, impute it with 42 based on external knowledge.

Impute steps can be chained together into a `ImputeMissingValuesBuilder` object, using the builder function `impute_missing_values()`. The `impute_columns` parameter accepts an array of `ImputeColumnArguments` objects. Call the `learn()` function to store the impute steps, and then apply to a dataflow object using `to_dataflow()`.

```

# impute with MEAN
impute_mean = dprep.ImputeColumnArguments(column_id='Latitude',
                                             impute_function=dprep.ReplaceValueFunction.MEAN)
# impute with custom value 42
impute_custom = dprep.ImputeColumnArguments(column_id='Longitude',
                                              custom_impute_value=42)
# get instance of ImputeMissingValuesBuilder
impute_builder = df.builders.impute_missing_values(impute_columns=[impute_mean, impute_custom],
                                                    group_by_columns=['Arrest'])

impute_builder.learn()
df_imputed = impute_builder.to_dataflow()
df_imputed.head(3)

```

	ID	ARREST	LATITUDE	LONGITUDE
0	10140490	false	41.973309	-87.800175
1	10139776	false	42.008124	-87.659550
2	10140270	false	41.878961	42.000000

As shown in the result above, the missing latitude was imputed with the `MEAN` value of `Arrest=='false'` group. The missing longitude was imputed with 42.

```

imputed_longitude = df_imputed.to_pandas_dataframe()['Longitude'][2]
assert imputed_longitude == 42

```

## Derive column by example

One of the more advanced tools in the Azure Machine Learning Data Prep SDK is the ability to derive columns using examples of desired results. This lets you give the SDK an example so it can generate code to achieve the intended transformation.

```

import azureml.dataprep as dprep
dataflow = dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/BostonWeather.csv')
dataflow.head(4)

```

	DATE	REPORTTPYE	HOURLYDRYBULB TEMPF	HOURLYRELATIVE HUMIDITY	HOURLYWINDSPEE D
0	1/1/2015 0:54	FM-15	22	50	10
1	1/1/2015 1:00	FM-12	22	50	10
2	1/1/2015 1:54	FM-15	22	50	10
3	1/1/2015 2:54	FM-15	22	50	11

Assume that you need to join this file with a dataset where date and time are in a format 'Mar 10, 2018 | 2AM-4AM'.

```
builder = dataflow.builders.derive_column_by_example(source_columns=['DATE'], new_column_name='date_timerange')
builder.add_example(source_data=df.iloc[1], example_value='Jan 1, 2015 12AM-2AM')
builder.preview(count=5)
```

	DATE	DATE_TIMERANGE
0	1/1/2015 0:54	Jan 1, 2015 12AM-2AM
1	1/1/2015 1:00	Jan 1, 2015 12AM-2AM
2	1/1/2015 1:54	Jan 1, 2015 12AM-2AM
3	1/1/2015 2:54	Jan 1, 2015 2AM-4AM
4	1/1/2015 3:54	Jan 1, 2015 2AM-4AM

The code above first creates a builder for the derived column. You provide an array of source columns to consider (`DATE`), and a name for the new column to be added. As the first example, you pass in the second row (index 1) and give an expected value for the derived column.

Finally, you call `builder.preview(skip=30, count=5)` and can see the derived column next to the source column. The format seems correct, but you only see values for the same date "Jan 1, 2015".

Now, pass in the number of rows you want to `skip` from the top to see rows further down.

#### NOTE

The `preview()` function skips rows but does not re-number the output index. In the example below, the index 0 in the table corresponds to index 30 in the dataflow.

```
builder.preview(skip=30, count=5)
```

	DATE	DATE_TIMERANGE
0	1/1/2015 22:54	Jan 1, 2015 10PM-12AM
1	1/1/2015 23:54	Jan 1, 2015 10PM-12AM
2	1/1/2015 23:59	Jan 1, 2015 10PM-12AM
3	1/2/2015 0:54	Feb 1, 2015 12AM-2AM
4	1/2/2015 1:00	Feb 1, 2015 12AM-2AM

Here you see an issue with the generated program. Based solely on the one example you provided above, the derive program chose to parse the date as "Day/Month/Year", which is not what you want in this case. To fix this issue, target a specific record index and provide another example using the `add_example()` function on the `builder` variable.

```
builder.add_example(source_data=df.iloc[3], example_value='Jan 2, 2015 12AM-2AM')
builder.preview(skip=30, count=5)
```

	<b>DATE</b>	<b>DATE_TIMERANGE</b>
0	1/1/2015 22:54	Jan 1, 2015 10PM-12AM
1	1/1/2015 23:54	Jan 1, 2015 10PM-12AM
2	1/1/2015 23:59	Jan 1, 2015 10PM-12AM
3	1/2/2015 0:54	Jan 2, 2015 12AM-2AM
4	1/2/2015 1:00	Jan 2, 2015 12AM-2AM

Now rows correctly handle '1/2/2015' as 'Jan 2, 2015', but if you look beyond index 76 of the derived column, you see that values at the end have nothing in derived column.

```
builder.preview(skip=75, count=5)
```

	<b>DATE</b>	<b>DATE_TIMERANGE</b>
0	1/3/2015 7:00	Jan 3, 2015 6AM-8AM
1	1/3/2015 7:54	Jan 3, 2015 6AM-8AM
2	1/29/2015 6:54	None
3	1/29/2015 7:00	None
4	1/29/2015 7:54	None

```
builder.add_example(source_data=df.iloc[77], example_value='Jan 29, 2015 6AM-8AM')
builder.preview(skip=75, count=5)
```

	<b>DATE</b>	<b>DATE_TIMERANGE</b>
0	1/3/2015 7:00	Jan 3, 2015 6AM-8AM
1	1/3/2015 7:54	Jan 3, 2015 6AM-8AM
2	1/29/2015 6:54	Jan 29, 2015 6AM-8AM
3	1/29/2015 7:00	Jan 29, 2015 6AM-8AM
4	1/29/2015 7:54	Jan 29, 2015 6AM-8AM

To see a list of current example derivations call `list_examples()` on the builder object.

```
examples = builder.list_examples()
```

	DATE	EXAMPLE	EXAMPLE_ID
0	1/1/2015 1:00	Jan 1, 2015 12AM-2AM	-1
1	1/2/2015 0:54	Jan 2, 2015 12AM-2AM	-2
2	1/29/2015 20:54	Jan 29, 2015 8PM-10PM	-3

In certain cases if you want to delete examples that are incorrect, you can pass in either `example_row` from the pandas DataFrame, or `example_id` value. For example, if you run `builder.delete_example(example_id=-1)`, it deletes the first transformation example.

Call `to_dataflow()` on the builder, which returns a data flow with the desired derived columns added.

```
dataflow = builder.to_dataflow()
df = dataflow.to_pandas_dataframe()
```

## Filtering

The SDK includes the methods `Dataflow.drop_columns()` and `Dataflow.filter()` to let you filter out columns or rows.

### Initial setup

```
import azureml.dataprep as dprep
from datetime import datetime
dataflow = dprep.read_csv(path='https://dprepdata.blob.core.windows.net/demo/green-small/*')
dataflow.head(5)
```

	LPEP _PIC KUP_ DATE TIME	LPEP _DR OPO FF_D ATET IME	STO RE_A ND_F WD_ FLAG	RATE CODE ID	PICK UP_L ONGI TUDE	PICK UP_L ATIT UDE	DRO POFF _LON GITU DE	DRO POFF _LATI TUDE	PASS ENGE R_CO UNT	TRIP_ DIST ANCE	TIP_A MOU NT	TOLL S_AM OUN T	TOTA L_AM OUN T
0	None	None	None	None	None	None	None	None	None	None	None	None	None
1	2013-08-01 08:14:37	2013-08-01 09:09:06	N	1	0	0	0	0	1	.00	0	0	21.25
2	2013-08-01 09:13:00	2013-08-01 11:38:00	N	1	0	0	0	0	2	.00	0	0	75

	LPEP_DATE TIME	LPEP_DR OPO_FF_D ATETIME	STO RE_A ND_F WD_FLAG	RATE CODE ID	PICK UP_L ONGI TUDE	PICK UP_L ATIT UDE	DRO POFF _LON GITUD E	DRO POFF _LATI TUDE	PASS ENGE R_CO UNT	TRIP_ DIST ANCE	TIP_A MOU NT	TOLL S_AM OUN T	TOTAL L_AM OUN T
3	2013-08-01 09:48:00	2013-08-01 09:49:00	N	5	0	0	0	0	1	.00	0	1	2.1
4	2013-08-01 10:38:35	2013-08-01 10:38:51	N	1	0	0	0	0	1	.00	0	0	3.25

## Filtering columns

To filter columns, use `Dataflow.drop_columns()`. This method takes a list of columns to drop or a more complex argument called `ColumnSelector`.

### Filtering columns with list of strings

In this example, `drop_columns` takes a list of strings. Each string should exactly match the desired column to drop.

```
dataflow = dataflow.drop_columns(['Store_and_fwd_flag', 'RateCodeID'])
dataflow.head(2)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PICKUP_P_LONGITUDE	PICKUP_LATITUDE	DROP_OFF_LONGITUDE	DROP_OFF_LATITUDE	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	None	None	None	None	None	None	None	None	None	None	None
1	2013-08-01 08:14:37	2013-08-01 09:09:06	0	0	0	0	1	.00	0	0	21.25

### Filtering columns with regex

Alternatively, use the `ColumnSelector` expression to drop columns that match a regex expression. In this example, you drop all the columns that match the expression `Column*|.*longitude|.*latitude`.

```
dataflow = dataflow.drop_columns(dprep.ColumnSelector('Column*|.*longitude|.*latitude', True, True))
dataflow.head(2)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	None	None	None	None	None	None	None
1	2013-08-01 08:14:37	2013-08-01 09:09:06	1	.00	0	0	21.25

# Filtering rows

To filter rows, use `DataFlow.filter()`. This method takes an Azure Machine Learning Data Prep SDK expression as an argument, and returns a new data flow with the rows that the expression evaluates as True. Expressions are built using expression builders (`col`, `f_not`, `f_and`, `f_or`) and regular operators (`>`, `<`, `>=`, `<=`, `==`, `!=`).

## Filtering rows with simple Expressions

Use the expression builder `col`, specify the column name as a string argument `col('column_name')`. Use this expression in combination with one of the following standard operators `>`, `<`, `>=`, `<=`, `==`, `!=` to build an expression such as `col('Tip_amount') > 0`. Finally, pass the built expression into the `Dataflow.filter` function.

In this example, `dataflow.filter(col('Tip_amount') > 0)` returns a new data flow with the rows in which the value of `Tip_amount` is greater than 0.

### NOTE

`Tip_amount` is first converted to numeric, which allows us to build an expression comparing it against other numeric values.

```
dataflow = dataflow.to_number(['Tip_amount'])
dataflow = dataflow.filter(dprep.col('Tip_amount') > 0)
dataflow.head(2)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	2013-08-01 19:33:28	2013-08-01 19:35:21	5	.00	0.08	0	4.58
1	2013-08-05 13:16:38	2013-08-05 13:18:24	1	.00	0.30	0	3.8

## Filtering rows with complex expressions

To filter using complex expressions, combine one or more simple expressions with the expression builders `f_not`, `f_and`, or `f_or`.

In this example, `Dataflow.filter()` returns a new data flow with the rows where `'Passenger_count'` is less than 5 and `'Tolls_amount'` is greater than 0.

```
dataflow = dataflow.to_number(['Passenger_count', 'Tolls_amount'])
dataflow = dataflow.filter(dprep.f_and(dprep.col('Passenger_count') < 5, dprep.col('Tolls_amount') > 0))
dataflow.head(2)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	2013-08-08 12:16:00	2013-08-08 12:16:00	1.0	.00	2.25	5.00	19.75
1	2013-08-12 14:43:53	2013-08-12 15:04:50	1.0	5.28	6.46	5.33	32.29

It is also possible to filter rows combining more than one expression builder to create a nested expression.

#### NOTE

`lpep_pickup_datetime` and `Lpep_dropoff_datetime` are first converted to datetime, which allows us to build an expression comparing it against other datetime values.

```
dataflow = dataflow.to_datetime(['lpep_pickup_datetime', 'Lpep_dropoff_datetime'], ['%Y-%m-%d %H:%M:%S'])
dataflow = dataflow.to_number(['Total_amount', 'Trip_distance'])
mid_2013 = datetime(2013,7,1)
dataflow = dataflow.filter(
    dprep.f_and(
        dprep.f_or(
            dprep.col('lpep_pickup_datetime') > mid_2013,
            dprep.col('Lpep_dropoff_datetime') > mid_2013),
        dprep.f_and(
            dprep.col('Total_amount') > 40,
            dprep.col('Trip_distance') < 10)))
dataflow.head(2)
```

	LPEP_PICKUP_DATETIME	LPEP_DROPOFF_DATETIME	PASSENGER_COUNT	TRIP_DISTANCE	TIP_AMOUNT	TOLLS_AMOUNT	TOTAL_AMOUNT
0	2013-08-13 06:11:06+00:00	2013-08-13 06:30:28+00:00	1.0	9.57	7.47	5.33	44.80
1	2013-08-23 12:28:20+00:00	2013-08-23 12:50:28+00:00	2.0	8.22	8.08	5.33	40.41

## Custom Python transforms

There will always be scenarios when the easiest option for making a transformation is writing your own script. The SDK provides three extension points that you can use for custom Python scripts.

- New script column
- New script filter
- Transform partition

Each of the extensions is supported in both the scale-up and scale-out runtime. A key advantage of using these extension points is that you don't need to pull all of the data in order to create a data frame. Your custom Python code will run just like other transforms, at scale, by partition, and typically in parallel.

### Initial data preparation

Start by loading some data from Azure Blob.

```
import azureml.dataprep as dprep
col = dprep.col

df =
dprep.read_csv(path='https://dpreptestfiles.blob.core.windows.net/testfiles/read_csv_duplicate_headers.csv',
skip_rows=1)
df.head(2)
```

	<b>STNAM</b>	<b>FIPST</b>	<b>LEAID</b>	<b>LEANM10</b>	<b>NCESSCH</b>	<b>MAM_MTH00NUMVALID_1011</b>
0	ALABAMA	1	101710	Hale County	10171002158	
1	ALABAMA	1	101710	Hale County	10171002162	

Trim down the data set and do some basic transforms including removing columns, replacing values and converting types.

```
df = df.keep_columns(['stnam', 'leanm10', 'ncessch', 'MAM_MTH00numvalid_1011'])
df = df.replace_na(columns=['leanm10', 'MAM_MTH00numvalid_1011'], custom_na_list='.')
df = df.to_number(['ncessch', 'MAM_MTH00numvalid_1011'])
df.head(2)
```

	<b>STNAM</b>	<b>LEANM10</b>	<b>NCESSCH</b>	<b>MAM_MTH00NUMVALID_1011</b>
0	ALABAMA	Hale County	1.017100e+10	None
1	ALABAMA	Hale County	1.017100e+10	None

Look for null values using the following filter.

```
df.filter(col('MAM_MTH00numvalid_1011').is_null()).head(2)
```

	<b>STNAM</b>	<b>LEANM10</b>	<b>NCESSCH</b>	<b>MAM_MTH00NUMVALID_1011</b>
0	ALABAMA	Hale County	1.017100e+10	None
1	ALABAMA	Hale County	1.017100e+10	None

## Transform partition

Use `transform_partition()` to replace all null values with a 0. This code will be run by partition, not on the entire data set at one time. This means that on a large data set, this code may run in parallel as the runtime processes the data, partition by partition.

The Python script must define a function called `transform()` that takes two arguments, `df` and `index`. The `df` argument will be a pandas dataframe that contains the data for the partition and the `index` argument is a unique identifier of the partition. The transform function can fully edit the passed in dataframe, but must return a dataframe. Any libraries that the Python script imports must exist in the environment where the dataflow is run.

```
df = df.transform_partition("""
def transform(df, index):
    df['MAM_MTH00numvalid_1011'].fillna(0,inplace=True)
    return df
""")
df.head(2)
```

	STNAM	LEANM10	NCESSCH	MAM_MTH00NUMVALID_1011
0	ALABAMA	Hale County	1.017100e+10	0.0
1	ALABAMA	Hale County	1.017100e+10	0.0

## New script column

You can use a Python script to create a new column that has the county name and the state name, and also to capitalize the state name. To do this, use the `new_script_column()` method on the data flow.

The Python script must define a function called `newvalue()` that takes a single argument `row`. The `row` argument is a dict (`key`: column name, `val`: current value) and will be passed to this function for each row in the data set. This function must return a value to be used in the new column. Any libraries that the Python script imports must exist in the environment where the dataflow is run.

```
df = df.new_script_column(new_column_name='county_state', insert_after='leanm10', script="""
def newvalue(row):
    return row['leanm10'] + ', ' + row['stnam'].title()
""")
df.head(2)
```

	STNAM	LEANM10	COUNTY_STATE	NCESSCH	MAM_MTH00NUMVALID_1011
0	ALABAMA	Hale County	Hale County, Alabama	1.017100e+10	0.0
1	ALABAMA	Hale County	Hale County, Alabama	1.017100e+10	0.0

## New Script Filter

Build a Python expression using `new_script_filter()` to filter the data set to only rows where 'Hale' is not in the new `county_state` column. The expression returns `True` if we want to keep the row, and `False` to drop the row.

```
df = df.new_script_filter("""
def includerow(row):
    val = row['county_state']
    return 'Hale' not in val
""")
df.head(2)
```

	STNAM	LEANM10	COUNTY_STATE	NCESSCH	MAM_MTH00NUMVALID_1011
0	ALABAMA	Jefferson County	Jefferson County, Alabama	1.019200e+10	1.0
1	ALABAMA	Jefferson County	Jefferson County, Alabama	1.019200e+10	0.0

## Next steps

- See the SDK [overview](#) for design patterns and usage examples

- See the Azure Machine Learning Data Prep SDK [tutorial](#) for an example of solving a specific scenario

# Write and configure data using Azure Machine Learning

3/6/2019 • 4 minutes to read

In this article, you learn different methods to write data using the [Azure Machine Learning Data Prep Python SDK](#) and how to configure that data for experimentation with the [Azure Machine Learning SDK for Python](#). Output data can be written at any point in a dataflow. Writes are added as steps to the resulting data flow, and these steps run every time the data flow runs. Data is written to multiple partition files to allow parallel writes.

Since there are no limitations to how many write steps there are in a pipeline, you can easily add additional write steps to get intermediate results for troubleshooting or for other pipelines.

Each time you run a write step, a full pull of the data in the data flow occurs. For example, a data flow with three write steps will read and process every record in the data set three times.

## Supported data types and location

The following file formats are supported

- Delimited files (CSV, TSV, etc.)
- Parquet files

Using the Azure Machine Learning Data Prep Python SDK, you can write data to:

- a local file system
- Azure Blob Storage
- Azure Data Lake Storage

## Spark considerations

When running a data flow in Spark, you must write to an empty folder. Attempting to run a write to an existing folder results in a failure. Make sure your target folder is empty or use a different target location for each run, or the write will fail.

## Monitoring write operations

For your convenience, a sentinel file named `SUCCESS` is generated once a write is completed. Its presence helps you identify when an intermediate write has completed without having to wait for the whole pipeline to complete.

## Example write code

For this example, start by loading data into a data flow using `auto_read_file()`. You reuse this data with different formats.

```
import azureml.dataprep as dprep
t = dprep.auto_read_file('./data/fixed_width_file.txt')
t = t.to_number('Column3')
t.head(5)
```

Example output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	None	NO	NO	ENRS	NaN	NaN	NaN
1	10003.0	99999.0	None	NO	NO	ENSO	NaN	NaN	NaN
2	10010.0	99999.0	None	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	None	NO	NO		NaN	NaN	NaN
4	10014.0	99999.0	None	NO	NO	ENSO	59783.0	5350.0	500.0

### Delimited file example

The following code uses the `write_to_csv()` function to write data to a delimited file.

```
# Create a new data flow using `write_to_csv`
write_t = t.write_to_csv(directory_path=dprep.LocalFileOutput('./test_out/'))

# Run the data flow to begin the write operation.
write_t.run_local()

written_files = dprep.read_csv('./test_out/part-*')
written_files.head(5)
```

Example output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	ERROR	NO	NO	ENRS	NaN	NaN	NaN
1	10003.0	99999.0	ERROR	NO	NO	ENSO	NaN	NaN	NaN
2	10010.0	99999.0	ERROR	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	ERROR	NO	NO		NaN	NaN	NaN
4	10014.0	99999.0	ERROR	NO	NO	ENSO	59783.0	5350.0	500.0

In the preceding output, several errors appear in the numeric columns because of numbers that were not parsed correctly. When written to CSV, null values are replaced with the string "ERROR" by default.

Add parameters as part of your write call and specify a string to use to represent null values.

```
write_t = t.write_to_csv(directory_path=dprep.LocalFileOutput('./test_out/'),
                        error='BadData',
                        na='NA')

write_t.run_local()
written_files = dprep.read_csv('./test_out/part-*')
written_files.head(5)
```

The preceding code produces this output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	BadData	NO	NO	ENRS	NaN	NaN	NaN
1	10003.0	99999.0	BadData	NO	NO	ENSO	NaN	NaN	NaN
2	10010.0	99999.0	BadData	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	BadData	NO	NO		NaN	NaN	NaN
4	10014.0	99999.0	BadData	NO	NO	ENSO	59783.0	5350.0	500.0

### Parquet file example

Similar to `write_to_csv()`, the `write_to_parquet()` function returns a new data flow with a write Parquet step that is executed when the data flow runs.

```
write_parquet_t = t.write_to_parquet(directory_path=dprep.LocalFileOutput('./test_parquet_out/'),
error='MiscreantData')
```

Run the data flow to start the write operation.

```
write_parquet_t.run_local()

written_parquet_files = dprep.read_parquet_file('./test_parquet_out/part-*')
written_parquet_files.head(5)
```

The preceding code produces this output:

	COLUMN 1	COLUMN 2	COLUMN 3	COLUMN 4	COLUMN 5	COLUMN 6	COLUMN 7	COLUMN 8	COLUMN 9
0	10000.0	99999.0	MiscreantData	NO	NO	ENRS	MiscreantData	MiscreantData	MiscreantData
1	10003.0	99999.0	MiscreantData	NO	NO	ENSO	MiscreantData	MiscreantData	MiscreantData
2	10010.0	99999.0	MiscreantData	NO	JN	ENJA	70933.0	-8667.0	90.0
3	10013.0	99999.0	MiscreantData	NO	NO		MiscreantData	MiscreantData	MiscreantData
4	10014.0	99999.0	MiscreantData	NO	NO	ENSO	59783.0	5350.0	500.0

## Configure data for automated machine learning training

Pass your newly written data file into an `AutoMLConfig` object in preparation for automated machine learning training.

The following code example illustrates how to convert your dataflow to a Pandas dataframe and subsequently, split it into training and test datasets for automated machine learning training.

```

from azureml.train.automl import AutoMLConfig
from sklearn.model_selection import train_test_split

dataflow = dprep.auto_read_file(path="")
X_dflow = dataflow.keep_columns([feature_1, feature_2, feature_3])
y_dflow = dataflow.keep_columns("target")

X_df = X_dflow.to_pandas_dataframe()
y_df = y_dflow.to_pandas_dataframe()

X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.2, random_state=223)

# flatten y_train to 1d array
y_train.values.flatten()

#configure
automated_ml_config = AutoMLConfig(task = 'regression',
                                     X = X_train.values,
                                     y = y_train.values.flatten(),
                                     iterations = 30,
                                     Primary_metric = "AUC_weighted",
                                     n_cross_validation = 5
                                     )

```

If you do not require any intermediate data preparation steps like in the preceding example, you can pass your dataflow directly into `AutoMLConfig`.

```

automated_ml_config = AutoMLConfig(task = 'regression',
                                     X = X_dflow,
                                     y = y_dflow,
                                     iterations = 30,
                                     Primary_metric = "AUC_weighted",
                                     n_cross_validation = 5
                                     )

```

## Next steps

- See the SDK [overview](#) for design patterns and usage examples
- See the automated machine learning [tutorial](#) for a regression model example

# Access data from your datastores

3/5/2019 • 4 minutes to read

In this article, you learn different ways to access and interact with your data in Azure Machine Learning workflows via datastores.

This how-to shows examples for the following tasks:

- [Choose a datastore](#)
- [Get a datastore](#)
- [Upload and download data to datastores](#)
- Access datastore during training

## Prerequisites

To use datastores, you need a [workspace](#) first.

Start by either [creating a new workspace](#) or retrieving an existing one:

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

Or, [follow this Python quickstart](#) to use the SDK to create your workspace and get started.

## Choose a datastore

You can use the default datastore or bring your own.

### Use the default datastore in your workspace

No need to create or configure a storage account since each workspace has a default datastore. You can use that datastore right away as it is already registered in the workspace.

To get the workspace's default datastore:

```
ds = ws.get_default_datastore()
```

### Register your own datastore with the workspace

If you have existing Azure Storage, you can register it as a datastore on your workspace. All the register methods are on the [Datastore](#) class and have the form register\_azure\_\*.

The following examples show you to register an Azure Blob Container or an Azure File Share as a datastore.

- For an **Azure Blob Container Datastore**, use [register\\_azure\\_blob-container\(\)](#)

```
ds = Datastore.register_azure_blob_container(workspace=ws,
                                              datastore_name='your datastore name',
                                              container_name='your azure blob container name',
                                              account_name='your storage account name',
                                              account_key='your storage account key',
                                              create_if_not_exists=True)
```

- For an **Azure File Share Datastore**, use `register_azure_file_share()`. For example:

```
ds = Datastore.register_azure_file_share(workspace=ws,
                                         datastore_name='your datastore name',
                                         container_name='your file share name',
                                         account_name='your storage account name',
                                         account_key='your storage account key',
                                         create_if_not_exists=True)
```

## Get data in your datastore

To get a specified datastore registered in the current workspace, use `get()`:

```
#get named datastore from current workspace
ds = Datastore.get(ws, datastore_name='your datastore name')
```

To get a list of all datastores in a given workspace, use this code:

```
#list all datastores registered in current workspace
datastores = ws.datastores
for name, ds in datastores.items():
    print(name, ds.datastore_type)
```

To define a different default datastore for the current workspace, use `set_default_datastore()`:

```
#define default datastore for current workspace
ws.set_default_datastore('your datastore name')
```

## Upload and download data

The `upload()` and `download()` methods described in the following examples are specific to and operate identically for the `AzureBlobDatastore` and `AzureFileDatastore` classes.

### Upload

Upload either a directory or individual files to the datastore using the Python SDK.

To upload a directory to a datastore `ds`:

```
import azureml.data
from azureml.data import AzureFileDatastore, AzureBlobDatastore

ds.upload(src_dir='your source directory',
          target_path='your target path',
          overwrite=True,
          show_progress=True)
```

`target_path` specifies the location in the file share (or blob container) to upload. It defaults to `None`, in which case

the data gets uploaded to root. `overwrite=True` will overwrite any existing data at `target_path`.

Or upload a list of individual files to the datastore via the datastore's `upload_files()` method.

## Download

Similarly, download data from a datastore to your local file system.

```
ds.download(target_path='your target path',
            prefix='your prefix',
            show_progress=True)
```

`target_path` is the location of the local directory to download the data to. To specify a path to the folder in the file share (or blob container) to download, provide that path to `prefix`. If `prefix` is `None`, all the contents of your file share (or blob container) will get downloaded.

## Access datastores during training

You can access a datastore during a training run (for example, for training or validation data) on a remote compute target via the Python SDK using the [DataReference](#) class.

There are several ways to make your datastore available on the remote compute.

WAY	METHOD	DESCRIPTION
Mount	<code>as_mount()</code>	Use to mount a datastore on the remote compute.
Download	<code>as_download()</code>	Use to download data from the location specified by <code>path_on_compute</code> on your datastore to the remote compute.
Upload	<code>as_upload()</code>	Use to upload data to the root of your datastore from the location specified by <code>path_on_compute</code> .

```
import azureml.data
from azureml.data import DataReference

ds.as_mount()
ds.as_download(path_on_compute='your path on compute')
ds.as_upload(path_on_compute='yourfilename')
```

## Reference files/folders

To reference a specific folder or file in your datastore, use the datastore's `path()` function.

```
#download the contents of the `./bar` directory from the datastore
ds.path('./bar').as_download()
```

## Examples

Any `ds` or `ds.path` object resolves to an environment variable name of the format

`"$AZUREML_DATAREFERENCE_XXXX"` whose value represents the mount/download path on the remote compute. The datastore path on the remote compute might not be the same as the execution path for the script.

To access your datastore during training, pass it into your training script as a command-line argument via

`script_params` from the `Estimator` class.

```
from azureml.train.estimator import Estimator

script_params = {
    '--data_dir': ds.as_mount()
}

est = Estimator(source_directory='your code directory',
                script_params=script_params,
                compute_target=compute_target,
                entry_script='train.py')
```

`as_mount()` is the default mode for a datastore, so you could also directly pass `ds` to the `--data_dir` argument.

Or pass in a list of datastores to the Estimator constructor `inputs` parameter to mount or copy to/from your datastore(s). This code example:

- Downloads all the contents in datastore `ds1` to the remote compute before your training script `train.py` is run
- Downloads the folder `'./foo'` in datastore `ds2` to the remote compute before `train.py` is run
- Uploads the file `'./bar.pkl'` from the remote compute up to the datastore `ds3` after your script has run

```
est = Estimator(source_directory='your code directory',
                compute_target=compute_target,
                entry_script='train.py',
                inputs=[ds1.as_download(), ds2.path('./foo').as_download(),
                        ds3.as_upload(path_on_compute='./bar.pkl')])
```

## Next steps

- [Train a model](#)
- [Deploy a model](#)

# Set up compute targets for model training

2/26/2019 • 17 minutes to read

With Azure Machine Learning service, 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"](#).

You can create and manage a compute target using the Azure Machine Learning SDK, Azure portal, or Azure CLI. If you have compute targets that were created through another service (for example, an HDInsight cluster), you can use them by attaching them to your Azure Machine Learning service workspace.

In this article, you learn how to use various compute targets for model training. The steps for all compute targets follow the same workflow:

1. **Create** a compute target if you don't already have one.
2. **Attach** the compute target to your workspace.
3. **Configure** the compute target so that it contains the Python environment and package dependencies needed by your script.

## NOTE

Code in this article was tested with Azure Machine Learning SDK version 1.0.6.

## Compute targets for training

Azure Machine Learning service has varying support across different compute targets. A typical model development lifecycle starts with dev/experimentation on a small amount of data. At this stage, we recommend using a local environment. For example, your local computer or a cloud-based VM. As you scale up your training on larger data sets, or do distributed training, we recommend using 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 various scenarios may vary as detailed below:

COMPUTE TARGET FOR TRAINING	GPU ACCELERATION	AUTOMATED HYPERPARAMETER TUNING	AUTOMATED MACHINE LEARNING	AZURE MACHINE LEARNING PIPELINES
Local computer	Maybe		✓	
Azure Machine Learning Compute	✓	✓	✓	✓
Remote VM	✓	✓	✓	✓
Azure Databricks			✓	✓
Azure Data Lake Analytics				✓
Azure HDInsight				✓

COMPUTE TARGET FOR TRAINING	GPU ACCELERATION	AUTOMATED HYPERPARAMETER TUNING	AUTOMATED MACHINE LEARNING	AZURE MACHINE LEARNING PIPELINES
Azure Batch				✓

All compute targets can be reused for multiple training jobs. For example, once you attach a remote VM to your workspace, you can reuse it for multiple jobs.

#### NOTE

Azure Machine Learning Compute can be created as a persistent resource or created dynamically when you request a run. Run-based creation removes the compute target after the training run is complete, so you cannot reuse compute targets created this way.

## What's a run configuration?

When training, it is common to start on your local computer, and later run that training script on a different compute target. With Azure Machine Learning service, you can run your script on various compute targets without having to change your script.

All you need to do is define the environment for each compute target with a **run configuration**. Then, when you want to run your training experiment on a different compute target, specify the run configuration for that compute.

Learn more about [submitting experiments](#) at the end of this article.

### Manage environment and dependencies

When you create a run configuration, you need to decide how to manage the environment and dependencies on the compute target.

#### System-managed environment

Use a system-managed environment when you want [Conda](#) to manage the Python environment and the script dependencies for you. A system-managed environment is assumed by default and the most common choice. It is useful on remote compute targets, especially when you cannot configure that target.

All you need to do is specify each package dependency using the [CondaDependency class](#). Then Conda creates a file named **conda\_dependencies.yml** in the **aml\_config** directory in your workspace with your list of package dependencies and sets up your Python environment when you submit your training experiment.

The initial setup of a new environment can take several minutes depending on the size of the required dependencies. As long as the list of packages remains unchanged, the setup time happens only once.

The following code shows an example for a system-managed environment requiring scikit-learn:

```
from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies

run_system_managed = RunConfiguration()

# Specify the conda dependencies with scikit-learn
run_system_managed.environment.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'])
```

#### User-managed environment

For a user-managed environment, you're responsible for setting up your environment and installing every package your training script needs on the compute target. If your training environment is already configured

(such as on your local machine), you can skip the setup step by setting `user_managed_dependencies` to True. Conda will not check your environment or install anything for you.

The following code shows an example of configuring training runs for a user-managed environment:

```
from azureml.core.runconfig import RunConfiguration

run_user_managed = RunConfiguration()
run_user_managed.environment.python.user_managed_dependencies = True

# Choose a specific Python environment by pointing to a Python path. For example:
# run_config.environment.python.interpreter_path = '/home/ninghai/miniconda3/envs/sdk2/bin/python'
```

## Set up compute targets with Python

Use the sections below to configure these compute targets:

- [Local computer](#)
- [Azure Machine Learning Compute](#)
- [Remote virtual machines](#)
- [Azure HDInsight](#)

### Local computer

1. **Create and attach:** There's no need to create or attach a compute target to use your local computer as the training environment.
2. **Configure:** When you use your local computer as a compute target, the training code is run in your [development environment](#). If that environment already has the Python packages you need, use the user-managed environment.

```
from azureml.core.runconfig import RunConfiguration

# Edit a run configuration property on the fly.
run_local = RunConfiguration()

run_local.environment.python.user_managed_dependencies = True
```

Now that you've attached the compute and configured your run, the next step is to [submit the training run](#).

### Azure Machine Learning Compute

Azure Machine Learning Compute is a managed-compute infrastructure that allows the user 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](#).

You can use Azure Machine Learning Compute to distribute the training 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](#).

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](#).

You can create an Azure Machine Learning compute environment on demand when you schedule a run, or as a persistent resource.

#### Run-based creation

You can create Azure Machine Learning Compute as a compute target at run time. The compute is automatically created for your run. The cluster scales up to the number of **max\_nodes** that you specify in your run config. The compute is deleted automatically once the run completes.

#### IMPORTANT

Run-based creation of Azure Machine Learning compute is currently in Preview. Don't use run-based creation if you use automated hyperparameter tuning or automated machine learning. To use hyperparameter tuning or automated machine learning, create a [persistent compute](#) target instead.

1. **Create, attach, and configure:** The run-based creation performs all the necessary steps to create, attach, and configure the compute target with the run configuration.

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

# First, list the supported VM families for Azure Machine Learning Compute
print(AmlCompute.supported_vmsizes(workspace=ws))

from azureml.core.runconfig import RunConfiguration
# Create a new runconfig object
run_temp_compute = RunConfiguration()

# Signal that you want to use AmlCompute to execute the script
run_temp_compute.target = "amlcompute"

# AmlCompute is created in the same region as your workspace
# Set the VM size for AmlCompute from the list of supported_vmsizes
run_temp_compute.amlcompute.vm_size = 'STANDARD_D2_V2'
```

Now that you've attached the compute and configured your run, the next step is to [submit the training run](#).

#### Persistent compute

A persistent Azure Machine Learning Compute can be reused across jobs. The compute can be shared with other users in the workspace and is kept between jobs.

1. **Create and attach:** 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. The compute autoscales down to zero nodes when it isn't used. Dedicated VMs are created to run your jobs as needed.

- **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 = "cpucluster"

# 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.

Or you can create and attach a persistent Azure Machine Learning Compute resource [in the Azure portal](#).

## 2. **Configure:** Create a run configuration for the persistent compute target.

```

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_amlcompute = RunConfiguration()

# Use the cpu_cluster you created above.
run_amlcompute.target = cpu_cluster

# Enable Docker
run_amlcompute.environment.docker.enabled = True

# Set Docker base image to the default CPU-based image
run_amlcompute.environment.docker.base_image = DEFAULT_CPU_IMAGE

# Use conda_dependencies.yml to create a conda environment in the Docker image for execution
run_amlcompute.environment.python.user_managed_dependencies = False

# Auto-prepare the Docker image when used for execution (if it is not already prepared)
run_amlcompute.auto_prepare_environment = True

# Specify CondaDependencies obj, add necessary packages
run_amlcompute.environment.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'])

```

Now that you've attached the compute and configured your run, the next step is to [submit the training run](#).

### 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 service. 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.

2. **Attach:** To attach an existing virtual machine as a compute target, you must provide the fully qualified domain name (FQDN), user name, and password for the virtual machine. In the example, replace <fqdn> with the public FQDN of the VM, or the public IP address. Replace <username> and <password> with the SSH user name and password for the VM.

```
from azureml.core.compute import RemoteCompute, ComputeTarget

# Create the compute config
compute_target_name = "attach-dsvm"
attach_config = RemoteCompute.attach_configuration(address = "<fqdn>",
                                                    ssh_port=22,
                                                    username='<username>',
                                                    password="<password>")

# If you authenticate with SSH keys instead, use this code:
#                                                     ssh_port=22,
#                                                     username='<username>',
#                                                     password=None,
#                                                     private_key_file="<path-to-file>",
#                                                     private_key_passphrase="<passphrase>")

# 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 the Azure portal](#).

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.

```

import azureml.core
from azureml.core.runconfig import RunConfiguration
from azureml.core.conda_dependencies import CondaDependencies

run_dsvm = RunConfiguration(framework = "python")

# Set the compute target to the Linux DSVM
run_dsvm.target = compute_target_name

# Use Docker in the remote VM
run_dsvm.environment.docker.enabled = True

# Use the CPU base image
# To use GPU in DSVM, you must also use the GPU base Docker image
"azureml.core.runconfig.DEFAULT_GPU_IMAGE"
run_dsvm.environment.docker.base_image = azureml.core.runconfig.DEFAULT_CPU_IMAGE
print('Base Docker image is:', run_dsvm.environment.docker.base_image)

# Specify the CondaDependencies object
run_dsvm.environment.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn'])

```

Now that you've attached the compute and configured your run, the next step is to [submit the training run](#).

## 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](#).

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 <clusternamespace>-ssh.azurehdinsight.net, where <clusternamespace> is the name that you provided for the cluster.

2. **Attach:** To attach an HDInsight cluster as a compute target, you must provide the hostname, user name, and password for the HDInsight cluster. The following example uses the SDK to attach a cluster to your workspace. In the example, replace <clusternamespace> with the name of your cluster. Replace <username> and <password> with the SSH user name and password for the cluster.

```

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(address='<clusternamespace>-ssh.azureinsight.net',
                                                          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 the Azure portal](#).

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

batch_compute_name = 'mybatchcompute' # Name to associate with new compute in workspace

# Batch account details needed to attach as compute to workspace
batch_account_name = "<batch_account_name>" # Name of the Batch account
batch_resource_group = "<batch_resource_group>" # Name of the resource group which contains this account

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))
```

## Set up compute in the Azure portal

You can access the compute targets that are associated with your workspace in the Azure portal. You can use the portal to:

- [View compute targets attached to your workspace](#)
- [Create a compute target](#) in your workspace
- [Attach a compute target](#) that was created outside the workspace

After a target is created and attached to your workspace, you will use it in your run configuration with a `ComputeTarget` object:

```
from azureml.core.compute import ComputeTarget  
myvm = ComputeTarget(workspace=ws, name='my-vm-name')
```

## View compute targets

To see the compute targets for your workspace, use the following steps:

1. Navigate to the [Azure portal](#) and open your workspace.
2. Under **Applications**, select **Compute**.

The screenshot shows the Azure Machine Learning service workspace named 'get-started-space'. The left sidebar has a 'Compute' section with a red box around the 'Compute' button. The main area shows workspace details: Resource group 'mygroup', Location 'East US 2', Subscription 'documentationteam', and a long Subscription ID. Below this is a 'Getting Started' section with three cards: 'Explore your Azure Machine Learning service workspace', 'View Documentation', and 'Get Started in Azure Notebooks'. At the bottom is a 'View more samples at GitHub' card.

## Create a 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 plus sign (+) to add a compute target.

The screenshot shows the 'Compute' page with a red box around the '+ Add Compute' button. The page displays the message 'There are no computes in this workspace.'

2. Enter a name for the compute target.
3. Select **Machine Learning Compute** as the type of compute to use for **Training**.

**NOTE**

Azure Machine Learning Compute is the only managed-compute resource you can create in the Azure portal. All other compute resources can be attached after they are created.

4. Fill out the form. Provide values for the required properties, especially **VM Family**, and the **maximum nodes** to use to spin up the compute.

The screenshot shows the 'Add Compute' page in the Azure portal. The 'Compute' tab is active. The form includes fields for Compute name (my-compute), Compute type (Machine Learning Compute), Region (East US 2), Virtual machine size (Choose Virtual machine size), Virtual machine priority (Dedicated), Minimum number of nodes (0), Maximum number of nodes (120), and Idle seconds before scale down (120). A note about Machine Learning Compute is displayed below the form. The 'Virtual machine size' and 'Maximum number of nodes' fields are highlighted with red boxes.

5. Select **Create**.
6. View the status of the create operation by selecting the compute target from the list:

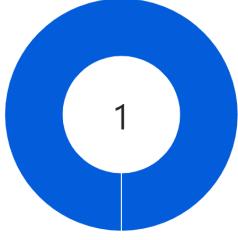
The screenshot shows the 'Compute' list page in the Azure portal. It displays a single row for a compute target named 'compute1'. The columns show the NAME (compute1), TYPE (Machine Learning Compute), PROVISIONING STATE (Succeeded), CREATED DATE (Mon Nov 19 2018), and VIRTUAL MACHINE SIZE (STANDARD\_D1\_V2). The 'compute1' entry is highlighted with a red box.

7. You then see the details for the compute target:

## compute1

[Back to Compute List](#) [Refresh](#) [Edit](#) [Delete](#) [Detach](#)

**CLUSTER NODE STATUS**



Idle	1
Leaving	0
Preparing	0
Running	0
Preempted	0
Unusable	0

Show usable nodes only

**CLUSTER STATE**

Allocation state	steady
Allocation state transition time	November 28, 2018, 2:18 AM
Provisioning state	Succeeded
Created	November 28, 2018, 2:18 AM
Current node count	1

**ATTRIBUTES**

Compute name	compute1
Compute type	Machine Learning Compute
Subscription ID	
Resource group	pgunda
Workspace	pgeastus
Region	eastus

**RESOURCE PROPERTIES**

Virtual Machine size	STANDARD_D1_V2
Virtual Machine priority	dedicated
Minimum number of nodes	1
Maximum number of nodes	1
Idle seconds before scale down	120
Subnet	--

## Attach compute targets

To use compute targets created outside the Azure Machine Learning service workspace, you must attach them. Attaching a compute target makes it available to your workspace.

Follow the steps described earlier to view the list of compute targets. Then use the following steps to attach a compute target:

1. Select the plus sign (+) to add a compute target.
2. Enter a name for the compute target.
3. Select the type of compute to attach for **Training**:

### IMPORTANT

Not all compute types can be attached from the Azure portal. 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

4. 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](#)

5. Select **Attach**.

6. View the status of the attach operation by selecting the compute target from the list.

## Set up compute with the CLI

You can access the compute targets that are associated with your workspace using the [CLI extension](#) for Azure Machine Learning service. You can use the CLI to:

- Create a managed compute target
- Update a managed compute target
- Attach an unmanaged compute target

For more information, see [Resource management](#).

## Submit training run

After you create a run configuration, you use it to run your experiment. The code pattern to submit a training run is the same for all types of compute targets:

1. Create an experiment to run
2. Submit the run.
3. Wait for the run to complete.

### Create an experiment

First, create an experiment in your workspace.

```
from azureml.core import Experiment
experiment_name = 'my_experiment'

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

### Submit the experiment

Submit the experiment with a `ScriptRunConfig` object. This object includes the:

- **source\_directory**: The source directory that contains your training script
- **script**: Identify the training script
- **run\_config**: The run configuration, which in turn defines where the training will occur.

When you submit a training run, a snapshot of the directory that contains your training scripts is created and sent to the compute target. For more information, see [Snapshots](#).

For example, to use [the local target](#) configuration:

```
from azureml.core import ScriptRunConfig
import os

script_folder = os.getcwd()
src = ScriptRunConfig(source_directory = script_folder, script = 'train.py', run_config = run_local)
run = exp.submit(src)
run.wait_for_completion(show_output = True)
```

Switch the same experiment to run in a different compute target by using a different run configuration, such as the [amlcompute target](#):

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory = script_folder, script = 'train.py', run_config = run_amlcompute)
run = exp.submit(src)
run.wait_for_completion(show_output = True)
```

Or you can:

- Submit the experiment with an `Estimator` object as shown in [Train ML models with estimators](#).
- Submit an experiment [using the CLI extension](#).

## 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

- [Tutorial: Train a model](#) uses a managed compute target to train a model.
- Once you have a trained model, learn [how and where to deploy models](#).
- View the [RunConfiguration class](#) SDK reference.
- [Use Azure Machine Learning service with Azure Virtual Networks](#)

# Train models with Azure Machine Learning using estimator

2/15/2019 • 4 minutes to read

With Azure Machine Learning, you can easily submit your training script to [various compute targets](#), using [RunConfiguration object](#) and [ScriptRunConfig object](#). That pattern gives you a lot of flexibility and maximum control.

To facilitate deep learning model training, the Azure Machine Learning Python SDK provides an alternative higher-level abstraction, the estimator class, which allows users to easily construct run configurations. You can create and use a generic [Estimator](#) to submit training script using any learning framework you choose (such as scikit-learn) you want to run on any compute target, whether it's your local machine, a single VM in Azure, or a GPU cluster in Azure. For PyTorch, TensorFlow and Chainer tasks, Azure Machine Learning also provides respective [PyTorch](#), [TensorFlow](#) and [Chainer](#) estimators to simplify using these frameworks.

## Train with an estimator

Once you've created your [workspace](#) and set up your [development environment](#), training a model in Azure Machine Learning involves the following steps:

1. Create a [remote compute target](#) (note you can also use local computer as compute target)
2. Upload your [training data](#) to datastore (Optional)
3. Create your [training script](#)
4. Create an [Estimator](#) object
5. Submit the estimator to an experiment object under the workspace

This article focuses on steps 4-5. For steps 1-3, refer to the [train a model tutorial](#) for an example.

### Single-node training

Use an [Estimator](#) for a single-node training run on remote compute in Azure for a scikit-learn model. You should have already created your [compute target](#) object `compute_target` and your [datastore](#) object `ds`.

```
from azureml.train.estimator import Estimator

script_params = {
    '--data-folder': ds.as_mount(),
    '--regularization': 0.8
}

sk_est = Estimator(source_directory='./my-sklearn-proj',
                    script_params=script_params,
                    compute_target=compute_target,
                    entry_script='train.py',
                    conda_packages=['scikit-learn'])
```

This code snippet specifies the following parameters to the [Estimator](#) constructor.

PARAMETER	DESCRIPTION
-----------	-------------

PARAMETER	DESCRIPTION
<code>source_directory</code>	Local directory that contains all of your code needed for the training job. This folder gets copied from your local machine to the remote compute
<code>script_params</code>	Dictionary specifying the command-line arguments to your training script <code>entry_script</code> , in the form of <command-line argument, value> pairs
<code>compute_target</code>	Remote compute target that your training script will run on, in this case an Azure Machine Learning Compute ( <a href="#">AmlCompute</a> ) cluster. (Please note even though AmlCompute cluster is the commonly used target, it is also possible to choose other compute target types such as Azure VMs or even local computer.)
<code>entry_script</code>	Filepath (relative to the <code>source_directory</code> ) of the training script to be run on the remote compute. This file, and any additional files it depends on, should be located in this folder
<code>conda_packages</code>	List of Python packages to be installed via conda needed by your training script.

The constructor has another parameter called `pip_packages` that you use for any pip packages needed

Now that you've created your `Estimator` object, submit the training job to be run on the remote compute with a call to the `submit` function on your `Experiment` object `experiment`.

```
run = experiment.submit(sk_est)
print(run.get_portal_url())
```

### 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')
```

### Distributed training and custom Docker images

There are two additional training scenarios you can carry out with the `Estimator`:

- Using a custom Docker image
- Distributed training on a multi-node cluster

The following code shows how to carry out distributed training for a Keras model. In addition, instead of using the default Azure Machine Learning images, it specifies a custom docker image from Docker Hub `continuumio/miniconda` for training.

You should have already created your `compute target` object `compute_target`. You create the estimator as follows:

```
from azureml.train.estimator import Estimator

estimator = Estimator(source_directory='./my-keras-proj',
                      compute_target=compute_target,
                      entry_script='train.py',
                      node_count=2,
                      process_count_per_node=1,
                      distributed_backend='mpi',
                      conda_packages=['tensorflow', 'keras'],
                      custom_docker_base_image='continuumio/miniconda')
```

The above code exposes the following new parameters to the `Estimator` constructor:

PARAMETER	DESCRIPTION	DEFAULT
<code>custom_docker_base_image</code>	Name of the image you want to use. Only provide images available in public docker repositories (in this case Docker Hub). To use an image from a private docker repository, use the constructor's <code>environment_definition</code> parameter instead. <a href="#">See example</a> .	<code>None</code>
<code>node_count</code>	Number of nodes to use for your training job.	<code>1</code>
<code>process_count_per_node</code>	Number of processes (or "workers") to run on each node. In this case, you use the <code>2</code> GPUs available on each node.	<code>1</code>
<code>distributed_backend</code>	Backend for launching distributed training, which the Estimator offers via MPI. To carry out parallel or distributed training (e.g. <code>node_count &gt; 1</code> or <code>process_count_per_node &gt; 1</code> or both), set <code>distributed_backend='mpi'</code> . The MPI implementation used by AML is <a href="#">Open MPI</a> .	<code>None</code>

Finally, submit the training job:

```
run = experiment.submit(estimator)
print(run.get_portal_url())
```

## Examples

For a notebook that shows the basics of estimator pattern, see:

- [how-to-use-azureml/training-with-deep-learning/how-to-use-estimator](#)

For a notebook that trains an scikit-learn model using estimator, see:

- [tutorials/img-classification-part1-training.ipynb](#)

For notebooks on training models using deep-learning-framework specific estimators, see:

- [how-to-use-azureml/training-with-deep-learning](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

## Next steps

- [Track run metrics during training](#)
- [Train PyTorch models](#)
- [Train TensorFlow models](#)
- [Tune hyperparameters](#)
- [Deploy a trained model](#)

# Train PyTorch models with Azure Machine Learning service

3/4/2019 • 2 minutes to read

For deep neural network (DNN) training using PyTorch, Azure Machine Learning provides a custom [PyTorch](#) class of the [Estimator](#). The Azure SDK's [PyTorch](#) estimator enables you to easily submit PyTorch training jobs for both single-node and distributed runs on Azure compute.

## Single-node training

Training with the [PyTorch](#) estimator is similar to using the base [Estimator](#), so first read through the how-to article and make sure you understand the concepts introduced there.

To run a PyTorch job, instantiate a [PyTorch](#) object. You should have already created your [compute target](#) object [compute\\_target](#) and your [datastore](#) object [ds](#).

```
from azureml.train.dnn import PyTorch

script_params = {
    '--data_dir': ds
}

pt_est = PyTorch(source_directory='./my-pytorch-proj',
                  script_params=script_params,
                  compute_target=compute_target,
                  entry_script='train.py',
                  use_gpu=True)
```

Here, we specify the following parameters to the PyTorch constructor:

PARAMETER	DESCRIPTION
<a href="#">source_directory</a>	Local directory that contains all of your code needed for the training job. This folder gets copied from your local machine to the remote compute
<a href="#">script_params</a>	Dictionary specifying the command-line arguments to your training script <a href="#">entry_script</a> , in the form of <command-line argument, value> pairs
<a href="#">compute_target</a>	Remote compute target that your training script will run on, in this case an Azure Machine Learning Compute ( <a href="#">AmlCompute</a> ) cluster
<a href="#">entry_script</a>	Filepath (relative to the <a href="#">source_directory</a> ) of the training script to be run on the remote compute. This file, and any additional files it depends on, should be located in this folder
<a href="#">conda_packages</a>	List of Python packages to be installed via conda needed by your training script. The constructor has another parameter called <a href="#">pip_packages</a> that you can use for any pip packages needed

PARAMETER	DESCRIPTION
<code>use_gpu</code>	Set this flag to <code>True</code> to leverage the GPU for training. Defaults to <code>False</code>

Since you are using the `PyTorch` estimator, the container used for training will include the PyTorch package and related dependencies needed for training on CPUs and GPUs.

Then, submit the PyTorch job:

```
run = exp.submit(pt_est)
```

## Distributed training

The `PyTorch` estimator also enables you to train your models at scale across CPU and GPU clusters of Azure VMs. You can easily run distributed PyTorch training with a few API calls, while Azure Machine Learning will manage behind the scenes all the infrastructure and orchestration needed to carry out these workloads.

Azure Machine Learning currently supports MPI-based distributed training of PyTorch using the Horovod framework.

### Horovod

[Horovod](#) is an open-source ring-allreduce framework for distributed training developed by Uber.

To run distributed PyTorch using the Horovod framework, create the PyTorch object as follows:

```
from azureml.train.dnn import PyTorch

pt_est = PyTorch(source_directory='./my-pytorch-project',
                 script_params={},
                 compute_target=compute_target,
                 entry_script='train.py',
                 node_count=2,
                 process_count_per_node=1,
                 distributed_backend='mpi',
                 use_gpu=True)
```

This code exposes the following new parameters to the PyTorch constructor:

PARAMETER	DESCRIPTION	DEFAULT
<code>node_count</code>	Number of nodes to use for your training job.	<code>1</code>
<code>process_count_per_node</code>	Number of processes (or "workers") to run on each node.	<code>1</code>
<code>distributed_backend</code>	Backend for launching distributed training, which the Estimator offers via MPI. To carry out parallel or distributed training (e.g. <code>node_count</code> > 1 or <code>process_count_per_node</code> > 1 or both) with MPI (and Horovod), set <code>distributed_backend='mpi'</code> . The MPI implementation used by Azure Machine Learning is <a href="#">Open MPI</a> .	<code>None</code>

The above example will run distributed training with two workers, one worker per node.

Horovod and its dependencies will be installed for you, so you can simply import it in your training script `train.py` as follows:

```
import torch
import horovod
```

Finally, submit your distributed PyTorch job:

```
run = exp.submit(pt_est)
```

## Examples

For notebooks on distributed deep learning, see:

- [how-to-use-azureml/training-with-deep-learning](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

## Next steps

- [Track run metrics during training](#)
- [Tune hyperparameters](#)
- [Deploy a trained model](#)

# Train TensorFlow and Keras models with Azure Machine Learning service

3/4/2019 • 5 minutes to read

For deep neural network (DNN) training using TensorFlow, Azure Machine Learning provides a custom `TensorFlow` class of the `Estimator`. The Azure SDK's `TensorFlow` estimator (not to be conflated with the `tf.estimator.Estimator` class) enables you to easily submit TensorFlow training jobs for both single-node and distributed runs on Azure compute.

## Single-node training

Training with the `TensorFlow` estimator is similar to using the base `Estimator`, so first read through the how-to article and make sure you understand the concepts introduced there.

To run a TensorFlow job, instantiate a `TensorFlow` object. You should have already created your `compute target` object `compute_target`.

```
from azureml.train.dnn import TensorFlow

script_params = {
    '--batch-size': 50,
    '--learning-rate': 0.01,
}

tf_est = TensorFlow(source_directory='./my-tf-proj',
                    script_params=script_params,
                    compute_target=compute_target,
                    entry_script='train.py',
                    conda_packages=['scikit-learn'], # in case you need scikit-learn in train.py
                    use_gpu=True)
```

Here, we specify the following parameters to the `TensorFlow` constructor:

PARAMETER	DESCRIPTION
<code>source_directory</code>	Local directory that contains all of your code needed for the training job. This folder gets copied from your local machine to the remote compute
<code>script_params</code>	Dictionary specifying the command-line arguments to your training script <code>entry_script</code> , in the form of <command-line argument, value> pairs
<code>compute_target</code>	Remote compute target that your training script will run on, in this case an Azure Machine Learning Compute ( <a href="#">AmlCompute</a> ) cluster
<code>entry_script</code>	Filepath (relative to the <code>source_directory</code> ) of the training script to be run on the remote compute. This file, and any additional files it depends on, should be located in this folder

PARAMETER	DESCRIPTION
<code>conda_packages</code>	List of Python packages to be installed via conda needed by your training script. In this case training script uses <code>sklearn</code> for loading the data, so specify this package to be installed. The constructor has another parameter called <code>pip_packages</code> that you can use for any pip packages needed
<code>use_gpu</code>	Set this flag to <code>True</code> to leverage the GPU for training. Defaults to <code>False</code> .

Since you are using the TensorFlow estimator, the container used for training will default include the TensorFlow package and related dependencies needed for training on CPUs and GPUs.

Then, submit the TensorFlow job:

```
run = exp.submit(tf_est)
```

## Keras support

[Keras](#) is a popular high-level DNN Python API that supports TensorFlow, CNTK, or Theano as backends. If you use TensorFlow as backend, you can easily use the TensorFlow estimator to train a Keras model. Here is an example of a TensorFlow estimator with Keras added to it:

```
from azureml.train.dnn import TensorFlow

keras_est = TensorFlow(source_directory='./my-keras-proj',
                      script_params=script_params,
                      compute_target=compute_target,
                      entry_script='keras_train.py',
                      pip_packages=['keras'], # just add keras through pip
                      use_gpu=True)
```

The above TensorFlow estimator constructor instructs Azure Machine Learning service to install Keras through pip to the execution environment. And your `keras_train.py` can then import Keras API to train a Keras model. For a complete example, explore [this Jupyter notebook](#).

## Distributed training

The TensorFlow Estimator also enables you to train your models at scale across CPU and GPU clusters of Azure VMs. You can easily run distributed TensorFlow training with a few API calls, while Azure Machine Learning will manage behind the scenes all the infrastructure and orchestration needed to carry out these workloads.

Azure Machine Learning supports two methods of distributed training in TensorFlow:

- MPI-based distributed training using the [Horovod](#) framework
- Native [distributed TensorFlow](#) via the parameter server method

### Horovod

[Horovod](#) is an open-source ring-allreduce framework for distributed training developed by Uber.

To run distributed TensorFlow using the Horovod framework, create the TensorFlow object as follows:

```

from azureml.train.dnn import TensorFlow

tf_est = TensorFlow(source_directory='./my-tf-proj',
                    script_params={},
                    compute_target=compute_target,
                    entry_script='train.py',
                    node_count=2,
                    process_count_per_node=1,
                    distributed_backend='mpi',
                    use_gpu=True)

```

The above code exposes the following new parameters to the TensorFlow constructor:

PARAMETER	DESCRIPTION	DEFAULT
<code>node_count</code>	Number of nodes to use for your training job.	<code>1</code>
<code>process_count_per_node</code>	Number of processes (or "workers") to run on each node.	<code>1</code>
<code>distributed_backend</code>	Backend for launching distributed training, which the Estimator offers via MPI. If you want to carry out parallel or distributed training (for example, <code>node_count &gt; 1</code> or <code>process_count_per_node &gt; 1</code> or both) with MPI (and Horovod), set <code>distributed_backend='mpi'</code> . The MPI implementation used by Azure Machine Learning is <a href="#">Open MPI</a> .	<code>None</code>

The above example will run distributed training with two workers, one worker per node.

Horovod and its dependencies will be installed for you, so you can import it in your training script `train.py` as follows:

```

import tensorflow as tf
import horovod

```

Finally, submit the TensorFlow job:

```

run = exp.submit(tf_est)

```

## Parameter server

You can also run [native distributed TensorFlow](#), which uses the parameter server model. In this method, you train across a cluster of parameter servers and workers. The workers calculate the gradients during training, while the parameter servers aggregate the gradients.

Construct the TensorFlow object:

```

from azureml.train.dnn import TensorFlow

tf_est = TensorFlow(source_directory='./my-tf-proj',
                    script_params={},
                    compute_target=compute_target,
                    entry_script='train.py',
                    node_count=2,
                    worker_count=2,
                    parameter_server_count=1,
                    distributed_backend='ps',
                    use_gpu=True)

```

Pay attention to the following parameters to the TensorFlow constructor in the above code:

PARAMETER	DESCRIPTION	DEFAULT
<code>worker_count</code>	Number of workers.	<code>1</code>
<code>parameter_server_count</code>	Number of parameter servers.	<code>1</code>
<code>distributed_backend</code>	Backend to use for distributed training. To do distributed training via parameter server, set <code>distributed_backend='ps'</code>	<code>None</code>

**Note on `TF_CONFIG`**

You will also need the network addresses and ports of the cluster for the `tf.train.ClusterSpec`, so Azure Machine Learning sets the `TF_CONFIG` environment variable for you.

The `TF_CONFIG` environment variable is a JSON string. Here is an example of the variable for a parameter server:

```

TF_CONFIG='{
  "cluster": {
    "ps": ["host0:2222", "host1:2222"],
    "worker": ["host2:2222", "host3:2222", "host4:2222"],
  },
  "task": {"type": "ps", "index": 0},
  "environment": "cloud"
}'

```

If you are using TensorFlow's high level `tf.estimator` API, TensorFlow will parse this `TF_CONFIG` variable and build the cluster spec for you.

If you are instead using TensorFlow's lower-level core APIs for training, you need to parse the `TF_CONFIG` variable and build the `tf.train.ClusterSpec` yourself in your training code. In [this example](#), you would do so in **your training script** as follows:

```

import os, json
import tensorflow as tf

tf_config = os.environ.get('TF_CONFIG')
if not tf_config or tf_config == "":
    raise ValueError("TF_CONFIG not found.")
tf_config_json = json.loads(tf_config)
cluster_spec = tf.train.ClusterSpec(tf_config_json)

```

Once you've finished writing your training script and creating the TensorFlow object, you can submit your training

job:

```
run = exp.submit(tf_est)
```

## Examples

Explore various [notebooks on distributed deep learning on Github](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

## Next steps

- [Track run metrics during training](#)
- [Tune hyperparameters](#)
- [Deploy a trained model](#)

# Tune hyperparameters for your model with Azure Machine Learning service

1/29/2019 • 12 minutes to read

Efficiently tune hyperparameters for your model using Azure Machine Learning service. Hyperparameter tuning includes the following steps:

- Define the parameter search space
- Specify a primary metric to optimize
- Specify early termination criteria for poorly performing runs
- Allocate resources for hyperparameter tuning
- Launch an experiment with the above configuration
- Visualize the training runs
- Select the best performing configuration for your model

## What are hyperparameters?

Hyperparameters are adjustable parameters you choose to train a model that govern the training process itself. For example, to train a deep neural network, you decide the number of hidden layers in the network and the number of nodes in each layer prior to training the model. These values usually stay constant during the training process.

In deep learning / machine learning scenarios, model performance depends heavily on the hyperparameter values selected. The goal of hyperparameter exploration is to search across various hyperparameter configurations to find a configuration that results in the best performance. Typically, the hyperparameter exploration process is painstakingly manual, given that the search space is vast and evaluation of each configuration can be expensive.

Azure Machine Learning allows you to automate hyperparameter exploration in an efficient manner, saving you significant time and resources. You specify the range of hyperparameter values and a maximum number of training runs. The system then automatically launches multiple simultaneous runs with different parameter configurations and finds the configuration that results in the best performance, measured by the metric you choose. Poorly performing training runs are automatically early terminated, reducing wastage of compute resources. These resources are instead used to explore other hyperparameter configurations.

## Define search space

Automatically tune hyperparameters by exploring the range of values defined for each hyperparameter.

### Types of hyperparameters

Each hyperparameter can either be discrete or continuous.

#### 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` takes on one of the values [16, 32, 64, 128] and `number_of_hidden_layers` takes on one of the values [1, 2, 3, 4].

Advanced discrete hyperparameters can also be specified using a distribution. The following distributions are supported:

- `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

Continuous hyperparameters are specified as a distribution over a continuous range of values. Supported distributions include:

- `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**

You can also specify the parameter sampling method to use over the hyperparameter space definition. Azure Machine Learning service supports random sampling, grid sampling, and Bayesian sampling.

#### **Random sampling**

In random sampling, hyperparameter values are randomly selected from the defined search space. Random sampling allows the search space to include both discrete and continuous hyperparameters.

```

from azureml.train.hyperdrive import RandomParameterSampling
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 performs a simple grid search over all feasible values in the defined search space. It can only be used with hyperparameters specified using `choice`. For example, the following space has a total of six samples:

```
from azureml.train.hyperdrive import GridParameterSampling
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 and makes intelligent choices on the hyperparameter values to sample next. It picks the sample based on how the previous samples performed, such that the new sample improves the reported primary metric.

When you use Bayesian sampling, the number of concurrent runs has an impact on the effectiveness of the tuning process. Typically, a smaller number of concurrent runs can lead to better sampling convergence, since the smaller degree of parallelism increases the number of runs that benefit from previously completed runs.

Bayesian sampling supports only `choice` and `uniform` distributions over the search space.

```
from azureml.train.hyperdrive import BayesianParameterSampling
param_sampling = BayesianParameterSampling( {
    "learning_rate": uniform(0.05, 0.1),
    "batch_size": choice(16, 32, 64, 128)
})
```

### NOTE

Bayesian sampling does not support any early termination policy (See [Specify an early termination policy](#)). When using Bayesian parameter sampling, set `early_termination_policy = None`, or leave off the `early_termination_policy` parameter.

## Specify primary metric

Specify the primary metric you want the hyperparameter tuning experiment to optimize. Each training run is evaluated for the primary metric. Poorly performing runs (where the primary metric does not meet criteria set by the early termination policy) will be terminated. In addition to the primary metric name, you also specify the goal of the optimization - whether to maximize or minimize the primary metric.

- `primary_metric_name`: The name of the primary metric to optimize. The name of the primary metric needs to exactly match the name of the metric logged by the training script. See [Log metrics for hyperparameter tuning](#).
- `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
```

Optimize the runs to maximize "accuracy". Make sure to log this value in your training script.

### Log metrics for hyperparameter tuning

The training script for your model must log the relevant metrics during model training. When you configure the

hyperparameter tuning, you specify the primary metric to use for evaluating run performance. (See [Specify a primary metric to optimize](#).) In your training script, you must log this metric so it is available to the hyperparameter tuning process.

Log this 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 "accuracy", which is used as the primary metric. Each time the metric is logged it is received by the hyperparameter tuning service. It is up to the model developer to determine how frequently to report this metric.

## Specify early termination policy

Terminate poorly performing runs automatically with an early termination policy. Termination reduces wastage of resources and instead uses these resources for exploring other parameter configurations.

When using an early termination policy, you can configure the following parameters that control when a policy is applied:

- `evaluation_interval`: the frequency for applying the policy. Each time the training script logs the primary metric counts as one interval. Thus 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 the training script reports the primary metric. If not specified, `evaluation_interval` is set to 1 by default.
- `delay_evaluation`: delays the first policy evaluation for a specified number of intervals. It is an optional parameter that allows all configurations to run for an initial minimum number of intervals, avoiding premature termination of training runs. If specified, the policy applies every multiple of `evaluation_interval` that is greater than or equal to `delay_evaluation`.

Azure Machine Learning service supports the following Early Termination Policies.

### Bandit policy

Bandit is a termination policy based on slack factor/slack amount and evaluation interval. The policy early terminates any runs where the primary metric is not within the specified slack factor / slack amount with respect to the best performing training run. It takes 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 being applied at interval 10. Assume that the best performing run at interval 10 reported a primary metric 0.8 with a goal to maximize the primary metric. If the policy was specified with a `slack_factor` of 0.2, any training runs, whose best metric at interval 10 is less than  $0.66 (0.8/(1 + slack\_factor))$  will be terminated. If instead, the policy was specified with a `slack_amount` of 0.2, any training runs, whose best metric at interval 10 is less than  $0.6 (0.8 - slack\_amount)$  will be terminated.

- `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 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 whose performance is worse than the median of the running 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 will be 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 given percentage of lowest performing runs at each evaluation interval. Runs are compared based on their performance on the primary metric and the lowest X% are terminated. It takes the following configuration parameters:

- `truncation_percentage` : the percentage of lowest performing runs to terminate at each evaluation interval. Specify an integer value between 1 and 99.
- `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 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 will be terminated 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

If you want all training runs to run to completion, set policy to None. This will have the effect of not applying any early termination policy.

```
policy=None
```

### Default policy

If no policy is specified, the hyperparameter tuning service will let all training runs run to completion.

#### NOTE

If you are looking for a conservative policy that provides savings without terminating promising jobs, you can use 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).

## Allocate resources

Control your resource budget for your hyperparameter tuning experiment by specifying the maximum total number of training runs. Optionally specify the maximum duration for your hyperparameter tuning experiment.

- `max_total_runs` : Maximum total number of training runs that will be created. Upper bound - there may be fewer runs, for instance, if the hyperparameter space is finite and has fewer samples. Must be a number between 1 and 1000.
- `max_duration_minutes` : Maximum duration in minutes of the hyperparameter tuning experiment. Parameter is optional, and if present, any runs that would be running after this duration are automatically 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` : Maximum number of runs to run concurrently at any given moment. If none specified, all `max_total_runs` will be launched in parallel. If specified, must be a number between 1 and 100.

### NOTE

The number of concurrent runs is gated on the resources available in the specified compute target. Hence, you need to ensure that the compute target has the available resources for the desired concurrency.

Allocate resources for hyperparameter tuning:

```
max_total_runs=20,  
max_concurrent_runs=4
```

This code configures the hyperparameter tuning experiment to use a maximum of 20 total runs, running 4 configurations at a time.

## Configure experiment

Configure your hyperparameter tuning experiment using the defined hyperparameter search space, early termination policy, primary metric, and resource allocation from the sections above. Additionally, provide an `estimator` that will be called with the sampled hyperparameters. The `estimator` describes the training script you run, the resources per job (single or multi-gpu), and the compute target to use. Since concurrency for your hyperparameter tuning experiment is gated on the resources available, ensure that the compute target specified in the `estimator` has sufficient resources for your desired concurrency. (For more information on estimators, see [how to train models](#).)

Configure your hyperparameter tuning experiment:

```
from azureml.train.hyperdrive import HyperDriveRunConfig
hyperdrive_run_config = HyperDriveRunConfig(estimator=estimator,
                                             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

Once you define your hyperparameter tuning configuration, submit an experiment:

```
from azureml.core.experiment import Experiment
experiment = Experiment(workspace, experiment_name)
hyperdrive_run = experiment.submit(hyperdrive_run_config)
```

`experiment_name` is the name you assign to your hyperparameter tuning experiment, and `workspace` is the workspace in which you want to create the experiment (For more information on experiments, see [How does Azure Machine Learning service work?](#))

## Visualize experiment

The Azure Machine Learning SDK provides a Notebook widget that visualizes 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	
Status	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

## Output Logs

[2018-09-11T21:57:36.389083][CONTROLLER][INFO]Experiment was 'ExperimentStatus.RUNNING', is 'ExperimentStatus.FINISHED'.

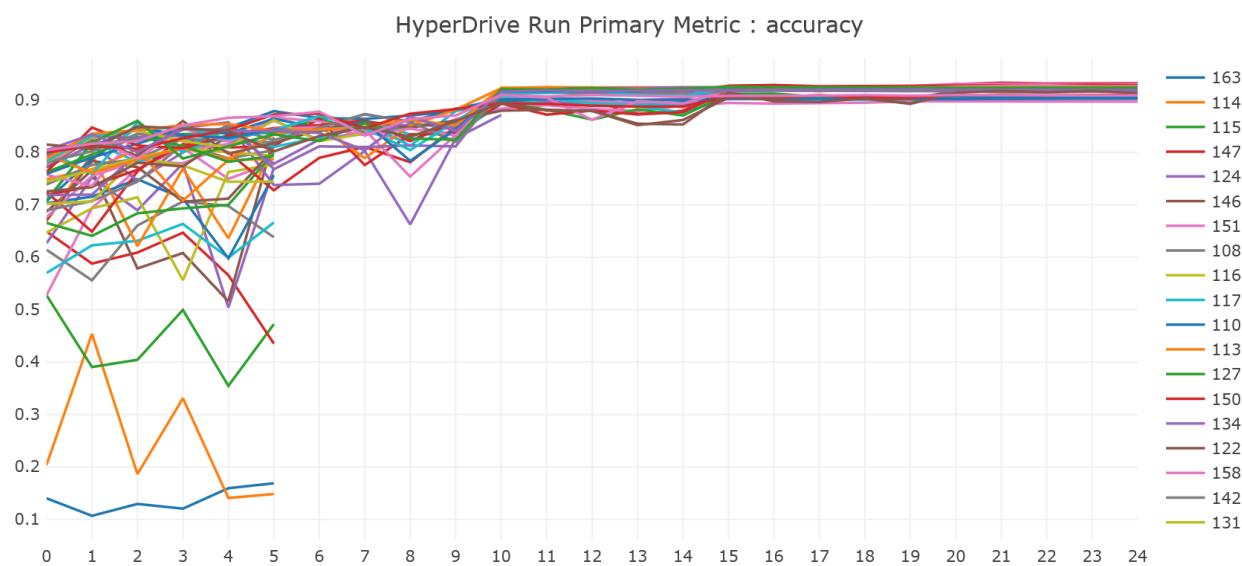
Run is completed.

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	
105	0.9283999800682068	Completed	Sep 4, 2018 2:56 PM	6:04:28	swatig-cifar_1536098154351_3	
104	0.9247000217437744	Completed	Sep 4, 2018 2:56 PM	3:32:35	swatig-cifar_1536098154351_2	
106	0.9251000285148621	Completed	Sep 4, 2018 2:56 PM	6:01:17	swatig-cifar_1536098154351_1	
107	0.9108999967575073	Completed	Sep 4, 2018 6:29 PM	4:18:35	swatig-cifar_1536098154351_4	

Pages: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ... Next Last

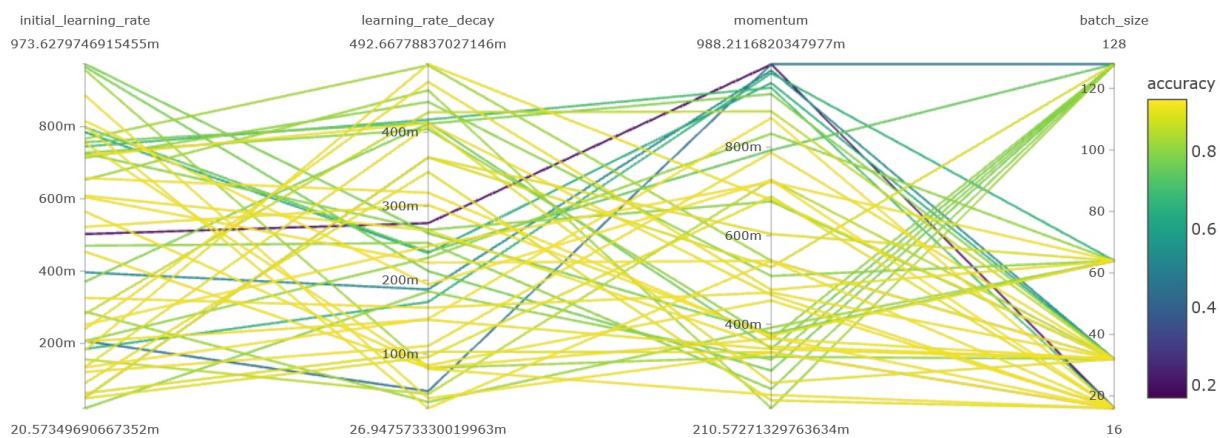
\* The best metric field is obtained from the min/max of primary metric achieved by a run

You can also visualize the performance of each of the runs as training progresses.

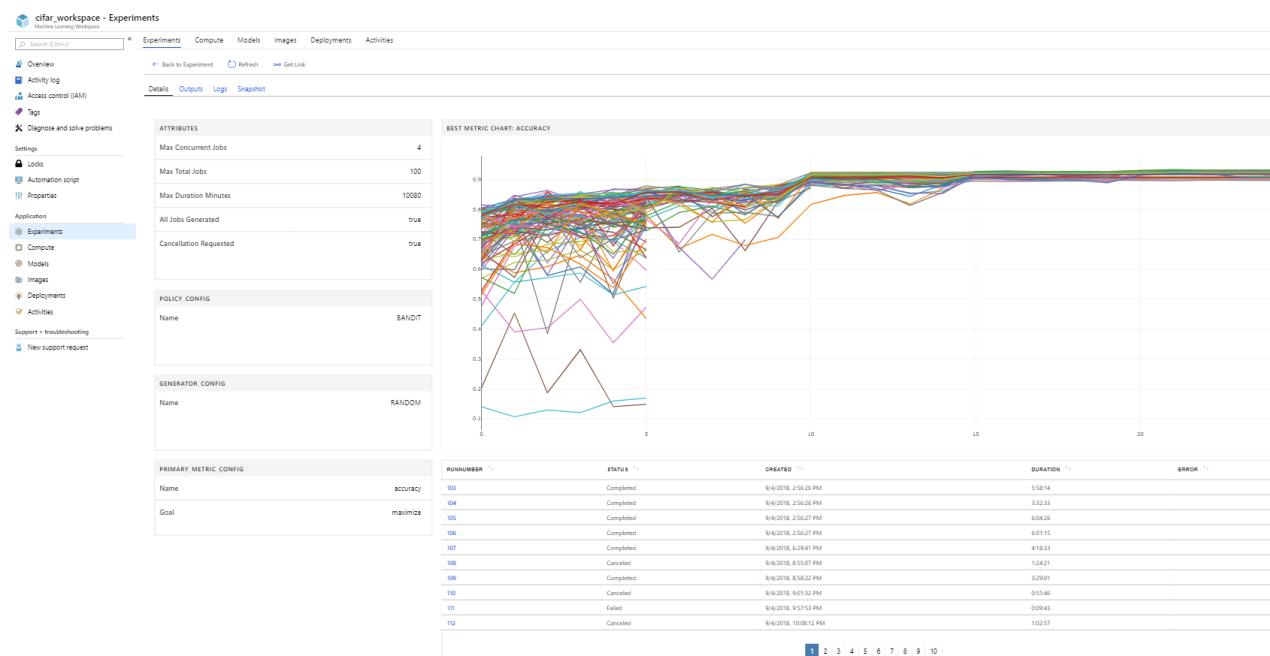


Additionally, you can visually identify the correlation between performance and values of individual hyperparameters using a Parallel Coordinates Plot.

### Parallel Coordinates Chart



You can visualize all your hyperparameter tuning runs in the Azure web portal as well. For more information on how to view an experiment in the web 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 the corresponding 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 these notebooks:

- [how-to-use-azureml/training-with-deep-learning/train-hyperparameter-tune-deploy-with-pytorch](#)
- [how-to-use-azureml/training-with-deep-learning/train-hyperparameter-tune-deploy-with-tensorflow](#)

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 experiments and training metrics in Azure Machine Learning

3/4/2019 • 15 minutes to read

In the Azure Machine Learning service, you can track your experiments and monitor metrics to enhance the model creation process. In this article, you'll learn about the different ways to add logging to your training script, how to submit the experiment with **start\_logging** and **ScriptRunConfig**, how to check the progress of a running job, and how to view the results of a run.

## List of training metrics

The following metrics can be added to a run while training an experiment. To view a more detailed list of what can be tracked on a run, see the [Run class reference documentation](#).

Type	Python Function	Notes
Scalar values	Function: <pre>run.log(name, value, description='')</pre> Example: <pre>run.log("accuracy", 0.95)</pre>	Log a numerical or string value to the run with the given name. Logging a metric to a run causes that metric to be stored in the run record in the experiment. You can log the same metric multiple times within a run, the result being considered a vector of that metric.
Lists	Function: <pre>run.log_list(name, value, description='')</pre> Example: <pre>run.log_list("accuracies", [0.6, 0.7, 0.87])</pre>	Log a list of values to the run with the given name.
Row	Function: <pre>'run.log_row(name, description=None, **kwargs)</pre> Example: <pre>run.log_row("Y over X", x=1, y=0.4)</pre>	Using <i>log_row</i> creates a metric with multiple columns as described in <i>kwargs</i> . Each named parameter generates a column with the value specified. <i>log_row</i> can be called once to log an arbitrary tuple, or multiple times in a loop to generate a complete table.
Table	Function: <pre>run.log_table(name, value, description='')</pre> Example: <pre>run.log_table("Y over X", {"x":[1, 2, 3], "y":[0.6, 0.7, 0.89]})</pre>	Log a dictionary object to the run with the given name.
Images	Function: <pre>run.log_image(name, path=None, plot=None)</pre> Example: <pre>run.log_image("ROC", plt)</pre>	Log an image to the run record. Use <i>log_image</i> to log an image file or a matplotlib plot to the run. These images will be visible and comparable in the run record.

TYPE	PYTHON FUNCTION	NOTES
Tag a run	<p>Function:</p> <pre>run.tag(key, value=None)</pre> <p>Example:</p> <pre>run.tag("selected", "yes")</pre>	Tag the run with a string key and optional string value.
Upload file or directory	<p>Function:</p> <pre>run.upload_file(name, path_or_stream)</pre> <p>Example:</p> <pre>run.upload_file("best_model.pkl", "./model.pkl")</pre>	Upload a file to the run record. Runs automatically capture file in the specified output directory, which defaults to "./outputs" for most run types. Use <code>upload_file</code> only when additional files need to be uploaded or an output directory is not specified. We suggest adding <code>outputs</code> to the name so that it gets uploaded to the outputs directory. You can list all of the files that are associated with this run record by called <code>run.get_file_names()</code>

#### NOTE

Metrics for scalars, lists, rows, and tables can have type: float, integer, or string.

## Start logging metrics

If you want to track or monitor your experiment, you must add code to start logging when you submit the run. The following are ways to trigger the run submission:

- **Run.start\_logging** - Add logging functions to your training script and start an interactive logging session in the specified experiment. **start\_logging** creates an interactive run for use in scenarios such as notebooks. Any metrics that are logged during the session are added to the run record in the experiment.
- **ScriptRunConfig** - Add logging functions to your training script and load the entire script folder with the run. **ScriptRunConfig** is a class for setting up configurations for script runs. With this option, you can add monitoring code to be notified of completion or to get a visual widget to monitor.

## Set up the workspace

Before adding logging and submitting an experiment, you must set up the workspace.

1. Load the workspace. To learn more about setting the workspace configuration, follow the [quickstart](#).

```
from azureml.core import Experiment, Run, Workspace
import azureml.core

ws = Workspace(workspace_name = <<workspace_name>>,
               subscription_id = <<subscription_id>>,
               resource_group = <<resource_group>>)
```

## Option 1: Use start\_logging

**start\_logging** creates an interactive run for use in scenarios such as notebooks. Any metrics that are logged during the session are added to the run record in the experiment.

The following example trains a simple sklearn Ridge model locally in a local Jupyter notebook. To learn more

about submitting experiments to different environments, see [Set up compute targets for model training with Azure Machine Learning service](#).

1. Create a training script in a local Jupyter notebook.

```
# load diabetes dataset, a well-known small dataset that comes with scikit-learn
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
from sklearn.externals import joblib

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}
}
reg = Ridge(alpha = 0.03)
reg.fit(data['train']['X'], data['train']['y'])
preds = reg.predict(data['test']['X'])
print('Mean Squared Error is', mean_squared_error(preds, data['test']['y']))
joblib.dump(value = reg, filename = 'model.pkl');
```

2. Add experiment tracking using the Azure Machine Learning service SDK, and upload a persisted model into the experiment run record. The following code adds tags, logs, and uploads a model file to the experiment run.

```
# Get an experiment object from Azure Machine Learning
experiment = Experiment(workspace = ws, name = "train-within-notebook")

# Create a run object in the experiment
run = experiment.start_logging()# Log the algorithm parameter alpha to the run
run.log('alpha', 0.03)

# Create, fit, and test the scikit-learn Ridge regression model
regression_model = Ridge(alpha=0.03)
regression_model.fit(data['train']['X'], data['train']['y'])
preds = regression_model.predict(data['test']['X'])

# Output the Mean Squared Error to the notebook and to the run
print('Mean Squared Error is', mean_squared_error(data['test']['y'], preds))
run.log('mse', mean_squared_error(data['test']['y'], preds))

# Save the model to the outputs directory for capture
joblib.dump(value=regression_model, filename='outputs/model.pkl')

# Take a snapshot of the directory containing this notebook
run.take_snapshot('./')

# Complete the run
run.complete()
```

The script ends with `run.complete()`, which marks the run as completed. This function is typically used in interactive notebook scenarios.

## Option 2: Use ScriptRunConfig

**ScriptRunConfig** is a class for setting up configurations for script runs. With this option, you can add monitoring code to be notified of completion or to get a visual widget to monitor.

This example expands on the basic sklearn Ridge model from above. It does a simple parameter sweep to sweep over alpha values of the model to capture metrics and trained models in runs under the experiment. The example runs locally against a user-managed environment.

1. Create a training script `train.py`.

```
# train.py

import os
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
from azureml.core.run import Run
from sklearn.externals import joblib

import numpy as np

#os.makedirs('./outputs', exist_ok = True)

X, y = load_diabetes(return_X_y = True)

run = Run.get_context()

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}}

# list of numbers from 0.0 to 1.0 with a 0.05 interval
alphas = mylib.get_alphas()

for alpha in alphas:
    # Use Ridge algorithm to create a regression model
    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"])
    # log the alpha and mse values
    run.log('alpha', alpha)
    run.log('mse', mse)

    model_file_name = 'ridge_{0:.2f}.pkl'.format(alpha)
    # save model in the outputs folder so it automatically get uploaded
    with open(model_file_name, "wb") as file:
        joblib.dump(value = reg, filename = model_file_name)

    # upload the model file explicitly into artifacts
    run.upload_file(name = model_file_name, path_or_stream = model_file_name)

    # register the model
    #run.register_model(file_name = model_file_name)

    print('alpha is {0:.2f}, and mse is {1:.2f}'.format(alpha, mse))
```

2. The `train.py` script references `mylib.py` which allows you to get the list of alpha values to use in the ridge model.

```
# mylib.py

import numpy as np

def get_alphas():
    # list of numbers from 0.0 to 1.0 with a 0.05 interval
    return np.arange(0.0, 1.0, 0.05)
```

### 3. Configure a user-managed local environment.

```
from azureml.core.runconfig import RunConfiguration

# Editing a run configuration property on-fly.
run_config_user_managed = RunConfiguration()

run_config_user_managed.environment.python.user_managed_dependencies = True

# You can choose a specific Python environment by pointing to a Python path
#run_config.environment.python.interpreter_path = '/home/user/miniconda3/envs/sdk2/bin/python'
```

### 4. Submit the `train.py` script to run in the user-managed environment. This whole script folder is submitted for training, including the `mylib.py` file.

```
from azureml.core import ScriptRunConfig

experiment = Experiment(workspace=ws, name="train-on-local")
src = ScriptRunConfig(source_directory = '.', script = 'train.py', run_config =
run_config_user_managed)
run = experiment.submit(src)
```

## Cancel a run

After a run is submitted, you can cancel it even if you have lost the object reference, as long as you know the experiment name and run id.

```
from azureml.core import Experiment
exp = Experiment(ws, "my-experiment-name")

# if you don't know the run id, you can list all runs under an experiment
for r in exp.get_runs():
    print(r.id, r.get_status())

# if you know the run id, you can "rehydrate" the run
from azureml.core import get_run
r = get_run(experiment=exp, run_id="my_run_id", rehydrate=True)

# check the returned run type and status
print(type(r), r.get_status())

# you can cancel a run if it hasn't completed or failed
if r.get_status() not in ['Complete', 'Failed']:
    r.cancel()
```

Please note that currently only `ScriptRun` and `PipelineRun` types support cancel operation.

Additionally, you can cancel a run through the CLI using the following command:

```
az ml run cancel -r <run_id> -p <project_path>
```

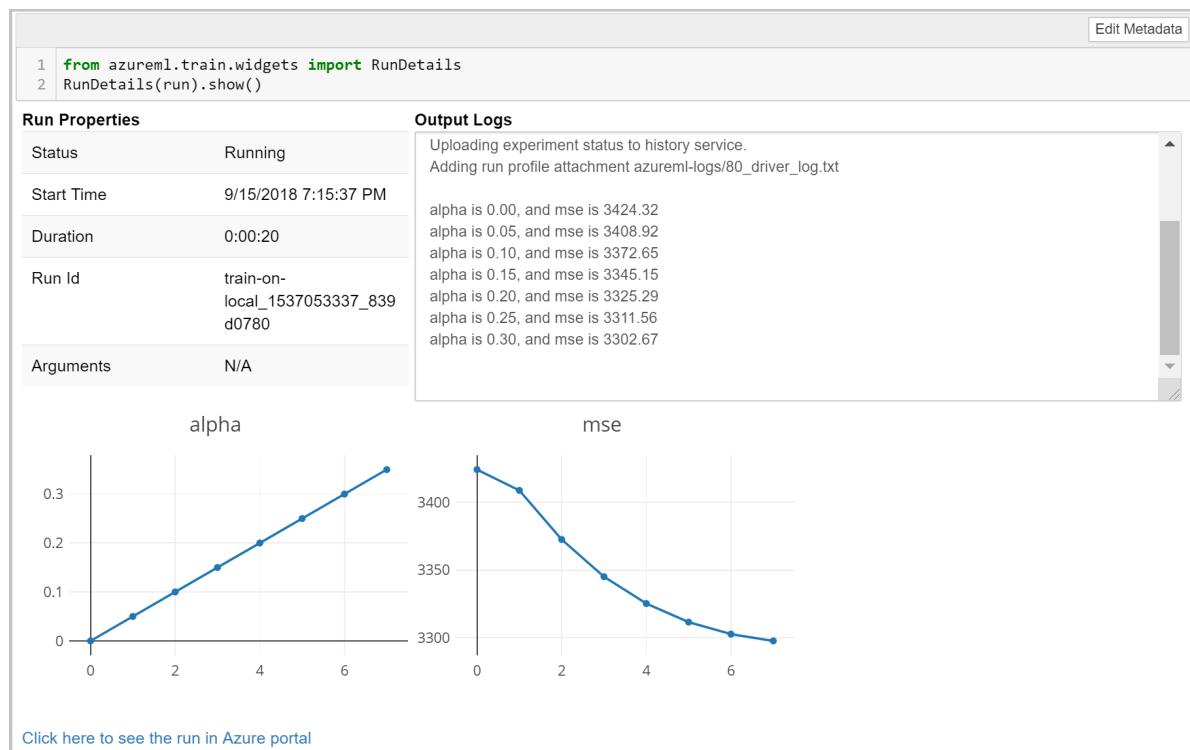
## View run details

### Monitor run with Jupyter notebook widgets

When you use the **ScriptRunConfig** method to submit runs, you can watch the progress of the run with a Jupyter notebook widget. Like the run submission, the widget is asynchronous and provides live updates every 10-15 seconds until the job completes.

1. View the Jupyter widget while waiting for the run to complete.

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



2. [For automated machine learning runs] To access the charts from a previous run. Please 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**

Status -	0	2	4	6	8	10	12	14	16	18	20	22	24

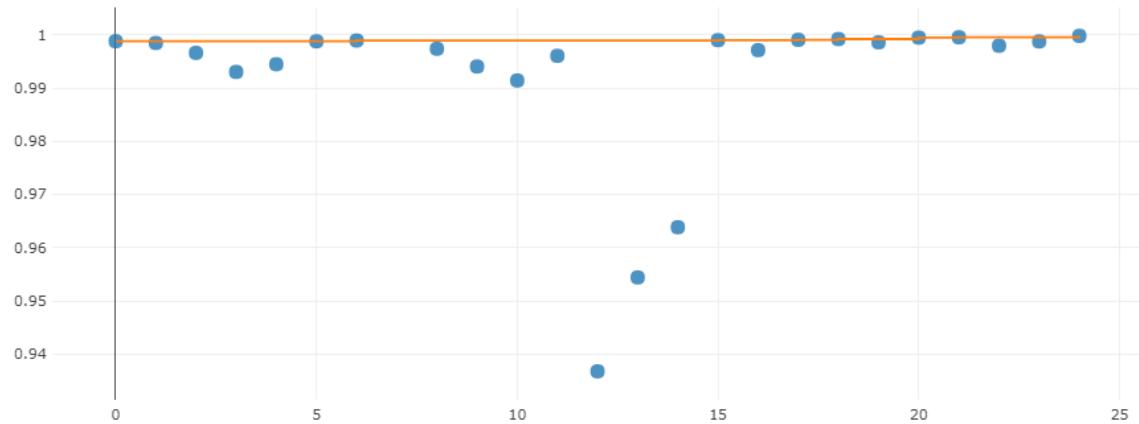
  

Iteration	Pipeline	Iteration metric	Best metric	Status	Duration	Started
0	StandardScalerWrapper, KNN	0.99883057	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
1	StandardScalerWrapper, KNN	0.99849239	0.99883057	Completed	0:00:00	Nov 26, 2018 1:30 PM
2	MaxAbsScaler, LightGBM	0.99664006	0.99883057	Completed	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](#)

To view further details of a pipeline click on the Pipeline you would like to explore in the table, and the charts will render in a pop-up from the Azure portal.

### Get log results upon completion

Model training and monitoring occur in the background so that you can run other tasks while you wait. You can also wait until the model has completed training before running more code. 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.

### Query run metrics

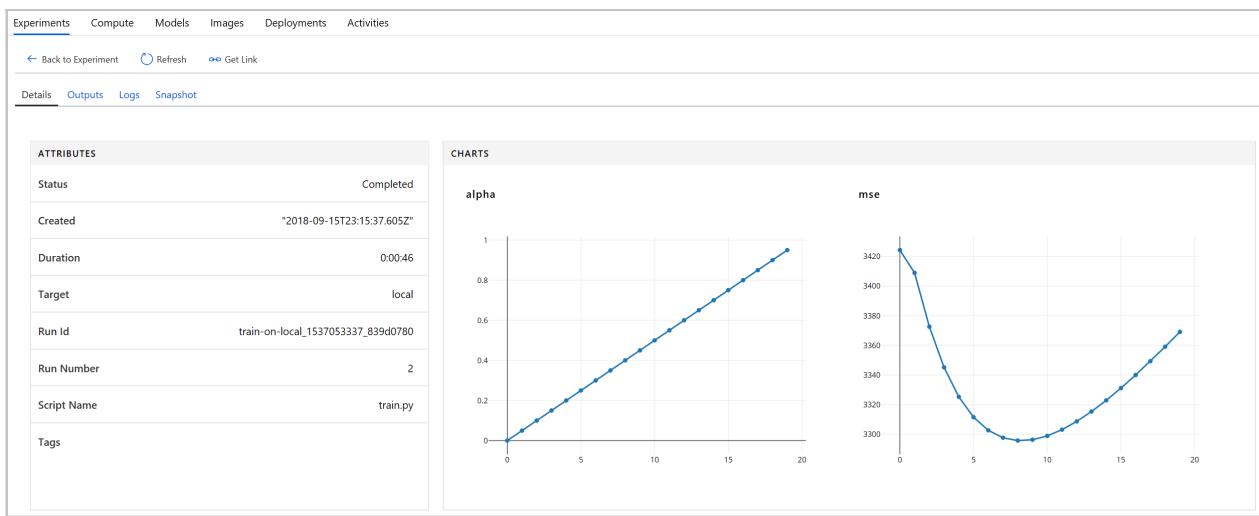
You can view the metrics of a trained model using `run.get_metrics()`. You can now get all of the metrics that were logged in the example above to determine the best model.

## View the experiment in the Azure portal

When an experiment has finished running, you can browse to the recorded experiment run record. You can do access the history in two ways:

- Get the URL to the run directly `print(run.get_portal_url())`
- View the run details by submitting the name of the run (in this case, `run`). This way points you to the experiment name, ID, type, status, details page, a link to the Azure portal, and a link to documentation.

The link for the run brings you directly to the run details page in the Azure portal. Here you can see any properties, tracked metrics, images, and charts that are logged in the experiment. In this case, we logged MSE and the alpha values.



You can also view any outputs or logs for the run, or download the snapshot of the experiment you submitted so you can share the experiment folder with others.

### Viewing charts in run details

There are various ways to use the logging APIs to record different types of metrics during a run and view them as charts in the Azure portal.

LOGGED VALUE	EXAMPLE CODE	VIEW 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

## Understanding automated ML charts

After submitting an automated ML job in a notebook, a history of these runs can be found in your machine learning service workspace.

Learn more about:

- [Charts and curves for classification models](#)
- [Charts and graphs for regression models](#)
- [Model explainability](#)

### View the run charts

1. Go to your workspace.
2. Select **Experiments** in the leftmost panel of your workspace.

The screenshot shows the Azure Machine Learning studio interface. On the left, there's a navigation sidebar with sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Locks, Automation script, Properties, Application, and Experiments. The 'Experiments' link is circled in red.

3. Select the experiment you are interested in.

## Experiments

Refresh

NAME ↑↓	CREATED ↑↓
<a href="#">energydemandforecasting</a>	10/31/2018, 10:44:51 AM
<a href="#">local-classification_1016</a>	10/16/2018, 11:02:25 AM
<a href="#">local-regression_ss_Demo</a>	10/9/2018, 8:37:07 AM
<a href="#">local-regression</a>	10/1/2018, 1:07:41 PM
<a href="#">local-classification</a>	10/1/2018, 1:07:03 PM
<a href="#">local-missing-data</a>	10/1/2018, 1:02:18 PM

4. In the table, select the Run Number.



5. In the table, select the Iteration Number for the model that you would like to explore further.

Iterations							
Iteration	Run_Preprocessor	Run_Algorithm	Normalized_Root_Mean_Squared_Error	Status	Created	Duration	
5	SparseNormalizer	DecisionTreeRegressor	0.06312059746871934	Completed	10/31/2018, 10:47:23 AM	23:58:34	
2	MaxAbsScaler	RandomForestRegressor	0.07021324452854548	Completed	10/31/2018, 10:46:00 AM	23:58:37	
1	SparseNormalizer	DecisionTreeRegressor	0.08182806797817412	Completed	10/31/2018, 10:45:33 AM	23:58:35	
9	StandardScalerWrapper	LassoLars	0.10910856865252272	Completed	10/31/2018, 10:49:11 AM	23:58:34	
7	StandardScalerWrapper	ElasticNet	0.10984011331262775	Completed	10/31/2018, 10:48:18 AM	23:58:33	
6	StandardScalerWrapper	ElasticNet	0.10988317343439677	Completed	10/31/2018, 10:47:50 AM	23:58:35	
4	MaxAbsScaler	ElasticNet	0.13711681172618576	Completed	10/31/2018, 10:46:56 AM	23:58:34	
3	SparseNormalizer	DecisionTreeRegressor	0.13961105249753472	Completed	10/31/2018, 10:46:30 AM	23:58:33	
8	StandardScalerWrapper	ElasticNet	0.14459679782725773	Completed	10/31/2018, 10:48:44 AM	23:58:34	
0	TruncatedSVDWrapper	DecisionTreeRegressor	0.14460494174552974	Completed	10/31/2018, 10:45:05 AM	23:58:35	

## Classification

For every classification model that you build by using the automated machine learning capabilities of Azure Machine Learning, you can see the following charts:

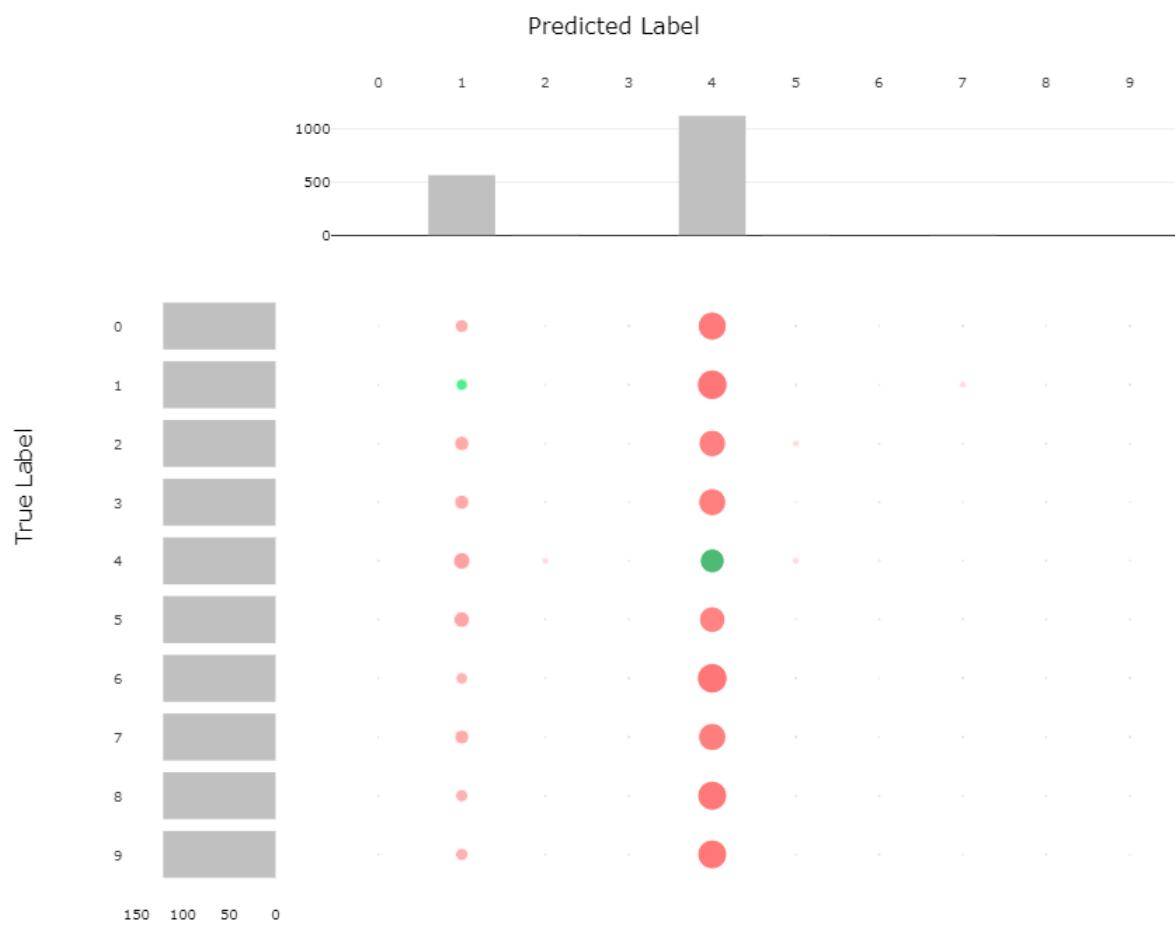
- [Confusion matrix](#)
- [Precision-Recall chart](#)
- [Receiver operating characteristics \(or ROC\)](#)
- [Lift curve](#)
- [Gains curve](#)
- [Calibration plot](#)

### Confusion matrix

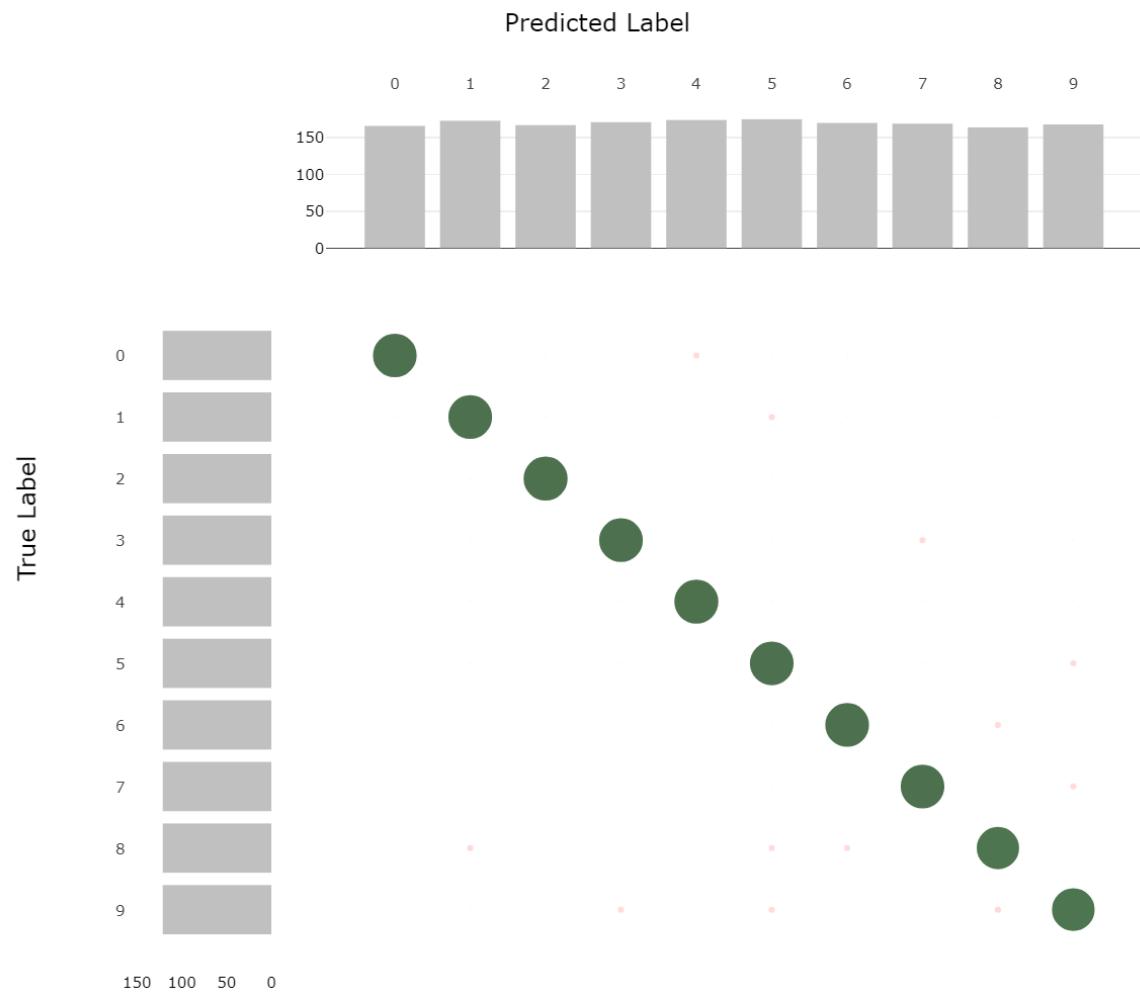
A confusion matrix is used to describe the performance of a classification model. Each row displays the instances of the true class, and each column represents the instances of the predicted class. The confusion matrix shows the correctly classified labels and the incorrectly classified labels for a given model.

For classification problems, Azure Machine Learning automatically provides a confusion matrix for each model that is built. For each confusion matrix, automated ML will show the correctly classified labels as green, and incorrectly classified labels as red. The size of the circle represents the number of samples in that bin. In addition, the frequency count of each predicted label and each true label is provided in the adjacent bar charts.

Example 1: A classification model with poor accuracy



Example 2: A classification model with high accuracy (ideal)

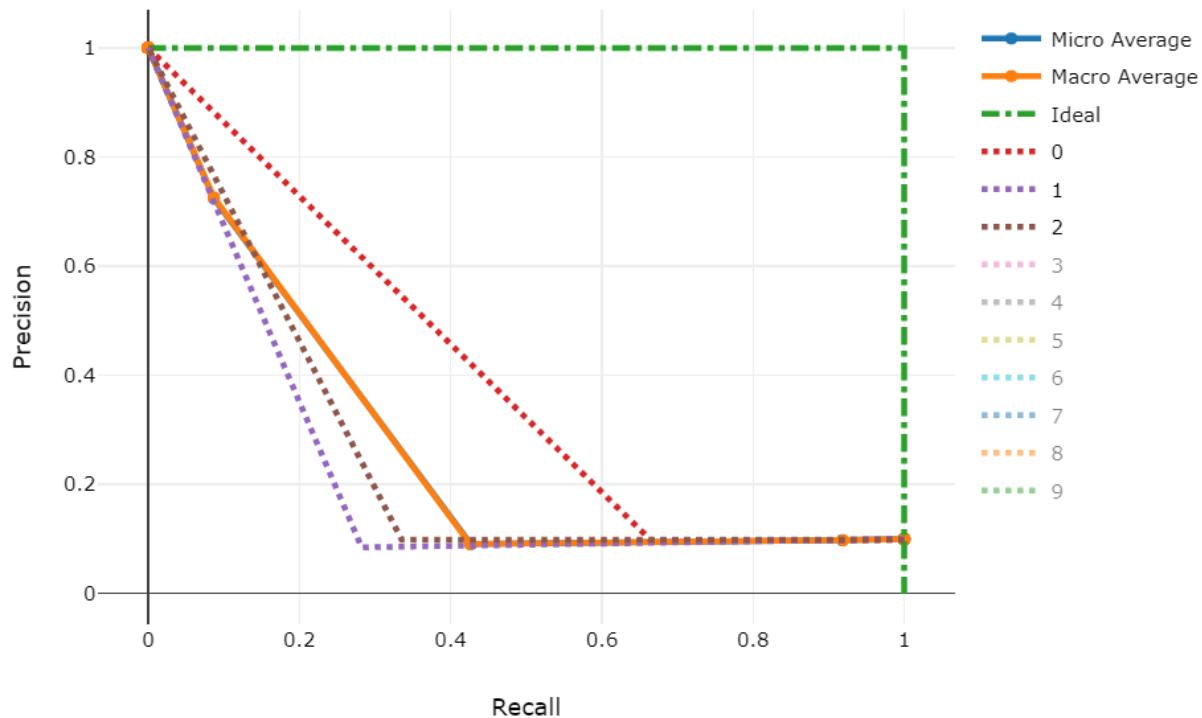


### Precision-recall chart

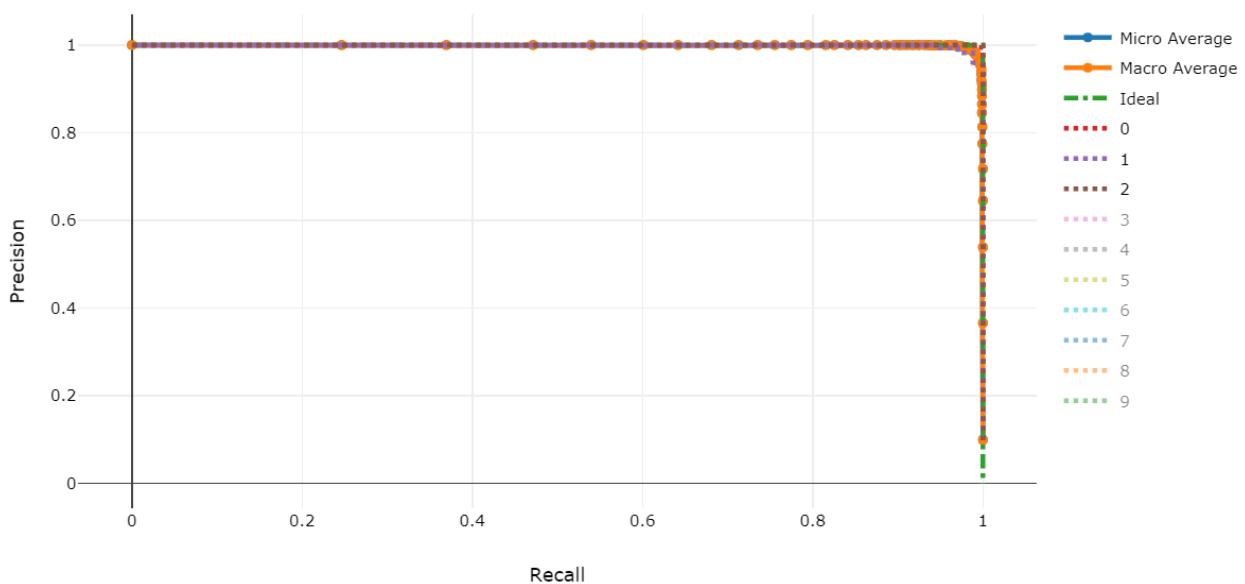
With this chart, you can compare the precision-recall curves for each model to determine which model has an acceptable relationship between precision and recall for your particular business problem. This chart shows Macro Average Precision-Recall, Micro Average Precision-Recall, and the precision-recall associated with all classes for a model.

The term Precision represents that ability for a classifier to label all instances correctly. Recall represents the ability for a classifier to find all instances of a particular label. The precision-recall curve shows the relationship between these two concepts. Ideally, the model would have 100% precision and 100% accuracy.

Example 1: A classification model with low precision and low recall



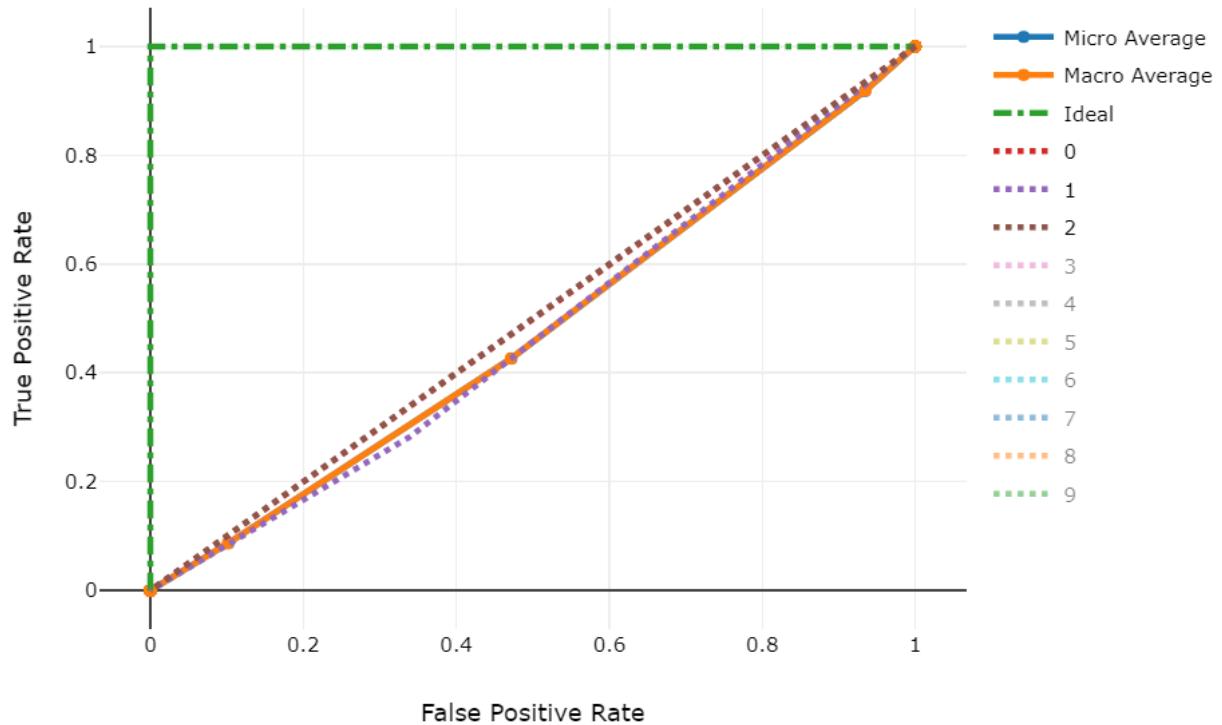
Example 2: A classification model with ~100% precision and ~100% recall (ideal)



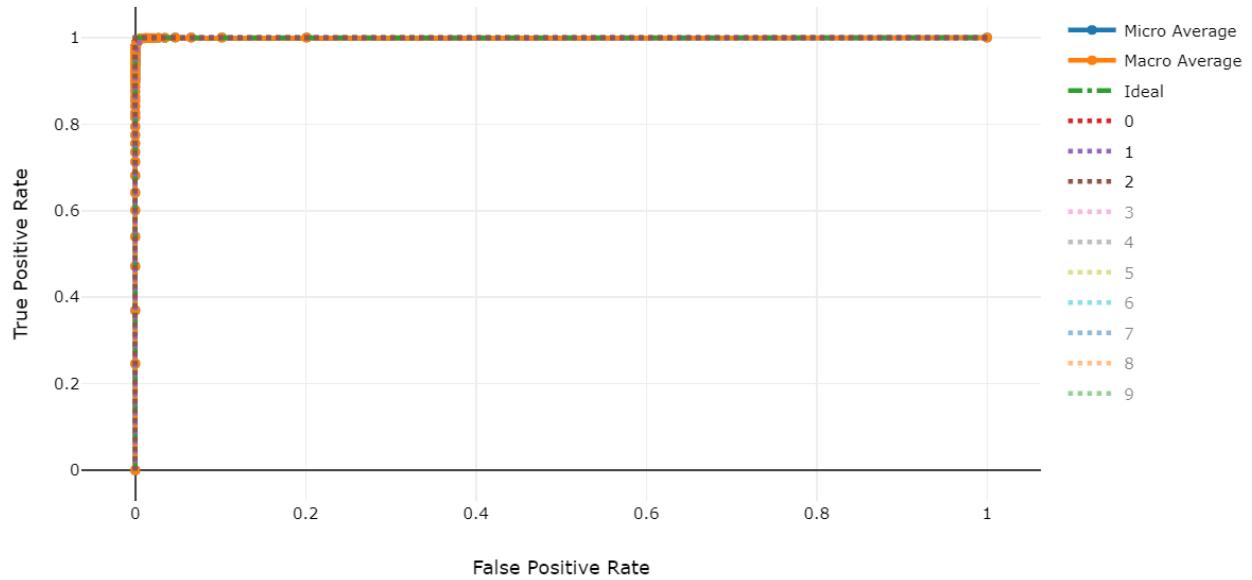
### ROC

Receiver operating characteristic (or ROC) is a plot of the correctly classified labels vs. the incorrectly classified labels for a particular model. The ROC curve can be less informative when training models on datasets with high bias, as it will not show the false positive labels.

### Example 1: A classification model with low true labels and high false labels



### Example 2: A classification model with high true labels and low false labels

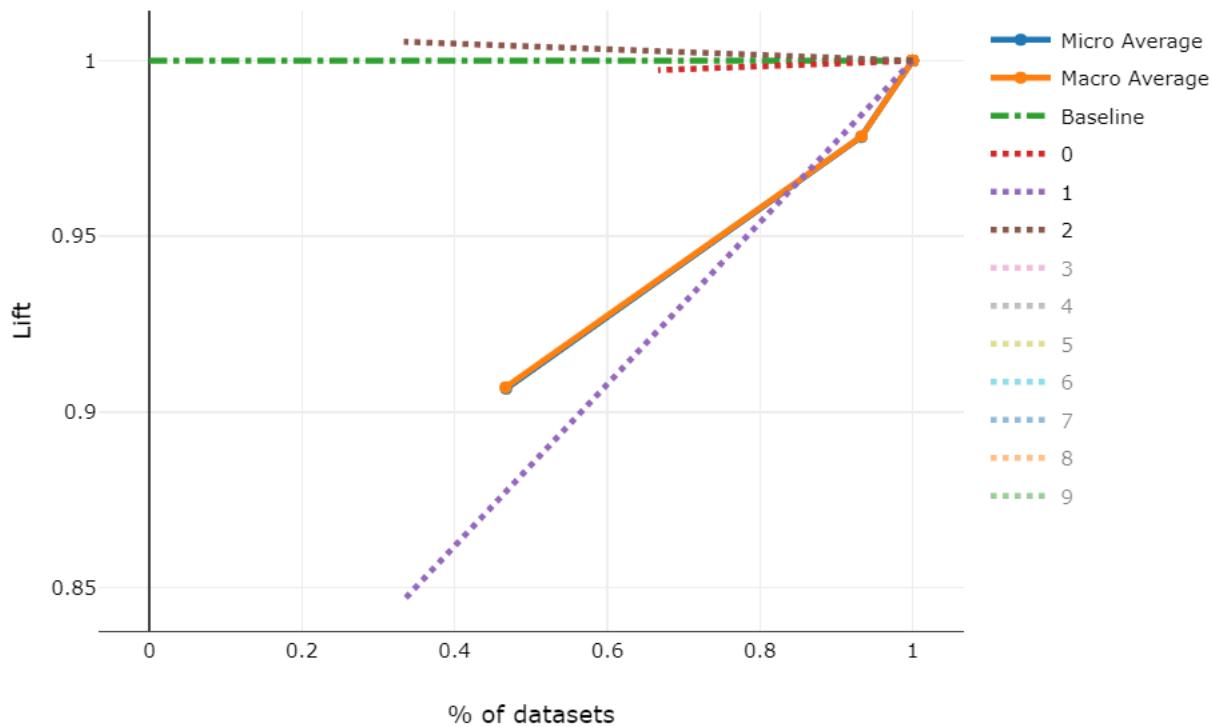


### Lift curve

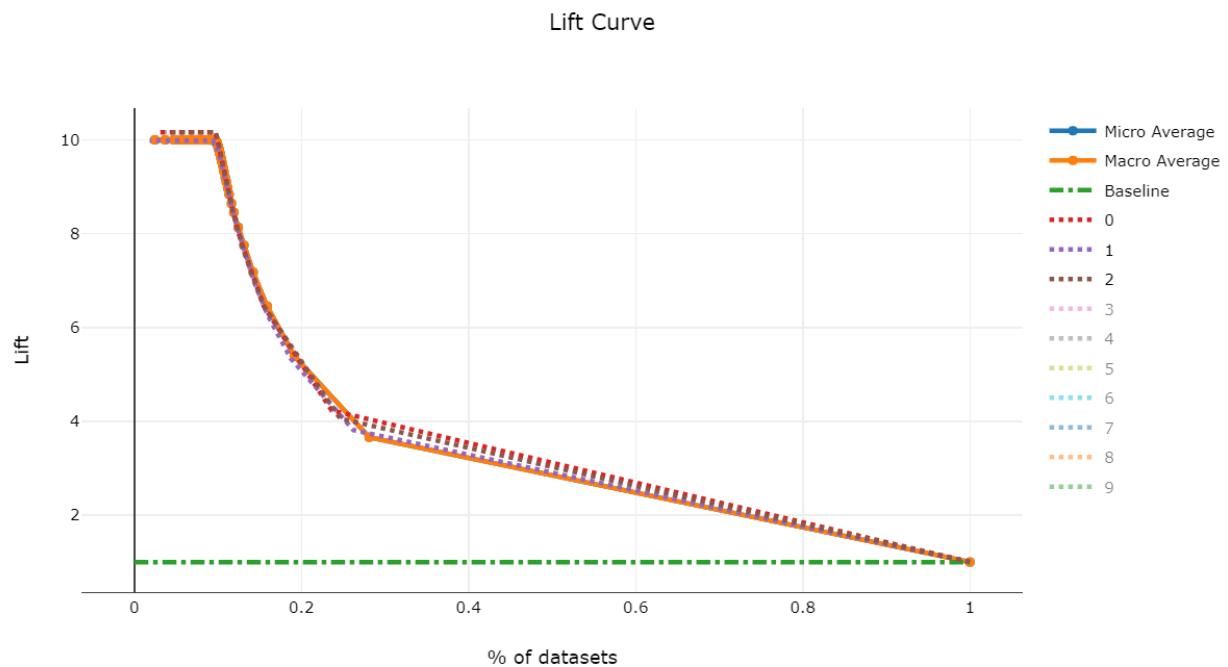
You can compare the lift of the model built automatically with Azure Machine Learning to the baseline in order to view the value gain of that particular model.

Lift charts are used to evaluate the performance of a classification model. It shows how much better you can expect to do with a model compared to without a model.

Example 1: Model performs worse than a random selection model



Example 2: Model performs better than a random selection model

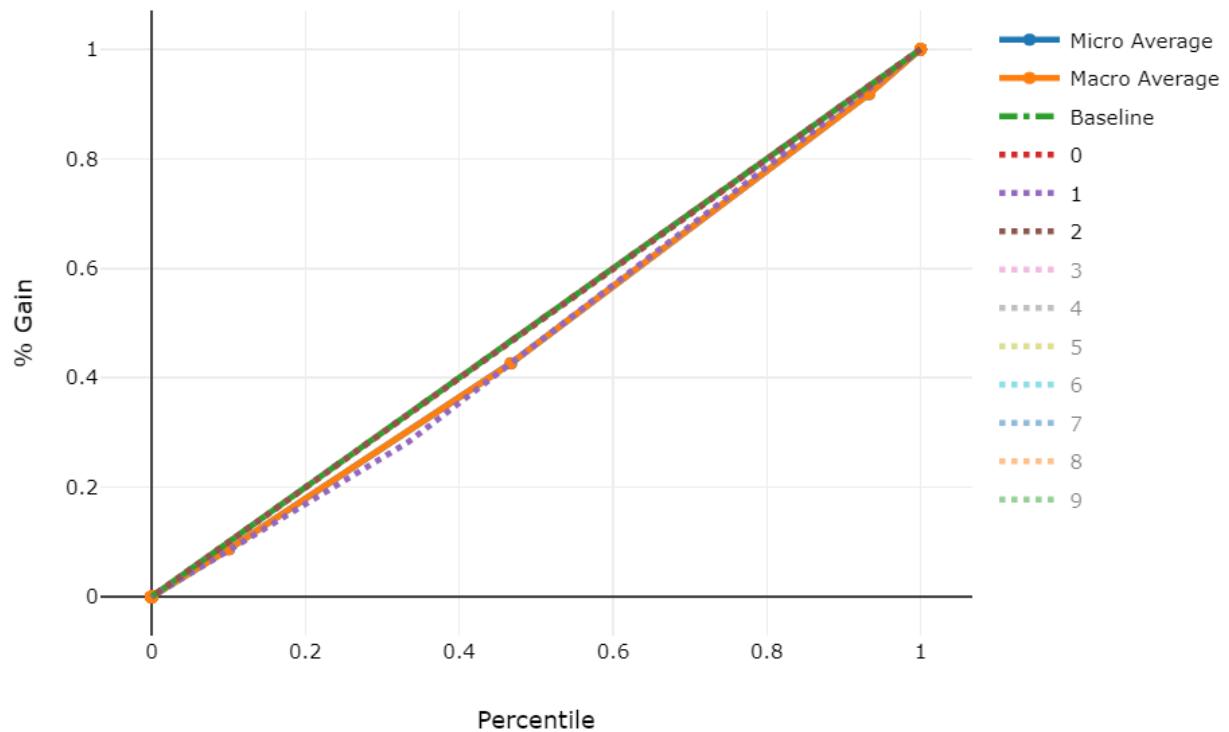


#### Gains curve

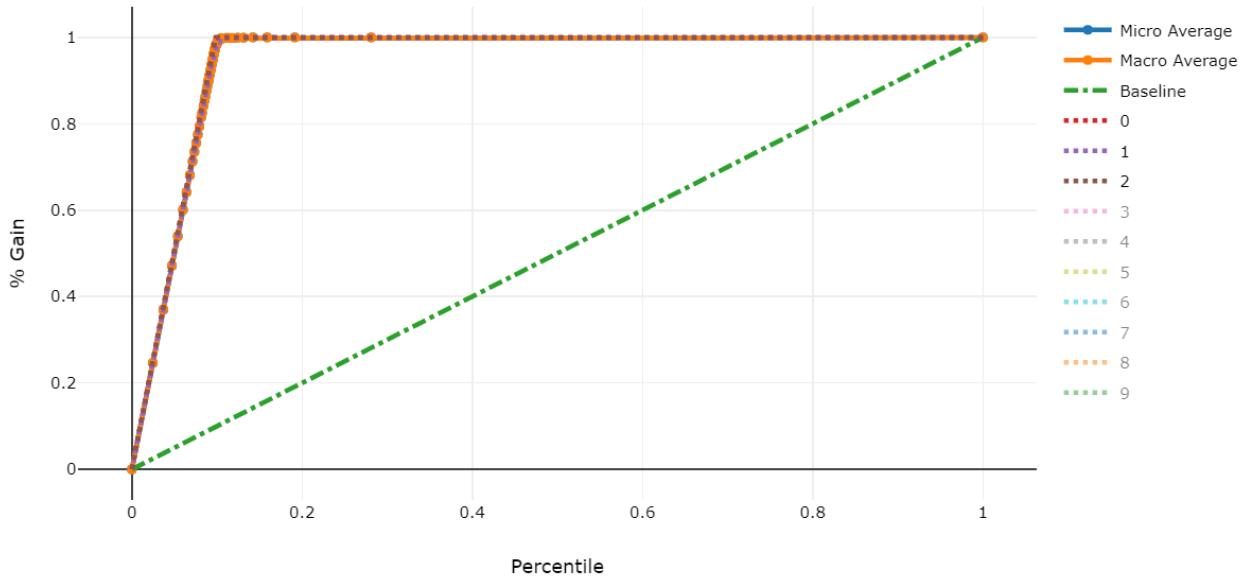
A gains chart evaluates the performance of a classification model by each portion of the data. It shows for each percentile of the data set, how much better you can expect to perform compared against a random selection model.

Use the cumulative gains chart to help you choose the classification cutoff using a percentage that corresponds to a desired gain from the model. This information provides another way of looking at the results in the accompanying lift chart.

Example 1: A classification model with minimal gain



Example 2: A classification model with significant gain

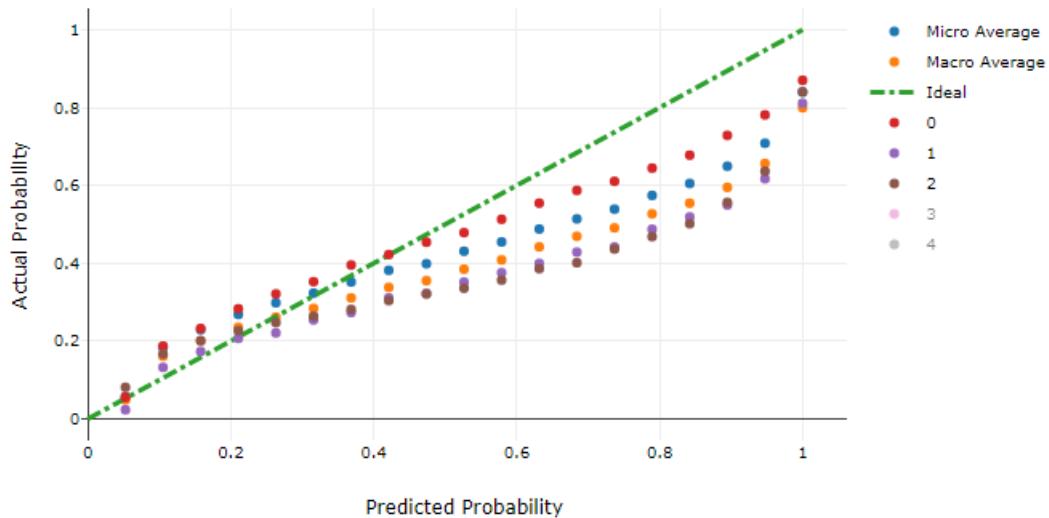


#### Calibration plot

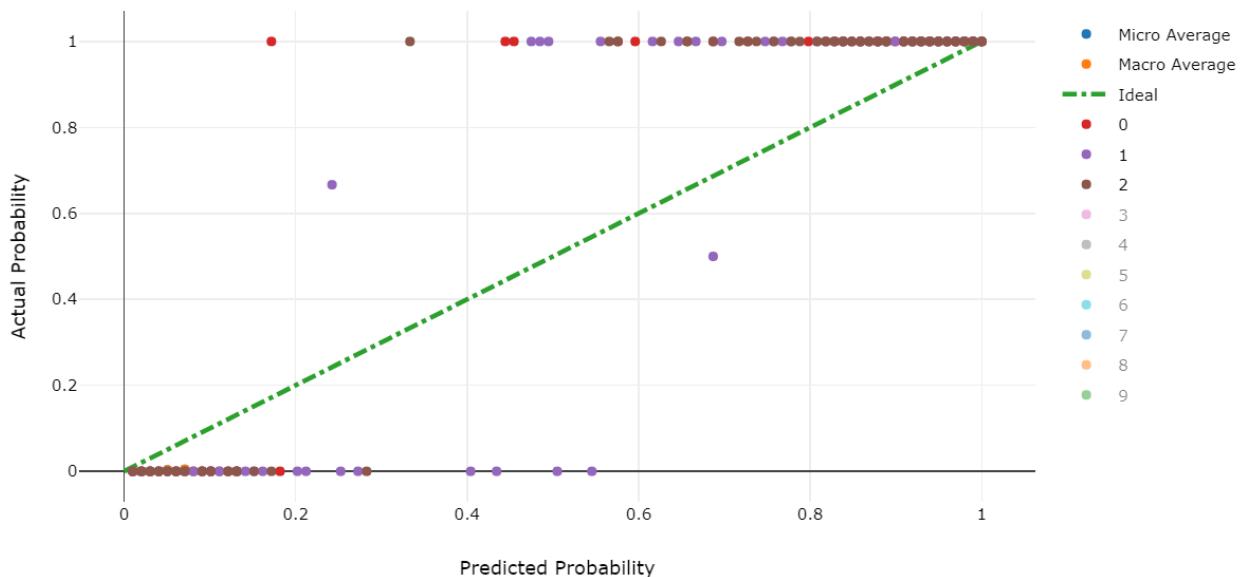
For all classification problems, you can review the calibration line for micro-average, macro-average, and each class in a given predictive model.

A calibration plot is used to display the confidence of a predictive model. It does this by showing the relationship between the predicted probability and the actual probability, where "probability" represents the likelihood that a particular instance belongs under some label. A well calibrated model aligns with the  $y=x$  line, where it is reasonably confident in its predictions. An over-confident model aligns with the  $y=0$  line, where the predicted probability is present but there is no actual probability.

Example 1: A more well-calibrated model



Example 2: An over-confident model



## Regression

For every regression model, you build using the automated machine learning capabilities of Azure Machine Learning, you can see the following charts:

- [Predicted vs. True](#)
- [Histogram of residuals](#)

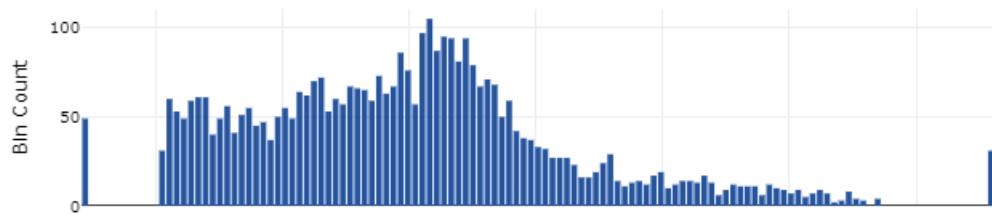
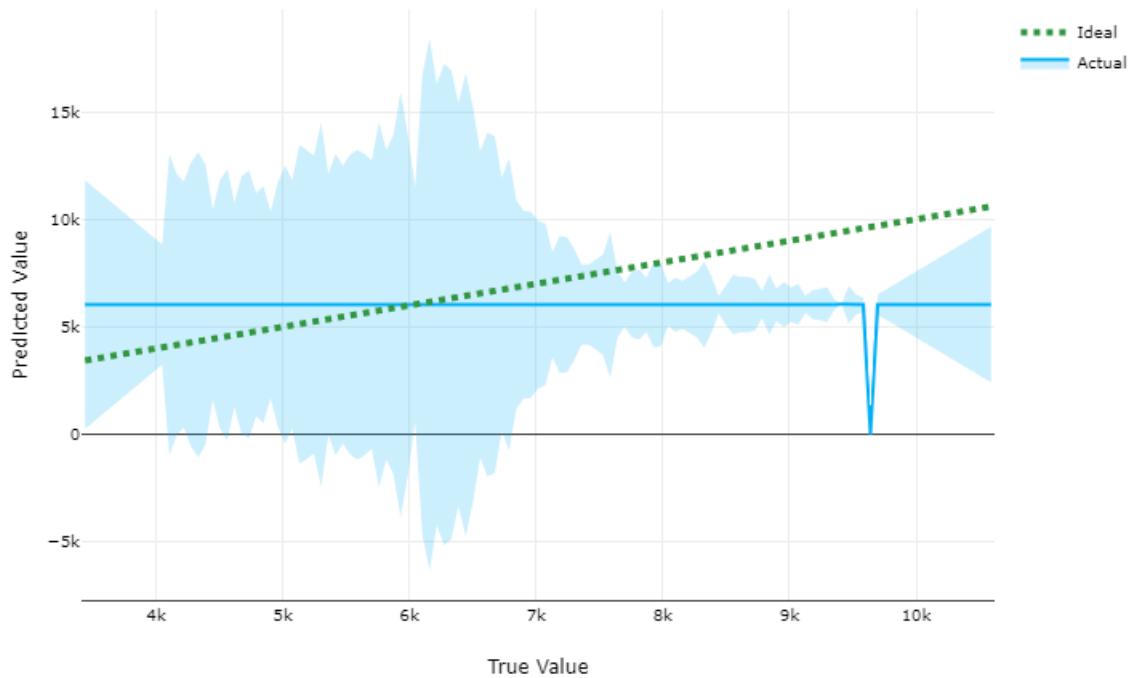
### Predicted vs. True

Predicted vs. True shows the relationship between a predicted value and its correlating true value for a regression problem. This graph can be used to measure performance of a model as the closer to the  $y=x$  line the predicted values are, the better the accuracy of a predictive model.

After each run, you can see a predicted vs. true graph for each regression model. To protect data privacy, values are binned together and the size of each bin is shown as a bar graph on the bottom portion of the chart area. You can compare the predictive model, with the lighter shade area showing error margins, against the ideal value of where the model should be.

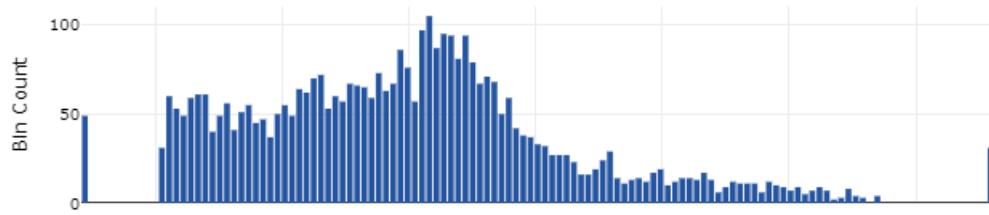
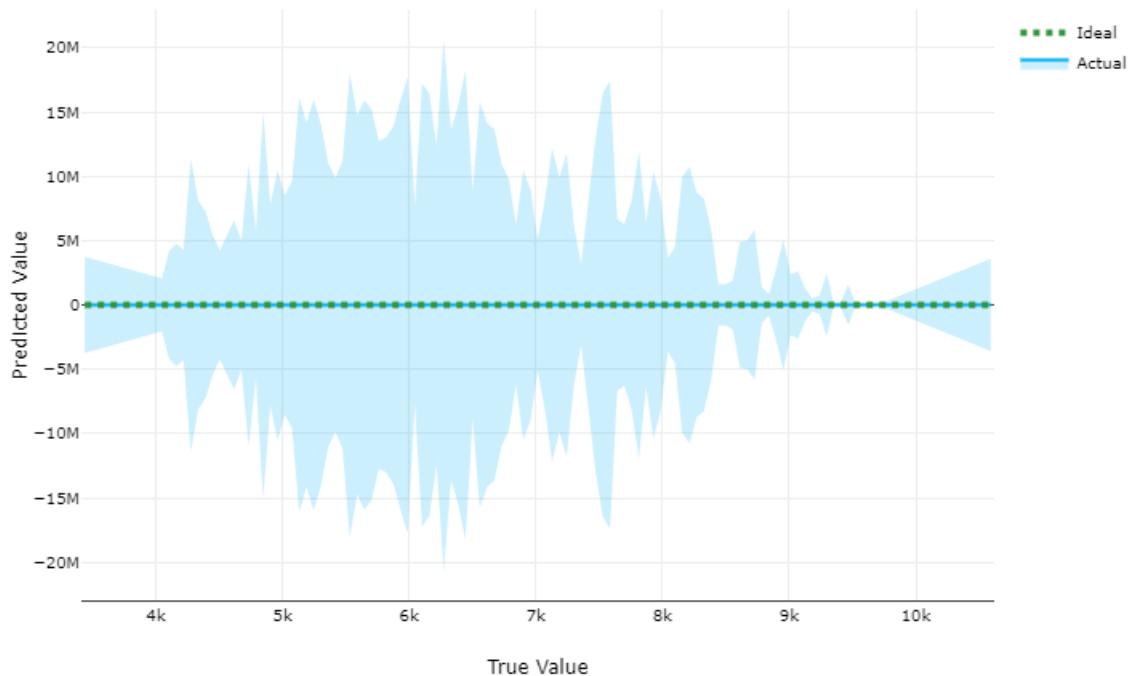
Example 1: A regression model with low accuracy in predictions

### Predicted vs. True



Example 2: A regression model with high accuracy in its predictions

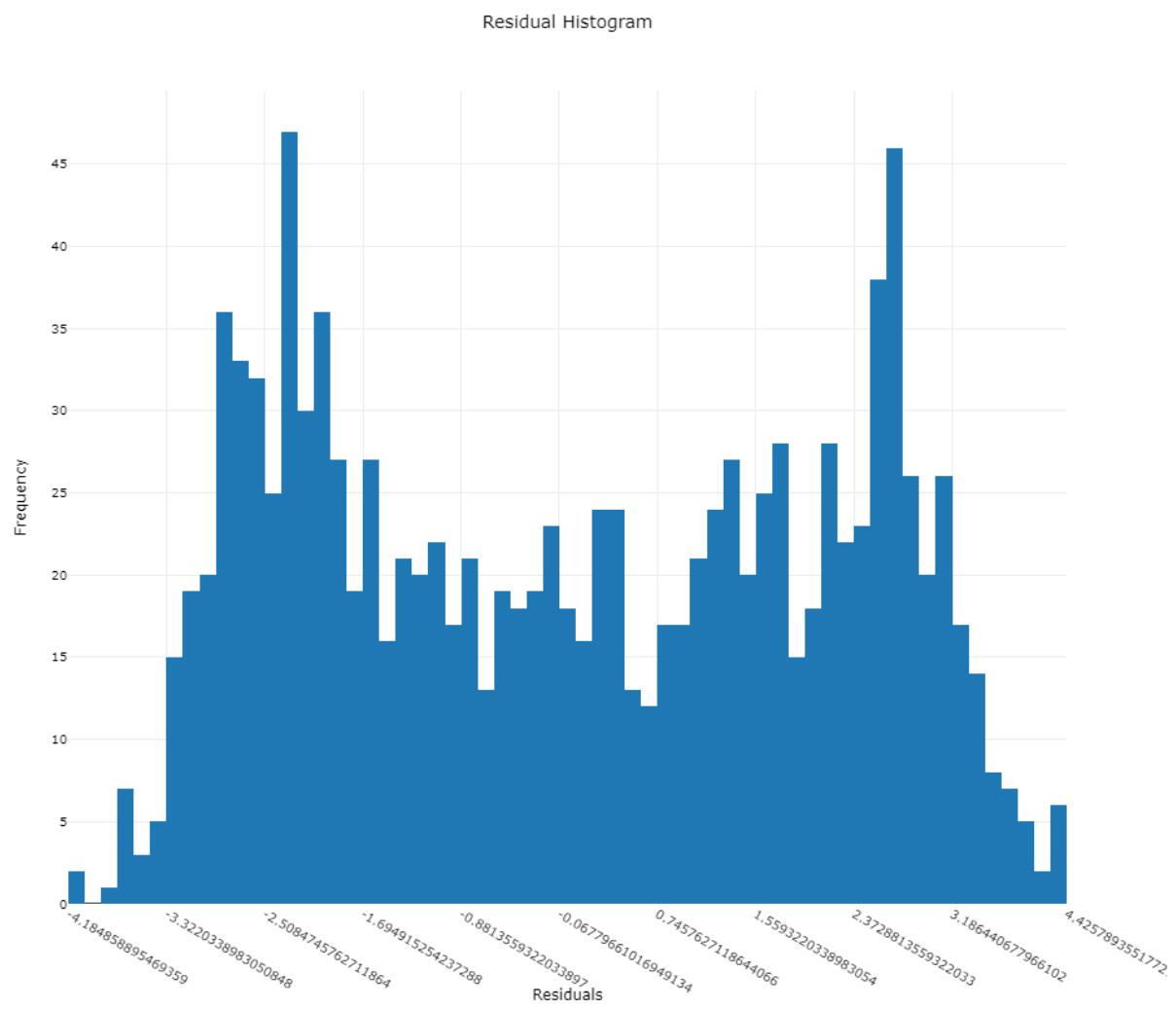
### Predicted vs. True



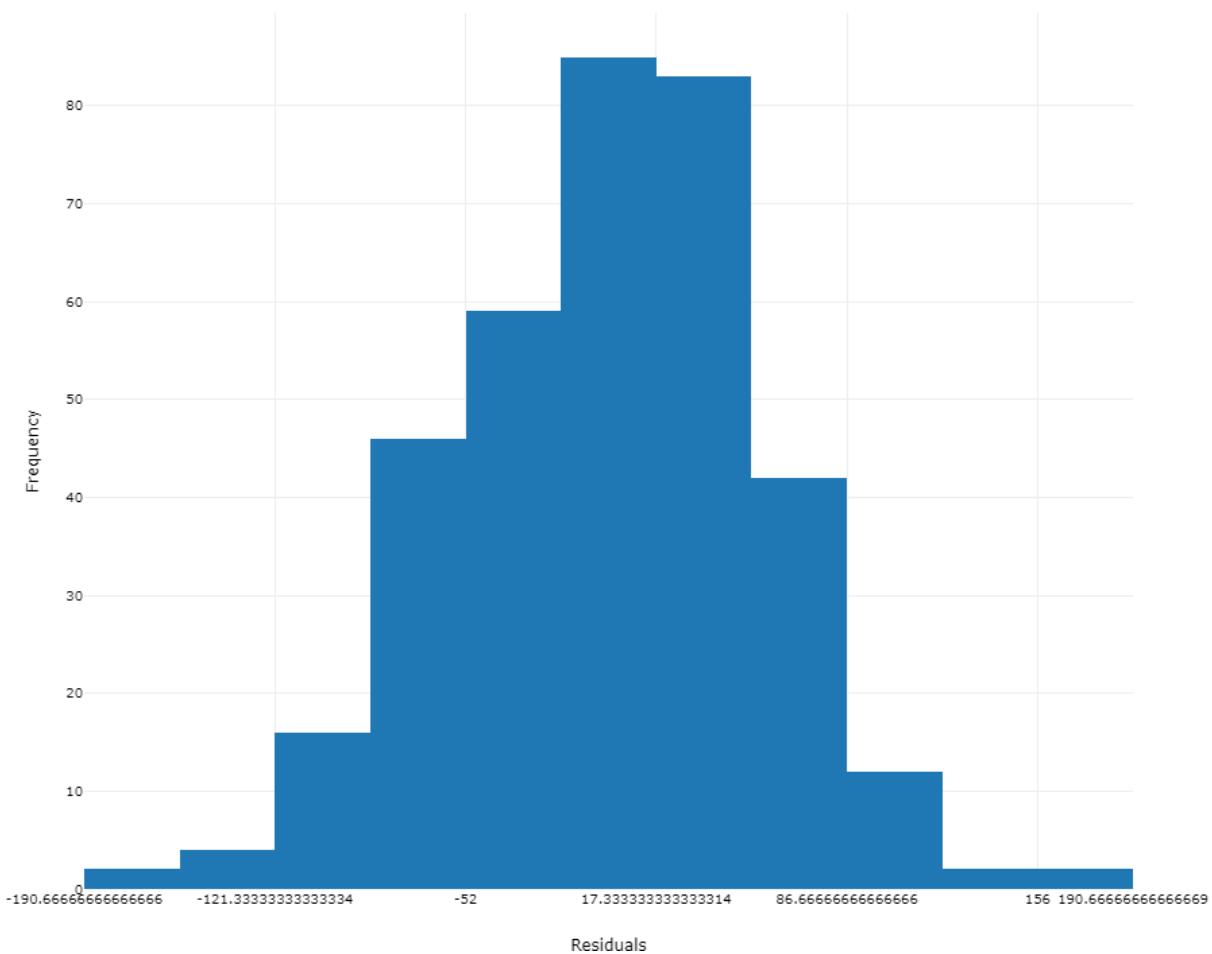
#### Histogram of residuals

A residual represents an observed  $y$  – the predicted  $\hat{y}$ . To show a margin of error with low bias, the histogram of residuals should be shaped as a bell curve, centered around 0.

Example 1: A regression model with bias in its errors

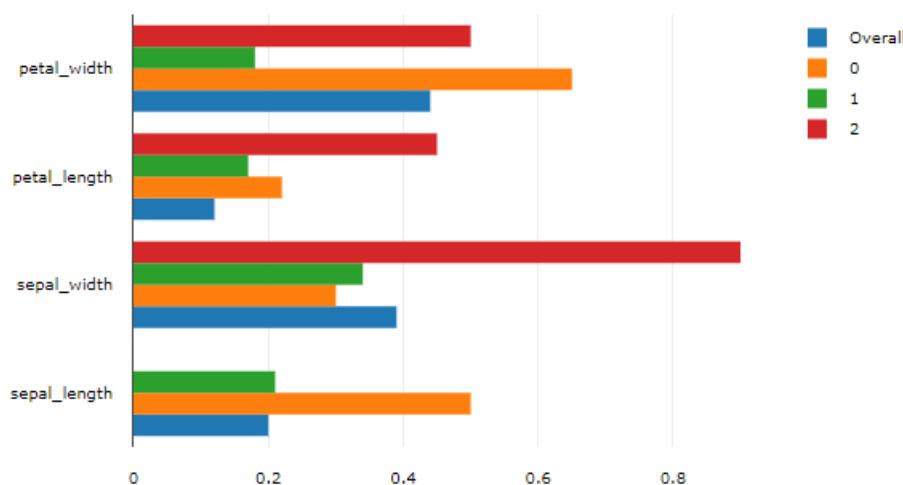


Example 2: A regression model with more even distribution of errors



## Model explainability and feature importance

Feature importance gives a score that indicates how valuable each feature was in the construction of a model. You can review the feature importance score for the model overall as well as per class on a predictive model. You can see per feature how the importance compares against each class and overall.



## Example notebooks

The following notebooks demonstrate concepts in this article:

- [how-to-use-azureml/training/train-within-notebook](#)
- [how-to-use-azureml/training/train-on-local](#)
- [how-to-use-azureml/training/logging-api/logging-api.ipynb](#)

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

## Next steps

Try these next steps to learn how to use the Azure Machine Learning SDK for Python:

- 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](#).
- Learn how to [Train PyTorch Models with Azure Machine Learning](#).

# Configure automated machine learning experiments

2/4/2019 • 15 minutes to read

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. In this guide, learn how to define various configuration settings.

To view examples of an 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 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

## 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.

While automated machine learning capabilities are generally available, **forecasting is still in public preview**.

Automated machine learning supports the following algorithms during the automation and tuning process. As a user, there is no need for you to specify the algorithm.

CLASSIFICATION	REGRESSION	FORECASTING
Logistic Regression	Elastic Net	Elastic Net
Stochastic Gradient Descent (SGD)	Light GBM	Light GBM
Naive Bayes	Gradient Boosting	Gradient Boosting
C-Support Vector Classification (SVC)	Decision Tree	Decision Tree
Linear SVC	K Nearest Neighbors	K Nearest Neighbors
K Nearest Neighbors	LARS Lasso	LARS Lasso
Decision Tree	Stochastic Gradient Descent (SGD)	Stochastic Gradient Descent (SGD)
Random Forest	Random Forest	Random Forest
Extremely Randomized Trees	Extremely Randomized Trees	Extremely Randomized Trees

CLASSIFICATION	REGRESSION	FORECASTING
Gradient Boosting		
Light GBM		

## 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 scikit-learn supported data formats. You can read the data into:

- Numpy arrays X (features) and y (target variable or also known as label)
- Pandas dataframe

Examples:

- Numpy arrays

```
digits = datasets.load_digits()
X_digits = digits.data
y_digits = digits.target
```

- Pandas dataframe

```
import pandas as pd
df = pd.read_csv("https://automldemos.blob.core.windows.net/datasets/PlayaEvents2016,_1.6MB,_3.4k-
rows.cleaned.2.tsv", delimiter="\t", quotechar='''')
# get integer labels
df = df.drop(["Label"], axis=1)
df_train, _, y_train, _ = train_test_split(df, y, test_size=0.1, random_state=42)
```

## Fetch data for running experiment on remote compute

If you are using a remote compute to run your experiment, the data fetch must be wrapped in a separate python script `get_data()`. This script is run on the remote compute where the automated machine learning experiment is run. `get_data` eliminates the need to fetch the data over the wire for each iteration. Without `get_data`, your experiment will fail when you run on remote compute.

Here is an example of `get_data`:

```
%%writefile $project_folder/get_data.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
def get_data(): # Burning man 2016 data
    df = pd.read_csv("https://automldemos.blob.core.windows.net/datasets/PlayaEvents2016,_1.6MB,_3.4k-
rows.cleaned.2.tsv", delimiter="\t", quotechar='''')
    # get integer labels
    le = LabelEncoder()
    le.fit(df["Label"].values)
    y = le.transform(df["Label"].values)
    df = df.drop(["Label"], axis=1)
    df_train, _, y_train, _ = train_test_split(df, y, test_size=0.1, random_state=42)
    return { "X" : df, "y" : y }
```

In your `AutoMLConfig` object, you specify the `data_script` parameter and provide the path to the `get_data` script

file similar to below:

```
automl_config = AutoMLConfig(***, data_script=project_folder + "/get_data.py", *** )
```

get\_data script can return:

KEY	TYPE	MUTUALLY EXCLUSIVE WITH	DESCRIPTION
X	Pandas Dataframe or Numpy Array	data_train, label, columns	All features to train with
y	Pandas Dataframe or Numpy Array	label	Label data to train with. For classification, should be an array of integers.
X_valid	Pandas Dataframe or Numpy Array	data_train, label	<i>Optional</i> All features to validate with. If not specified, X is split between train and validate
y_valid	Pandas Dataframe or Numpy Array	data_train, label	<i>Optional</i> The label data to validate with. If not specified, y is split between train and validate
sample_weight	Pandas Dataframe or Numpy Array	data_train, label, columns	<i>Optional</i> A weight value for each sample. Use when you would like to assign different weights for your data points
sample_weight_valid	Pandas Dataframe or Numpy Array	data_train, label, columns	<i>Optional</i> A weight value for each validation sample. If not specified, sample_weight is split between train and validate
data_train	Pandas Dataframe	X, y, X_valid, y_valid	All data (features+label) to train with
label	string	X, y, X_valid, y_valid	Which column in data_train represents the label
columns	Array of strings		<i>Optional</i> Whitelist of columns to use for features
cv_splits_indices	Array of integers		<i>Optional</i> List of indexes to split the data for cross validation

## Load and prepare data using DataPrep SDK

Automated machine learning experiments supports data loading and transforms using the dataprep SDK. Using the SDK provides the ability to

- Load from many file types with parsing parameter inference (encoding, separator, headers)
- Type-conversion using inference during file loading
- Connection support for MS SQL Server and Azure Data Lake Storage

- Add column using an expression
- Impute missing values
- Derive column by example
- Filtering
- Custom Python transforms

To learn about the data prep sdk refer the [How to prepare data for modeling article](#). Below is an example loading data using data prep sdk.

```
# The data referenced here was pulled from `sklearn.datasets.load_digits()`.  
simple_example_data_root = 'https://dprepdata.blob.core.windows.net/automl-notebook-data/'  
X = dprep.auto_read_file(simple_example_data_root + 'X.csv').skip(1) # Remove the header row.  
# You can use `auto_read_file` which intelligently figures out delimiters and datatypes of a file.  
  
# Here we read a comma delimited file and convert all columns to integers.  
y = dprep.read_csv(simple_example_data_root + 'y.csv').to_long(dprep.ColumnSelector(term='.*', use_regex = True))
```

## Train and validation data

You can specify separate train and validation set either through `get_data()` or directly in the `AutoMLConfig` method.

## Cross validation split options

### K-Folds Cross Validation

Use `n_cross_validations` setting to specify the number of cross validations. The training data set will be randomly split into `n_cross_validations` folds of equal size. During each cross validation round, one of the folds will be used for validation of the model trained on the remaining folds. This process repeats for `n_cross_validations` rounds until each fold is used once as validation set. The average scores across all `n_cross_validations` rounds will be reported, and the corresponding model will be retrained on the whole training data set.

### Monte Carlo Cross Validation (a.k.a. Repeated Random Sub-Sampling)

Use `validation_size` to specify the percentage of the training dataset that should be used for validation, and use `n_cross_validations` to specify the number of cross validations. During each cross validation round, a subset of size `validation_size` will be randomly selected for validation of the model trained on the remaining data. Finally, the average scores across all `n_cross_validations` rounds will be reported, and the corresponding model will be retrained on the whole training data set.

### Custom validation dataset

Use custom validation dataset if random split is not acceptable (usually time series data or imbalanced data). You can specify your own validation dataset. The model will be evaluated against the validation dataset specified instead of random dataset.

## 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:

- Your local machine such as a local desktop or laptop – Generally when you have small dataset and you are still in the exploration stage.
- 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 the [GitHub site](#) for example notebooks with local and remote compute targets.

## 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.

Some examples include:

1. Classification experiment using AUC weighted as the primary metric with a max time of 12,000 seconds per iteration, with the experiment to end after 50 iterations and 2 cross validation folds.

```
automl_classifier = AutoMLConfig(  
    task='classification',  
    primary_metric='AUC_weighted',  
    max_time_sec=12000,  
    iterations=50,  
    X=X,  
    y=y,  
    n_cross_validations=2)
```

2. Below is an example of a regression experiment set to end after 100 iterations, with each iteration lasting up to 600 seconds with 5 validation cross folds.

```
automl_regressor = AutoMLConfig(  
    task='regression',  
    max_time_sec=600,  
    iterations=100,  
    primary_metric='r2_score',  
    X=X,  
    y=y,  
    n_cross_validations=5)
```

There are three different `task` parameter values, which determine the list of algorithms to apply. Use the `whitelist` or `blacklist` parameters to further modify iterations with the available algorithms to include or exclude.

- Classification
  - LogisticRegression
  - SGD
  - MultinomialNaiveBayes
  - BernoulliNaiveBayes
  - SVM
  - LinearSVM
  - KNN
  - DecisionTree
  - RandomForest
  - ExtremeRandomTrees
  - LightGBM
  - GradientBoosting
  - TensorFlowDNN
  - TensorFlowLinearClassifier
- Regression
  - ElasticNet

- GradientBoosting
- DecisionTree
- KNN
- LassoLars
- SGD
- RandomForest
- ExtremeRandomTree
- LightGBM
- TensorFlowLinearRegressor
- TensorFlowDNN
- Forecasting
  - ElasticNet
  - GradientBoosting
  - DecisionTree
  - KNN
  - LassoLars
  - SGD
  - RandomForest
  - ExtremeRandomTree
  - LightGBM
  - TensorFlowLinearRegressor
  - TensorFlowDNN

See the [AutoMLConfig class](#) for a full list of parameters.

## Data pre-processing and featurization

If you use `preprocess=True`, the following data preprocessing steps are performed automatically for you:

1. Drop high cardinality or no variance features
  - Drop features with no useful information from training and validation sets. These include features with all values missing, same value across all rows or with extremely high cardinality (e.g., hashes, IDs or GUIDs).
2. Missing value imputation
  - For numerical features, impute missing values with average of values in the column.
  - For categorical features, impute missing values with most frequent value.
3. Generate additional features
  - For DateTime features: Year, Month, Day, Day of week, Day of year, Quarter, Week of the year, Hour, Minute, Second.
  - For Text features: Term frequency based on word unigram, bi-grams, and tri-gram, Count vectorizer.
4. Transformations and encodings
  - Numeric features with very few unique values transformed into categorical features.
  - Depending on cardinality of categorical features, perform label encoding or (hashing) one-hot encoding.

## Run experiment

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.

## Explore model metrics

You can view your results in a widget or inline if you are in a notebook. See [Track and evaluate models](#) for more details.

### Classification metrics

The following metrics are saved in each iteration for a classification task.

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
AUC_Macro	AUC is the Area under the Receiver Operating Characteristic Curve. Macro is the arithmetic mean of the AUC for each class.	<a href="#">Calculation</a>	average="macro"
AUC_Micro	AUC is the Area under the Receiver Operating Characteristic Curve. Micro is computed globally by combining the true positives and false positives from each class	<a href="#">Calculation</a>	average="micro"
AUC_Weighted	AUC is the Area under the Receiver Operating Characteristic Curve. Weighted is the arithmetic mean of the score for each class, weighted by the number of true instances in each class	<a href="#">Calculation</a>	average="weighted"
accuracy	Accuracy is the percent of predicted labels that exactly match the true labels.	<a href="#">Calculation</a>	None
average_precision_score_macro	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. Macro is the arithmetic mean of the average precision score of each class	<a href="#">Calculation</a>	average="macro"

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
average_precision_score_micro	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. Micro is computed globally by combining the true positives and false positives at each cutoff	<a href="#">Calculation</a>	average="micro"
average_precision_score_weighted	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. Weighted is the arithmetic mean of the average precision score for each class, weighted by the number of true instances in each class	<a href="#">Calculation</a>	average="weighted"
balanced_accuracy	Balanced accuracy is the arithmetic mean of recall for each class.	<a href="#">Calculation</a>	average="macro"
f1_score_macro	F1 score is the harmonic mean of precision and recall. Macro is the arithmetic mean of F1 score for each class	<a href="#">Calculation</a>	average="macro"
f1_score_micro	F1 score is the harmonic mean of precision and recall. Micro is computed globally by counting the total true positives, false negatives, and false positives	<a href="#">Calculation</a>	average="micro"
f1_score_weighted	F1 score is the harmonic mean of precision and recall. Weighted mean by class frequency of F1 score for each class	<a href="#">Calculation</a>	average="weighted"

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
log_loss	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. For a single sample with true label $y_t$ in {0,1} and estimated probability $y_p$ that $y_t = 1$ , the log loss is $-\log P(y_t y_p) = -(y_t \log(y_p) + (1 - y_t) \log(1 - y_p))$	<a href="#">Calculation</a>	None
norm_macro_recall	Normalized Macro Recall is Macro Recall normalized so that random performance has a score of 0 and perfect performance has a score of 1. This is achieved by $\text{norm\_macro\_recall} := (\text{recall\_score\_macro} - R)/(1 - R)$ , where $R$ is the expected value of recall_score_macro for random predictions (i.e., $R=0.5$ for binary classification and $R=(1/C)$ for C-class classification problems)	<a href="#">Calculation</a>	average = "macro" and then $(\text{recall\_score\_macro} - R)/(1 - R)$ , where $R$ is the expected value of recall_score_macro for random predictions (i.e., $R=0.5$ for binary classification and $R=(1/C)$ for C-class classification problems)
precision_score_macro	Precision is the percent of elements labeled as a certain class that actually are in that class. Macro is the arithmetic mean of precision for each class	<a href="#">Calculation</a>	average="macro"
precision_score_micro	Precision is the percent of elements labeled as a certain class that actually are in that class. Micro is computed globally by counting the total true positives and false positives	<a href="#">Calculation</a>	average="micro"
precision_score_weighted	Precision is the percent of elements labeled as a certain class that actually are in that class. Weighted is the arithmetic mean of precision for each class, weighted by number of true instances in each class	<a href="#">Calculation</a>	average="weighted"

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
recall_score_macro	Recall is the percent of elements actually in a certain class that are correctly labeled. Macro is the arithmetic mean of recall for each class	<a href="#">Calculation</a>	average="macro"
recall_score_micro	Recall is the percent of elements actually in a certain class that are correctly labeled. Micro is computed globally by counting the total true positives, false negatives	<a href="#">Calculation</a>	average="micro"
recall_score_weighted	Recall is the percent of elements actually in a certain class that are correctly labeled. Weighted is the arithmetic mean of recall for each class, weighted by number of true instances in each class	<a href="#">Calculation</a>	average="weighted"
weighted_accuracy	Weighted accuracy is accuracy where the weight given to each example is equal to the proportion of true instances in that example's true class	<a href="#">Calculation</a>	sample_weight is a vector equal to the proportion of that class for each element in the target

## Regression and forecasting metrics

The following metrics are saved in each iteration for a regression or forecasting task.

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
explained_variance	Explained variance is the proportion to which a mathematical model accounts for the variation of a given data set. 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 explained variance.	<a href="#">Calculation</a>	None
r2_score	R2 is the coefficient of determination or the percent reduction in squared errors compared to a baseline model that outputs the mean. When the mean of the errors is 0, it is equal to explained variance.	<a href="#">Calculation</a>	None

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
spearman_correlation	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, this one varies between -1 and +1 with 0 implying no correlation. Correlations of -1 or +1 imply an exact monotonic relationship. Positive correlations imply that as x increases, so does y. Negative correlations imply that as x increases, y decreases.	<a href="#">Calculation</a>	None
mean_absolute_error	Mean absolute error is the expected value of absolute value of difference between the target and the prediction	<a href="#">Calculation</a>	None
normalized_mean_absolute_error	Normalized mean absolute error is mean Absolute Error divided by the range of the data	<a href="#">Calculation</a>	Divide by range of the data
median_absolute_error	Median absolute error is the median of all absolute differences between the target and the prediction. This loss is robust to outliers.	<a href="#">Calculation</a>	None
normalized_median_absolute_error	Normalized median absolute error is median absolute error divided by the range of the data	<a href="#">Calculation</a>	Divide by range of the data
root_mean_squared_error	Root mean squared error is the square root of the expected squared difference between the target and the prediction	<a href="#">Calculation</a>	None
normalized_root_mean_squared_error	Normalized root mean squared error is root mean squared error divided by the range of the data	<a href="#">Calculation</a>	Divide by range of the data

PRIMARY METRIC	DESCRIPTION	CALCULATION	EXTRA PARAMETERS
root_mean_squared_log_err or	Root mean squared log error is the square root of the expected squared logarithmic error	<a href="#">Calculation</a>	None
normalized_root_mean_squared_log_error	Normalized Root mean squared log error is root mean squared log error divided by the range of the data	<a href="#">Calculation</a>	Divide by range of the data

## Explain the model

While automated machine learning capabilities are generally available, **the model explainability feature is still in public preview.**

Automated machine learning allows you to understand feature importance. During the training process, you can get global feature importance for the model. For classification scenarios, you can also get class-level feature importance. You must provide a validation dataset (X\_valid) to get feature importance.

There are two ways to generate feature importance.

- Once an experiment is complete, you can use `explain_model` method on any iteration.

```
from azureml.train.automl.automlexplainer import explain_model

shap_values, expected_values, overall_summary, overall_imp, per_class_summary, per_class_imp = \
    explain_model(fitted_model, X_train, X_test)

#Overall feature importance
print(overall_imp)
print(overall_summary)

#Class-level feature importance
print(per_class_imp)
print(per_class_summary)
```

- To view feature importance for all iterations, set `model_explainability` flag to `True` in AutoMLConfig.

```
automl_config = AutoMLConfig(task = 'classification',
                             debug_log = 'automl_errors.log',
                             primary_metric = 'AUC_weighted',
                             max_time_sec = 12000,
                             iterations = 10,
                             verbosity = logging.INFO,
                             X = X_train,
                             y = y_train,
                             X_valid = X_test,
                             y_valid = y_test,
                             model_explainability=True,
                             path=project_folder)
```

Once done, you can use `retrieve_model_explanation` method to retrieve feature importance for a specific iteration.

```
from azureml.train.automl.automlexplainer import retrieve_model_explanation

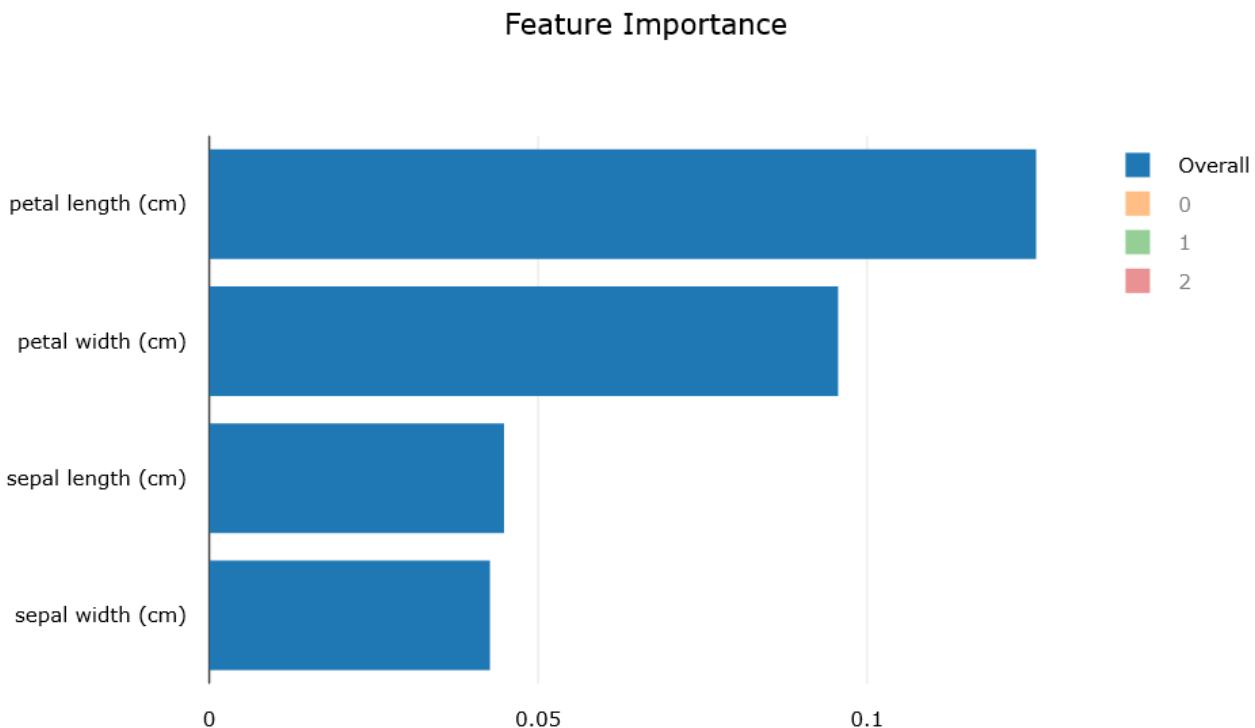
shap_values, expected_values, overall_summary, overall_imp, per_class_summary, per_class_imp = \
retrieve_model_explanation(best_run)

#Overall feature importance
print(overall_imp)
print(overall_summary)

#Class-level feature importance
print(per_class_imp)
print(per_class_summary)
```

You can visualize the feature importance chart in your workspace in the Azure portal. The chart is also shown when using the Jupyter widget in a notebook. To learn more about the charts refer to the [Sample Azure Machine Learning service notebooks article](#).

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



## Next steps

Learn more about [how and where to deploy a model](#).

Learn more about [how to train a classification model with Automated machine learning](#) or [how to train using Automated machine learning on a remote resource](#).

# Train models with automated machine learning in the cloud

3/4/2019 • 7 minutes to read

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 computer in the cloud.

You can easily scale up or scale out your machine learning experiment by adding additional compute targets. Compute target options include Ubuntu-based Data Science Virtual Machine (DSVM) or Azure Machine Learning Compute. The DSVM is a customized VM image on Microsoft's Azure cloud built specifically for doing data science. It has many popular data science and other tools pre-installed and pre-configured.

In this article, you learn how to build a model using automated ML on the DSVM.

## How does remote differ from local?

The tutorial "[Train a classification model with automated machine learning](#)" teaches you how to use a local computer to train model with automated ML. The workflow when training locally also applies to remote targets as well. However, with remote compute, automated ML experiment iterations are executed asynchronously. This functionality allows you to cancel a particular iteration, watch the status of the execution, or continue to work on other cells in the Jupyter notebook. To train remotely, you first create a remote compute target such as an Azure DSVM. 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 DSVM. The workspace object, `ws`, from the tutorial is used throughout the code here.

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

## Create resource

Create the DSVM in your workspace (`ws`) if it doesn't already exist. If the DSVM was previously created, this code skips the creation process and loads the existing resource detail into the `dsvm_compute` object.

**Time estimate:** Creation of the VM takes approximately 5 minutes.

```
from azureml.core.compute import DsvmCompute

dsvm_name = 'mydsvm' #Name your DSVM
try:
    dsvm_compute = DsvmCompute(ws, dsvm_name)
    print('found existing dsvm.')
except:
    print('creating new dsvm.')
    # Below is using a VM of SKU Standard_D2_v2 which is 2 core machine. You can check Azure virtual machines documentation for additional SKUs of VMs.
    dsvm_config = DsvmCompute.provisioning_configuration(vm_size = "Standard_D2_v2")
    dsvm_compute = DsvmCompute.create(ws, name = dsvm_name, provisioning_configuration = dsvm_config)
    dsvm_compute.wait_for_completion(show_output = True)
```

You can now use the `dsvm_compute` object as the remote compute target.

DSVM name restrictions include:

- Must be shorter than 64 characters.
- Cannot include any of the following characters: `\ ~ ! @ # $ % ^ & * ( ) = + _ [ ] { } \\ | ; ' \" , < > / ? ``

### WARNING

If creation fails with a message about Marketplace purchase eligibility:

1. Go to the [Azure portal](#)
2. Start creating a DSVM
3. Select "Want to create programmatically" to enable programmatic creation
4. Exit without actually creating the VM
5. Rerun the creation code

This code doesn't create a user name or password for the DSVM that is provisioned. If you want to connect directly to the VM, go to the [Azure portal](#) to create credentials.

### Attach existing Linux DSVM

You can also attach an existing Linux DSVM as the compute target. This example utilizes an existing DSVM, but doesn't create a new resource.

### NOTE

The following code uses the [RemoteCompute](#) target class to attach an existing VM as your compute target. The [DsvmCompute](#) class will be deprecated in future releases in favor of this design pattern.

Run the following code to create the compute target from a pre-existing Linux DSVM.

```
from azureml.core.compute import ComputeTarget, RemoteCompute

attach_config = RemoteCompute.attach_configuration(username='<username>',
                                                 address='<ip_address_or_fqdn>',
                                                 ssh_port=22,
                                                 private_key_file='./.ssh/id_rsa')

compute_target = ComputeTarget.attach(workspace=ws,
                                      name='attached_vm',
                                      attach_configuration=attach_config)

compute_target.wait_for_completion(show_output=True)
```

You can now use the `compute_target` object as the remote compute target.

## Access data using get\_data file

Provide the remote resource access to your training data. For automated machine learning experiments running on remote compute, the data needs to be fetched using a `get_data()` function.

To provide access, you must:

- Create a `get_data.py` file containing a `get_data()` function
- Place that file in a directory accessible as an absolute path

You can encapsulate code to read data from a blob storage or local disk in the `get_data.py` file. In the following code sample, the data comes from the `sklearn` package.

## WARNING

If you are using remote compute, then you must use `get_data()` where your data transformations are performed. If you need to install additional libraries for data transformations as part of `get_data()`, there are additional steps to be followed. Refer to the [auto-ml-dataprep sample notebook](#) for details.

```
# Create a project_folder if it doesn't exist
if not os.path.exists(project_folder):
    os.makedirs(project_folder)

#Write the get_data file.
%%writefile $project_folder/get_data.py

from sklearn import datasets
from scipy import sparse
import numpy as np

def get_data():

    digits = datasets.load_digits()
    X_digits = digits.data[10:,:]
    y_digits = digits.target[10:]

    return { "X" : X_digits, "y" : y_digits }
```

## Configure experiment

Specify the settings for `AutoMLConfig`. (See a [full list of parameters](#) and their possible values.)

In the settings, `run_configuration` is set to the `run_config` object, which contains the settings and configuration for the DSVM.

```
from azureml.train.automl import AutoMLConfig
import time
import logging

automl_settings = {
    "name": "AutoML_Demo_Experiment_{0}".format(time.time()),
    "iteration_timeout_minutes": 10,
    "iterations": 20,
    "n_cross_validations": 5,
    "primary_metric": 'AUC_weighted',
    "preprocess": False,
    "max_concurrent_iterations": 10,
    "verbosity": logging.INFO
}

automl_config = AutoMLConfig(task='classification',
                             debug_log='automl_errors.log',
                             path=project_folder,
                             compute_target = dsvm_compute,
                             data_script=project_folder + "/get_data.py",
                             **automl_settings,
                             )
```

## Enable model explanations

Set the optional `model_explainability` parameter in the `AutoMLConfig` constructor. Additionally, a validation dataframe object must be passed as a parameter `X_valid` to use the model explainability feature.

```

automl_config = AutoMLConfig(task='classification',
                             debug_log='automl_errors.log',
                             path=project_folder,
                             compute_target = dsvm_compute,
                             data_script=project_folder + "/get_data.py",
                             **automl_settings,
                             model_explainability=True,
                             X_valid = X_test
)

```

## 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 the one in [the training tutorial](#) to see a graph and table of results.

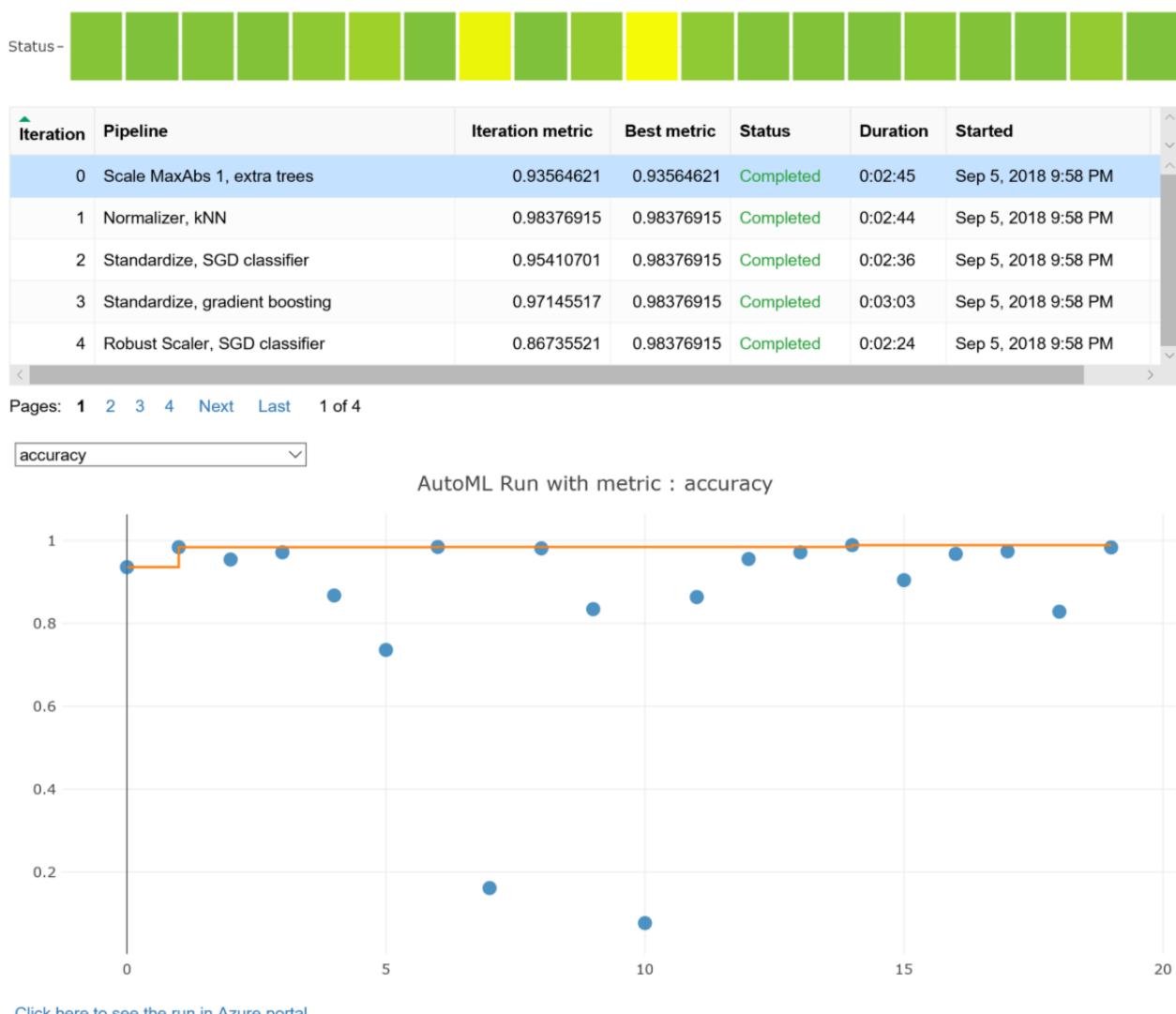
```

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.



The widget displays a URL you can use to see and explore the individual run details.

### View logs

Find logs on the DSVM under `/tmp/azureml_run/{iterationid}/azureml-logs`.

## Best model explanation

Retrieving model explanation data allows you to see detailed information about the models to increase transparency into what's running on the back-end. In this example, you run model explanations only for the best fit model. If you run for all models in the pipeline, it will result in significant run time. Model explanation information includes:

- `shap_values`: The explanation information generated by shap lib
- `expected_values`: The expected value of the model applied to set of `X_train` data.
- `overall_summary`: The model level feature importance values sorted in descending order
- `overall_imp`: The feature names sorted in the same order as in `overall_summary`
- `per_class_summary`: The class level feature importance values sorted in descending order. Only available for the classification case
- `per_class_imp`: The feature names sorted in the same order as in `per_class_summary`. Only available for the classification case

Use the following code to select the best pipeline from your iterations. The `get_output` method returns the best

run and the fitted model for the last fit invocation.

```
best_run, fitted_model = remote_run.get_output()
```

Import the `retrieve_model_explanation` function and run on the best model.

```
from azureml.train.automl.automlexplainer import retrieve_model_explanation

shap_values, expected_values, overall_summary, overall_imp, per_class_summary, per_class_imp = \
    retrieve_model_explanation(best_run)
```

Print results for the `best_run` explanation variables you want to view.

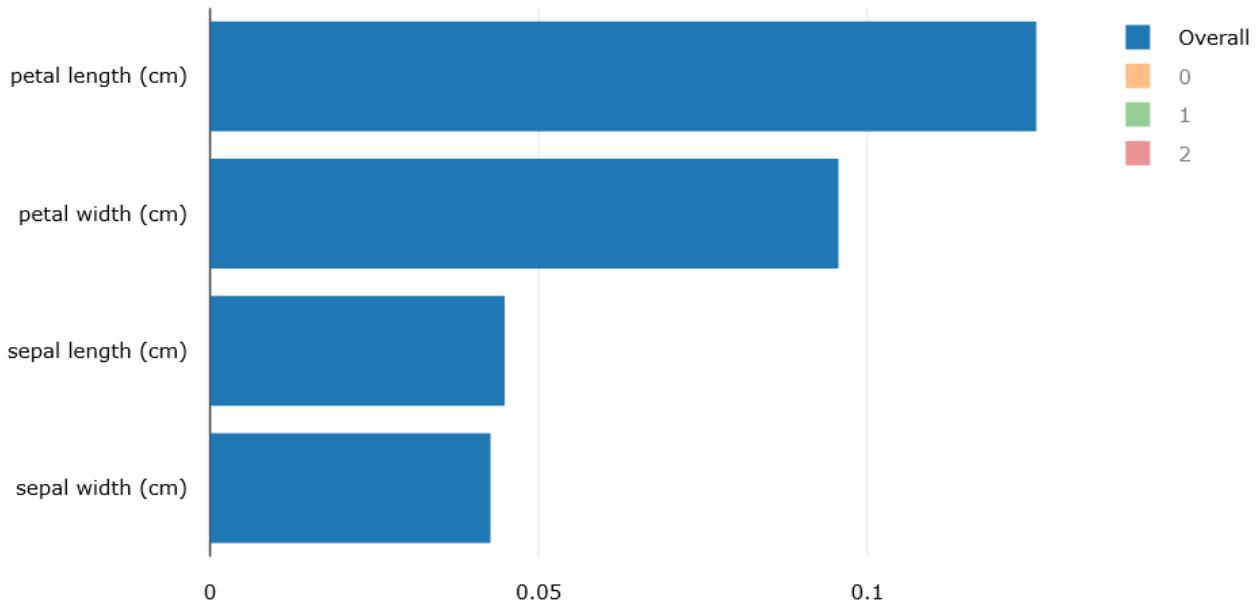
```
print(overall_summary)
print(overall_imp)
print(per_class_summary)
print(per_class_imp)
```

Printing the `best_run` explanation summary variables results in the following output.

```
[0.13751896361164054, 0.12196113064848417, 0.02515800367003573, 0.0204136524293061]
['petal width (cm)', 'petal length (cm)', 'sepal length (cm)', 'sepal width (cm)']
[[0.13935022276497785, 0.12224503499384598, 0.030073645391740296, 0.028297085859246965], [0.14292484561842992, 0.11960261236451923, 0.02612530890966221, 0.02358220878544405], [0.14738701023064565, 0.10693055681595545, 0.02181815683292284, 0.00681856251900913]]
[['petal length (cm)', 'petal width (cm)', 'sepal length (cm)', 'sepal width (cm)'], ['petal width (cm)', 'petal length (cm)', 'sepal width (cm)', 'sepal length (cm)'], ['petal width (cm)', 'petal length (cm)', 'sepal length (cm)', 'sepal width (cm)']]
```

You can also visualize feature importance through the widget UI as well as the web UI on Azure portal inside your workspace.

Feature Importance



## Example

The [how-to-use-azureml/automated-machine-learning/remote-execution/auto-ml-remote-execution.ipynb](#) 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](#).

# ONNX and Azure Machine Learning: Create and deploy interoperable AI models

2/26/2019 • 4 minutes to read

The [Open Neural Network Exchange \(ONNX\)](#) format is an open standard for representing machine learning models. ONNX is supported by a [community of partners](#), including Microsoft, who create compatible frameworks and tools. Microsoft is committed to open and interoperable AI so that data scientists and developers can:

- Use the framework of their choice to create and train models
- Deploy models cross-platform with minimal integration work

Microsoft supports ONNX across its products including Azure and Windows to help you achieve these goals.

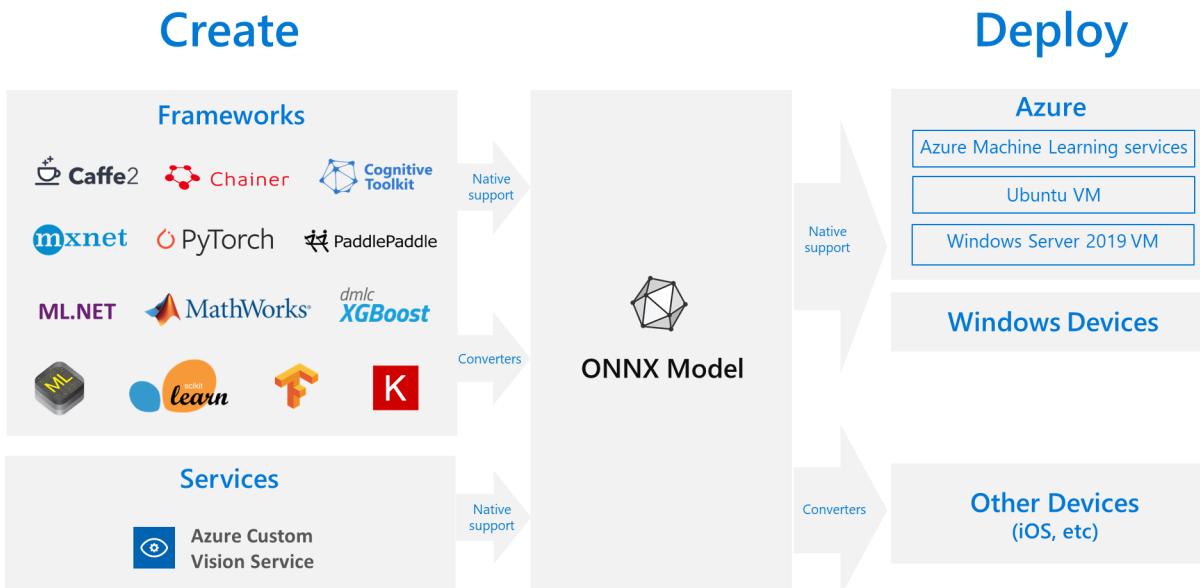
## Why choose ONNX?

The interoperability you get with ONNX makes it possible to get great ideas into production faster. With ONNX, data scientists can choose their preferred framework for the job. Similarly, developers can spend less time getting models ready for production, and deploy across the cloud and edge.

You can create ONNX models from many frameworks, including PyTorch, Chainer, Microsoft Cognitive Toolkit (CNTK), MXNet, ML.NET, TensorFlow, Keras, SciKit-Learn, and more.

There is also an ecosystem of tools for visualizing and accelerating ONNX models. A number of pre-trained ONNX models are also available for common scenarios.

[ONNX models can be deployed](#) to the cloud using Azure Machine Learning and ONNX Runtime. They can also be deployed to Windows 10 devices using [Windows ML](#). They can even be deployed to other platforms using converters that are available from the ONNX community.



## Get ONNX models

You can obtain ONNX models in several ways:

- Get a pre-trained ONNX model from the [ONNX Model Zoo](#) (see example at the bottom of this article)

- Generate a customized ONNX model from [Azure Custom Vision service](#)
- Convert existing model from another format to ONNX (see example at the bottom of this article)
- Train a new ONNX model in Azure Machine Learning service (see example at the bottom of this article)

## Save/convert your models to ONNX

You can convert existing models to ONNX or save them as ONNX at the end of your training.

FRAMEWORK FOR MODEL	CONVERSION EXAMPLE OR TOOL
PyTorch	<a href="#">Jupyter notebook</a>
Microsoft Cognitive Toolkit (CNTK)	<a href="#">Jupyter notebook</a>
TensorFlow	<a href="#">tensorflow-onnx converter</a>
Chainer	<a href="#">Jupyter notebook</a>
MXNet	<a href="#">Jupyter notebook</a>
Keras, Scikit-Learn, CoreML XGBoost, and libSVM	<a href="#">WinMLTools</a>

You can find the latest list of supported frameworks and converters at the [ONNX Tutorials site](#).

## Deploy ONNX models in Azure

With Azure Machine Learning service, 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 a full example Jupyter notebook at the end of this article to try it out for yourself.

### Install and configure ONNX Runtime

ONNX Runtime is an open source high-performance inference engine for ONNX models. It provides hardware acceleration on both CPU and GPU, with APIs available for Python, C#, and C. ONNX Runtime supports ONNX 1.2+ models and runs on Linux, Windows, and Mac. Python packages are available on [PyPi.org](#) ([CPU](#), [GPU](#)), and [C# package](#) is on [Nuget.org](#). See more about the project on [GitHub](#).

To install ONNX Runtime for Python, use:

```
pip install onnxruntime
```

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](#).

## Example deployment steps

Here is an example for deploying an ONNX model:

1. Initialize your Azure Machine Learning service workspace. If you don't have one yet, learn how to create a workspace in [this quickstart](#).

```
from azureml.core import Workspace

ws = Workspace.from_config()
print(ws.name, ws.resource_group, ws.location, ws.subscription_id, sep = '\n')
```

2. Register the model with Azure Machine Learning.

```
from azureml.core.model import Model

model = Model.register(model_path = "model.onnx",
                      model_name = "MyONNXmodel",
                      tags = ["onnx"],
                      description = "test",
                      workspace = ws)
```

3. Create an image with the model and any dependencies.

```
from azureml.core.image import ContainerImage

image_config = ContainerImage.image_configuration(execution_script = "score.py",
                                                 runtime = "python",
                                                 conda_file = "myenv.yml",
                                                 description = "test",
                                                 tags = ["onnx"]
)

image = ContainerImage.create(name = "myonnxmodelimage",
                             # this is the model object
                             models = [model],
                             image_config = image_config,
                             workspace = ws)

image.wait_for_creation(show_output = True)
```

The file `score.py` contains the scoring logic and needs to be included in the image. This file is used to run the model in the image. See this [tutorial](#) for instructions on how to create a scoring script. An example file for an ONNX model is shown below:

```

import onnxruntime
import json
import numpy as np
import sys
from azureml.core.model import Model

def init():
    global model_path
    model_path = Model.get_model_path(model_name = 'MyONNXmodel')

def run(raw_data):
    try:
        data = json.loads(raw_data)['data']
        data = np.array(data)

        sess = onnxruntime.InferenceSession(model_path)
        result = sess.run(["outY"], {"inX": data})

        return json.dumps({"result": result.tolist()})
    except Exception as e:
        result = str(e)
        return json.dumps({"error": result})

```

The file `myenv.yml` describes the dependencies needed for the image. See this [tutorial](#) for instructions on how to create an environment file, such as this sample file:

```

from azureml.core.conda_dependencies import CondaDependencies

myenv = CondaDependencies()
myenv.add_pip_package("numpy")
myenv.add_pip_package("azureml-core")
myenv.add_pip_package("onnxruntime")

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())

```

4. To deploy your model, see the [How to deploy and where](#) document.

## Examples

See [how-to-use-azureml/deployment/onnx](#) for example notebooks that create and deploy ONNX models.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

## 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 GitHub Repo](#)

# Deploy models with the Azure Machine Learning service

3/6/2019 • 16 minutes to read

The Azure Machine Learning service provides several ways you can deploy your trained model using the SDK. In this document, learn how to deploy your model as a web service in the Azure cloud, or to IoT Edge devices.

## IMPORTANT

Cross-origin resource sharing (CORS) is not currently supported when deploying a model as a web service.

You can deploy models to the following compute targets:

COMPUTE TARGET	DEPLOYMENT TYPE	DESCRIPTION
Azure Kubernetes Service (AKS)	Real-time inference	Good for high-scale production deployments. Provides autoscaling, and fast response times.
Azure ML Compute	Batch inference	Run batch prediction on serverless compute. Supports normal and low priority VMs.
Azure Container Instances (ACI)	Testing	Good for development or testing. <b>Not suitable for production workloads.</b>
Azure IoT Edge	(Preview) IoT module	Deploy models on IoT devices. Inferencing happens on the device.
Field-programmable gate array (FPGA)	(Preview) Web service	Ultra-low latency for real-time inferencing.

The process of deploying a model is similar for all compute targets:

1. Train and register a model.
2. Configure and register an image that uses the model.
3. Deploy the image to a compute target.
4. Test the deployment

For more information on the concepts involved in the deployment workflow, see [Manage, deploy, and monitor models with Azure Machine Learning 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 service](#) today.
- An Azure Machine Learning service workspace and the Azure Machine Learning SDK for Python

installed. Learn how to get these prerequisites using the [Get started with Azure Machine Learning quickstart](#).

- A trained model. If you do not have a trained model, use the steps in the [Train models](#) tutorial to train and register one with the Azure Machine Learning service.

#### NOTE

While the Azure Machine Learning service can work with any generic model that can be loaded in Python 3, the examples in this document demonstrate using a model stored in pickle format.

For more information on using ONNX models, see the [ONNX and Azure Machine Learning](#) document.

## Register a trained model

The model registry is a way to store and organize your trained models in the Azure cloud. Models are registered in your Azure Machine Learning service workspace. The model can be trained using Azure Machine Learning, or another service. To register a model from file, use the following code:

```
from azureml.core.model import Model

model = Model.register(model_path = "model.pkl",
                       model_name = "MyModel",
                       tags = {"key": "0.1"},
                       description = "test",
                       workspace = ws)
```

**Time estimate:** Approximately 10 seconds.

For more information, see the reference documentation for the [Model class](#).

## Create and register an image

Deployed models are packaged as an image. The image contains the dependencies needed to run the model.

For **Azure Container Instance**, **Azure Kubernetes Service**, and **Azure IoT Edge** deployments, the [azureml.core.image.ContainerImage](#) class is used to create an image configuration. The image configuration is then used to create a new Docker image.

The following code demonstrates how to create a new image configuration:

```
from azureml.core.image import ContainerImage

# Image configuration
image_config = ContainerImage.image_configuration(execution_script = "score.py",
                                                   runtime = "python",
                                                   conda_file = "myenv.yml",
                                                   description = "Image with ridge regression model",
                                                   tags = {"data": "diabetes", "type": "regression"})
```

**Time estimate:** Approximately 10 seconds.

The important parameters in this example described in the following table:

PARAMETER	DESCRIPTION
<code>execution_script</code>	Specifies a Python script that is used to receive requests submitted to the service. In this example, the script is contained in the <code>score.py</code> file. For more information, see the <a href="#">Execution script</a> section.
<code>runtime</code>	Indicates that the image uses Python. The other option is <code>spark-py</code> , which uses Python with Apache Spark.
<code>conda_file</code>	Used to provide a conda environment file. This file defines the conda environment for the deployed model. For more information on creating this file, see <a href="#">Create an environment file (myenv.yml)</a> .

For more information, see the reference documentation for [ContainerImage class](#)

## Execution script

The execution script receives data submitted to a deployed image, 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 script usually contains two functions that load and run the model:

- `init()` : Typically this function loads the model into a global object. This function is run only once when the Docker container is started.
- `run(input_data)` : This 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. You can also work with raw binary data. You can transform the data before sending to the model, or before returning to the client.

## Working with JSON data

The following example script accepts and returns JSON data. The `run` function transforms the data from JSON into a format that the model expects, and then transforms the response to JSON before returning it:

```
# import things required by this script
import json
import numpy as np
import os
import pickle
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression

from azureml.core.model import Model

# load the model
def init():
    global model
    # retrieve the path to the model file using the model name
    model_path = Model.get_model_path('sklearn_mnist')
    model = joblib.load(model_path)

# Passes data to the model and returns the prediction
def run(raw_data):
    data = np.array(json.loads(raw_data)['data'])
    # make prediction
    y_hat = model.predict(data)
    return json.dumps(y_hat.tolist())
```

## Working with Binary data

If your model accepts **binary data**, use `AMLRequest`, `AMLResponse`, and `rawhttp`. The following example script accepts binary data and returns the reversed bytes for POST requests. For GET requests, it returns the full URL in the response body:

```
from azureml.contrib.servicesaml_request import AMLRequest, rawhttp
from azureml.contrib.servicesaml_response import AMLResponse

def init():
    print("This is init()")

# Accept and return binary data
@rawhttp
def run(request):
    print("This is run()")
    print("Request: [{0}]".format(request))
    # handle GET requests
    if request.method == 'GET':
        respBody = str.encode(request.full_path)
        return AMLResponse(respBody, 200)
    # handle POST requests
    elif request.method == 'POST':
        reqBody = request.get_data(False)
        respBody = bytarray(reqBody)
        respBody.reverse()
        respBody = bytes(respBody)
        return AMLResponse(respBody, 200)
    else:
        return AMLResponse("bad request", 500)
```

### IMPORTANT

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.

If you need to test this on your local development environment, you can install the components in the `contrib` namespace by using the following command:

```
pip install azureml-contrib-services
```

### Register the image

Once you have created the image configuration, you can use it to register an image. This image is stored in the container registry for your workspace. Once created, you can deploy the same image to multiple services.

```
# Register the image from the image configuration
image = ContainerImage.create(name = "myimage",
                               models = [model], #this is the model object
                               image_config = image_config,
                               workspace = ws
                             )
```

**Time estimate:** Approximately 3 minutes.

Images are versioned automatically when you register multiple images with the same name. For example, the first image registered as `myimage` is assigned an ID of `myimage:1`. The next time you register an image as `myimage`, the ID of the new image is `myimage:2`.

For more information, see the reference documentation for [ContainerImage class](#).

# Deploy the image

When you get to deployment, the process is slightly different depending on the compute target that you deploy to. Use the information in the following sections to learn how to deploy to:

- [Azure Container Instances](#)
- [Azure Kubernetes Services](#)
- [Project Brainwave \(field-programmable gate arrays\)](#)
- [Azure IoT Edge devices](#)

## NOTE

When **deploying as a web service**, there are three deployment methods you can use:

METHOD	NOTES
<code>deploy_from_image</code>	You must register the model and create an image before using this method.
<code>deploy</code>	When using this method, you do not need to register the model or create the image. However you cannot control the name of the model or image, or associated tags and descriptions.
<code>deploy_from_model</code>	When using this method, you do not need to create an image. But you do not have control over the name of the image that is created.

The examples in this document use `deploy_from_image`.

## Deploy to Azure Container Instances (DEVTEST)

Use Azure Container Instances for deploying your models as a web service if one or more of the following conditions is true:

- You need to quickly deploy and validate your model. ACI deployment is finished in less than 5 minutes.
- You are testing a model that is under development. To see quota and region availability for ACI, see the [Quotas and region availability for Azure Container Instances](#) document.

To deploy to Azure Container Instances, use the following steps:

1. Define the deployment configuration. The following example defines a configuration that uses one CPU core and 1 GB of memory:

```
from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores = 1,
                                              memory_gb = 1,
                                              tags = {"data": "mnist", "type": "classification"},
                                              description = 'Handwriting recognition')
```

2. To deploy the image created in the [Create the image](#) section of this document, use the following code:

```

from azureml.core.webservice import Webservice

service_name = 'aci-mnist-13'
service = Webservice.deploy_from_image(deployment_config = aciconfig,
                                       image = image,
                                       name = service_name,
                                       workspace = ws)
service.wait_for_deployment(show_output = True)
print(service.state)

```

**Time estimate:** Approximately 3 minutes.

For more information, see the reference documentation for the [AciWebservice](#) and [Webservice](#) classes.

## Deploy to Azure Kubernetes Service (PRODUCTION)

To deploy your model as a high-scale production web service, use Azure Kubernetes Service (AKS). You can use an existing AKS cluster or create a new one using the Azure Machine Learning SDK, CLI, or the Azure portal.

Creating an AKS cluster is a one time process for your workspace. You can reuse this cluster for multiple deployments. If you delete the cluster, then you must create a new cluster the next time you need to deploy.

Azure Kubernetes Service provides the following capabilities:

- Autoscaling
- Logging
- Model data collection
- Fast response times for your web services
- TLS termination
- Authentication

### Autoscaling

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 the 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.

#### Create a new cluster

To create a new Azure Kubernetes Service cluster, use the following code:

##### IMPORTANT

Creating the AKS cluster is a one time process for your workspace. Once created, you can reuse this cluster for multiple deployments. If you delete the cluster or the resource group that contains it, then you must create a new cluster the next time you need to deploy. For `provisioning_configuration()`, if you pick custom values for `agent_count` and `vm_size`, then you need to make sure `agent_count` multiplied by `vm_size` is greater than or equal to 12 virtual CPUs. For example, if you use a `vm_size` of "Standard\_D3\_v2", which has 4 virtual CPUs, then you should pick an `agent_count` of 3 or greater.

```

from azureml.core.compute import AksCompute, ComputeTarget

# Use the default configuration (you can also provide parameters to customize this)
prov_config = AksCompute.provisioning_configuration()

aks_name = 'aml-aks-1'
# 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)
print(aks_target.provisioning_state)
print(aks_target.provisioning_errors)

```

**Time estimate:** Approximately 20 minutes.

#### Use an existing cluster

If you already have AKS cluster in your Azure subscription, and it is version 1.11.\*, you can use it to deploy your image. The following code demonstrates how to attach an existing cluster to your workspace:

```

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 = 'mycluster'

# Attach the cluster to your workgroup
attach_config = AksCompute.attach_configuration(resource_group = resource_group,
                                                 cluster_name = cluster_name)
aks_target = ComputeTarget.attach(ws, 'mycompute', attach_config)

# Wait for the operation to complete
aks_target.wait_for_completion(True)

```

**Time estimate:** Approximately 3 minutes.

#### Deploy the image

To deploy the image created in the [Create the image](#) section of this document to the Azure Kubernetes Server cluster, use the following code:

```

from azureml.core.webservice import Webservice, AksWebservice

# Set configuration and service name
aks_config = AksWebservice.deploy_configuration()
aks_service_name = 'aks-service-1'
# Deploy from image
service = Webservice.deploy_from_image(workspace = ws,
                                         name = aks_service_name,
                                         image = image,
                                         deployment_config = aks_config,
                                         deployment_target = aks_target)

# Wait for the deployment to complete
service.wait_for_deployment(show_output = True)
print(service.state)

```

**Time estimate:** Approximately 3 minutes.

For more information, see the reference documentation for the [AksWebservice](#) and [Webservice](#) classes.

#### Inference with Azure ML Compute

Azure ML compute targets are created and managed by the Azure Machine Learning service. They can be used for batch prediction from Azure ML Pipelines.

For a walkthrough of batch inference with Azure ML Compute, read the [How to Run Batch Predictions](#) document.

#### Deploy to field-programmable gate arrays (FPGA)

Project Brainwave makes it possible to achieve ultra-low latency for real-time inferencing requests. Project Brainwave accelerates deep neural networks (DNN) deployed on field-programmable gate arrays in the Azure cloud. Commonly used DNNs are available as featurizers for transfer learning, or customizable with weights trained from your own data.

For a walkthrough of deploying a model using Project Brainwave, see the [Deploy to a FPGA](#) document.

#### Deploy to Azure IoT Edge

An Azure IoT Edge device is a Linux or Windows-based device that runs the Azure IoT Edge runtime. Using the Azure IoT Hub, you can deploy machine learning models to these devices as IoT Edge modules.

Deploying a model to an IoT Edge device allows the device to use the model directly, instead of having to send data to the cloud for processing. You get faster response times and less data transfer.

Azure IoT Edge modules are deployed to your device from a container registry. When you create an image from your model, it is stored in the container registry for your workspace.

#### IMPORTANT

The information in this section assumes that you are already familiar with Azure IoT Hub and Azure IoT Edge modules. While some of the information in this section is specific to Azure Machine Learning service, the majority of the process to deploy to an edge device happens in the Azure IoT service.

If you are unfamiliar with Azure IoT, see [Azure IoT Fundamentals](#) and [Azure IoT Edge](#) for basic information. Then use the other links in this section to learn more about specific operations.

#### Set up your environment

- A development environment. For more information, see the [How to configure a development environment](#) document.
- An [Azure IoT Hub](#) in your Azure subscription.
- A trained model. For an example of how to train a model, see the [Train an image classification model with Azure Machine Learning](#) document. A pre-trained model is available on the [AI Toolkit for Azure IoT Edge GitHub repo](#).

#### Get the container registry credentials

To deploy an IoT Edge module to your device, Azure IoT needs the credentials for the container registry that Azure Machine Learning service stores docker images in.

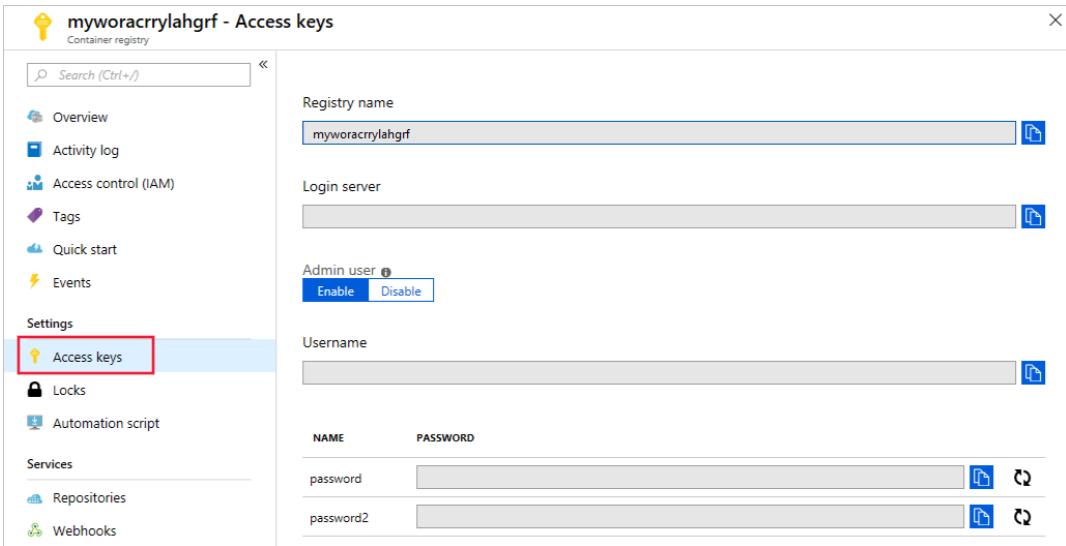
You can get the credentials in two ways:

- **In the Azure portal:**

1. Sign in to the [Azure portal](#).
2. Go to your Azure Machine Learning service workspace and select **Overview**. To go to the container registry settings, select the **Registry** link.

The screenshot shows the Azure Machine Learning Workspace - PREVIEW interface. The left sidebar has a 'Search (Ctrl+ /)' bar and links for 'Overview' (which is highlighted with a red box), 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. The main area shows resource details for 'mygroup': Resource group 'mygroup', Location 'East US 2', Subscription 'documentationteam', and Subscription ID. On the right, there's a 'Delete' button. Below these are 'Storage' ('myworstoragenwhzkzka'), 'Registry' (which is also highlighted with a red box and shows the value 'myworacrylahgrf'), 'Key Vault' ('myworkeyvaulttmvmuwhc'), and 'Application Insights' ('myworinsightsxxtlsmu').

3. Once in the container registry, select **Access Keys** and then enable the admin user.



4. Save the values for **login server**, **username**, and **password**.

- **With a Python script:**

1. Use the following Python script after the code you ran above to create a container:

```
# Getting your container details
container_reg = ws.get_details()["containerRegistry"]
reg_name=container_reg.split("/")[-1]
container_url = "\"" + image.image_location + "\","
subscription_id = ws.subscription_id
from azure.mgmt.containerregistry import ContainerRegistryManagementClient
from azure.mgmt import containerregistry
client = ContainerRegistryManagementClient(ws._auth,subscription_id)
result= client.registries.list_credentials(resource_group_name, reg_name,
custom_headers=None, raw=False)
username = result.username
password = result.passwords[0].value
print('ContainerURL{}'.format(image.image_location))
print('Servername: {}'.format(reg_name))
print('Username: {}'.format(username))
print('Password: {}'.format(password))
```

2. Save the values for ContainerURL, servername, username, and password.

These credentials are necessary to provide the IoT Edge device access to images in your private container registry.

#### Prepare the IoT device

You must register your device with Azure IoT Hub, and then install the IoT Edge runtime on the device. If you are not familiar with this process, see [Quickstart: Deploy your first IoT Edge module to a Linux x64 device](#).

Other methods of registering a device are:

- [Azure portal](#)
- [Azure CLI](#)
- [Visual Studio Code](#)

#### Deploy the model to the device

To deploy the model to the device, use the registry information gathered in the [Get container registry credentials](#) section with the module deployment steps for IoT Edge modules. For example, when [Deploying Azure IoT Edge modules from the Azure portal](#), you must configure the **Registry settings** for the device. Use the **login server**, **username**, and **password** for your workspace container registry.

You can also deploy using [Azure CLI](#) and [Visual Studio Code](#).

## Testing web service deployments

To test a web service deployment, you can use the `run` method of the `Webservice` object. In the following example, a JSON document is set to a web service and the result is displayed. The data sent must match what the model expects. In this example, the data format matches the input expected by the diabetes model.

```
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)
```

The `webservice` is a REST API, so you can create client applications in a variety of programming languages. For more information, see [Create client applications to consume webservices](#).

## Update the web service

When you create a new image, you must manually update each service that you want to use the new image. To update the web service, use the `update` method. The following code demonstrates how to update the web service to use a new image:

```
from azureml.core.webservice import Webservice
from azureml.core.image import Image

service_name = 'aci-mnist-3'
# Retrieve existing service
service = Webservice(name = service_name, workspace = ws)

# point to a different image
new_image = Image(workspace = ws, id="myimage2:1")

# Update the image used by the service
service.update(image = new_image)
print(service.state)
```

For more information, see the reference documentation for the [Webservice](#) class.

## Clean up

To delete a deployed web service, use `service.delete()`.

To delete an image, use `image.delete()`.

To delete a registered model, use `model.delete()`.

For more information, see the reference documentation for [WebService.delete\(\)](#), [Image.delete\(\)](#), and [Model.delete\(\)](#).

## Troubleshooting

- If there are errors during deployment, use `service.get_logs()` to view the service logs. The logged

information may indicate the cause of the error.

- The logs may contain an error that instructs you to **set logging level to DEBUG**. To set the logging level, add the following lines to your scoring script, create the image, and then create a service using the image:

```
import logging  
logging.basicConfig(level=logging.DEBUG)
```

This change enables additional logging, and may return more information on why the error is occurring.

## Next steps

- [Secure Azure Machine Learning web services with SSL](#)
- [Consume a ML Model deployed as a web service](#)
- [How to run batch predictions](#)
- [Monitor your Azure Machine Learning models with Application Insights](#)
- [Collect data for models in production](#)
- [Azure Machine Learning service SDK](#)
- [Use Azure Machine Learning service with Azure Virtual Networks](#)
- [Best practices for building recommendation systems](#)
- [Build a real-time recommendation API on Azure](#)

# Deploy a model as a web service on an FPGA with Azure Machine Learning service

3/5/2019 • 3 minutes to read

You can deploy a model as a web service on [field programmable gate arrays \(FPGAs\)](#). Using FPGAs provides ultra-low latency inferencing, even with a single batch size. These models are currently available:

- ResNet 50
- ResNet 152
- DenseNet-121
- VGG-16

## 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 service](#) today.
- An Azure Machine Learning service workspace and the Azure Machine Learning SDK for Python installed. Learn how to get these prerequisites using the [How to configure a development environment](#) document.
  - Your workspace needs to be in the *East US 2* region.
  - Install the contrib extras:

```
pip install --upgrade azureml-sdk[contrib]
```

- Currently only tensorflow version  $<=1.10$  is supported, so install it after all other installations are complete:

```
pip install "tensorflow==1.10"
```

## Get the notebook

For your convenience, this tutorial is available as a Jupyter notebook. Follow the code here or run the [quickstart notebook](#).

## Create and deploy your model

Create a pipeline to preprocess the input image, make it a feature using ResNet 50 on an FPGA, and then run the features through a classifier trained on the ImageNet data set.

Follow the instructions to:

- Define the model pipeline
- Deploy the model
- Consume the deployed model
- Delete deployed services

## IMPORTANT

To optimize latency and throughput, your client should be in the same Azure region as the endpoint. Currently the APIs are created in the East US Azure region.

## Preprocess image

The first stage of the pipeline is to preprocess the images.

```
import os
import tensorflow as tf

# Input images as a two-dimensional tensor containing an arbitrary number of images represented as strings
import azureml.contrib.brainwave.models.utils as utils
in_images = tf.placeholder(tf.string)
image_tensors = utils.preprocess_array(in_images)
print(image_tensors.shape)
```

## Add Featurizer

Initialize the model and download a TensorFlow checkpoint of the quantized version of ResNet50 to be used as a featurizer.

```
from azureml.contrib.brainwave.models import QuantizedResnet50
model_path = os.path.expanduser('~/models')
model = QuantizedResnet50(model_path, is_frozen = True)
feature_tensor = model.import_graph_def(image_tensors)
print(model.version)
print(feature_tensor.name)
print(feature_tensor.shape)
```

## Add Classifier

This classifier has been trained on the ImageNet data set.

```
classifier_output = model.get_default_classifier(feature_tensor)
```

## Create service definition

Now that you have defined the image preprocessing, featurizer, and classifier that runs on the service, you can create a service definition. The service definition is a set of files generated from the model that is deployed to the FPGA service. The service definition consists of a pipeline. The pipeline is a series of stages that are run in order. TensorFlow stages, Keras stages, and BrainWave stages are supported. The stages are run in order on the service, with the output of each stage becoming the input into the subsequent stage.

To create a TensorFlow stage, specify a session containing the graph (in this case default graph is used) and the input and output tensors to this stage. This information is used to save the graph so that it can be run on the service.

```

from azureml.contrib.brainwave.pipeline import ModelDefinition, TensorflowStage, BrainWaveStage

save_path = os.path.expanduser('~/models/save')
model_def_path = os.path.join(save_path, 'model_def.zip')

model_def = ModelDefinition()
with tf.Session() as sess:
    model_def.pipeline.append(TensorflowStage(sess, in_images, image_tensors))
    model_def.pipeline.append(BrainWaveStage(sess, model))
    model_def.pipeline.append(TensorflowStage(sess, feature_tensor, classifier_output))
model_def.save(model_def_path)
print(model_def_path)

```

## Deploy model

Create a service from the service definition. Your workspace needs to be in the East US 2 location.

```

from azureml.core import Workspace

ws = Workspace.from_config()
print(ws.name, ws.resource_group, ws.location, ws.subscription_id, sep = '\n')

from azureml.core.model import Model
model_name = "resnet-50-rtai"
registered_model = Model.register(ws, model_def_path, model_name)

from azureml.core.webservice import Webservice
from azureml.exceptions import WebServiceException
from azureml.contrib.brainwave import BrainwaveWebservice, BrainwaveImage
service_name = "imagenet-infer"
service = None
try:
    service = Webservice(ws, service_name)
except WebServiceException:
    image_config = BrainwaveImage.image_configuration()
    deployment_config = BrainwaveWebservice.deploy_configuration()
    service = Webservice.deploy_from_model(ws, service_name, [registered_model], image_config,
deployment_config)
    service.wait_for_deployment(True)

```

## Test the service

To send an image to the API and test the response, add a mapping from the output class ID to the ImageNet class name.

```

import requests
classes_entries =
requests.get("https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.txt"
).text.splitlines()

```

Call your service and replace the "your-image.jpg" file name below with an image from your machine.

```

with open('your-image.jpg') as f:
    results = service.run(f)
# 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 service

Delete the service.

```
service.delete()  
registered_model.delete()
```

## Secure FPGA web services

For information on securing FPGA web services, see the [Secure web services](#) document.

## Next steps

Learn how to [Consume a ML Model deployed as a web service](#).

# Troubleshooting Azure Machine Learning service AKS and ACI deployments

3/5/2019 • 10 minutes to read

In this article, you will learn how to work around or solve the common Docker deployment errors with Azure Container Instances (ACI) and Azure Kubernetes Service (AKS) using Azure Machine Learning service.

When deploying a model in Azure Machine Learning service, the system performs a number of tasks. This is a complex sequence of events and sometimes issues arise. The deployment tasks are:

1. Register the model in the workspace model registry.
2. Build a Docker image, including:
  - a. Download the registered model from the registry.
  - b. Create a dockerfile, with a Python environment based on the dependencies you specify in the environment yaml file.
  - c. Add your model files and the scoring script you supply in the dockerfile.
  - d. Build a new Docker image using the dockerfile.
  - e. Register the Docker image with the Azure Container Registry associated with the workspace.
3. Deploy the Docker image to Azure Container Instance (ACI) service or to Azure Kubernetes Service (AKS).
4. Start up a new container (or containers) in ACI or AKS.

Learn more about this process in the [Model Management](#) introduction.

## Before you begin

If you run into any issue, the first thing to do is to break down the deployment task (previous described) into individual steps to isolate the problem.

This is helpful if you are using the `Webservice.deploy` API, or `Webservice.deploy_from_model` API, since those functions group together the aforementioned steps into a single action. Typically those APIs are convenient, but it helps to break up the steps when troubleshooting by replacing them with the below API calls.

1. Register the model. Here's some sample code:

```
# register a model out of a run record
model = best_run.register_model(model_name='my_best_model', model_path='outputs/my_model.pkl')

# or, you can register a file or a folder of files as a model
model = Model.register(model_path='my_model.pkl', model_name='my_best_model', workspace=ws)
```

2. Build the image. Here's some sample code:

```

# configure the image
image_config = ContainerImage.image_configuration(runtime="python",
                                                 execution_script="score.py",
                                                 conda_file="myenv.yml")

# create the image
image = Image.create(name='myimg', models=[model], image_config=image_config, workspace=ws)

# wait for image creation to finish
image.wait_for_creation(show_output=True)

```

3. Deploy the image as service. Here's some sample code:

```

# configure an ACI-based deployment
aci_config = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1)

aci_service = Webservice.deploy_from_image(deployment_config=aci_config,
                                            image=image,
                                            name='mysvc',
                                            workspace=ws)
aci_service.wait_for_deployment(show_output=True)

```

Once you have broken down the deployment process into individual tasks, we can look at some of the most common errors.

## Image building fails

If system is unable to build the Docker image, the `image.wait_for_creation()` call fails with some error messages that can offer some clues. You can also find out more details about the errors from the image build log. Below is some sample code showing how to discover the image build log uri.

```

# if you already have the image object handy
print(image.image_build_log_uri)

# if you only know the name of the image (note there might be multiple images with the same name but different
# version number)
print(ws.images['myimg'].image_build_log_uri)

# list logs for all images in the workspace
for name, img in ws.images.items():
    print (img.name, img.version, img.image_build_log_uri)

```

The image log uri is a SAS URL pointing to a log file stored in your Azure blob storage. Simply copy and paste the uri into a browser window and you can download and view the log file.

### Azure Key Vault access policy and Azure Resource Manager templates

The image build can also fail due to a problem with the access policy on Azure Key Vault. This can occur 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. This breaks access to the Key Vault for any existing workspace that is using it. This results in errors when you try to create new images. The following are examples of the errors that you can receive:

**Portal:**

```
Create image "myimage": An internal server error occurred. Please try again. If the problem persists, contact support.
```

## SDK:

```
image = ContainerImage.create(name = "myimage", models = [model], image_config = image_config, workspace = ws)
Creating image
Traceback (most recent call last):
  File "C:\Python37\lib\site-packages\azureml\core\image\image.py", line 341, in create
    resp.raise_for_status()
  File "C:\Python37\lib\site-packages\requests\models.py", line 940, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 500 Server Error: Internal Server Error for url:
https://eastus.modelmanagement.azureml.net/api/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.MachineLearningServices/workspaces/<workspace-name>/images?api-version=2018-11-19

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Python37\lib\site-packages\azureml\core\image\image.py", line 346, in create
    'Content: {}'.format(resp.status_code, resp.headers, resp.content))
azureml.exceptions._azureml_exception.WebServiceException: Received bad response from Model Management Service:
Response Code: 500
Headers: {'Date': 'Tue, 26 Feb 2019 17:47:53 GMT', 'Content-Type': 'application/json', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive', 'api-supported-versions': '2018-03-01-preview, 2018-11-19', 'x-ms-client-request-id': '3cdcf791f1214b9cbac93076ebfb5167', 'x-ms-client-session-id': '', 'Strict-Transport-Security': 'max-age=15724800; includeSubDomains; preload'}
Content: b'{"code":"InternalServerError","statusCode":500,"message":"An internal server error occurred. Please try again. If the problem persists, contact support"}'
```

## CLI:

```
ERROR: {'Azure-cli-ml Version': None, 'Error': WebServiceException('Received bad response from Model Management Service:\nResponse Code: 500\nHeaders: {'Date': 'Tue, 26 Feb 2019 17:34:05 GMT', 'Content-Type': 'application/json', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive', 'api-supported-versions': '2018-03-01-preview, 2018-11-19', 'x-ms-client-request-id': 'bc89430916164412abe3d82acb1d1109', 'x-ms-client-session-id': '', 'Strict-Transport-Security': 'max-age=15724800; includeSubDomains; preload'}\nContent: b'{"code":"InternalServerError","statusCode":500,"message":"An internal server error occurred. Please try again. If the problem persists, contact support"}')}
```

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 use this to set the `accessPolicies` property of the template.
- Check if the Key Vault resource already exists. If it does, do not recreate it through the template. For example, add a parameter that allows you to disable the creation of the Key Vault resource if it already exists.

## Service launch fails

After the image is successfully built, the system attempts to start a container in either ACI or AKS depending on your deployment configuration. It is recommended to try an ACI deployment first, since it is a simpler single-container deployment. This way you can then rule out any AKS-specific problem.

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. Below are some tips to help you troubleshoot the problem.

### Inspect the Docker log

You can print out detailed Docker engine log messages from the service object.

```
# 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())
```

## Debug the Docker image locally

Some times the Docker log does not emit enough information about what is going wrong. You can go one step further and pull down the built Docker image, start a local container, and debug directly inside the live container interactively. To start a local container, you must have a Docker engine running locally, and it would be a lot easier if you also have [azure-cli](#) installed.

First we need to find out the image location:

```
# print image location
print(image.image_location)
```

The image location has this format: <acr-name>.azurecr.io/<image-name>:<version-number>, such as  
myworkspaceacr.azurecr.io/myimage:3 .

Now go to your command-line window. If you have azure-cli installed, you can type the following commands to sign in to the ACR (Azure Container Registry) associated with the workspace where the image is stored.

```
# log on to Azure first if you haven't done so before
$ az login

# make sure you set the right subscription in case you have access to multiple subscriptions
$ az account set -s <subscription_name_or_id>

# now let's log in to the workspace ACR
# note the acr-name is the domain name WITHOUT the ".azurecr.io" postfix
# e.g.: az acr login -n myworkspaceacr
$ az acr login -n <acr-name>
```

If you don't have azure-cli installed, you can use `docker login` command to log into the ACR. But you need to retrieve the user name and password of the ACR from Azure portal first.

Once you have logged in to the ACR, you can pull down the Docker image and start a container locally, and then launch a bash session for debugging by using the `docker run` command:

```
# note the image_id is <acr-name>.azurecr.io/<image-name>:<version-number>
# for example: myworkspaceacr.azurecr.io/myimage:3
$ docker run -it <image_id> /bin/bash
```

Once you launch a bash session the running container, you can find your scoring scripts in the `/var/azureml-app` folder. You can then launch a Python session to debug your scoring scripts.

```
# enter the directory where scoring scripts live
cd /var/azureml-app

# find what Python packages are installed in the python environment
pip freeze

# sanity-check on score.py
# you might want to edit the score.py to trigger init().
# as most of the errors happen in init() when you are trying to load the model.
python score.py
```

In case you need a text editor to modify your scripts, you can install vim, nano, Emacs, or your other favorite editor.

```
# update package index
apt-get update

# install a text editor of your choice
apt-get install vim
apt-get install nano
apt-get install emacs

# launch emacs (for example) to edit score.py
emacs score.py

# exit the container bash shell
exit
```

You can also start up the web service locally and send HTTP traffic to it. The Flask server in the Docker container is running on port 5001. You can map to any other ports available on the host machine.

```
# you can find the scoring API at: http://localhost:8000/score
$ docker run -p 8000:5001 <image_id>
```

## 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. This is often a source of failure if the model file or folder cannot be found. The easiest way to debug this error is to run the below Python code in the Container shell:

```
import logging
logging.basicConfig(level=logging.DEBUG)
from azureml.core.model import Model
print(Model.get_model_path(model_name='my-best-model'))
```

This would print 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 provide 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. It might not be a good idea to do this in a production environment for security reasons.

## HTTP status code 503

Azure Kubernetes Service deployments support autoscaling, which allows replicas to be added to support additional load. However, 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.

There are two things that can help prevent 503 status codes:

- Change the utilization level at which autoscaling creates new replicas.

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%. 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.

## Next steps

Learn more about deployment:

- [How to deploy and where](#)
- [Tutorial: Train & deploy models](#)

# Consume an Azure Machine Learning model deployed as a web service

2/19/2019 • 8 minutes to read

Deploying an Azure Machine Learning model as a web service creates a REST API. You can send data to this API 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 an image to Azure Container Instances, Azure Kubernetes Service, or Project Brainwave (field programmable gate arrays). You create images from registered models and scoring files. You retrieve the URI used to access a web service by using the [Azure Machine Learning SDK](#). If authentication is enabled, you can also use the SDK to get the authentication keys.

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.

## 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.WebService` class provides the information you need to create a client. The following `WebService` properties are useful for creating a client application:

- `auth_enabled` - If authentication is enabled, `True`; otherwise, `False`.
- `scoring_uri` - The REST API address.

There are three ways to retrieve this information for deployed web services:

- When you deploy a model, a `WebService` object is returned with information about the service:

```
service = WebService.deploy_from_model(name='myservice',
                                       deployment_config=myconfig,
                                       models=[model],
                                       image_config=image_config,
                                       workspace=ws)
print(service.scoring_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)
```

- 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)
```

## Authentication key

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`.

## 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.

For example, the model in the [Train within notebook](#) example expects an array of 10 numbers. The scoring script for this example creates a Numpy array from the request, and passes it to the model. The following example shows the data this service expects:

```
{
  "data": [
    [
      0.0199132141783263,
      0.0506801187398187,
      0.104808689473925,
      0.0700725447072635,
      -0.0359677812752396,
      -0.0266789028311707,
      -0.0249926566315915,
      -0.00259226199818282,
      0.00371173823343597,
      0.0403433716478807
    ]
  ]
}
```

The web service can accept multiple sets of data in one request. It returns a JSON document containing an array of responses.

## Binary data

If your model accepts binary data, such as an image, you must modify the `score.py` file used for your deployment to accept raw HTTP requests. Here's an example of a `score.py` that accepts binary data, and returns the reversed bytes for POST requests. For GET requests, it returns the full URL in the response body:

```
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':
        respBody = str.encode(request.full_path)
        return AMLResponse(respBody, 200)
    elif request.method == 'POST':
        reqBody = request.get_data(False)
        respBody = bytearray(reqBody)
        respBody.reverse()
        respBody = bytes(respBody)
        return AMLResponse(respBody, 200)
    else:
        return AMLResponse("bad request", 500)
```

## IMPORTANT

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.

If you need to test this on your local development environment, you can install the components in the `contrib` namespace by using the following command:

```
pip install azureml-contrib-services
```

## 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
            string scoringUri = "<your web service URI>";
            string authKey = "<your key>";

            // 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 (if any) for your service
var authKey string = "<your key>"

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
        String key = "<your key>";
        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
key = '<your key>'

# 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,
            0.060618394448076,
            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]
```

# Run batch predictions on large data sets with Azure Machine Learning service

1/29/2019 • 6 minutes to read

In this article, you'll learn how to make predictions on large quantities of data asynchronously, by using the Azure Machine Learning service.

Batch prediction (or batch scoring) provides cost-effective inference, with unparalleled throughput for asynchronous applications. Batch prediction pipelines can scale to perform inference on terabytes of production data. Batch prediction is optimized for high throughput, fire-and-forget predictions for a large collection of data.

## TIP

If your system requires low-latency processing (to process a single document or small set of documents quickly), use [real-time scoring](#) instead of batch prediction.

In the following steps, you create a [machine learning pipeline](#) to register a pretrained computer vision model ([Inception-V3](#)). Then you use the pretrained model to do batch scoring on images available in your Azure Blob storage account. These images used for scoring are unlabeled images from the [ImageNet](#) dataset.

## 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 service](#).
- Configure your development environment to install the Azure Machine Learning SDK. For more information, see [Configure a development environment for Azure Machine Learning](#).
- Create an Azure Machine Learning workspace that will hold all your pipeline resources. You can use the following code, or for more options, see [Create a workspace configuration file](#).

```
ws = Workspace.create(  
    name = '<workspace-name>',  
    subscription_id = '<subscription-id>',  
    resource_group = '<resource-group>',  
    location = '<workspace_region>',  
    exist_ok = True)
```

## Set up machine learning resources

The following steps set up the resources you need to run a pipeline:

- Access the datastore that already has the pretrained model, input labels, and images to score (this is already set up for you).
- Set up a datastore to store your outputs.
- Configure `DataReference` objects to point to the data in the preceding datastores.
- Set up compute machines or clusters where the pipeline steps will run.

### Access the datastores

First, access the datastore that has the model, labels, and images.

You'll use a public blob container, named *sampledata*, in the *pipelinedata* account that holds images from the ImageNet evaluation set. The datastore name for this public container is *images\_datastore*. Register this datastore with your workspace:

```
# Public blob container details
account_name = "pipelinedata"
datastore_name="images_datastore"
container_name="sampledata"

batchscore_blob = Datastore.register_azure_blob_container(ws,
    datastore_name=datastore_name,
    container_name= container_name,
    account_name=account_name,
    overwrite=True)
```

Next, set up to use the default datastore for the outputs.

When you create your workspace, [Azure Files](#) and [Blob storage](#) are attached to the workspace by default. Azure Files is the default datastore for a workspace, but you can also use Blob storage as a datastore. For more information, see [Azure storage options](#).

```
def_data_store = ws.get_default_datastore()
```

## Configure data references

Now, reference the data in your pipeline as inputs to pipeline steps.

A data source in a pipeline is represented by a [DataReference](#) object. The [DataReference](#) object points to data that lives in, or is accessible from, a datastore. You need [DataReference](#) objects for the directory used for input images, the directory in which the pretrained model is stored, the directory for labels, and the output directory.

```
input_images = DataReference(datastore=batchscore_blob,
    data_reference_name="input_images",
    path_on_datastore="batchscoring/images",
    mode="download")

model_dir = DataReference(datastore=batchscore_blob,
    data_reference_name="input_model",
    path_on_datastore="batchscoring/models",
    mode="download")

label_dir = DataReference(datastore=batchscore_blob,
    data_reference_name="input_labels",
    path_on_datastore="batchscoring/labels",
    mode="download")

output_dir = PipelineData(name="scores",
    datastore=def_data_store,
    output_path_on_compute="batchscoring/results")
```

## Set up compute target

In Azure Machine Learning, *compute* (or *compute target*) refers to the machines or clusters that perform the computational steps in your machine learning pipeline. For example, you can create an

[Azure Machine Learning compute](#).

```

compute_name = "gpucluster"
compute_min_nodes = 0
compute_max_nodes = 4
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. just use it. ' + compute_name)
else:
    print('Creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(
        vm_size = vm_size, # NC6 is GPU-enabled
        vm_priority = 'lowpriority', # optional
        min_nodes = compute_min_nodes,
        max_nodes = compute_max_nodes)

# create the cluster
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)

```

## Prepare the model

Before you can use the pretrained model, you'll need to download the model and register it with your workspace.

### Download the pretrained model

Download the pretrained computer vision model (InceptionV3) from

[http://download.tensorflow.org/models/inception\\_v3\\_2016\\_08\\_28.tar.gz](http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz). Then extract it to the `models` subfolder.

```

import os
import tarfile
import urllib.request

model_dir = 'models'
if not os.path.isdir(model_dir):
    os.mkdir(model_dir)

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

```

### Register the model

Here's how to register the model:

```
import shutil
from azureml.core.model import Model

# register downloaded model
model = Model.register(
    model_path = "models/inception_v3.ckpt",
    model_name = "inception", # This is the name of the registered model
    tags = {'pretrained': "inception"},
    description = "Imagenet trained tensorflow inception",
    workspace = ws)
```

## Write your scoring script

### WARNING

The following code is only a sample of what is contained in the `batch_score.py` used by the [sample notebook](#). You'll need to create your own scoring script for your scenario.

The `batch_score.py` script takes input images in `dataset_path`, pretrained models in `model_dir`, and outputs `results-label.txt` to `output_dir`.

```

# Snippets from a sample scoring script
# Refer to the accompanying batch-scoring Notebook
# https://github.com/Azure/MachineLearningNotebooks/blob/master/pipeline/pipeline-batch-scoring.ipynb
# for the implementation script

# Get labels
def get_class_label_dict(label_file):
    label = []
    proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

class DataIterator:
    # Definition of the DataIterator here

def main(_):
    # Refer to batch-scoring Notebook for implementation.
    label_file_name = os.path.join(args.label_dir, "labels.txt")
    label_dict = get_class_label_dict(label_file_name)
    classes_num = len(label_dict)
    test_feeder = DataIterator(data_dir=args.dataset_path)
    total_size = len(test_feeder.labels)

    # get model from model registry
    model_path = Model.get_model_path(args.model_name)
    with tf.Session() as sess:
        test_images = test_feeder.input_pipeline(batch_size=args.batch_size)
        with slim.arg_scope(inception_v3.inception_v3_arg_scope()):
            input_images = tf.placeholder(tf.float32, [args.batch_size, 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)

        sess.run(tf.global_variables_initializer())
        sess.run(tf.local_variables_initializer())
        coord = tf.train.Coordinator()
        threads = tf.train.start_queue_runners(sess=sess, coord=coord)
        saver = tf.train.Saver()
        saver.restore(sess, model_path)
        out_filename = os.path.join(args.output_dir, "result-labels.txt")

        # copy the file to artifacts
        shutil.copy(out_filename, "./outputs/")

```

## Build and run the batch scoring pipeline

You have everything you need to build the pipeline, so now put it all together.

### Prepare the run environment

Specify the conda dependencies for your script. You'll need this object later, when you create the pipeline step.

```

from azureml.core.runconfig import DEFAULT_GPU_IMAGE

cd = CondaDependencies.create(pip_packages=["tensorflow-gpu==1.10.0", "azureml-defaults"])

# Runconfig
amlcompute_run_config = RunConfiguration(conda_dependencies=cd)
amlcompute_run_config.environment.docker.enabled = True
amlcompute_run_config.environment.docker.gpu_support = True
amlcompute_run_config.environment.docker.base_image = DEFAULT_GPU_IMAGE
amlcompute_run_config.environment.spark.precache_packages = False

```

## Specify the parameter for your pipeline

Create a pipeline parameter by using a [PipelineParameter](#) object with a default value.

```
batch_size_param = PipelineParameter(  
    name="param_batch_size",  
    default_value=20)
```

## Create the pipeline step

Create the pipeline step by using the script, environment configuration, and parameters. Specify the compute target you already attached to your workspace as the target of execution of the script. Use [PythonScriptStep](#) to create the pipeline step.

```
inception_model_name = "inception_v3.ckpt"  
  
batch_score_step = PythonScriptStep(  
    name="batch_scoring",  
    script_name="batch_score.py",  
    arguments=[>--dataset_path", input_images,  
              "--model_name", "inception",  
              "--label_dir", label_dir,  
              "--output_dir", output_dir,  
              "--batch_size", batch_size_param],  
    compute_target=compute_target,  
    inputs=[input_images, label_dir],  
    outputs=[output_dir],  
    runconfig=amlcompute_run_config,  
    source_directory=scripts_folder
```

## Run the pipeline

Now run the pipeline, and examine the output it produced. The output has a score corresponding to each input image.

```
# Run the pipeline  
pipeline = Pipeline(workspace=ws, steps=[batch_score_step])  
pipeline_run = Experiment(ws, 'batch_scoring').submit(pipeline, pipeline_params={"param_batch_size": 20})  
  
# Wait for the run to finish (this might take several minutes)  
pipeline_run.wait_for_completion(show_output=True)  
  
# Download and review the output  
step_run = list(pipeline_run.get_children())[0]  
step_run.download_file("./outputs/result-labels.txt")  
  
import pandas as pd  
df = pd.read_csv("result-labels.txt", delimiter=":", header=None)  
df.columns = ["Filename", "Prediction"]  
df.head()
```

## Publish the pipeline

After you're satisfied with the outcome of the run, publish the pipeline so you can run it with different input values later. When you publish a pipeline, you get a REST endpoint. This endpoint accepts invoking of the pipeline with the set of parameters you have already incorporated by using [PipelineParameter](#).

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

## Rerun the pipeline by using the REST endpoint

To rerun the pipeline, you'll need an Azure Active Directory authentication header token, as described in [AzureCliAuthentication class](#).

```
from azureml.pipeline.core import PublishedPipeline

rest_endpoint = published_pipeline.endpoint
# specify batch size when running the pipeline
response = requests.post(rest_endpoint,
    headers=aad_token,
    json={"ExperimentName": "batch_scoring",
          "ParameterAssignments": {"param_batch_size": 50}})

# Monitor the run
from azureml.pipeline.core.run import PipelineRun
published_pipeline_run = PipelineRun(ws.experiments["batch_scoring"], run_id)

RunDetails(published_pipeline_run).show()
```

## Next steps

To see this working end-to-end, try the batch scoring notebook in [GitHub](#).

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

# Collect data for models in production

2/25/2019 • 4 minutes to read

In this article, you can learn how to collect input model data from the Azure Machine Learning services you've deployed into Azure Kubernetes Cluster (AKS) into an Azure Blob storage.

Once enabled, this data you collect helps you:

- Monitor data drifts as production data enters your model
- Make better decisions on when to retrain or optimize your model
- Retrain your model with the data collected

## What is collected and where does it go?

The following data can be collected:

- Model **input** data from web services deployed in Azure Kubernetes Cluster (AKS) (Voice, images, and video are **not** collected)
- Model predictions using production input data.

### NOTE

Pre-aggregation or pre-calculations on this data are not part of the service at this time.

The output gets saved in an Azure Blob. Since the data gets added into an Azure Blob, you can then 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>/<identifier>/<year>/<month>/<day>/data.csv  
# example: /modeldata/1a2b3c4d-5e6f-7g8h-9i10-j11k12l13m14/myresourcegrp/myworkspace/aks-w-col1v9/best_model/10/inputs/2018/12/31/data.csv
```

## 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 service](#) today.
- An Azure Machine Learning service workspace, a local directory containing your scripts, and the Azure Machine Learning SDK for Python installed. Learn how to get these prerequisites using the [How to configure a development environment](#) document.
- A trained machine learning model to be deployed to Azure Kubernetes Service (AKS). If you don't have one, see the [train image classification model](#) tutorial.
- An Azure Kubernetes Service cluster. For information on how to create and deploy to one, see the [How to deploy and where](#) document.
- [Set up your environment](#) and install the [Monitoring SDK](#).

# Enable data collection

Data collection can be enabled regardless of the model being deployed through Azure Machine Learning Service or other tools.

To enable it, 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", identifier="inputs", feature_names=["feat1", "feat2",
"feat3", "feat4", "feat5", "feat6"])
prediction_dc = ModelDataCollector("best_model", identifier="predictions", feature_names=
["prediction1", "prediction2"])
```

*CorrelationId* is an optional parameter, you do not need to set it up if your model doesn't require it. Having a correlationId in place does help you for easier mapping with other data. (Examples include: LoanNumber, CustomerId, etc.)

*Identifier* is later used for building the folder structure in your Blob, it can be used to divide "raw" data versus "processed".

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, so you must update your configuration file such as:

```
aks_config = AksWebservice.deploy_configuration(collect_model_data=True)
```

AppInsights for service monitoring can also be turned on 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 service, see the [How to deploy and where](#) document.

If you already have a service with the dependencies installed in your **environment file** and **scoring file**, enable data collection by:

1. Go to [Azure Portal](#).
2. Open your workspace.
3. Go to **Deployments** -> **Select service** -> **Edit**.

Home > Machine Learning Workspaces > doc-ws

**doc-ws**  
Machine Learning workspace

Experiments Compute Models Images Deployments Activities

## cvscodebservice

← Back to Deployments  Edit  Delete

Details Models Images

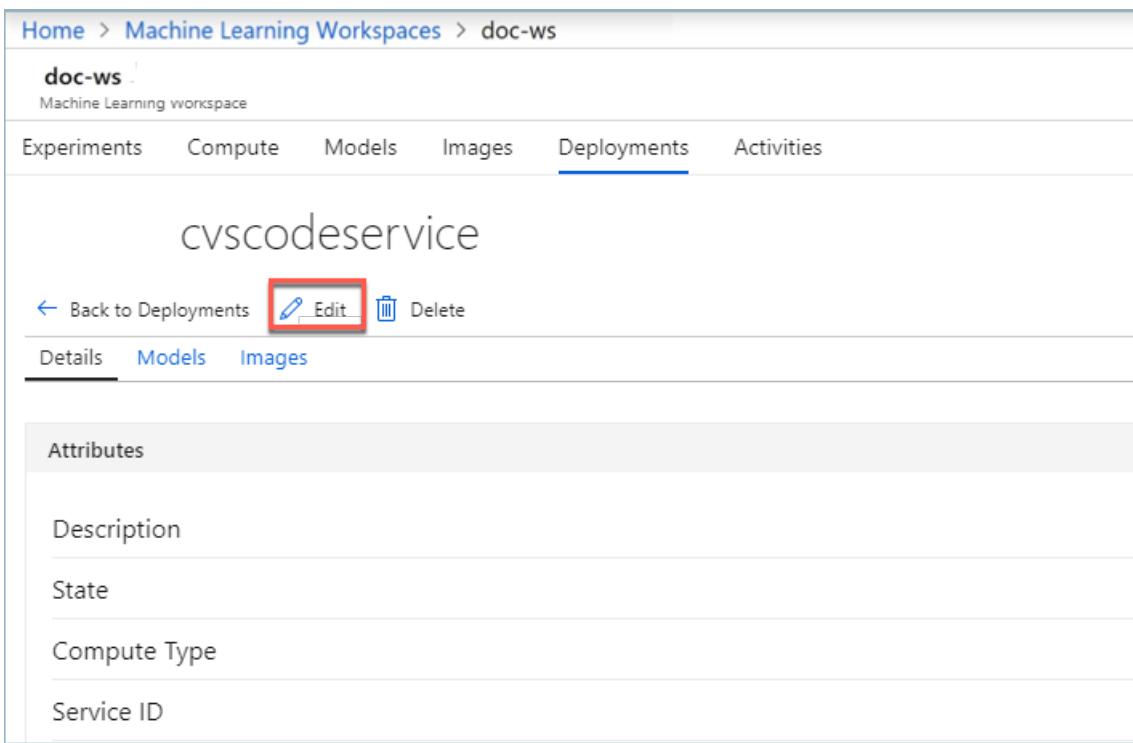
Attributes

Description

State

Compute Type

Service ID



4. In **Advanced Settings**, deselect **Enable Model data collection**.

Home > Machine Learning Workspaces > doc-ws

**doc-ws**  
Machine Learning Workspace

Experiments Compute Models Images Deployments Activities

## Update Deployment

\* Name  
cvscodebservice

Description

Tags  
Separate tags with commas

Compute Settings

\* Compute Type  
AKS

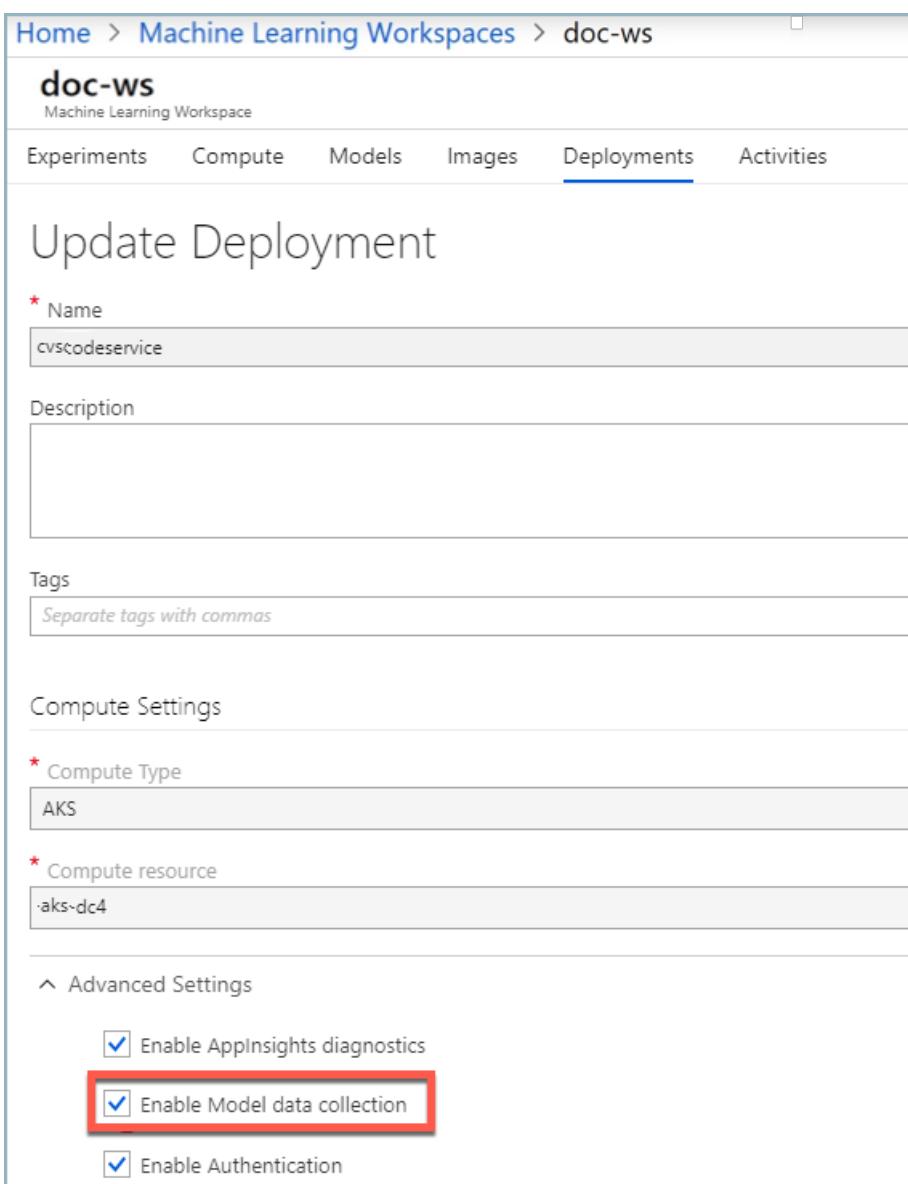
\* Compute resource  
aks-dc4

Advanced Settings

Enable AppInsights diagnostics

**Enable Model data collection**

Enable Authentication



In this window, you can also choose to "Enable Appinsights diagnostics" to track the health of your service.

5. Select **Update** to apply the change.

## Disable data collection

You can stop collecting data any time. Use Python code or the Azure portal to disable data collection.

- Option 1 - Disable in the Azure portal:

1. Sign in to [Azure portal](#).
2. Open your workspace.
3. Go to **Deployments** -> **Select service** -> **Edit**.

The screenshot shows the Azure Machine Learning Workspaces interface. The top navigation bar includes 'Home', 'Machine Learning Workspaces', and 'doc-ws'. Below this, the workspace name 'doc-ws' is shown as a 'Machine Learning workspace'. The main menu has tabs for 'Experiments', 'Compute', 'Models', 'Images', 'Deployments' (which is underlined), and 'Activities'. Under 'Deployments', the service 'cvscodeservice' is listed. Below the service name are buttons for 'Back to Deployments', 'Edit' (which is highlighted with a red box), and 'Delete'. A sub-menu below 'Edit' shows 'Details', 'Models', and 'Images', with 'Details' being the active tab. The 'Attributes' section contains fields for 'Description', 'State', 'Compute Type', and 'Service ID'.

4. In **Advanced Settings**, deselect **Enable Model data collection**.

The screenshot shows the 'Advanced Settings' section. It includes three checkboxes: 'Enable AppInsights diagnostics' (checked), 'Enable Model data collection' (unchecked and highlighted with a red box), and 'Enable Authentication' (checked).

5. Select **Update** to apply the change.

- Option 2 - Use Python to disable data collection:

```
## replace <service_name> with the name of the web service
<service_name>.update(collect_model_data=False)
```

## Validate your data and analyze it

You can choose any tool of your preference to analyze the data collected into your Azure Blob.

To quickly access the data from your blob:

1. Sign in to [Azure portal](#).

2. Open your workspace.

3. Click on **Storage**.

The screenshot shows the Azure Machine Learning service workspace - PREVIEW interface. On the left, there's a sidebar with links: Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main area displays workspace details: Resource group (<resource group>), Location East US 2, Subscription (<subscription name>), Subscription ID (<SubscriptionID>), and Storage (<Blob storage name>). A red box highlights the 'Storage' section. Below the workspace details, there are links for Registry (<registry name>), Key Vault (<key vault name>), Application Insights (<App Insights name>), and a 'Delete' button.

4. Follow the path to the output data in the blob with this syntax:

```
/modeldata/<subscriptionid>/<resourcegroup>/<workspace>/<webservice>/<model>/<version>/<identifier>/<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
```

## Analyzing model data through Power BI

1. Download and Open [PowerBI Desktop](#)

2. Select **Get Data** and click on [Azure Blob Storage](#).

The screenshot shows the Power BI 'Get Data' interface. A search bar at the top contains 'azure'. The left sidebar has a tree view with 'All' selected, showing 'All', 'Azure', and 'Online Services'. Under 'All', 'Azure Blob Storage' is highlighted with a yellow bar and selected. The right pane lists various Azure services: Azure SQL database, Azure SQL Data Warehouse, Azure Analysis Services database, 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), and Azure KustoDB (Beta). At the bottom, there are links for 'Certified Connectors', 'Connect' (in a yellow button), and 'Cancel'.

3. Add your storage account name and enter your storage key. You can find this information in your blob's **Settings** >> Access keys.

4. Select the container **modeldata** and click on **Edit**.

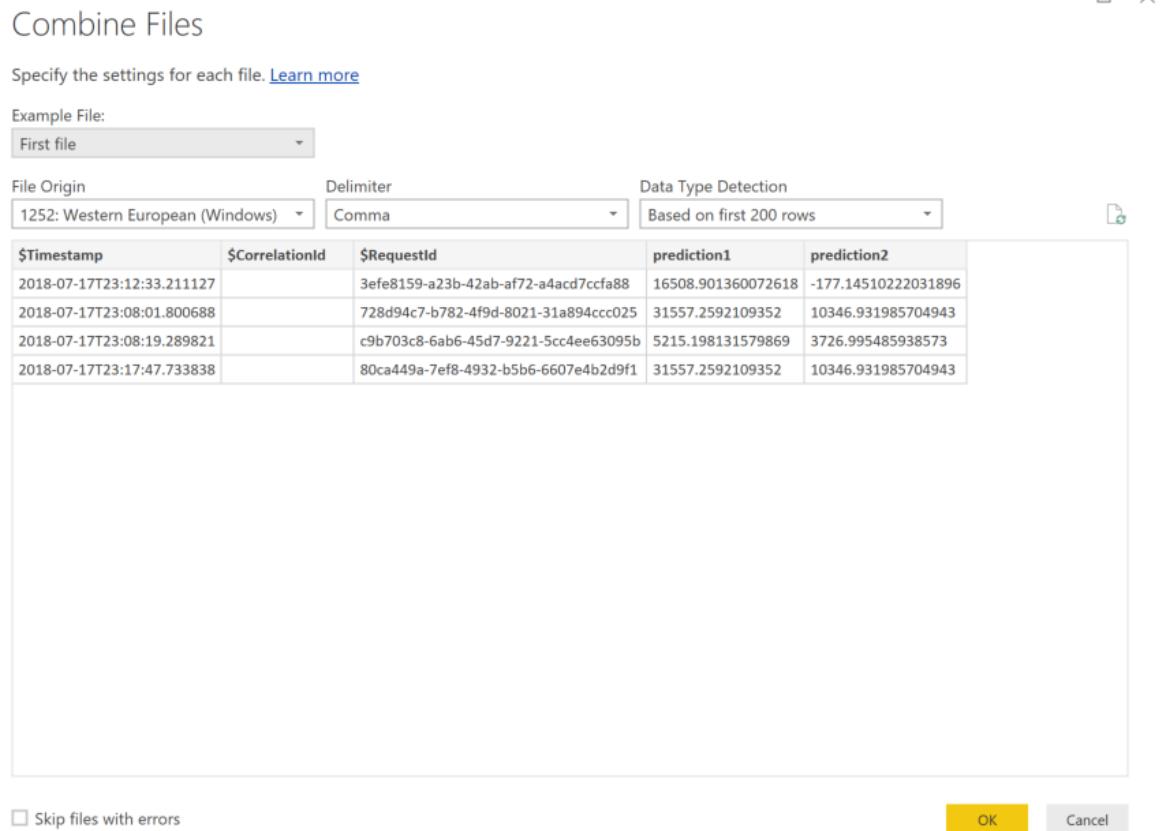
## Navigator

The screenshot shows the Azure ML Navigator interface. On the left, there's a 'Display Options' dropdown menu with several items: mystorage, amlbdpackages, azureml, and modeldata. The 'modeldata' item is selected, indicated by a checked checkbox. To the right of the menu is a large list of file paths under the heading 'modeldata'. The paths listed are all identical: 'earxg.blob.core.windows.net/modeldata/'. At the bottom of the interface are three buttons: 'Load' (yellow), 'Edit' (red box), and 'Cancel'.

5. In the query editor, click under "Name" column and add your Storage account 1. Model path into the filter.  
Note: if you want to only look into files from a specific year or month, just expand the filter path. For example, just look into March data:  
`/modeldata/subscriptionid>/resourcegroupname>/workspacename>/webservicename>/modelname>/modelversion>/identifier>/year>/3`
6. Filter the data that is relevant to you based on **Name**. If you stored **predictions** and **inputs** you'll need to do create a query per each.
7. Click on the double arrow aside the **Content** column to combine the files.

The screenshot shows the 'Content' pane of the Azure ML interface. It lists four items, all labeled 'Binary'. To the right of the list is a double arrow icon, which is highlighted with a red box. This icon is used to merge multiple files into a single combined file.

8. Click OK and the data will preload.



9. You can now click **Close and Apply**.

10. If you added inputs and predictions your tables will automatically correlate by **RequestId**.

11. Start building your custom reports on your model data.

### Analyzing model data using Databricks

1. Create a [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. 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 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 = "Jhsduhwe908rjjfoieudnf 98h v9udfhgb987yh g908ufgb98yfgb9 ufgb78gy8 gfo89ug98yfdg7yfg8tg9fyg87"
```

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)
```

6. Follow the steps on the template in order to view and analyze your data.

## Example notebook

The [how-to-use-azureml/deployment/enable-data-collection-for-models-in-aks/enable-data-collection-for-models-in-aks.ipynb](#) notebook demonstrates concepts in this article.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

# Monitor your Azure Machine Learning models with Application Insights

2/25/2019 • 2 minutes to read

In this article, you learn how to set up Azure Application Insights for your Azure Machine Learning service. Application Insights gives you the opportunity to monitor:

- Request rates, response times, and failure rates.
- Dependency rates, response times, and failure rates.
- Exceptions.

[Learn more about Application Insights.](#)

## 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 service](#) today.
- An Azure Machine Learning workspace, a local directory that contains your scripts, and the Azure Machine Learning SDK for Python installed. To learn how to get these prerequisites, see [How to configure a development environment](#).
- A trained machine learning model to be deployed to Azure Kubernetes Service (AKS) or Azure Container Instance (ACI). If you don't have one, see the [Train image classification model](#) tutorial.

## Enable and disable from the SDK

### Update a deployed 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 Application Insights.

```
aks_service.update(enable_app_insights=True)
```

### Log custom traces in your service

If you want 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.

```
print ("model initialized" + time.strftime("%H:%M:%S"))
```

2. Update the service configuration.

```
config = Webservice.deploy_configuration(enable_app_insights=True)
```

3. Build an image and deploy it on [AKS](#) or [ACI](#).

### Disable tracking in Python

To disable Application Insights, use the following code:

```
## replace <service_name> with the name of the web service
<service_name>.update(enable_app_insights=False)
```

## Enable and disable in the portal

You can enable and disable Application Insights in the Azure portal.

1. In the [Azure portal](#), open your workspace.
2. On the **Deployments** tab, select the service where you want to enable Application Insights.

The screenshot shows the 'Myworkspace' machine learning workspace in the Azure portal. The 'Deployments' tab is selected. The page header includes 'Home > All resources > Myworkspace > Myworkspace'. Below the header, the workspace name 'Myworkspace' and description 'Machine Learning Workspace' are displayed. The 'Deployments' tab is underlined, indicating it is active. Other tabs shown include 'Experiments', 'Compute', 'Models', 'Images', 'Activities', and 'Deployments'. The main content area is titled 'Deployments' and contains a table of deployment history. The table has columns for 'NAME' and 'LAST UPDATED'. Five entries are listed:

NAME	LAST UPDATED
acimlc1	09/21/2018, 5:14:47 PM UTC
aks-w-dc2	09/19/2018, 8:29:22 PM UTC
aks-w-dc3	01/01/1, 12:00:00 AM UTC
aks-w-dc1	01/01/1, 12:00:00 AM UTC

At the top of the deployment list, there are buttons for 'Refresh', 'Edit', and 'Delete'.

3. Select **Edit**.

Home > All resources > Myworkspace > Myworkspace

## Myworkspace

Machine Learning Workspace

Experiments Compute Models Images Deployments Activities

### aks-w-dc2

Back to Deployments Edit Delete

Details Models Images

Attributes	
Description	
State	Healthy
Compute Type	AKS
Service ID	aks-w-dc2
Tags	
Creation date	09/19/2018, 8:04:06 PM UTC
Last updated	09/19/2018, 8:29:22 PM UTC

4. In **Advanced Settings**, select the **Enable AppInsights diagnostics** check box.

Advanced Settings

- Enable AppInsights diagnostics
- Enable Model data collection
- Enable Authentication

5. Select **Update** at the bottom of the screen to apply the changes.

### Disable

1. In the [Azure portal](#), open your workspace.
2. Select **Deployments**, select the service, and select **Edit**.

Home > All resources > Myworkspace > Myworkspace

## Myworkspace

Machine Learning Workspace

Experiments Compute Models Images Deployments Activities

aks-w-dc2

[← Back to Deployments](#) [Edit](#) [Delete](#)

Details Models Images

Attributes	
Description	
State	Healthy
Compute Type	AKS
Service ID	aks-w-dc2
Tags	
Creation date	09/19/2018, 8:04:06 PM UTC
Last updated	09/19/2018, 8:29:22 PM UTC

3. In **Advanced Settings**, clear the **Enable AppInsights diagnostics** check box.

^ Advanced Settings

- Enable AppInsights diagnostics**
- Enable Model data collection
- Enable Authentication

4. Select **Update** at the bottom of the screen to apply the changes.

## Evaluate data

Your service's data is stored in your Application Insights account, within the same resource group as your Azure Machine Learning service. To view it:

1. Go to your Machine Learning service workspace in the [Azure portal](#) and click on Application Insights link.

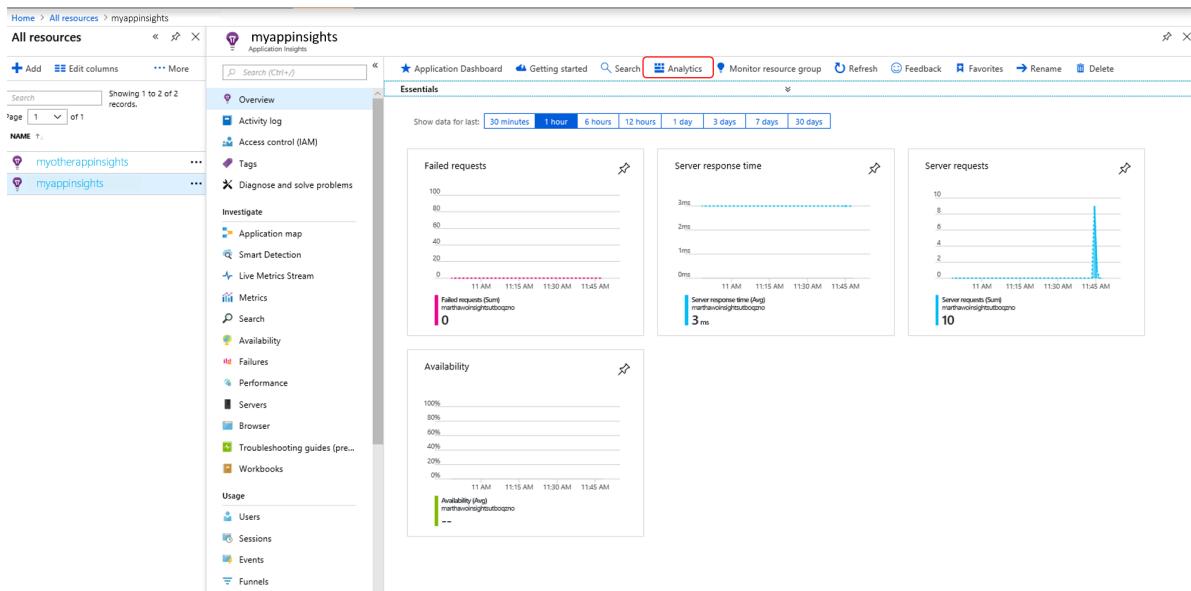
My Workspace  
Machine Learning service workspace - PREVIEW

Search (Ctrl+/  
Delete

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

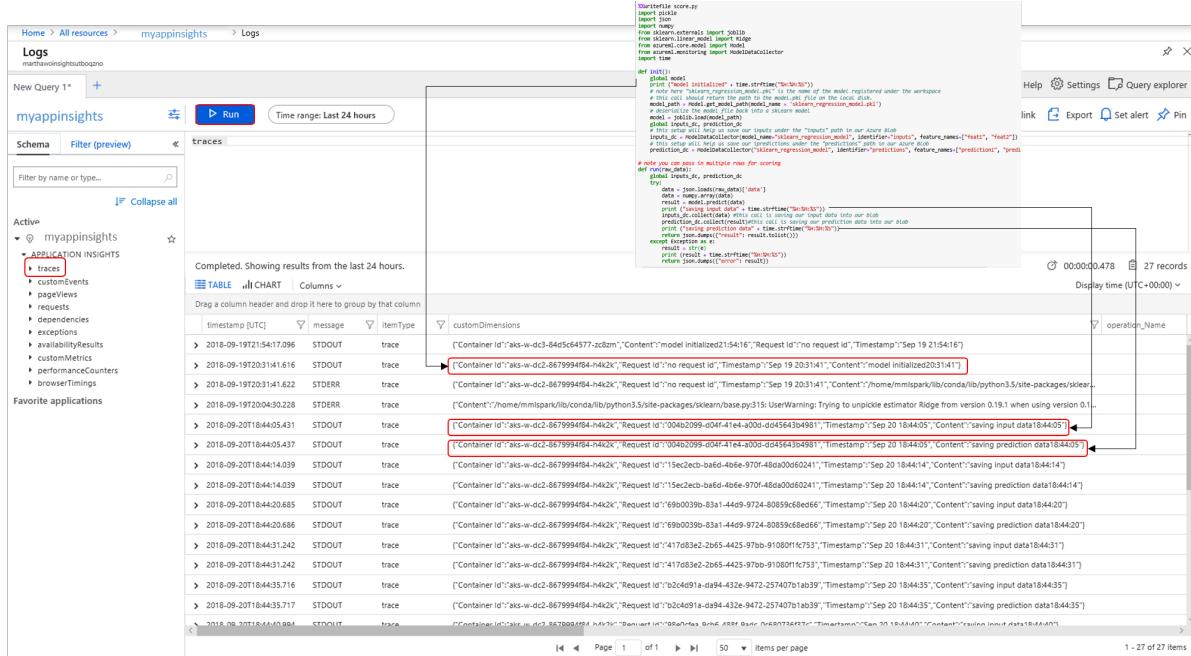
Resource group <resource group>	Storage <blob storage name>
Location East US 2	Registry <registry name>
Subscription <subscription name>	Key Vault <key vault name>
Subscription ID <SubscriptionID>	Application Insights <App Insights name>

2. Select the **Overview** tab to see a basic set of metrics for your service.



3. To look into your custom traces, select **Analytics**.

4. In the schema section, select **Traces**. Then select **Run** to run your query. Data should appear in a table format and should map to your custom calls in your scoring file.



To learn more about how to use Application Insights, see [What is Application Insights?](#).

## Example notebook

The [how-to-use-azureml/deployment/enable-app-insights-in-production-service/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](#).

## Next steps

You can also collect data on your models in production. Read the article [Collect data for models in production](#).

## Other references

- [Azure Monitor for containers](#)



# Securely run experiments and inferencing inside an Azure virtual network

3/1/2019 • 7 minutes to read

In this article, you learn how to run your experiments and inferencing inside a virtual network. A virtual network acts as a security boundary, isolating your Azure resources from the public internet. You can also join an Azure virtual network to your on-premises network. It allows you to securely train your models and access your deployed models for inferencing.

The Azure Machine Learning service relies on other Azure services for compute resources. Compute resources (compute targets) are used to train and deploy models. These compute targets can be created inside a virtual network. For example, you can use the Microsoft Data Science Virtual Machine to train a model and then deploy the model to Azure Kubernetes Service (AKS). For more information about virtual networks, see the [Azure Virtual Network overview](#).

## Prerequisites

This document assumes that you are familiar with Azure Virtual Networks, and IP networking in general. This document also assumes that you have created a virtual network and subnet to use with your compute resources. If you are not familiar with Azure Virtual Networks, read the following articles to learn about the service:

- [IP addressing](#)
- [Security groups](#)
- [Quickstart: Create a virtual network](#)
- [Filter network traffic](#)

## Storage account for your workspace

When you create an Azure Machine Learning service workspace, it requires an Azure Storage account. Don't turn on firewall rules for this storage account. The Azure Machine Learning service requires unrestricted access to the storage account.

If you aren't sure if you've modified these settings or not, see **Change the default network access rule** in [Configure Azure Storage firewalls and virtual networks](#). Use the steps to allow access from all networks.

## Use Machine Learning Compute

To use Azure Machine Learning Compute in a virtual network, use the following information about network requirements:

- The virtual network must be in the same subscription and region as the Azure Machine Learning service workspace.
- The subnet specified for the Machine Learning Compute cluster must have enough unassigned IP addresses to accommodate the number of VMs targeted for the cluster. If the subnet doesn't have enough unassigned IP addresses, the cluster will be partially allocated.
- If you plan to secure the virtual network by restricting traffic, leave some ports open for the Machine Learning Compute service. For more information, see [Required ports](#).
- Check whether your security policies or locks on the virtual network's subscription or resource group

restrict permissions to manage the virtual network.

- If you are going to put multiple Machine Learning Compute clusters in one virtual network, you may need to request a quota increase for one or more of your resources.

Machine Learning Compute automatically allocates additional networking resources in the resource group that contains the virtual network. For each Machine Learning Compute cluster, the Azure Machine Learning service allocates the following resources:

- One network security group (NSG)
- One public IP address
- One load balancer

These resources are limited by the subscription's [resource quotas](#).

## Required ports

Machine Learning Compute currently uses the Azure Batch service to provision VMs in the specified virtual network. The subnet must allow inbound communication from the Batch service. This communication is used to schedule runs on the Machine Learning Compute nodes and to communicate with Azure Storage and other resources. Batch adds 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 Batch service role IP addresses.
- Inbound TCP traffic on port 22 to permit remote access.
- Outbound traffic on any port to the virtual network.
- Outbound traffic on any port to the internet.

Exercise caution if you modify or add inbound/outbound rules in Batch-configured NSGs. If an NSG blocks communication to the compute nodes, then the Machine Learning Compute services sets the state of the compute nodes to unusable.

You don't need to specify NSGs at the subnet level because Batch configures its own NSGs. However, if the specified subnet has associated NSGs and/or a firewall, configure the inbound and outbound security rules as mentioned earlier. The following screenshots show how the rule configuration looks in the Azure portal:

**Inbound security rules:**

PRIORITY	NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
1010	JupyterHub	8000	TCP	Any	Any	Allow
1020	RStudioServer	8787	TCP	Any	Any	Allow
1030	Port_22_AML	22	Any	AzureMachineLe...	Any	Allow
1040	Port_29876-29877	29876-29877	TCP	Any	Any	Allow
1050	Port_22_SSH	22	TCP	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalanc...	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

**Outbound security 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

## Create Machine Learning Compute in a virtual network

To create a Machine Learning Compute cluster by using the Azure portal, follow these steps:

1. In the [Azure portal](#), select your Azure Machine Learning service workspace.
2. In the **Application** section, select **Compute**. Then select **Add compute**.

NAME	TYPE
mydsvm	Virtual Machine
dsvmvmnetval2	Virtual Machine
aks-vnet	Kubernetes Service
aks-new	Kubernetes Service
my-aks-1	Kubernetes Service
my-aks-2	Kubernetes Service
my-aks-3	Kubernetes Service
my-aks-4	Kubernetes Service
my-aks-6	Kubernetes Service

3. To configure this compute resource to use a virtual network, use these options:

- **Network configuration:** Select **Advanced**.
- **Resource group:** Select the resource group that contains the virtual network.
- **Virtual network:** Select the virtual network that contains the subnet.
- **Subnet:** Select the subnet to use.

The sidebar shows the following navigation items:

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Settings**
- Locks
- Automation script
- Properties
- Application**
- Experiments
- Pipelines
- Compute** (selected)
- Models
- Images
- Deployments
- Activities
- Support + troubleshooting
- Usage + quotas
- New support request

## Add Compute

\* Compute name

\* Compute type

**i** Machine Learning Compute is a managed training environment consisting of one or more nodes. [Learn more.](#)

\* Region

\* Virtual Machine size  Choose Virtual Machine size

\* Virtual Machine priority  Dedicated  Low-priority

\* Minimum number of nodes

\* Maximum number of nodes

\* Idle seconds before scale down

\* Network configuration  Advanced

Resource group

Virtual network

Subnet

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 cpubluster")
except ComputeTargetException:
    print("Creating new cpubluster")

# 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 complete, show the output log
cpu_cluster.wait_for_completion(show_output=True)
```

When the creation process finishes, you can train your model by using the cluster. For more information, see [Select and use a compute target for training](#).

# Use a virtual machine or HDInsight cluster

To use a virtual machine or Azure HDInsight cluster in a virtual network with your workspace, follow these steps:

## IMPORTANT

The Azure Machine Learning service only supports virtual machines that are running Ubuntu.

1. Create a VM or HDInsight cluster by using the Azure portal or Azure CLI, and put it in an Azure virtual network. For more information, see the following documents:
  - [Create and manage Azure virtual networks for Linux VMs](#)
  - [Extend HDInsight using an Azure virtual network](#)
2. To allow the Azure Machine Learning service to communicate with the SSH port on the VM or cluster, you must configure a source entry for the NSG. The SSH port is usually port 22. To allow traffic from this source, use the following information:
  - **Source:** Select **Service Tag**.
  - **Source service tag:** Select **AzureMachineLearning**.
  - **Source port ranges:** Select **\***.
  - **Destination:** Select **Any**.
  - **Destination port ranges:** Select **22**.
  - **Protocol:** Select **Any**.
  - **Action:** Select **Allow**.

The screenshot shows the Azure portal interface for managing Network Security Groups (NSGs). On the left, the 'Inbound security rules' section is selected under the 'Network interfaces' blade. A new rule is being added, as indicated by the 'Add' button in the top right of the main pane. The 'Basic' tab is selected in the dialog. The configuration for the new rule is as follows:

- Source:** Service Tag (highlighted)
- Source service tag:** AzureMachineLearning (highlighted)
- Source port ranges:** \*
- Destination:** Any (highlighted)
- Destination port ranges:** 22 (highlighted)
- Protocol:** Any (TCP and UDP are also listed)
- Action:** Allow (highlighted)
- Priority:** 1030
- Name:** Port\_22\_AML
- Description:** (empty)

Keep the default outbound rules for the NSG. For more information, see the default security rules in [Security groups](#).

3. Attach the VM or HDInsight cluster to your Azure Machine Learning service workspace. For more information, see [Set up compute targets for model training](#).

## Use Azure Kubernetes Service

## IMPORTANT

Check the prerequisites and plan IP addressing for your cluster before proceeding with the steps. For more information, see [Configure advanced networking in Azure Kubernetes Service](#).  
Keep the default outbound rules for the NSG. For more information, see the default security rules in [Security groups](#).  
Azure Kubernetes Service and the Azure virtual network should be in the same region.

To add Azure Kubernetes Service in a virtual network to your workspace, follow these steps in the Azure portal:

1. In the [Azure portal](#), select your Azure Machine Learning service workspace.
2. In the **Application** section, select **Compute**. Then select **Add compute**.

NAME	TYPE
mydsvm	Virtual Machine
dsvmvnetval2	Virtual Machine
aks-vnet	Kubernetes Service
aks-new	Kubernetes Service
my-aks-1	Kubernetes Service
my-aks-2	Kubernetes Service
my-aks-3	Kubernetes Service
my-aks-4	Kubernetes Service
my-aks-6	Kubernetes Service

3. To configure this compute resource to use a virtual network, use these options:

- **Network configuration:** Select **Advanced**.
- **Resource group:** Select the resource group that contains the virtual network.
- **Virtual network:** Select the virtual network that contains the subnet.
- **Subnet:** Select the subnet.
- **Kubernetes Service address range:** Select the Kubernetes service address range. This address range uses a CIDR notation IP range to define the IP addresses available for the cluster. It must not overlap with any subnet IP ranges. For example: 10.0.0.0/16.
- **Kubernetes DNS service IP address:** Select 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.

- **Docker bridge address:** Select 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.

## Add Compute

\* Compute name

\* Compute type

\* Kubernetes Service  Create new  Use existing

\* Region

\* Virtual Machine size

\* Number of nodes

\* Network configuration   Basic  Advanced

---

\* Resource group

\* Virtual network

\* Subnet

\* Kubernetes Service address range

\* Kubernetes DNS service IP address

\* Docker bridge address

TIP

If you already have an AKS cluster in a virtual network, you can attach it to the workspace. For more information, see [How to deploy to AKS](#).

You can also use the **Azure Machine Learning SDK** to add Azure Kubernetes Service in a virtual network. The following code creates a new Azure Kubernetes Service instance in the `default` subnet of a virtual network named `mynetwork`:

When the creation process is completed, you can do inferencing on an AKS cluster behind a virtual network. For more information, see [How to deploy to AKS](#).

## Next steps

- [Set up training environments](#)
- [Where to deploy models](#)
- [Securely deploy models with SSL](#)

# Use SSL to secure web services with Azure Machine Learning service

2/11/2019 • 3 minutes to read

In this article, you will learn how to secure a web service deployed with the Azure Machine Learning service. You can restrict access to web services and secure the data submitted by clients using secure socket layers (SSL) and key-based authentication.

## WARNING

If you do not enable SSL, any user on the internet will be able to make calls to the web service.

SSL encrypts data sent between the client and the web service. It also used by the client to verify the identity of the server. Authentication is only enabled for services that have provided an SSL certificate and key. If you enable SSL, an authentication key is required when accessing the web service.

Whether you deploy a web service enabled with SSL or you enable SSL for existing deployed web service, the steps are the same:

1. Get a domain name.
2. Get an SSL certificate.
3. Deploy or update the web service with the SSL setting enabled.
4. Update your DNS to point to the web service.

There are slight differences when securing web services across the [deployment targets](#).

## Get a domain name

If you do not already own a domain name, you can purchase one from a **domain name registrar**. The process differs between registrars, as does the cost. The registrar also provides you with tools for managing the domain name. These tools are used to map a fully qualified domain name (such as www.contoso.com) to the IP address hosting your web service.

## Get an SSL certificate

There are many ways to get an SSL certificate. The most common is to purchase one from a **Certificate Authority** (CA). Regardless of where you obtain the certificate, you need the following files:

- A **certificate**. The certificate must contain the full certificate chain, and must be PEM-encoded.
- A **key**. The key must be PEM-encoded.

When requesting a certificate, you must provide the fully qualified domain name (FQDN) of the address you plan to use for the web service. For example, www.contoso.com. The address stamped into the certificate and the address used by the clients are compared when validating the identity of the web service. If the addresses do not match, the clients will receive an error.

**TIP**

If the Certificate Authority cannot provide the certificate and key as PEM-encoded files, you can use a utility such as [OpenSSL](#) to change the format.

**WARNING**

Self-signed certificates should be used only for development. They should not be used in production. Self-signed certificates can cause problems in your client applications. For more information, see the documentation for the network libraries used in your client application.

## Enable SSL and deploy

To deploy (or re-deploy) the service with SSL enabled, set the `ssl_enabled` parameter to `True`, wherever applicable. Set the `ssl_certificate` parameter to the value of the `certificate` file and the `ssl_key` to the value of the `key` file.

- **Deploy on Azure Kubernetes Service (AKS)**

While provisioning the AKS cluster, provide values for SSL-related parameters as shown in the code snippet:

```
from azureml.core.compute import AksCompute

provisioning_config = AksCompute.provisioning_configuration(ssl_cert_pem_file="cert.pem",
    ssl_key_pem_file="key.pem", ssl_cname="www.contoso.com")
```

- **Deploy on Azure Container Instances (ACI)**

While deploying to ACI, provide values for SSL-related parameters as shown in the code snippet:

```
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")
```

- **Deploy on Field Programmable Gate Arrays (FPGA)**

While deploying to FPGA, provide values for the SSL-related parameters as shown in the code snippet:

```
from azureml.contrib.brainwave import BrainwaveWebservice

deployment_config = BrainwaveWebservice.deploy_configuration(ssl_enabled=True,
    ssl_cert_pem_file="cert.pem", ssl_key_pem_file="key.pem")
```

## Update your DNS

Next, you must update your DNS to point to the web service.

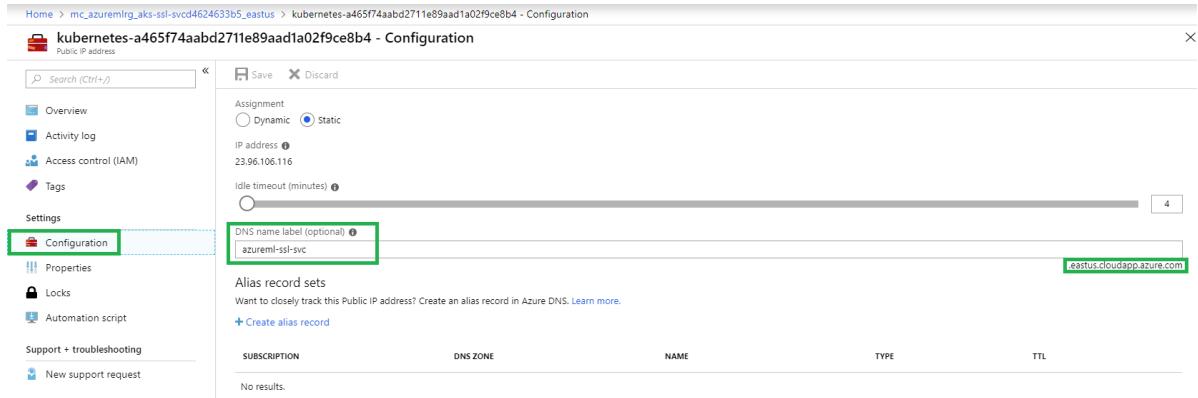
- **For ACI:**

Use the tools provided by your domain name registrar to update the DNS record for your domain name. The record must point to the IP address of the service.

Depending on the registrar, and the time to live (TTL) configured for the domain name, it can take several minutes to several hours before clients can resolve the domain name.

- **For AKS:**

Update the DNS under the "Configuration" tab of the "Public IP Address" of the AKS cluster as shown in the image. You can find the Public IP Address as one of the resource types created under the resource group that contains the AKS agent nodes and other networking resources.



## Next steps

Learn how to:

- [Consume a machine learning model deployed as a web service](#)
- [Securely run experiments and inferencing inside an Azure Virtual Network](#)

# Create and run a machine learning pipeline by using Azure Machine Learning SDK

3/1/2019 • 9 minutes to read

In this article, you learn how to create, publish, run, and track a [machine learning pipeline](#) by using the [Azure Machine Learning SDK](#). These pipelines help create and manage the workflows that stitch together various machine learning phases. Each phase of a pipeline, such as data preparation and model training, can include one or more steps.

The pipelines you create are visible to the members of your Azure Machine Learning service [workspace](#).

Pipelines use remote compute targets for computation and the storage of the intermediate and final data associated with that pipeline. 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 service](#).

## Prerequisites

- [Configure your development environment](#) to install the Azure Machine Learning SDK.
- Create an [Azure Machine Learning workspace](#) to hold all your pipeline resources.

```
ws = Workspace.create(  
    name = '<workspace-name>',  
    subscription_id = '<subscription-id>',  
    resource_group = '<resource-group>',  
    location = '<workspace_region>',  
    exist_ok = True)
```

## Set up machine learning resources

Create the resources required to run a pipeline:

- Set up a datastore used to access the data needed in the pipeline steps.
- Configure a [DataReference](#) object to point to data that lives in, or is accessible in, a datastore.
- 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 by default. Azure Files is the default datastore for a workspace, but you can also use Blob storage as a datastore. To learn more, see [Deciding when to use Azure Files, Azure Blobs, or Azure Disks](#).

```
# Default datastore (Azure file storage)
def_data_store = ws.get_default_datastore()

# The above call is equivalent to this
def_data_store = Datastore(ws, "workspacefilestore")

# Get blob storage associated with the workspace
def_blob_store = Datastore(ws, "workspaceblobstore")
```

Upload data files or directories to the datastore for them to be accessible from your pipelines. This example uses the Blob storage version of the datastore:

```
def_blob_store.upload_files(
    ["./data/20news.pk1"],
    target_path="20newsgroups",
    overwrite=True)
```

A pipeline consists of one or more steps. A step is a unit run on a compute target. Steps might consume data sources and produce “intermediate” 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.

### Configure data reference

You just created a data source that can be referenced in a pipeline as an input to a step. A data source in a pipeline is represented by a [DataReference](#) object. The `DataReference` object points to data that lives in or is accessible from a datastore.

```
blob_input_data = DataReference(
    datastore=def_blob_store,
    data_reference_name="test_data",
    path_on_datastore="20newsgroups/20news.pk1")
```

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.

```
output_data1 = PipelineData(
    "output_data1",
    datastore=def_blob_store,
    output_name="output_data1")
```

## Set up 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 how to create and attach them to your workspace. The process for creating and or attaching a compute target is the same regardless of 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.

Below are examples of creating and attaching compute targets for:

- Azure Machine Learning Compute
- Azure Databricks
- Azure Data Lake Analytics

## Azure Machine Learning Compute

You can create an Azure Machine Learning compute for running your steps.

```
compute_name = "aml-compute"
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', # NC6 is GPU-enabled
                                                               min_nodes = 1,
                                                               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())
```

## 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 these 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:

```

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)

```

## 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)

```

#### 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.

## Construct your pipeline steps

Once you create and attach a compute target to your workspace, you are ready to define a pipeline step. There are many built-in steps available via the Azure Machine Learning SDK. The most basic of these steps is a [PythonScriptStep](#), which runs a Python script in a specified compute target:

```

trainStep = PythonScriptStep(
    script_name="train.py",
    arguments=["--input", blob_input_data, "--output", processed_data1],
    inputs=[blob_input_data],
    outputs=[processed_data1],
    compute_target=compute_target,
    source_directory=project_folder
)

```

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 the Azure Machine Learning service when you define the steps or build the pipeline.

```
# list of steps to run
compareModels = [trainStep, extractStep, compareStep]

# Build the pipeline
pipeline1 = Pipeline(workspace=ws, steps=[compareModels])
```

The following example uses the Azure Databricks compute target created earlier:

```
dbStep = DatabricksStep(
    name="databricksmodule",
    inputs=[step_1_input],
    outputs=[step_1_output],
    num_workers=1,
    notebook_path=notebook_path,
    notebook_params={'myparam': 'testparam'},
    run_name='demo run name',
    compute_target=databricks_compute,
    allow_reuse=False
)
# List of steps to run
steps = [dbStep]

# Build the pipeline
pipeline1 = Pipeline(workspace=ws, steps=steps)
```

For more information, see the [azure-pipeline-steps package](#) and [Pipeline class reference](#).

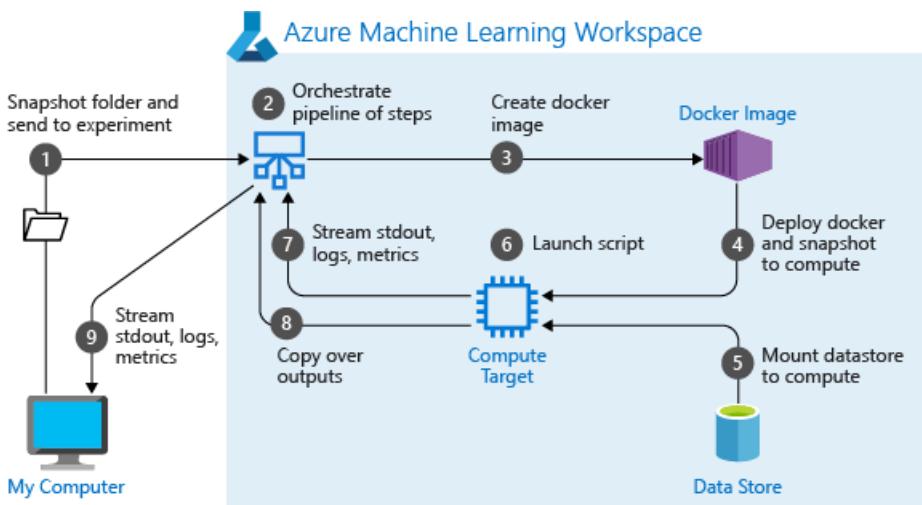
## Submit the pipeline

When you submit the pipeline, Azure Machine Learning service 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.

```
# 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.
- Mounts the datastore, if a `DataReference` object is specified in a step. If mount is not supported, 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.

## Publish a pipeline

You can publish a pipeline to run it with different inputs later. For the REST endpoint of an already published pipeline to accept parameters, you must parameterize the pipeline before publishing.

1. To create a pipeline parameter, use a [PipelineParameter](#) object with a default value.

```
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],
    target=compute_target,
    source_directory=project_folder)
```

3. Publish this pipeline that will accept a parameter when invoked.

```
published_pipeline1 = pipeline1.publish(
    name="My_Published_Pipeline",
    description="My Published Pipeline Description")
```

## Run a published pipeline

All published pipelines have a REST endpoint. This endpoint invokes the run of the pipeline from external systems, such as non-Python clients. This endpoint enables "managed repeatability" in batch scoring and retraining scenarios.

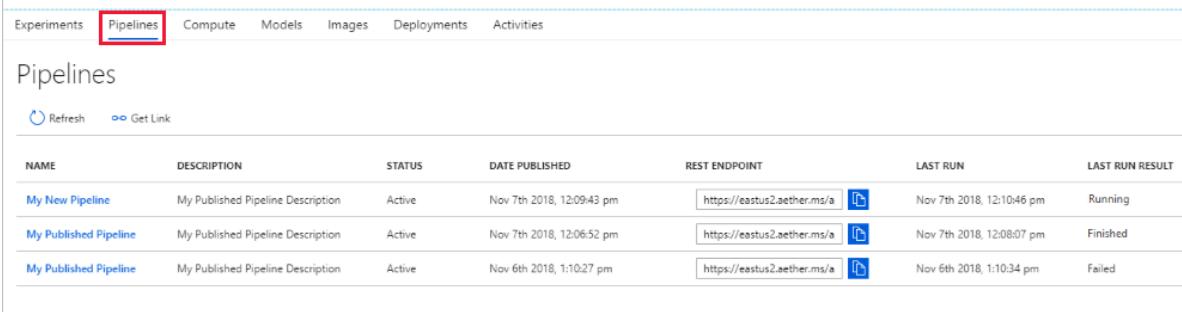
To invoke the run of the preceding pipeline, you need an Azure Active Directory authentication header token, as described in [AzureCliAuthentication class](#) or get more details at [Authentication in Azure Machine Learning](#) notebook.

```
response = requests.post(published_pipeline1.endpoint,
    headers=aad_token,
    json={"ExperimentName": "My_Pipeline",
        "ParameterAssignments": {"pipeline_arg": 20}})
```

## View results

See the list of all your pipelines and their run details:

1. Sign in to the [Azure portal](#).
2. [View your workspace](#) to find the list of pipelines.



The screenshot shows the Azure Machine Learning studio interface. At the top, there is a navigation bar with tabs: Experiments, Pipelines (which is highlighted with a red box), Compute, Models, Images, Deployments, and Activities. Below the navigation bar, the page title is 'Pipelines'. There are two buttons at the top left: 'Refresh' and 'Get Link'. The main content area displays a table of published pipelines. The columns are: NAME, DESCRIPTION, STATUS, DATE PUBLISHED, REST ENDPOINT, LAST RUN, and LAST RUN RESULT. The data in the table is as follows:

NAME	DESCRIPTION	STATUS	DATE PUBLISHED	REST ENDPOINT	LAST RUN	LAST RUN RESULT
My New Pipeline	My Published Pipeline Description	Active	Nov 7th 2018, 12:09:43 pm	<a href="https://eastus2.aether.ms/a">https://eastus2.aether.ms/a</a> 	Nov 7th 2018, 12:10:46 pm	Running
My Published Pipeline	My Published Pipeline Description	Active	Nov 7th 2018, 12:06:52 pm	<a href="https://eastus2.aether.ms/a">https://eastus2.aether.ms/a</a> 	Nov 7th 2018, 12:08:07 pm	Finished
My Published Pipeline	My Published Pipeline Description	Active	Nov 6th 2018, 1:10:27 pm	<a href="https://eastus2.aether.ms/a">https://eastus2.aether.ms/a</a> 	Nov 6th 2018, 1:10:34 pm	Failed

3. Select a specific pipeline to see the run results.

## Next steps

- Use [these Jupyter notebooks on GitHub](#) to explore machine learning pipelines further.
- Read the SDK reference help for the `azureml-pipelines-core` package and the `azureml-pipelines-steps` package.

Learn how to run notebooks by following the article, [Use Jupyter notebooks to explore this service](#).

# Enable logging in Azure Machine Learning service

3/4/2019 • 2 minutes to read

The Azure Machine Learning Python SDK allows you to enable logging using both the default Python logging package, as well as using SDK-specific functionality both for local logging and logging to your workspace in the portal. Logs provide developers with real-time information about the application state, and can help with diagnosing errors or warnings. In this article, you learn different ways of enabling logging in the following areas:

- Training models and compute targets
- Image creation
- Deployed models
- Python `logging` settings

Use the [guide](#) to install the SDK, and [get started](#) with the SDK to create a workspace in the Azure Portal.

## Training models and compute target logging

There are multiple ways to enable logging during the model training process, and the examples shown will illustrate common design patterns. You can easily log run-related data to your workspace in the cloud by using the `start_logging` function on the `Experiment` class.

```
from azureml.core import Experiment

exp = Experiment(workspace=ws, name='test_experiment')
run = exp.start_logging()
run.log("test-val", 10)
```

See the reference documentation for the [Run](#) class for additional logging functions.

To enable local logging of application state during training progress, use the `show_output` parameter. Enabling verbose logging allows you to see details from the training process as well as information about any remote resources or compute targets. Use the following code to enable logging upon experiment submission.

```
from azureml.core import Experiment

experiment = Experiment(ws, experiment_name)
run = experiment.submit(config=run_config_object, 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)
```

The SDK also supports using the default python logging package in certain scenarios for training. The following example enables a logging level of `INFO` in an `AutoMLConfig` object.

```
from azureml.train.automl import AutoMLConfig
import logging

automated_ml_config = AutoMLConfig(task = 'regression',
                                    verbosity=logging.INFO,
                                    X=your_training_features,
                                    y=your_training_labels,
                                    iterations=30,
                                    iteration_timeout_minutes=5,
                                    primary_metric="spearman_correlation")
```

You can also use the `show_output` parameter when creating a persistent compute target. Specify the parameter in the `wait_for_completion` function to enable logging during compute target creation.

```
from azureml.core.compute import ComputeTarget

compute_target = ComputeTarget.attach(workspace=ws, name="example", attach_configuration=config)
compute.wait_for_completion(show_output=True)
```

## Logging during image creation

Enabling logging during image creation will allow you to see any errors during the build process. Set the `show_output` param on the `wait_for_deployment()` function.

```
from azureml.core.webservice import Webservice

service = Webservice.deploy_from_image(deployment_config=your_config,
                                       image=image,
                                       name="example-image",
                                       workspace=ws)

service.wait_for_deployment(show_output=True)
```

## Logging for deployed models

To retrieve logs from a previously deployed web service, load the service and use the `get_logs()` function. The logs may contain detailed information about any errors that occurred during deployment.

```
from azureml.core.webservice import Webservice

# load existing web service
service = Webservice(name="service-name", workspace=ws)
logs = service.get_logs()
```

You can also log custom stack traces for your web service by enabling Application Insights, which allows you to monitor request/response times, failure rates, and exceptions. Call the `update()` function on an existing web service to enable Application Insights.

```
service.update(enable_app_insights=True)
```

See the [how-to](#) for more information on how to work with Application Insights in the Azure portal.

## Python native logging settings

Certain 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)
```

# Manage and request quotas for Azure resources

1/29/2019 • 7 minutes to read

As with other Azure services, there are limits on certain resources associated with the Azure Machine Learning service. These limits range from a cap on the number of workspaces you can create to limits on the actual underlying compute that gets used for training or inferencing your models. This article gives more details on the pre-configured limits on various Azure resources for your subscription and also contains handy links to request quota enhancements for each type of resource. These limits are put in place to prevent budget over-runs due to fraud, and to honor Azure capacity constraints.

Keep these quotas in mind as you design and scale up your Azure Machine Learning service resources for production workloads. For example, if your cluster doesn't reach the target number of nodes you specified, then you might have reached a Azure Machine Learning Compute cores limit for your subscription. If you want to raise the limit or quota above the Default Limit, open an online customer support request at no charge. The limits can't be raised above the Maximum Limit value shown in the following tables due to Azure Capacity constraints. If there is no Maximum Limit column, then the resource doesn't have adjustable limits.

## Special considerations

- A quota is a credit limit, not a capacity guarantee. If you have large-scale capacity needs, contact Azure support.
- Your quota is shared across all the services in your subscriptions including Azure Machine Learning service. The only exception is Azure Machine Learning compute which has a separate quota from the core compute quota. Be sure to calculate the quota usage across all services when evaluating your capacity needs.
- Default limits vary by offer Category Type, such as Free Trial, Pay-As-You-Go, and series, such as Dv2, F, G, and so on.

## Default resource quotas

Here is a breakdown of the quota limits by various resource types within your Azure subscription.

### IMPORTANT

Limits are subject to change. The latest can always be found at the service-level quota [document](#) for all of Azure.

### Virtual machines

There is a limit on the number of virtual machines you can provision on an Azure subscription across your services or in a standalone manner. This limit is at the region level both on the total cores and also on a per family basis.

It is important to emphasize that virtual machine cores have a regional total limit and a regional per size series (Dv2, F, etc.) limit that 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 not to exceed a total of 30 cores (for example, 10 A1 VMs and 20 D1 VMs).

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
VMs per <a href="#">subscription</a>	25,000 <sup>1</sup> per region.	25,000 per region.
VM total cores per <a href="#">subscription</a>	20 <sup>1</sup> per region.	Contact support.
VM per series, such as Dv2 and F, cores per <a href="#">subscription</a>	20 <sup>1</sup> per region.	Contact support.
<a href="#">Coadministrators</a> per subscription	Unlimited.	Unlimited.
<a href="#">Storage accounts</a> per region per subscription	200	200 <sup>2</sup>
<a href="#">Resource groups</a> per subscription	980	980
<a href="#">Availability sets</a> per subscription	2,000 per region.	2,000 per region.
Azure Resource Manager API request size	4,194,304 bytes.	4,194,304 bytes.
Tags per subscription <sup>3</sup>	Unlimited.	Unlimited.
Unique tag calculations per subscription <sup>3</sup>	10,000	10,000
<a href="#">Cloud services</a> per subscription	N/A <sup>4</sup>	N/A <sup>4</sup>
<a href="#">Affinity groups</a> per subscription	N/A <sup>4</sup>	N/A <sup>4</sup>
<a href="#">Subscription-level deployments</a> per location	800	800

<sup>1</sup>Default limits vary by offer category type, such as Free Trial and Pay-As-You-Go, and by series, such as Dv2, F, and G.

<sup>2</sup>Both Standard and Premium storage accounts are included. If you need more than 200 storage accounts, make a request through [Azure Support](#). The Azure Storage team reviews your business case and might approve up to 250 storage accounts.

<sup>3</sup>You can apply an unlimited number of tags per subscription. The number of tags per resource or resource group is limited to 15. 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.

<sup>4</sup>These features are no longer required with Azure resource groups and Resource Manager.

#### NOTE

Virtual machine cores have a regional total limit. They also have a limit for regional per-size series, such as Dv2 and F. These 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 can deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two not to exceed a total of 30 cores. An example of a combination is 10 A1 VMs and 20 D1 VMs.

For a more detailed and up-to-date list of quota limits, check the Azure-wide quota article [here](#).

## Azure Machine Learning Compute

For Azure Machine Learning Compute, there is a default quota limit on both the number of cores and number of unique compute resources allowed per region in a subscription. This quota is separate from the VM core quota above and the core limits are not shared currently between the two resource types.

Available resources:

- Dedicated cores per region have a default limit of 10 - 24. The number of dedicated cores per subscription can be increased. Contact Azure support to discuss increase options.
- Low-priority cores per region have a default limit of 10 - 24. The number of low-priority cores per subscription can be increased. Contact Azure support to discuss increase options.
- Clusters per region have a default limit of 100 and a maximum limit of 200. Contact Azure support if you want to request an increase beyond this limit.
- There are **other strict limits** which cannot be exceeded once hit.

RESOURCE	MAXIMUM LIMIT
Maximum workspaces per resource group	800
Maximum nodes in a single Azure Machine Learning Compute (AmlCompute) resource	100 nodes
Maximum GPU MPI processes per node	1-4
Maximum GPU workers per node	1-4
Maximum job lifetime	7 days <sup>1</sup>
Maximum parameter servers per node	1

<sup>1</sup> The maximum lifetime refers to the time that a run starts and when it finishes. Completed runs persist indefinitely; data for runs not completed within the maximum lifetime is not accessible.

## Container instances

There is also a limit on the number of container instances that you can spin up in a given time period (scoped hourly) or across your entire subscription.

RESOURCE	DEFAULT LIMIT
Container groups per <a href="#">subscription</a>	100 <sup>1</sup>
Number of containers per container group	60
Number of volumes per container group	20
Ports per IP	5
Container instance log size - running instance	4 MB
Container instance log size - stopped instance	16 KB or 1,000 lines
Container creates per hour	300 <sup>1</sup>

RESOURCE	DEFAULT LIMIT
Container creates per 5 minutes	100 <sup>1</sup>
Container deletes per hour	300 <sup>1</sup>
Container deletes per 5 minutes	100 <sup>1</sup>

<sup>1</sup>To request a limit increase, create an [Azure Support request](#).

For a more detailed and up-to-date list of quota limits, check the Azure-wide quota article [here](#).

## Storage

There is a limit on the number of storage accounts per region as well in a given subscription. The default limit is 200 and includes both Standard and Premium Storage accounts. If you require more than 200 storage accounts in a given region, make a request through [Azure Support](#). The Azure Storage team will review your business case and may approve up to 250 storage accounts for a given region.

## Find your quotas

Viewing your quota for various resources, such as Virtual Machines, Storage, Network, is easy through 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 are looking for.

**There is a caveat**, specifically for viewing the Azure Machine Learning Compute quota. As mentioned above, that quota is separate from the compute quota on your subscription.

3. On the left pane, select **Machine Learning service** and then select any workspace from the list shown
4. On the next blade, under the **Support + troubleshooting section** select **Usage + quotas** to view your current quota limits and usage.
5. Select a subscription to view the quota limits. Remember to filter to the region you are interested in.

## Request quota increases

If you want to raise the limit or quota above the default limit, [open an online customer support request](#) at no charge.

The limits can't be raised above the maximum limit value shown in the tables. If there is no maximum limit, then the resource doesn't have adjustable limits. [This](#) article covers the quota increase process in more detail.

When requesting a quota increase, you need to select the service you are requesting to raise the quota against, which could be services such as Machine Learning service quota, Container instances or Storage quota. In addition for Azure Machine Learning Compute, you can simply click on the **Request Quota** button while viewing the quota following the steps above.

### 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 [Free Trial subscription FAQ](#).

# Use the CLI extension for Azure Machine Learning service

2/19/2019 • 5 minutes to read

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 the Azure Machine Learning service from the command line. It allows you to create scripts that automate your machine learning workflows. For example, you can create scripts that perform the following actions:

- 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 such as:

- Creating compute resources
- Parameterized experiment submission
- Model registration
- Image creation
- Service deployment

## 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 service](#) today.
- The [Azure CLI](#).

## Install the extension

To install the Machine Learning CLI extension, use the following command:

```
az extension add -s https://azuremlsdktestpypi.blob.core.windows.net/wheels/sdk-release/Preview/E7501C02541B433786111FE8E140CAA1/azure_cli_ml-1.0.10-py2.py3-none-any.whl --pip-extra-index-urls https://azuremlsdktestpypi.azureedge.net/sdk-release/Preview/E7501C02541B433786111FE8E140CAA1
```

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
```

**TIP**

To update the extension you must **remove** it, and then **install** it. This installs the latest version.

## 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.

- Create an Azure Machine Learning service workspace:

```
az ml workspace create -n myworkspace -g myresourcegroup
```

- Set a default workspace:

```
az configure --defaults aml_workspace=myworkspace group=myresourcegroup
```

- Attach an AKS cluster

```
az ml computetarget attach aks -n myaks -i myaksresourceid -g myrg -w myworkspace
```

## Experiments

The following commands demonstrate how to use the CLI to work with experiments:

- Attach a project (run configuration) before submitting an experiment:

```
az ml project attach --experiment-name myhistory
```

- Start a run of your experiment. When using this command, specify the name of the runconfig file that contains the run configuration. The compute target uses the run configuration to create the training environment for the model. In this example, the run configuration is loaded from the `./aml_config/myrunconfig.runconfig` file.

```
az ml run submit -c myrunconfig train.py
```

For more information on the runconfig file, see the [RunConfig](#) section.

- View a list of submitted experiments:

```
az ml history list
```

# Model registration, image creation & 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 -m sklearn_regression_model.pkl
```

- Create an image that contains your machine learning model and dependencies:

```
az ml image create container -n myimage -r python -m mymodel:1 -f score.py -c myenv.yml
```

- Deploy an image to a compute target:

```
az ml service create aci -n myaciservice --image-id myimage:1
```

## Runconfig file

A run configuration is used to configure the training environment used to train your model. This configuration can be created in-memory using the SDK or it can be loaded from a runconfig file.

The runconfig file is a text document that describes the configuration for the training environment. For example, it lists the name of the training script and the file that contains the conda dependencies needed to train the model.

The Azure Machine Learning CLI creates two default `.runconfig` files named `docker.runconfig` and `local.runconfig` when you attach a project using the `az ml project attach` command.

If you have code that creates a run configuration using the `RunConfiguration` class, you can use the `save()` method to persist it to a `.runconfig` file.

The following is an example of the contents of a `.runconfig` file:

```
# The script to run.
script: train.py
# The arguments to the script file.
arguments: []
# The name of the compute target to use for this run.
target: local
# Framework to execute inside. Allowed values are "Python" , "PySpark", "CNTK", "TensorFlow", and "PyTorch".
framework: PySpark
# Communicator for the given framework. Allowed values are "None" , "ParameterServer", "OpenMpi", and "IntelMpi".
communicator: None
# Automatically prepare the run environment as part of the run itself.
autoPrepareEnvironment: true
# Maximum allowed duration for the run.
maxRunDurationSeconds:
# Number of nodes to use for running job.
nodeCount: 1
# Environment details.
environment:
# Environment variables set for the run.
environmentVariables:
    EXAMPLE_ENV_VAR: EXAMPLE_VALUE
# Python details
python:
# user_managed_dependencies=True indicates that the environment will be user managed. False indicates that
```

```
- - - .  
AzureML will manage the user environment.  
    userManagedDependencies: false  
# The python interpreter path  
    interpreterPath: python  
# Path to the conda dependencies file to use for this run. If a project  
# contains multiple programs with different sets of dependencies, it may be  
# convenient to manage those environments with separate files.  
    condaDependenciesFile: aml_config/conda_dependencies.yml  
# Docker details  
    docker:  
# Set True to perform this run inside a Docker container.  
    enabled: true  
# Base image used for Docker-based runs.  
    baseImage: mcr.microsoft.com/azureml/base:0.2.1  
# Set False if necessary to work around shared volume bugs.  
    sharedVolumes: true  
# Run with NVidia Docker extension to support GPUs.  
    gpuSupport: false  
# Extra arguments to the Docker run command.  
    arguments: []  
# Image registry that contains the base image.  
    baseImageRegistry:  
# DNS name or IP address of azure container registry(ACR)  
    address:  
# The username for ACR  
    username:  
# The password for ACR  
    password:  
# Spark details  
    spark:  
# List of spark repositories.  
    repositories:  
        - https://mmlspark.azureedge.net/maven  
    packages:  
        - group: com.microsoft.ml.spark  
          artifact: mmlspark_2.11  
          version: '0.12'  
    precachePackages: true  
# Databricks details  
    databricks:  
# List of maven libraries.  
    mavenLibraries: []  
# List of PyPi libraries  
    pypiLibraries: []  
# List of RCran libraries  
    rcranLibraries: []  
# List of JAR libraries  
    jarLibraries: []  
# List of Egg libraries  
    eggLibraries: []  
# History details.  
    history:  
# Enable history tracking -- this allows status, logs, metrics, and outputs  
# to be collected for a run.  
    outputCollection: true  
# whether to take snapshots for history.  
    snapshotProject: true  
# Spark configuration details.  
    spark:  
        configuration:  
            spark.app.name: Azure ML Experiment  
            spark.yarn.maxAppAttempts: 1  
# HDI details.  
    hdi:  
# Yarn deploy mode. Options are cluster and client.  
    yarnDeployMode: cluster  
# Tensorflow details.  
    tensorflow:  
# The number of worker tasks.
```

```
"# The number of worker tasks.
workerCount: 1
# The number of parameter server tasks.
parameterServerCount: 1
# Mpi details.
mpi:
# When using MPI, number of processes per node.
processCountPerNode: 1
# data reference configuration details
dataReferences: {}
# Project share datastore reference.
sourceDirectoryDataStore:
# AmlCompute details.
amlcompute:
# VM size of the Cluster to be created. Allowed values are Azure vm sizes. The list of vm sizes is available
in 'https://docs.microsoft.com/azure/cloud-services/cloud-services-sizes-specs
vmSize:
# VM priority of the Cluster to be created. Allowed values are "dedicated" , "lowpriority".
vmPriority:
# A bool that indicates if the cluster has to be retained after job completion.
retainCluster: false
# Name of the cluster to be created. If not specified, runId will be used as cluster name.
name:
# Maximum number of nodes in the AmlCompute cluster to be created. Minimum number of nodes will always be set
to 0.
clusterMaxNodeCount: 1
```

# Export or delete your Machine Learning service workspace data

1/29/2019 • 3 minutes to read

In Azure Machine Learning, you can export or delete your workspace data with the authenticated REST API. This article tells you how.

## NOTE

If you're interested in viewing or deleting personal data, please see the [Azure Data Subject Requests for the GDPR](#) article. If you're looking for general info 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 Services is available for export and deletion through the Azure portal, CLI, SDK, and authenticated REST APIs. Telemetry data can be accessed through the Azure Privacy portal.

In Azure Machine Learning Services, personal data consists of user information in run history documents and telemetry records of some user interactions with the service.

## Delete workspace data with the REST API

In order to delete data, the following API calls can be made with the HTTP DELETE verb. These are authorized by having an `Authorization: Bearer <arm-token>` header in the request, where `<arm-token>` is the AAD access token for the `https://management.core.windows.net/` endpoint.

To learn how to get this token and call Azure endpoints, see [Azure REST API documentation](#).

In the examples following, replace the text in {} with the instance names that determine the associated resource.

### Delete an entire workspace

Use this call to delete an entire workspace.

#### WARNING

All information will be deleted and the workspace will no longer be usable.

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}?api-version=2018-03-01-preview
```

### Delete models

Use this call to get a list of models and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models?api-version=2018-03-01-preview
```

Individual models can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models/{{id}}?api-version=2018-03-01-preview
```

## Delete assets

Use this call to get a list of assets and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets?api-version=2018-03-01-preview
```

Individual assets can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets/{{id}}?api-version=2018-03-01-preview
```

## Delete images

Use this call to get a list of images and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images?api-version=2018-03-01-preview
```

Individual images can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images/{{id}}?api-version=2018-03-01-preview
```

## Delete services

Use this call to get a list of services and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services?api-version=2018-03-01-preview
```

Individual services can be deleted with:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services/{{id}}?api-version=2018-03-01-preview
```

# Export service data with the REST API

In order to export data, the following API calls can be made with the HTTP GET verb. These are authorized by having an `Authorization: Bearer <arm-token>` header in the request, where `<arm-token>` is the AAD access token

for the endpoint `https://management.core.windows.net/`

To learn how to get this token and call Azure endpoints, see [Azure REST API documentation](#).

In the examples following, replace the text in {} with the instance names that determine the associated resource.

## Export Workspace information

Use this call to get a list of all workspaces:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces?api-version=2018-03-01-preview
```

Information about an individual workspace can be obtained by:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}?api-version=2018-03-01-preview
```

## Export Compute Information

All compute targets attached to a workspace can be obtained by:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroup/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/computes?api-version=2018-03-01-preview
```

Information about a single compute target can be obtained by:

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroup/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/computes/{computeName}?api-version=2018-03-01-preview
```

## Export run history data

Use this call to get a list of all experiments and their information:

```
https://[location].experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments
```

All the runs for a particular experiment can be obtained by:

```
https://[location].experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments/{experimentName}/runs
```

Run history items can be obtained by:

```
https://[location].experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments/{experimentName}/runs/{runId}
```

All run metrics for an experiment can be obtained by:

```
https://[location].experiments.azureml.net/history/v1.0/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.MachineLearningServices/workspaces/{workspaceName}/experiments/{experimentName}/metrics
```

A single run metric can be obtained by:

```
https://{{location}}.experiments.azureml.net/history/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/experiments/{{experimentName}}/metrics/{{metricId}}
```

## Export artifacts

Use this call to get a list of artifacts and their paths:

```
https://{{location}}.experiments.azureml.net/artifact/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/artifacts/origins/ExperimentRun/containers/{{runId}}
```

## Export notifications

Use this call to get a list of stored tasks:

```
https://{{location}}.experiments.azureml.net/notification/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/tasks
```

Notifications for a single task can be obtained by:

```
https://{{location}}.experiments.azureml.net/notification/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/tasks/{{taskId}}
```

## Export data stores

Use this call to get a list of data stores:

```
https://{{location}}.experiments.azureml.net/datastore/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/datastores
```

Individual data stores can be obtained by:

```
https://{{location}}.experiments.azureml.net/datastore/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/datastores/{{name}}
```

## Export models

Use this call to get a list of models and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models?api-version=2018-03-01-preview
```

Individual models can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/models/{{id}}?api-version=2018-03-01-preview
```

## Export assets

Use this call to get a list of assets and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets?api-version=2018-03-01-preview
```

Individual assets can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/assets/{{id}}?api-version=2018-03-01-preview
```

## Export images

Use this call to get a list of images and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images?api-version=2018-03-01-preview
```

Individual images can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/images/{{id}}?api-version=2018-03-01-preview
```

## Export services

Use this call to get a list of services and their IDs:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services?api-version=2018-03-01-preview
```

Individual services can be obtained by:

```
https://{{location}}.modelmanagement.azureml.net/api/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspace}}/services/{{id}}?api-version=2018-03-01-preview
```

## Export Pipeline Experiments

Individual experiments can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Experiments/{{experimentId}}
```

## Export Pipeline Graphs

Individual graphs can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Graphs/{{graphId}}
```

## Export Pipeline Modules

Modules can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Modules/{{id}}
```

## Export Pipeline Templates

Templates can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/Templates/{{templateId}}
```

## Export Pipeline Data Sources

Data Sources can be obtained by:

```
https://{{location}}.aether.ms/api/v1.0/subscriptions/{{subscriptionId}}/resourceGroups/{{resourceGroupName}}/providers/Microsoft.MachineLearningServices/workspaces/{{workspaceName}}/DataSources/{{id}}
```

# Azure Machine Learning service release notes

3/4/2019 • 12 minutes to read

In this article, learn about the Azure Machine Learning service releases. For a full description of each SDK, visit the reference docs for:

- The Azure Machine Learning's [main SDK for Python](#)
- The Azure Machine Learning [Data Prep SDK](#)

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 SDK now supports a [Chainer](#) estimator class to streamline the process of training and deploying a model using custom Chainer code.
  - Azure Machine Learning Pipelines added ability 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 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

## Azure Machine Learning Data Prep SDK v1.0.15

- **New features**
  - Data Prep now supports writing file streams from a dataflow. Also provides the ability to manipulate the file stream names to create new file names.
    - How-to guide: [Working With File Streams notebook](#)
- **Bug fixes and improvements**
  - Improved performance of t-Digest on large data sets.
  - Data Prep now supports reading data from a DataPath.
  - One hot encoding now works on boolean and numeric columns.

- Other miscellaneous bug fixes.

## 2019-02-11

### Azure Machine Learning SDK for Python v1.0.15

- **New features**

- Azure Machine Learning Pipelines added AzureBatchStep ([notebook](#)), HyperDriveStep ([notebook](#)) and time-based scheduling functionality ([notebook](#)).
- DataTransferStep updated to work with Azure SQL Server and Azure database for PostgreSQL ([notebook](#)).

- **Changes**

- Deprecated `PublishedPipeline.get_published_pipeline` in favor of `PublishedPipeline.get`.
- Deprecated `Schedule.get_schedule` in favor of `Schedule.get`.

### Azure Machine Learning Data Prep SDK v1.0.12

- **New features**

- Data Prep now supports reading from an Azure SQL database using Datastore.

- **Changes**

- Significantly improved the memory performance of certain operations on large data.
- `read_pandas_dataframe()` now requires `temp_folder` to be specified.
- The `name` property on `ColumnProfile` has been deprecated - use `column_name` instead.

## 2019-01-28

### Azure Machine Learning SDK for Python v1.0.10

- **Changes:**

- Azure ML SDK no longer has azure-cli packages as dependency. Specifically, azure-cli-core and azure-cli-profile dependencies have been removed from azureml-core. These are the user impacting changes:
  - If you are performing "az login" and then using azureml-sdk, the SDK will do the browser or device code login one more time. It won't use any credentials state created by "az login".
  - For Azure CLI authentication, such as using "az login", use `azureml.core.authentication.AzureCliAuthentication` class. For Azure CLI authentication, do `pip install azure-cli` in the Python environment where you have installed azureml-sdk.
  - If you are doing "az login" using a service principal for automation, we recommend using `azureml.core.authentication.ServicePrincipalAuthentication` class, as azureml-sdk won't use credentials state created by azure CLI.

- **Bug fixes:** This release mostly contains minor bug fixes

### Azure Machine Learning Data Prep SDK v1.0.8

- **Bug fixes**

- Significantly improved the performance of getting data profiles.
- Fixed minor bugs related to error reporting.

### Azure portal: new features

- New drag and drop charting experience for reports. Users can drag a column or attribute from the well to the chart area where the system will automatically select an appropriate chart type for the user based on the type of data. Users can change the chart type to other applicable types or add additional attributes.

Supported Chart Types:

- Line Chart
- Histogram
- Stacked Bar Chart
- Box Plot
- Scatter Plot
- Bubble Plot
- The portal now dynamically generates reports for experiments. When a user submits a run to an experiment, a report will automatically be generated with logged metrics and graphs to allow comparison across different runs.

## 2019-01-14

### Azure Machine Learning SDK for Python v1.0.8

- **Bug fixes:** This release mostly contains minor bug fixes

### Azure Machine Learning Data Prep SDK v1.0.7

- **New features**

- Datastore improvements (documented in [Datastore how-to-guide](#))
- Added ability to read from and write to Azure File Share and ADLS Datastores in scale-up.
- When using Datastores, Data Prep now supports using service principal authentication instead of interactive authentication.
- Added support for wasb and wasbs urls.

## 2019-01-09

### Azure Machine Learning Data Prep SDK v1.0.6

- **Bug fixes**

- Fixed bug with reading from public readable Azure Blob containers on Spark

## 2018-12-20

### Azure Machine Learning SDK for Python v1.0.6

- **Bug fixes:** This release mostly contains minor bug fixes

### Azure Machine Learning Data Prep SDK v1.0.4

- **New features**

- `to_bool` function now allows mismatched values to be converted to Error values. This is the new default mismatch behavior for `to_bool` and `set_column_types`, whereas the previous default behavior was to convert mismatched values to False.
- When calling `to_pandas_dataframe`, there is a new option to interpret null/missing values in numeric columns as NaN.
- Added ability to check the return type of some expressions to ensure type consistency and fail early.
- You can now call `parse_json` to parse values in a column as JSON objects and expand them into multiple columns.

- **Bug fixes**

- Fixed a bug that crashed `set_column_types` in Python 3.5.2.
- Fixed a bug that crashed when connecting to Datastore using an AML image.

- **Updates**

- [Example Notebooks](#) for getting started tutorials, case studies, and how-to guides.

## 2018-12-04: General Availability

Azure Machine Learning service is now generally available.

### Azure Machine Learning Compute

With this release, we are announcing a new managed compute experience through the [Azure Machine Learning Compute](#). This compute target replaces Azure Batch AI compute for Azure Machine Learning.

This compute target:

- Is used for model training and batch inferencing
- Is single- to multi-node compute
- Does the cluster management and job scheduling for the user
- Autoscales by default
- Support for both CPU and GPU resources
- Enables use of low-priority VMs for reduced cost

Azure Machine Learning Compute can be created in Python, using Azure portal, or the CLI. It must be created in the region of your workspace, and cannot be attached to any other workspace. This compute target uses a Docker container for your run, and packages your dependencies to replicate the same environment across all your nodes.

#### WARNING

We recommend creating a new workspace to use Azure Machine Learning Compute. There is a remote chance that users trying to create Azure Machine Learning Compute from an existing workspace might see an error. Existing compute in your workspace should continue to work unaffected.

### Azure Machine Learning SDK for Python v1.0.2

- **Breaking changes**

- With this release, we are removing support for creating a VM from Azure Machine Learning. You can still attach an existing cloud VM or a remote on-premises server.
- We are also removing support for BatchAI, all of which should be supported through Azure Machine Learning Compute now.

- **New**

- For machine learning pipelines:
  - [EstimatorStep](#)
  - [HyperDriveStep](#)
  - [MpStep](#)

- **Updated**

- For machine learning pipelines:
  - [DatabricksStep](#) now accepts runconfig
  - [DataTransferStep](#) now copies to and from a SQL datasource
  - Schedule functionality in SDK to create and update schedules for running published pipelines

### Azure Machine Learning Data Prep SDK v0.5.2

- **Breaking changes**

- `SummaryFunction.N` was renamed to `SummaryFunction.Count`.

- **Bug Fixes**

- Use latest AML Run Token when reading from and writing to datastores on remote runs. Previously, if the AML Run Token is updated in Python, the Data Prep runtime will not be updated with the updated AML Run Token.
- Additional clearer error messages
- `to_spark_dataframe()` will no longer crash when Spark uses `kryo` serialization
- Value Count Inspector can now show more than 1000 unique values
- Random Split no longer fails if the original Dataflow doesn't have a name

- **More information**

- [Azure Machine Learning Data Prep SDK](#)

## Docs and notebooks

- ML Pipelines
  - New and updated notebooks for getting started with pipelines, batch scoping, and style transfer examples: <https://aka.ms/aml-pipeline-notebooks>
  - Learn how to [create your first pipeline](#)
  - Learn how to [run batch predictions using pipelines](#)
- Azure Machine Learning compute target
  - [Sample notebooks](#) are now updated to use the new managed compute.
  - [Learn about this compute](#)

## Azure portal: new features

- Create and manage [Azure Machine Learning Compute](#) types in the portal.
- Monitor quota usage and [request quota](#) for Azure Machine Learning Compute.
- View Azure Machine Learning Compute cluster status in real time.
- Virtual network support was added for Azure Machine Learning Compute and Azure Kubernetes Service creation.
- Rerun your published pipelines with existing parameters.
- New [automated machine learning charts](#) for classification models (lift, gains, calibration, feature importance chart with model explainability) and regression models (residuals and feature importance chart with model explainability).
- Pipelines can be viewed in Azure portal

2018-11-20

## Azure Machine Learning SDK for Python v0.1.80

- **Breaking changes**

- `azuremltrain.widgets` namespace has moved to `azureml.widgets`.
- `azureml.core.compute.AmlCompute` deprecates the following classes - `azureml.core.compute.BatchAICompute` and `azureml.core.compute.DSVMCompute`. The latter class will be removed in subsequent releases. The AmlCompute class has an easier definition now, and simply needs a `vm_size` and the `max_nodes`, and will automatically scale your cluster from 0 to the `max_nodes` when a job is submitted. Our [sample notebooks](#) have been updated with this information and should give you usage examples. We hope you like this simplification and lots of more exciting features to come in a later release!

## Azure Machine Learning Data Prep SDK v0.5.1

Learn more about the Data Prep SDK by reading [reference docs](#).

- **New Features**

- Created a new DataPrep CLI to execute DataPrep packages and view the data profile for a dataset or dataflow
- Redesigned SetColumnType API to improve usability
- Renamed smart\_read\_file to auto\_read\_file
- Now includes skew and kurtosis in the Data Profile
- Can sample with stratified sampling
- Can read from zip files that contain CSV files
- Can split datasets row-wise with Random Split (for example, into test-train sets)
- Can get all the column data types from a dataflow or a data profile by calling `.dtypes`
- Can get the row count from a dataflow or a data profile by calling `.row_count`

- **Bug Fixes**

- Fixed long to double conversion
- Fixed assert after any add column
- Fixed an issue with FuzzyGrouping, where it would not detect groups in some cases
- Fixed sort function to respect multi-column sort order
- Fixed and/or expressions to be similar to how `pandas` handles them
- Fixed reading from dbfs path
- Made error messages more understandable
- Now no longer fails when reading on remote compute target using AML token
- Now no longer fails on Linux DSVM
- Now no longer crashes when non-string values are in string predicates
- Now handles assertion errors when Dataflow should fail correctly
- Now supports dbutils mounted storage locations on Azure Databricks

2018-11-05

#### Azure portal

The Azure portal for the Azure Machine Learning service has the following updates:

- A new **Pipelines** tab for published pipelines.
- Added support for attaching an existing HDInsight cluster as a compute target.

#### Azure Machine Learning SDK for Python v0.1.74

- **Breaking changes**

- *Workspace.compute\_targets, datastores, experiments, images, models, and \*webservices* are properties instead of methods. For example, replace `Workspace.compute_targets()` with `Workspace.compute_targets`.
- `Run.get_context` deprecates `Run.get_submitted_run`. The latter method will be removed in subsequent releases.
- `PipelineData` class now expects a datastore object as a parameter rather than `datastore_name`. Similarly, `Pipeline` accepts `default_datastore` rather than `default_datastore_name`.

- **New features**

- The Azure Machine Learning Pipelines [sample notebook](#) now uses MPI steps.
- The RunDetails widget for Jupyter notebooks is updated to show a visualization of the pipeline.

#### Azure Machine Learning Data Prep SDK v0.4.0

- **New features**

- Type Count added to Data Profile
- Value Count and Histogram is now available
- More percentiles in Data Profile
- The Median is available in Summarize
- Python 3.7 is now supported
- When you save a dataflow that contains datastores to a DataPrep package, the datastore information will be persisted as part of the DataPrep package
- Writing to datastore is now supported

- **Bug fixed**

- 64-bit unsigned integer overflows are now handled properly on Linux
- Fixed incorrect text label for plain text files in smart\_read
- String column type now shows up in metrics view
- Type count now is fixed to show ValueKinds mapped to single FieldType instead of individual ones
- Write\_to\_csv no longer fails when path is provided as a string
- When using Replace, leaving "find" blank will no longer fail

2018-10-12

**Azure Machine Learning SDK for Python v0.1.68**

- **New features**

- Multi-tenant support when creating new workspace.

- **Bugs fixed**

- You no longer need to pin the pynacl library version when deploying web service.

**Azure Machine Learning Data Prep SDK v0.3.0**

- **New features**

- Added method transform\_partition\_with\_file(script\_path), which allows users to pass in the path of a Python file to execute

2018-10-01

**Azure Machine Learning SDK for Python v0.1.65**

Version 0.1.65 includes new features, more documentation, bug fixes, and more [sample notebooks](#).

See [the list of known issues](#) to learn about known bugs and workarounds.

- **Breaking changes**

- Workspace.experiments, Workspace.models, Workspace.compute\_targets, Workspace.images, Workspace.web\_services return dictionary, previously returned list. See [azureml.core.Workspace](#) API documentation.
- Automated Machine Learning removed normalized mean square error from the primary metrics.

- **HyperDrive**

- Various HyperDrive bug fixes for Bayesian, Performance improvements for get Metrics calls.
- Tensorflow 1.10 upgrade from 1.9
- Docker image optimization for cold start.

- Jobs now report correct status even if they exit with error code other than 0.
- RunConfig attribute validation in SDK.
- HyperDrive run object supports cancel similar to a regular run: no need to pass any parameters.
- Widget improvements for maintaining state of drop-down values for distributed runs and HyperDrive runs.
- TensorBoard and other log files support fixed for Parameter server.
- Intel(R) MPI support on service side.
- Bugfix to parameter tuning for distributed run fix during validation in BatchAI.
- Context Manager now identifies the primary instance.

- **Azure portal experience**

- `log_table()` and `log_row()` are supported in Run details.
- Automatically create graphs for tables and rows with 1, 2 or 3 numerical columns and an optional categorical column.

- **Automated Machine Learning**

- Improved error handling and documentation
- Fixed run property retrieval performance issues.
- Fixed continue run issue.
- Fixed ensembling iteration issues.
- Fixed training hanging bug on MAC OS.
- Downsampling macro average PR/ROC curve in custom validation scenario.
- Removed extra index logic.
- Removed filter from `get_output` API.

- **Pipelines**

- Added a method `Pipeline.publish()` to publish a pipeline directly, without requiring an execution run first.
- Added a method `PipelineRun.get_pipeline_runs()` to fetch the pipeline runs that were generated from a published pipeline.

- **Project Brainwave**

- Updated support for new AI models available on FPGAs.

## Azure Machine Learning Data Prep SDK v0.2.0

[Version 0.2.0](#) includes following features and bug fixes:

- **New features**

- Support for one-hot encoding
- Support for quantile transform

- **Bug fixed:**

- Works with any Tornado version, no need to downgrade your Tornado version
- Value counts for all values, not just the top three

## 2018-09 (Public preview refresh)

A new, refreshed release of Azure Machine Learning: Read more about this release:

<https://azure.microsoft.com/blog/what-s-new-in-azure-machine-learning-service/>

## Next steps

Read the overview for [Azure Machine Learning service](#).

# Known issues and troubleshooting Azure Machine Learning service

2/25/2019 • 2 minutes to read

This article helps you find and correct errors or failures encountered when using the Azure Machine Learning service.

## SDK installation issues

### Error message: Cannot uninstall 'PyYAML'

Azure Machine Learning SDK for Python: PyYAML is a distutils installed project. Therefore, we cannot accurately determine which files belong to it if there is a partial uninstall. To continue installing the SDK while ignoring this error, use:

```
pip install --upgrade azureml-sdk[notebooks,automl] --ignore-installed PyYAML
```

## Trouble creating Azure Machine Learning Compute

There is a rare 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 Azure Machine Learning Compute 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.

## Image building failure

Image building failure when deploying web service. Workaround is to add "pynacl==1.2.1" as a pip dependency to Conda file for image configuration.

## Deployment failure

If you observe `['DaskOnBatch:context_managers.DaskOnBatch', 'setup.py']' died with <Signals.SIGKILL: 9>`, change the SKU for VMs used in your deployment to one that has more memory.

## FPGAs

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>

## Databricks

Databricks and Azure Machine Learning issues.

### Failure when installing packages

Azure Machine Learning SDK installation failures on Databricks when more packages are installed. Some packages, such as `psutil`, can cause conflicts. To avoid installation errors, install packages by freezing lib version. This issue is related to Databricks and not the Azure Machine Learning service SDK - you may face it with other libs 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 libs. This approach is not an officially supported approach. You can refer to [this doc](#).

### Cancel an automated ML run

When using automated machine learning capabilities on Databricks, if you want to cancel a run and start a new experiment run, restart your Azure Databricks cluster.

### >10 iterations for automated ML

In automated ml settings, if you have more than 10 iterations, set `show_output` to `False` when you submit the run.

## Azure portal

If you go directly to view your workspace from a share link from the SDK or the portal, you will not be able to view the normal Overview page with subscription information in the extension. You will also not be able to switch into another workspace. If you need to view another workspace, the workaround is to go directly to the [Azure portal](#) and search for the workspace name.

## Diagnostic logs

Sometimes it can be helpful if you can provide diagnostic information when asking for help. Here is where the log files live:

## Resource quotas

Learn about the [resource quotas](#) you might encounter when working with Azure Machine Learning.

## 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.

## Get more support

You can submit requests for support and get help from technical support, forums, and more. [Learn more...](#)

# Get support and training for Azure Machine Learning service

1/29/2019 • 2 minutes to read

This article provides information on how to learn more about Azure Machine Learning service as well as get support for your issues and questions.

## Learn more about Azure Machine Learning

See our learning resources:

- [Tutorials, release notes, and how-to articles](#)
- [Architecture overview](#)
- [Videos](#)

## Submit doc feedback

You can **submit requests** for additional learning materials using the **Content feedback** button at the end of each article.

## Get service support

Check out these support resources:

- **Technical support for Azure customers:** [Submit and manage support requests](#) through the Azure portal.
- **User forum:** Ask questions, answer questions, and connect with other users in the [Azure Machine Learning service support forum on MSDN](#).
- **Stack Overflow:** Visit the Azure Machine Learning community on [StackOverflow](#) tagged with "Azure-Machine-Learning".
- **Share product suggestions** and feature requests in our [Azure Machine Learning Feedback Channel](#). Select the **Product feedback** button at the end of each article to share yours.

# What are the machine learning products at Microsoft?

1/31/2019 • 5 minutes to read

Microsoft provides a variety of product options to build, deploy, and manage your machine learning models. Compare these products and choose what you need to develop your machine learning solutions most effectively.

## Cloud-based options

The following options are available for machine learning in the Azure cloud.

CLOUD OPTIONS	WHAT IT IS	WHAT YOU CAN DO WITH IT
Azure Machine Learning service	Managed cloud service for ML	Train, deploy, and manage models in Azure using Python and CLI
Azure Machine Learning Studio	Drag-and-drop visual interface for ML	Build, experiment, and deploy models using preconfigured algorithms (Python and R)
Azure Databricks	Spark-based analytics platform	Build and deploy models and data workflows
Azure Cognitive Services	Azure services with pre-built AI and ML models	Easily add intelligent features to your apps
Azure Data Science Virtual Machine	Virtual machine with pre-installed data science tools	Develop ML solutions in a pre-configured environment

## On-premises options

The following options are available for machine learning on-premises. On-premises servers can also run in a virtual machine in the cloud.

ON-PREMISES OPTIONS	WHAT IT IS	WHAT YOU CAN DO WITH IT
SQL Server Machine Learning Services	Analytics engine embedded in SQL	Build and deploy models inside SQL Server
Microsoft Machine Learning Server	Standalone enterprise server for predictive analysis	Build and deploy models with R and Python

## Development tools

The following development tools are available for machine learning.

DEVELOPMENT TOOLS	WHAT IT IS	WHAT YOU CAN DO WITH IT
ML.NET	Open-source, cross-platform ML SDK	Develop ML solutions for .NET applications

DEVELOPMENT TOOLS	WHAT IT IS	WHAT YOU CAN DO WITH IT
Windows ML	Windows 10 ML platform	Evaluate trained models on a Windows 10 device

## Azure Machine Learning service

[Azure Machine Learning service](#) is a fully managed cloud service used to train, deploy, and manage ML models at scale. It fully supports open-source technologies, so you can use tens of thousands of open-source Python packages such as TensorFlow, PyTorch, and scikit-learn. Rich tools are also available, such as [Azure notebooks](#), [Jupyter notebooks](#), or the [Azure Machine Learning for Visual Studio Code](#) extension to make it easy to explore and transform data, and then train and deploy models. Azure Machine Learning service includes features that automate model generation and tuning with ease, efficiency, and accuracy.

Use Azure Machine Learning service to train, deploy, and manage ML models using Python and CLI at cloud scale.

Try the [free or paid version of Azure Machine Learning service](#) today.

## Azure Machine Learning Studio

[Azure Machine Learning Studio](#) gives you an interactive, visual workspace that you can use to easily and quickly build, test, and deploy models using pre-built machine learning algorithms. Machine Learning Studio publishes models as web services that can easily be consumed by custom apps or BI tools such as Excel. No programming is required - you construct your machine learning model by connecting datasets and analysis modules on an interactive canvas, and then deploy it with a couple clicks.

Use Machine Learning Studio when you want to develop and deploy models with no code required.

Try [Azure Machine Learning Studio](#), available in paid or free options.

## Azure Databricks

[Azure Databricks](#) is an Apache Spark-based analytics platform optimized for the Microsoft Azure cloud services platform. Databricks is integrated with Azure to provide one-click setup, streamlined workflows, and an interactive workspace that enables collaboration between data scientists, data engineers, and business analysts. Use Python, R, Scala, and SQL code in web-based notebooks to query, visualize, and model data.

Use Databricks when you want to collaborate on building machine learning solutions on Apache Spark.

## Azure Cognitive Services

[Azure Cognitive Services](#) is a set of APIs that enable you to build apps that use natural methods of communication. These APIs allow your apps to see, hear, speak, understand, and interpret user needs with just a few lines of code. Easily add intelligent features to your apps, such as:

- Emotion and sentiment detection
- Vision and speech recognition
- Language understanding (LUIS)
- Knowledge and search

Use Cognitive Services to develop apps across devices and platforms. The APIs keep improving, and are easy to set up.

## Azure Data Science Virtual Machine

The [Azure Data Science Virtual Machine](#) is a customized virtual machine environment on the Microsoft Azure cloud built specifically for doing data science. It has many popular data science and other tools pre-installed and pre-configured to jump-start building intelligent applications for advanced analytics.

The Data Science Virtual Machine is supported as a target for Azure Machine Learning service. It is available in versions for both Windows and Linux Ubuntu (Azure Machine Learning service is not supported on Linux CentOS). For specific version information and a list of what's included, see [Introduction to the Azure Data Science Virtual Machine](#).

Use the Data Science VM when you need to run or host your jobs on a single node. Or if you need to remotely scale up your processing on a single machine.

## SQL Server Machine Learning Services

[SQL Server Microsoft Machine Learning Service](#) adds statistical analysis, data visualization, and predictive analytics in R and Python for relational data in SQL Server databases. R and Python libraries from Microsoft include advanced modeling and ML algorithms, which can run in parallel and at scale, in SQL Server.

Use SQL Server Machine Learning Services when you need built-in AI and predictive analytics on relational data in SQL Server.

## Microsoft Machine Learning Server

[Microsoft Machine Learning Server](#) is an enterprise server for hosting and managing parallel and distributed workloads of R and Python processes. Microsoft Machine Learning Server runs on Linux, Windows, Hadoop, and Apache Spark, and it is also available on [HDInsight](#). It provides an execution engine for solutions built using [RevoScaleR](#), [revoscalepy](#), and [MicrosoftML packages](#), and extends open-source R and Python with support for high-performance analytics, statistical analysis, machine learning, and massively large datasets. This functionality is provided through proprietary packages that install with the server. For development, you can use IDEs such as [R Tools for Visual Studio](#) and [Python Tools for Visual Studio](#).

Use Microsoft Machine Learning Server when you need to build and operationalize models built with R and Python on a server, or distribute R and Python training at scale on a Hadoop or Spark cluster.

## ML.NET

[ML.NET](#) is a free, open-source, and cross-platform machine learning framework that enables you to build custom machine learning solutions and integrate them into your .NET applications.

Use ML.NET when you want to integrate machine learning solutions into your .NET applications.

## Windows ML

[Windows ML](#) allows you to use trained machine learning models in your applications, evaluating trained models locally on Windows 10 devices.

Use Windows ML when you want to use trained machine learning models within your Windows applications.

## Next steps

- To learn about all the Artificial Intelligence (AI) development products available from Microsoft, see [Microsoft AI platform](#)
- For training in how to develop AI solutions, see [Microsoft AI School](#)