MLOps: Model management, deployment, and monitoring with Azure Machine Learning

Article • 04/04/2023

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

In this article, learn how to apply Machine Learning Operations (MLOps) practices in Azure Machine Learning for the purpose of managing the lifecycle of your models. Applying MLOps practices can improve the quality and consistency of your machine learning solutions.

What is MLOps?

MLOps is based on DevOps principles and practices that increase the efficiency of workflows. Examples include continuous integration, delivery, and deployment. MLOps applies these principles to the machine learning process, with the goal of:

- Faster experimentation and development of models.
- Faster deployment of models into production.
- Quality assurance and end-to-end lineage tracking.

MLOps in Machine Learning

Machine Learning provides the following MLOps capabilities:

- **Create reproducible machine learning pipelines.** Use machine learning pipelines to define repeatable and reusable steps for your data preparation, training, and scoring processes.
- **Create reusable software environments.** Use these environments for training and deploying models.
- **Register, package, and deploy models from anywhere.** You can also track associated metadata required to use the model.
- **Capture the governance data for the end-to-end machine learning lifecycle.** The logged lineage information can include who is publishing models and why changes were made. It can also include when models were deployed or used in production.

- **Notify and alert on events in the machine learning lifecycle.** Event examples include experiment completion, model registration, model deployment, and data drift detection.
- **Monitor machine learning applications for operational and machine learning-related issues.** Compare model inputs between training and inference. Explore model-specific metrics. Provide monitoring and alerts on your machine learning infrastructure.
- **Automate the end-to-end machine learning lifecycle with Machine Learning and Azure Pipelines.** By using pipelines, you can frequently update models. You can also test new models. You can continually roll out new machine learning models alongside your other applications and services.

For more information on MLOps, see Machine learning DevOps.

Create reproducible machine learning pipelines

Use machine learning pipelines from Machine Learning to stitch together all the steps in your model training process.

A machine learning pipeline can contain steps from data preparation to feature extraction to hyperparameter tuning to model evaluation. For more information, see Machine learning pipelines.

If you use the designer to create your machine learning pipelines, you can at any time select the **...** icon in the upper-right corner of the designer page. Then select **Clone**. When you clone your pipeline, you iterate your pipeline design without losing your old versions.

Create reusable software environments

By using Machine Learning environments, you can track and reproduce your projects' software dependencies as they evolve. You can use environments to ensure that builds are reproducible without manual software configurations.

Environments describe the pip and conda dependencies for your projects. You can use them for training and deployment of models. For more information, see What are Machine Learning environments?.

Register, package, and deploy models from anywhere

The following sections discuss how to register, package, and deploy models.

Register and track machine learning models

With model registration, you can store and version your models in the Azure cloud, in your workspace. The model registry makes it easy to organize and keep track of your trained models.

> **Tip**
>
> A registered model is a logical container for one or more files that make up your model. For example, if you have a model that's stored in multiple files, you can register them as a single model in your Machine Learning workspace. After registration, you can then download or deploy the registered model and receive all the files that were registered.

Registered models are identified by name and version. Each time you register a model with the same name as an existing one, the registry increments the version. More metadata tags can be provided during registration. These tags are then used when you search for a model. Machine Learning supports any model that can be loaded by using Python 3.5.2 or higher.

> **Tip**
>
> You can also register models trained outside Machine Learning.

> **Important**
>
> - When you use the **Filter by** `Tags` option on the **Models** page of Azure Machine Learning Studio, instead of using `TagName : TagValue`, use `TagName=TagValue` without spaces.
> - You can't delete a registered model that's being used in an active deployment.

For more information, Work with models in Azure Machine Learning.

Package and debug models

Before you deploy a model into production, it's packaged into a Docker image. In most cases, image creation happens automatically in the background during deployment. You

can manually specify the image.

If you run into problems with the deployment, you can deploy on your local development environment for troubleshooting and debugging.

For more information, see How to troubleshoot online endpoints.

Convert and optimize models

Converting your model to Open Neural Network Exchange (ONNX) might improve performance. On average, converting to ONNX can double performance.

For more information on ONNX with Machine Learning, see Create and accelerate machine learning models.

Use models

Trained machine learning models are deployed as endpoints in the cloud or locally. Deployments use CPU, GPU for inferencing.

When deploying a model as an endpoint, you provide the following items:

- The models that are used to score data submitted to the service or device.
- An entry script. This script accepts requests, uses the models to score the data, and returns a response.
- A Machine Learning environment that describes the pip and conda dependencies required by the models and entry script.
- Any other assets such as text and data that are required by the models and entry script.

You also provide the configuration of the target deployment platform. For example, the VM family type, available memory, and number of cores. When the image is created, components required by Azure Machine Learning are also added. For example, assets needed to run the web service.

Batch scoring

Batch scoring is supported through batch endpoints. For more information, see endpoints.

Online endpoints

You can use your models with an online endpoint. Online endpoints can use the following compute targets:

- Managed online endpoints
- Azure Kubernetes Service
- Local development environment

To deploy the model to an endpoint, you must provide the following items:

- The model or ensemble of models.
- Dependencies required to use the model. Examples are a script that accepts requests and invokes the model and conda dependencies.
- Deployment configuration that describes how and where to deploy the model.

For more information, see Deploy online endpoints.

Controlled rollout

When deploying to an online endpoint, you can use controlled rollout to enable the following scenarios:

- Create multiple versions of an endpoint for a deployment
- Perform A/B testing by routing traffic to different deployments within the endpoint.
- Switch between endpoint deployments by updating the traffic percentage in endpoint configuration.

For more information, see Controlled rollout of machine learning models.

Analytics

Microsoft Power BI supports using machine learning models for data analytics. For more information, see Machine Learning integration in Power BI (preview).

Capture the governance data required for MLOps

Machine Learning gives you the capability to track the end-to-end audit trail of all your machine learning assets by using metadata. For example:

- Machine Learning datasets help you track, profile, and version data.

- Interpretability allows you to explain your models, meet regulatory compliance, and understand how models arrive at a result for specific input.
- Machine Learning Job history stores a snapshot of the code, data, and computes used to train a model.
- The Machine Learning Model Registry captures all the metadata associated with your model. For example, metadata includes which experiment trained it, where it's being deployed, and if its deployments are healthy.
- Integration with Azure allows you to act on events in the machine learning lifecycle. Examples are model registration, deployment, data drift, and training (job) events.

> **Tip**
>
> While some information on models and datasets is automatically captured, you can add more information by using *tags*. When you look for registered models and datasets in your workspace, you can use tags as a filter.

Notify, automate, and alert on events in the machine learning lifecycle

Machine Learning publishes key events to Azure Event Grid, which can be used to notify and automate on events in the machine learning lifecycle. For more information, see Use Event Grid.

Automate the machine learning lifecycle

You can use GitHub and Azure Pipelines to create a continuous integration process that trains a model. In a typical scenario, when a data scientist checks a change into the Git repo for a project, Azure Pipelines starts a training job. The results of the job can then be inspected to see the performance characteristics of the trained model. You can also create a pipeline that deploys the model as a web service.

The Machine Learning extension makes it easier to work with Azure Pipelines. It provides the following enhancements to Azure Pipelines:

- Enables workspace selection when you define a service connection.
- Enables release pipelines to be triggered by trained models created in a training pipeline.

For more information on using Azure Pipelines with Machine Learning, see:

- Continuous integration and deployment of machine learning models with Azure Pipelines
- Machine Learning MLOps repository

Next steps

Learn more by reading and exploring the following resources:

- Set up MLOps with Azure DevOps
- Learning path: End-to-end MLOps with Azure Machine Learning
- How to deploy a model to an online endpoint with Machine Learning
- Tutorial: Train and deploy a model
- CI/CD of machine learning models with Azure Pipelines
- Machine learning at scale
- Azure AI reference architectures and best practices repo

Machine Learning registries for MLOps

Article • 05/23/2023

In this article, you'll learn how to scale MLOps across development, testing and production environments. Your environments can vary from few to many based on the complexity of your IT environment and is influenced by factors such as:

- Security and compliance policies - do production environments need to be isolated from development environments in terms of access controls, network architecture, data exposure, etc.?
- Subscriptions - Are your development environments in one subscription and production environments in a different subscription? Often separate subscriptions are used to account for billing, budgeting, and cost management purposes.
- Regions - Do you need to deploy to different Azure regions to support latency and redundancy requirements?

In such scenarios, you may be using different Azure Machine Learning workspaces for development, testing and production. This configuration presents the following challenges for model training and deployment:

- You need to train a model in a development workspace but deploy it an endpoint in a production workspace, possibly in a different Azure subscription or region. In this case, you must be able to trace back the training job. For example, to analyze the metrics, logs, code, environment, and data used to train the model if you encounter accuracy or performance issues with the production deployment.
- You need to develop a training pipeline with test data or anonymized data in the development workspace but retrain the model with production data in the production workspace. In this case, you may need to compare training metrics on sample vs production data to ensure the training optimizations are performing well with actual data.

Cross-workspace MLOps with registries

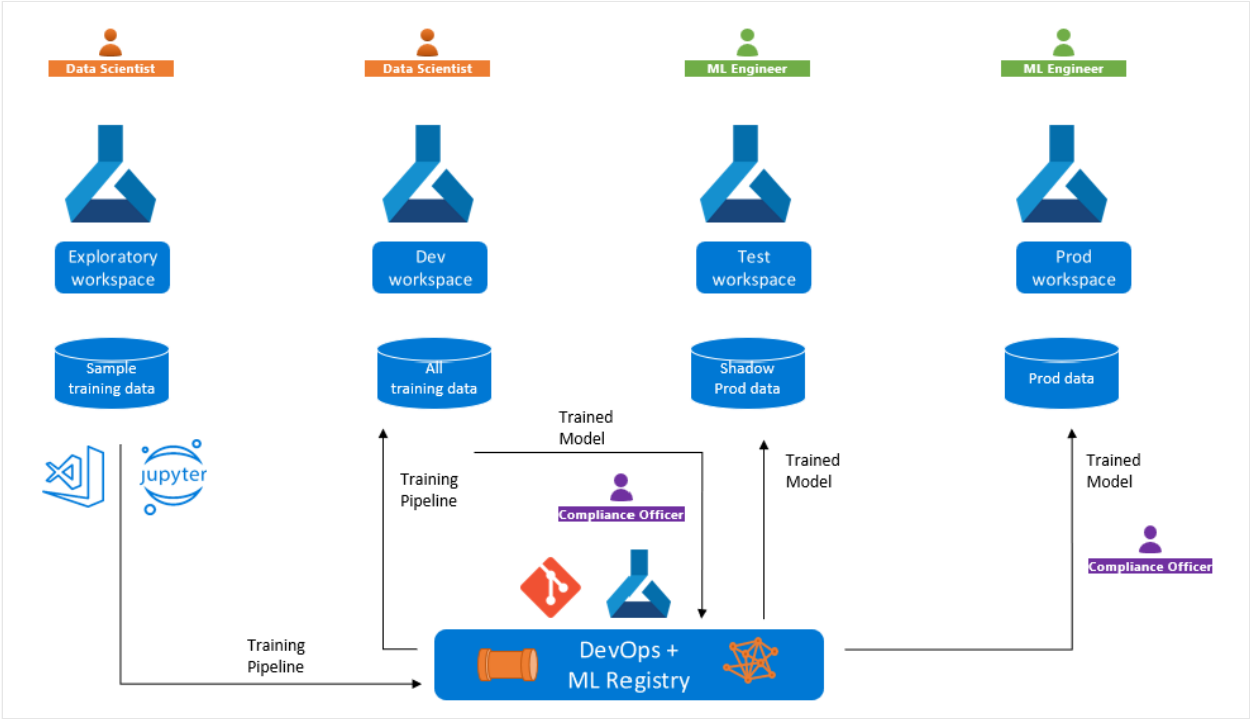
Registries, much like a Git repository, decouples ML assets from workspaces and hosts them in a central location, making them available to all workspaces in your organization.

If you want to promote models across environments (dev, test, prod), start by iteratively developing a model in dev. When you have a good candidate model, you can publish it to a registry. You can then deploy the model from the registry to endpoints in different workspaces.

> **Tip**
>
> If you already have models registered in a workspace, you can promote them to a registry. You can also register a model directly in a registry from the output of a training job.

If you want to develop a pipeline in one workspace and then run it in others, start by registering the components and environments that form the building blocks of the pipeline. When you submit the pipeline job, the workspace it runs in is selected by the compute and training data, which are unique to each workspace.

The following diagram illustrates promotion of pipelines between exploratory and dev workspaces, then model promotion between dev, test, and production.



Next steps

- Create a registry.
- Network isolation with registries.
- Share models, components, and environments using registries.

What are Azure Machine Learning pipelines?

Article • 04/04/2023

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

An Azure Machine Learning pipeline is an independently executable workflow of a complete machine learning task. An Azure Machine Learning pipeline helps to standardize the best practices of producing a machine learning model, enables the team to execute at scale, and improves the model building efficiency.

Why are Azure Machine Learning pipelines needed?

The core of a machine learning pipeline is to split a complete machine learning task into a multistep workflow. Each step is a manageable component that can be developed, optimized, configured, and automated individually. Steps are connected through well-defined interfaces. The Azure Machine Learning pipeline service automatically orchestrates all the dependencies between pipeline steps. This modular approach brings two key benefits:

- Standardize the Machine learning operation (MLOps) practice and support scalable team collaboration
- Training efficiency and cost reduction

Standardize the MLOps practice and support scalable team collaboration

Machine learning operation (MLOps) automates the process of building machine learning models and taking the model to production. This is a complex process. It usually requires collaboration from different teams with different skills. A well-defined machine learning pipeline can abstract this complex process into a multiple steps workflow, mapping each step to a specific task such that each team can work independently.

For example, a typical machine learning project includes the steps of data collection, data preparation, model training, model evaluation, and model deployment. Usually, the data engineers concentrate on data steps, data scientists spend most time on model

training and evaluation, the machine learning engineers focus on model deployment and automation of the entire workflow. By leveraging machine learning pipeline, each team only needs to work on building their own steps. The best way of building steps is using Azure Machine Learning component (v2), a self-contained piece of code that does one step in a machine learning pipeline. All these steps built by different users are finally integrated into one workflow through the pipeline definition. The pipeline is a collaboration tool for everyone in the project. The process of defining a pipeline and all its steps can be standardized by each company's preferred DevOps practice. The pipeline can be further versioned and automated. If the ML projects are described as a pipeline, then the best MLOps practice is already applied.

Training efficiency and cost reduction

Besides being the tool to put MLOps into practice, the machine learning pipeline also improves large model training's efficiency and reduces cost. Taking modern natural language model training as an example. It requires pre-processing large amounts of data and GPU intensive transformer model training. It takes hours to days to train a model each time. When the model is being built, the data scientist wants to test different training code or hyperparameters and run the training many times to get the best model performance. For most of these trainings, there's usually small changes from one training to another one. It will be a significant waste if every time the full training from data processing to model training takes place. By using machine learning pipeline, it can automatically calculate which steps result is unchanged and reuse outputs from previous training. Additionally, the machine learning pipeline supports running each step on different computation resources. Such that, the memory heavy data processing work and run-on high memory CPU machines, and the computation intensive training can run on expensive GPU machines. By properly choosing which step to run on which type of machines, the training cost can be significantly reduced.

Getting started best practices

Depending on what a machine learning project already has, the starting point of building a machine learning pipeline may vary. There are a few typical approaches to building a pipeline.

The first approach usually applies to the team that hasn't used pipeline before and wants to take some advantage of pipeline like MLOps. In this situation, data scientists typically have developed some machine learning models on their local environment using their favorite tools. Machine learning engineers need to take data scientists' output into production. The work involves cleaning up some unnecessary code from

original notebook or Python code, changes the training input from local data to parameterized values, split the training code into multiple steps as needed, perform unit test of each step, and finally wraps all steps into a pipeline.

Once the teams get familiar with pipelines and want to do more machine learning projects using pipelines, they'll find the first approach is hard to scale. The second approach is set up a few pipeline templates, each try to solve one specific machine learning problem. The template predefines the pipeline structure including how many steps, each step's inputs and outputs, and their connectivity. To start a new machine learning project, the team first forks one template repo. The team leader then assigns members which step they need to work on. The data scientists and data engineers do their regular work. When they're happy with their result, they structure their code to fit in the pre-defined steps. Once the structured codes are checked-in, the pipeline can be executed or automated. If there's any change, each member only needs to work on their piece of code without touching the rest of the pipeline code.

Once a team has built a collection of machine learnings pipelines and reusable components, they could start to build the machine learning pipeline from cloning previous pipeline or tie existing reusable component together. At this stage, the team's overall productivity will be improved significantly.

Azure Machine Learning offers different methods to build a pipeline. For users who are familiar with DevOps practices, we recommend using CLI. For data scientists who are familiar with python, we recommend writing pipeline using the Azure Machine Learning SDK v2. For users who prefer to use UI, they could use the designer to build pipeline by using registered components.

Which Azure pipeline technology should I use?

The Azure cloud provides several types of pipeline, each with a different purpose. The following table lists the different pipelines and what they're used for:

Scenario	Primary persona	Azure offering	OSS offering	Canonical pipe	Strengths
Model orchestration (Machine learning)	Data scientist	Azure Machine Learning Pipelines	Kubeflow Pipelines	Data -> Model	Distribution, caching, code-first, reuse
Data orchestration (Data prep)	Data engineer	Azure Data Factory pipelines	Apache Airflow	Data -> Data	Strongly typed movement, data-centric activities

Scenario	Primary persona	Azure offering	OSS offering	Canonical pipe	Strengths
Code & app orchestration (CI/CD)	App Developer / Ops	Azure Pipelines	Jenkins	Code + Model -> App/Service	Most open and flexible activity support, approval queues, phases with gating

Next steps

Azure Machine Learning pipelines are a powerful facility that begins delivering value in the early development stages.

- Define pipelines with the Azure Machine Learning CLI v2
- Define pipelines with the Azure Machine Learning SDK v2
- Define pipelines with Designer
- Try out CLI v2 pipeline example
- Try out Python SDK v2 pipeline example
- Learn about SDK and CLI v2 expressions that can be used in a pipeline.

What is an Azure Machine Learning component?

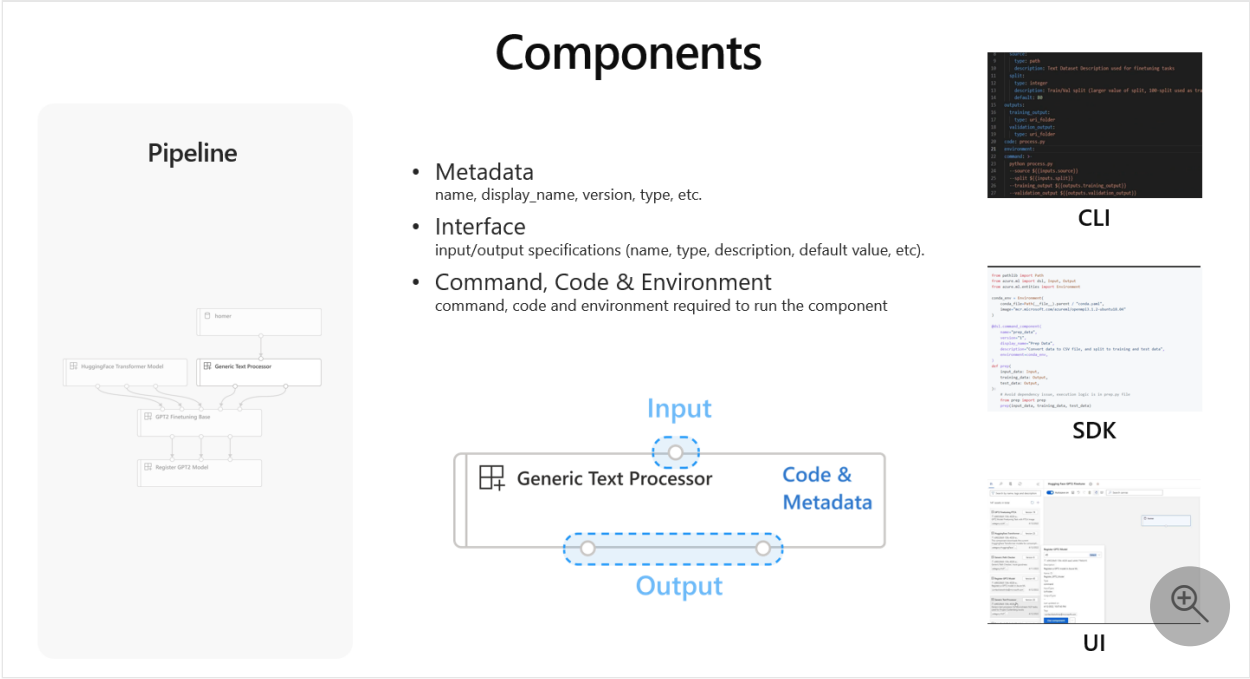
Article • 02/24/2023

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

An Azure Machine Learning component is a self-contained piece of code that does one step in a machine learning pipeline. A component is analogous to a function - it has a name, inputs, outputs, and a body. Components are the building blocks of the Azure Machine Learning pipelines.

A component consists of three parts:

- Metadata: name, display_name, version, type, etc.
- Interface: input/output specifications (name, type, description, default value, etc.).
- Command, Code & Environment: command, code and environment required to run the component.



Why should I use a component?

It's a good engineering practice to build a machine learning pipeline to split a complete machine learning task into a multi-step workflow. Such that, everyone can work on the specific step independently. In Azure Machine Learning, a component represents one

reusable step in a pipeline. Components are designed to help improve the productivity of pipeline building. Specifically, components offer:

- **Well-defined interface**: Components require a well-defined interface (input and output). The interface allows the user to build steps and connect steps easily. The interface also hides the complex logic of a step and removes the burden of understanding how the step is implemented.
- **Share and reuse**: As the building blocks of a pipeline, components can be easily shared and reused across pipelines, workspaces, and subscriptions. Components built by one team can be discovered and used by another team.
- **Version control**: Components are versioned. The component producers can keep improving components and publish new versions. Consumers can use specific component versions in their pipelines. This gives them compatibility and reproducibility.

Unit testable: A component is a self-contained piece of code. It's easy to write unit test for a component.

Component and Pipeline

A machine learning pipeline is the workflow for a full machine learning task. Components are the building blocks of a machine learning pipeline. When you're thinking of a component, it must be under the context of pipeline.

To build components, the first thing is to define the machine learning pipeline. This requires breaking down the full machine learning task into a multi-step workflow. Each step is a component. For example, considering a simple machine learning task of using historical data to train a sales forecasting model, you may want to build a sequential workflow with data processing, model training, and model evaluation steps. For complex tasks, you may want to further break down. For example, split one single data processing step into data ingestion, data cleaning, data pre-processing, and feature engineering steps.

Once the steps in the workflow are defined, the next thing is to specify how each step is connected in the pipeline. For example, to connect your data processing step and model training step, you may want to define a data processing component to output a folder that contains the processed data. A training component takes a folder as input and outputs a folder that contains the trained model. These inputs and outputs definition will become part of your component interface definition.

Now, it's time to develop the code of executing a step. You can use your preferred languages (python, R, etc.). The code must be able to be executed by a shell command. During the development, you may want to add a few inputs to control how this step is going to be executed. For example, for a training step, you may like to add learning rate, number of epochs as the inputs to control the training. These additional inputs plus the inputs and outputs required to connect with other steps are the interface of the component. The argument of a shell command is used to pass inputs and outputs to the code. The environment to execute the command and the code needs to be specified. The environment could be a curated Azure Machine Learning environment, a docker image or a conda environment.

Finally, you can package everything including code, cmd, environment, input, outputs, metadata together into a component. Then connects these components together to build pipelines for your machine learning workflow. One component can be used in multiple pipelines.

To learn more about how to build a component, see:

- How to build a component using Azure Machine Learning CLI v2.
- How to build a component using Azure Machine Learning SDK v2.

Next steps

- Define component with the Azure Machine Learning CLI v2.
- Define component with the Azure Machine Learning SDK v2.
- Define component with Designer.
- Component CLI v2 YAML reference.
- What is Azure Machine Learning Pipeline?.
- Try out CLI v2 component example.
- Try out Python SDK v2 component example.

Work with models in Azure Machine Learning

Article • 06/16/2023

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

Azure Machine Learning allows you to work with different types of models. In this article, you learn about using Azure Machine Learning to work with different model types, such as custom, MLflow, and Triton. You also learn how to register a model from different locations, and how to use the Azure Machine Learning SDK, the user interface (UI), and the Azure Machine Learning CLI to manage your models.

> **Tip**
>
> If you have model assets created that use the SDK/CLI v1, you can still use those with SDK/CLI v2. Full backward compatibility is provided. All models registered with the V1 SDK are assigned the type `custom`.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning.
- An Azure Machine Learning workspace.
- The Azure Machine Learning SDK v2 for Python.
- The Azure Machine Learning CLI v2.

Additionally, you will need to:

Azure CLI

- Install the Azure CLI and the ml extension to the Azure CLI. For more information, see Install, set up, and use the CLI (v2).

Supported paths

When you provide a model you want to register, you'll need to specify a `path` parameter that points to the data or job location. Below is a table that shows the different data locations supported in Azure Machine Learning and examples for the `path` parameter:

Location	**Examples**
A path on your local computer	`mlflow-model/model.pkl`
A path on an Azure Machine Learning Datastore	`azureml://datastores/<datastore-name>/paths/<path_on_datastore>`
A path from an Azure Machine Learning job	`azureml://jobs/<job-name>/outputs/<output-name>/paths/<path-to-model-relative-to-the-named-output-location>`
A path from an MLflow job	`runs:/<run-id>/<path-to-model-relative-to-the-root-of-the-artifact-location>`
A path from a Model Asset in Azure Machine Learning Workspace	`azureml:<model-name>:<version>`
A path from a Model Asset in Azure Machine Learning Registry	`azureml://registries/<registry-name>/models/<model-name>/versions/<version>`

Supported modes

When you run a job with model inputs/outputs, you can specify the *mode* - for example, whether you would like the model to be read-only mounted or downloaded to the compute target. The table below shows the possible modes for different type/mode/input/output combinations:

Type	**Input/Output**	`upload`	`download`	`ro_mount`	`rw_mount`	`direct`
`custom` file	Input					
`custom` folder	Input		✓	✓		✓
`mlflow`	Input		✓	✓		
`custom` file	Output	✓			✓	✓
`custom` folder	Output	✓			✓	✓
`mlflow`	Output	✓			✓	✓

Follow along in Jupyter Notebooks

You can follow along this sample in a Jupyter Notebook. In the azureml-examples repository, open the notebook: model.ipynb.

Create a model in the model registry

Model registration allows you to store and version your models in the Azure cloud, in your workspace. The model registry helps you organize and keep track of your trained models.

The code snippets in this section cover how to:

- Register your model as an asset in Machine Learning by using the CLI.
- Register your model as an asset in Machine Learning by using the SDK.
- Register your model as an asset in Machine Learning by using the UI.

These snippets use `custom` and `mlflow`.

- `custom` is a type that refers to a model file or folder trained with a custom standard not currently supported by Azure Machine Learning.
- `mlflow` is a type that refers to a model trained with mlflow. MLflow trained models are in a folder that contains the *MLmodel* file, the *model* file, the *conda dependencies* file, and the *requirements.txt* file.

Connect to your workspace

First, let's connect to Azure Machine Learning workspace where we are going to work on.

Azure CLI

Azure CLI

```
az account set --subscription <subscription>
az configure --defaults workspace=<workspace> group=<resource-group>
location=<location>
```

Register your model as an asset in Machine Learning by using the CLI

Use the following tabs to select where your model is located.

Local model

YAML

```yaml
$schema: https://azuremlschemas.azureedge.net/latest/model.schema.json
name: local-file-example
path: mlflow-model/model.pkl
description: Model created from local file.
```

Bash

```bash
az ml model create -f <file-name>.yml
```

For a complete example, see the model YAML.

Register your model as an asset in Machine Learning by using the SDK

Use the following tabs to select where your model is located.

Local model

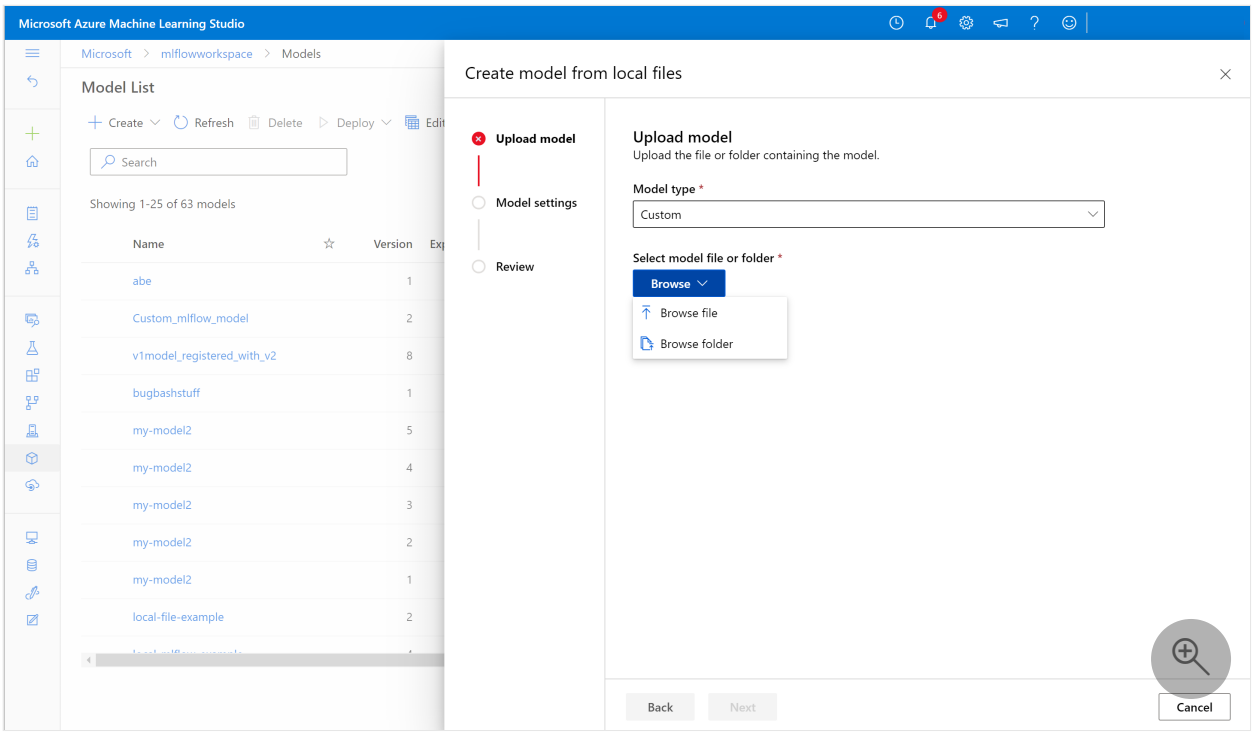
Python

```python
from azure.ai.ml.entities import Model
from azure.ai.ml.constants import AssetTypes

file_model = Model(
    path="mlflow-model/model.pkl",
    type=AssetTypes.CUSTOM_MODEL,
    name="local-file-example",
    description="Model created from local file.",
)
ml_client.models.create_or_update(file_model)
```

Register your model as an asset in Machine Learning by using the UI

To create a model in Machine Learning, from the UI, open the **Models** page. Select **Register model**, and select where your model is located. Fill out the required fields, and

then select **Register**.



Manage models

The SDK and CLI (v2) also allow you to manage the lifecycle of your Azure Machine Learning model assets.

List

List all the models in your workspace:

Azure CLI

cli

```
az ml model list
```

List all the model versions under a given name:

Azure CLI

cli

```
az ml model list --name run-model-example
```

Show

Get the details of a specific model:

Azure CLI

```cli
az ml model show --name run-model-example --version 1
```

Update

Update mutable properties of a specific model:

Azure CLI

```cli
az ml model update --name  run-model-example --version 1 --set
description="This is an updated description." --set tags.stage="Prod"
```

> **ⓘ Important**
>
> For model, only `description` and `tags` can be updated. All other properties are immutable; if you need to change any of those properties you should create a new version of the model.

Archive

Archiving a model will hide it by default from list queries (`az ml model list`). You can still continue to reference and use an archived model in your workflows. You can archive either all versions of a model or only a specific version.

If you don't specify a version, all versions of the model under that given name will be archived. If you create a new model version under an archived model container, that new version will automatically be set as archived as well.

Archive all versions of a model:

Azure CLI

cli

```
az ml model archive --name run-model-example
```

Archive a specific model version:

Azure CLI

cli

```
az ml model archive --name run-model-example --version 1
```

Use model for training

The SDK and CLI (v2) also allow you to use a model in a training job as an input or output.

Use model as input in a job

Azure CLI

Create a job specification YAML file (`<file-name>.yml`). Specify in the `inputs` section of the job:

1. The `type`; whether the model is a `mlflow_model`, `custom_model` or `triton_model`.
2. The `path` of where your data is located; can be any of the paths outlined in the Supported Paths section.

YAML

```
$schema:
https://azuremlschemas.azureedge.net/latest/commandJob.schema.json

# Possible Paths for models:
# AzureML Datastore: azureml://datastores/<datastore-
name>/paths/<path_on_datastore>
# MLflow run: runs:/<run-id>/<path-to-model-relative-to-the-root-of-the-
artifact-location>
```

```
# Job: azureml://jobs/<job-name>/outputs/<output-name>/paths/<path-to-
model-relative-to-the-named-output-location>
# Model Asset: azureml:<my_model>:<version>

command: |
  ls ${{inputs.my_model}}
inputs:
  my_model:
    type: mlflow_model # List of all model types here:
https://learn.microsoft.com/azure/machine-learning/reference-yaml-
model#yaml-syntax
    path: ../../assets/model/mlflow-model
environment: azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
```

Next, run in the CLI

Azure CLI

```
az ml job create -f <file-name>.yml
```

For a complete example, see the model GitHub repo.

Use model as output in a job

In your job you can write model to your cloud-based storage using *outputs*.

Azure CLI

Create a job specification YAML file (`<file-name>.yml`), with the `outputs` section populated with the type and path of where you would like to write your data to:

YAML

```
$schema:
https://azuremlschemas.azureedge.net/latest/commandJob.schema.json

# Possible Paths for Model:
# Local path: mlflow-model/model.pkl
# AzureML Datastore: azureml://datastores/<datastore-
name>/paths/<path_on_datastore>
# MLflow run: runs:/<run-id>/<path-to-model-relative-to-the-root-of-the-
artifact-location>
# Job: azureml://jobs/<job-name>/outputs/<output-name>/paths/<path-to-
model-relative-to-the-named-output-location>
# Model Asset: azureml:<my_model>:<version>

code: src
```

```
command: >-
  python hello-model-as-output.py
  --input_model ${{inputs.input_model}}
  --custom_model_output ${{outputs.output_folder}}
inputs:
  input_model:
    type: mlflow_model # mlflow_model,custom_model, triton_model
    path: ../../assets/model/mlflow-model
outputs:
  output_folder:
    type: custom_model # mlflow_model,custom_model, triton_model
environment: azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest
```

Next create a job using the CLI:

Azure CLI

```
az ml job create --file <file-name>.yml
```

For a complete example, see the model GitHub repo.

Next steps

- Install and set up Python SDK v2
- No-code deployment for MLflow models
- Learn more about MLflow and Azure Machine Learning

Git integration for Azure Machine Learning

Article • 06/02/2023

Git is a popular version control system that allows you to share and collaborate on your projects.

Azure Machine Learning fully supports Git repositories for tracking work - you can clone repositories directly onto your shared workspace file system, use Git on your local workstation, or use Git from a CI/CD pipeline.

When submitting a job to Azure Machine Learning, if source files are stored in a local git repository then information about the repo is tracked as part of the training process.

Since Azure Machine Learning tracks information from a local git repo, it isn't tied to any specific central repository. Your repository can be cloned from GitHub, GitLab, Bitbucket, Azure DevOps, or any other git-compatible service.

> **Tip**
>
> Use Visual Studio Code to interact with Git through a graphical user interface. To connect to an Azure Machine Learning remote compute instance using Visual Studio Code, see **Launch Visual Studio Code integrated with Azure Machine Learning (preview)**
>
> For more information on Visual Studio Code version control features, see **Using Version Control in VS Code** and **Working with GitHub in VS Code**.

Clone Git repositories into your workspace file system

Azure Machine Learning provides a shared file system for all users in the workspace. To clone a Git repository into this file share, we recommend that you create a compute instance & open a terminal. Once the terminal is opened, you have access to a full Git client and can clone and work with Git via the Git CLI experience.

We recommend that you clone the repository into your user directory so that others will not make collisions directly on your working branch.

> **Tip**
>
> There is a performance difference between cloning to the local file system of the compute instance or cloning to the mounted filesystem (mounted as the `~/cloudfiles/code` directory). In general, cloning to the local filesystem will have better performance than to the mounted filesystem. However, the local filesystem is lost if you delete and recreate the compute instance. The mounted filesystem is kept if you delete and recreate the compute instance.

You can clone any Git repository you can authenticate to (GitHub, Azure Repos, BitBucket, etc.)

For more information about cloning, see the guide on how to use Git CLI.

Authenticate your Git Account with SSH

Generate a new SSH key

1. Open the terminal window in the Azure Machine Learning Notebook Tab.
2. Paste the text below, substituting in your email address.

```Bash
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private rsa key pair.
```

3. When you're prompted to "Enter a file in which to save the key" press Enter. This accepts the default file location.
4. Verify that the default location is '/home/azureuser/.ssh' and press enter. Otherwise specify the location '/home/azureuser/.ssh'.

> **Tip**

Make sure the SSH key is saved in '/home/azureuser/.ssh'. This file is saved on the compute instance is only accessible by the owner of the Compute Instance

```
> Enter a file in which to save the key (/home/azureuser/.ssh/id_rsa):
[Press enter]
```

5. At the prompt, type a secure passphrase. We recommend you add a passphrase to your SSH key for added security

```
> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

Add the public key to Git Account

1. In your terminal window, copy the contents of your public key file. If you renamed the key, replace id_rsa.pub with the public key file name.

Bash

```
cat ~/.ssh/id_rsa.pub
```

Tip

Copy and Paste in Terminal

- Windows: `Ctrl-Insert` to copy and use `Ctrl-Shift-v` or `Shift-Insert` to paste.
- Mac OS: `Cmd-c` to copy and `Cmd-v` to paste.
- FireFox/IE may not support clipboard permissions properly.

2. Select and copy the SSH key output to your clipboard.
3. Next, follow the steps to add the SSH key to your preferred account type:

- GitHub
- GitLab

- Azure DevOps Start at **Step 2**.
- BitBucket. Follow **Step 4**.

Clone the Git repository with SSH

1. Copy the SSH Git clone URL from the Git repo.
2. Paste the url into the `git clone` command below, to use your SSH Git repo URL. This will look something like:

```
Bash

git clone git@example.com:GitUser/azureml-example.git
Cloning into 'azureml-example'...
```

You will see a response like:

```
Bash

The authenticity of host 'example.com (192.30.255.112)' can't be
established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'github.com,192.30.255.112' (RSA) to the list of
known hosts.
```

SSH may display the server's SSH fingerprint and ask you to verify it. You should verify that the displayed fingerprint matches one of the fingerprints in the SSH public keys page.

SSH displays this fingerprint when it connects to an unknown host to protect you from man-in-the-middle attacks. Once you accept the host's fingerprint, SSH will not prompt you again unless the fingerprint changes.

3. When you are asked if you want to continue connecting, type `yes`. Git will clone the repo and set up the origin remote to connect with SSH for future Git commands.

Track code that comes from Git repositories

When you submit a training job from the Python SDK or Machine Learning CLI, the files needed to train the model are uploaded to your workspace. If the `git` command is available on your development environment, the upload process uses it to check if the

files are stored in a git repository. If so, then information from your git repository is also uploaded as part of the training job. This information is stored in the following properties for the training job:

Property	Git command used to get the value	Description
`azureml.git.repository_uri`	`git ls-remote --get-url`	The URI that your repository was cloned from.
`mlflow.source.git.repoURL`	`git ls-remote --get-url`	The URI that your repository was cloned from.
`azureml.git.branch`	`git symbolic-ref --short HEAD`	The active branch when the job was submitted.
`mlflow.source.git.branch`	`git symbolic-ref --short HEAD`	The active branch when the job was submitted.
`azureml.git.commit`	`git rev-parse HEAD`	The commit hash of the code that was submitted for the job.
`mlflow.source.git.commit`	`git rev-parse HEAD`	The commit hash of the code that was submitted for the job.
`azureml.git.dirty`	`git status --porcelain .`	`True`, if the branch/commit is dirty; otherwise, `false`.

This information is sent for jobs that use an estimator, machine learning pipeline, or script run.

If your training files are not located in a git repository on your development environment, or the `git` command is not available, then no git-related information is tracked.

> **Tip**
>
> To check if the git command is available on your development environment, open a shell session, command prompt, PowerShell or other command line interface and type the following command:
>
> ```
> git --version
> ```

If installed, and in the path, you receive a response similar to `git version 2.4.1`.

For more information on installing git on your development environment, see the **Git website**.

View the logged information

The git information is stored in the properties for a training job. You can view this information using the Azure portal or Python SDK.

Azure portal

1. From the studio portal, select your workspace.
2. Select **Jobs**, and then select one of your experiments.
3. Select one of the jobs from the **Display name** column.
4. Select **Outputs + logs**, and then expand the **logs** and **azureml** entries. Select the link that begins with ###_**azure**.

The logged information contains text similar to the following JSON:

JSON

```
"properties": {
    "_azureml.ComputeTargetType": "batchai",
    "ContentSnapshotId": "5ca66406-cbac-4d7d-bc95-f5a51dd3e57e",
    "azureml.git.repository_uri":
"git@github.com:azure/machinelearningnotebooks",
    "mlflow.source.git.repoURL":
"git@github.com:azure/machinelearningnotebooks",
    "azureml.git.branch": "master",
    "mlflow.source.git.branch": "master",
    "azureml.git.commit": "4d2b93784676893f8e346d5f0b9fb894a9cf0742",
    "mlflow.source.git.commit": "4d2b93784676893f8e346d5f0b9fb894a9cf0742",
    "azureml.git.dirty": "True",
    "AzureML.DerivedImageName":
"azureml/azureml_9d3568242c6bfef9631879915768deaf",
    "ProcessInfoFile": "azureml-logs/process_info.json",
    "ProcessStatusFile": "azureml-logs/process_status.json"
}
```

View properties

After submitting a training run, a Job object is returned. The `properties` attribute of this object contains the logged git information. For example, the following code retrieves the commit hash:

Python SDK

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

Python

```
job.properties["azureml.git.commit"]
```

Next steps

- Access a compute instance terminal in your workspace

Share models, components, and environments across workspaces with registries

Article • 03/31/2023

Azure Machine Learning registry enables you to collaborate across workspaces within your organization. Using registries, you can share models, components, and environments.

There are two scenarios where you'd want to use the same set of models, components and environments in multiple workspaces:

- **Cross-workspace MLOps**: You're training a model in a `dev` workspace and need to deploy it to `test` and `prod` workspaces. In this case you, want to have end-to-end lineage between endpoints to which the model is deployed in `test` or `prod` workspaces and the training job, metrics, code, data and environment that was used to train the model in the `dev` workspace.
- **Share and reuse models and pipelines across different teams**: Sharing and reuse improve collaboration and productivity. In this scenario, you may want to publish a trained model and the associated components and environments used to train it to a central catalog. From there, colleagues from other teams can search and reuse the assets you shared in their own experiments.

In this article, you'll learn how to:

- Create an environment and component in the registry.
- Use the component from registry to submit a model training job in a workspace.
- Register the trained model in the registry.
- Deploy the model from the registry to an online-endpoint in the workspace, then submit an inference request.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning.

- An Azure Machine Learning registry to share models, components and environments. To create a registry, see Learn how to create a registry.
- An Azure Machine Learning workspace. If you don't have one, use the steps in the Quickstart: Create workspace resources article to create one.

  > **ⓘ Important**
  >
  > The Azure region (location) where you create your workspace must be in the list of supported regions for Azure Machine Learning registry

- The Azure CLI and the `ml` extension **or** the Azure Machine Learning Python SDK v2:

  Azure CLI

  To install the Azure CLI and extension, see Install, set up, and use the CLI (v2).

  > **ⓘ Important**
  >
  > - The CLI examples in this article assume that you are using the Bash (or compatible) shell. For example, from a Linux system or **Windows Subsystem for Linux**.
  > - The examples also assume that you have configured defaults for the Azure CLI so that you don't have to specify the parameters for your subscription, workspace, resource group, or location. To set default settings, use the following commands. Replace the following parameters with the values for your configuration:
  >   - Replace `<subscription>` with your Azure subscription ID.
  >   - Replace `<workspace>` with your Azure Machine Learning workspace name.
  >   - Replace `<resource-group>` with the Azure resource group that contains your workspace.
  >   - Replace `<location>` with the Azure region that contains your workspace.
  >
  > Azure CLI
  >
  > ```
  > az account set --subscription <subscription>
  > az configure --defaults workspace=<workspace> group=
  > <resource-group> location=<location>
  > ```

You can see what your current defaults are by using the `az configure -l` command.

Clone examples repository

The code examples in this article are based on the `nyc_taxi_data_regression` sample in the examples repository. To use these files on your development environment, use the following commands to clone the repository and change directories to the example:

Bash

```
git clone https://github.com/Azure/azureml-examples
cd azureml-examples
# changing branch is temporary until samples merge to main
git checkout mabables/registry
```

Azure CLI

For the CLI example, change directories to `cli/jobs/pipelines-with-components/nyc_taxi_data_regression` in your local clone of the examples repository.

Bash

```
cd cli/jobs/pipelines-with-components/nyc_taxi_data_regression
```

Create SDK connection

Tip

This step is only needed when using the Python SDK.

Create a client connection to both the Azure Machine Learning workspace and registry:

Python

```
ml_client_workspace = MLClient( credential=credential,
    subscription_id = "<workspace-subscription>",
```

```
    resource_group_name = "<workspace-resource-group",
    workspace_name = "<workspace-name>")
print(ml_client_workspace)

ml_client_registry = MLClient(credential=credential,
                        registry_name="<REGISTRY_NAME>",
                        registry_location="<REGISTRY_REGION>")
print(ml_client_registry)
```

Create environment in registry

Environments define the docker container and Python dependencies required to run training jobs or deploy models. For more information on environments, see the following articles:

- Environment concepts
- How to create environments (CLI) articles.

Azure CLI

> **Tip**
>
> The same CLI command `az ml environment create` can be used to create environments in a workspace or registry. Running the command with `--workspace-name` command creates the environment in a workspace whereas running the command with `--registry-name` creates the environment in the registry.

We'll create an environment that uses the `python:3.8` docker image and installs Python packages required to run a training job using the SciKit Learn framework. If you've cloned the examples repo and are in the folder `cli/jobs/pipelines-with-components/nyc_taxi_data_regression`, you should be able to see environment definition file `env_train.yml` that references the docker file `env_train/Dockerfile`. The `env_train.yml` is shown below for your reference:

YAML

```yaml
$schema:
https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: SKLearnEnv
version: 1
build:
  path: ./env_train
```

Create the environment using the `az ml environment create` as follows

Azure CLI

```
az ml environment create --file env_train.yml --registry-name <registry-name>
```

If you get an error that an environment with this name and version already exists in the registry, you can either edit the `version` field in `env_train.yml` or specify a different version on the CLI that overrides the version value in `env_train.yml`.

Azure CLI

```
# use shell epoch time as the version
version=$(date +%s)
az ml environment create --file env_train.yml --registry-name <registry-name> --set version=$version
```

> **Tip**
>
> `version=$(date +%s)` works only in Linux. Replace `$version` with a random number if this does not work.

Note down the `name` and `version` of the environment from the output of the `az ml environment create` command and use them with `az ml environment show` commands as follows. You'll need the `name` and `version` in the next section when you create a component in the registry.

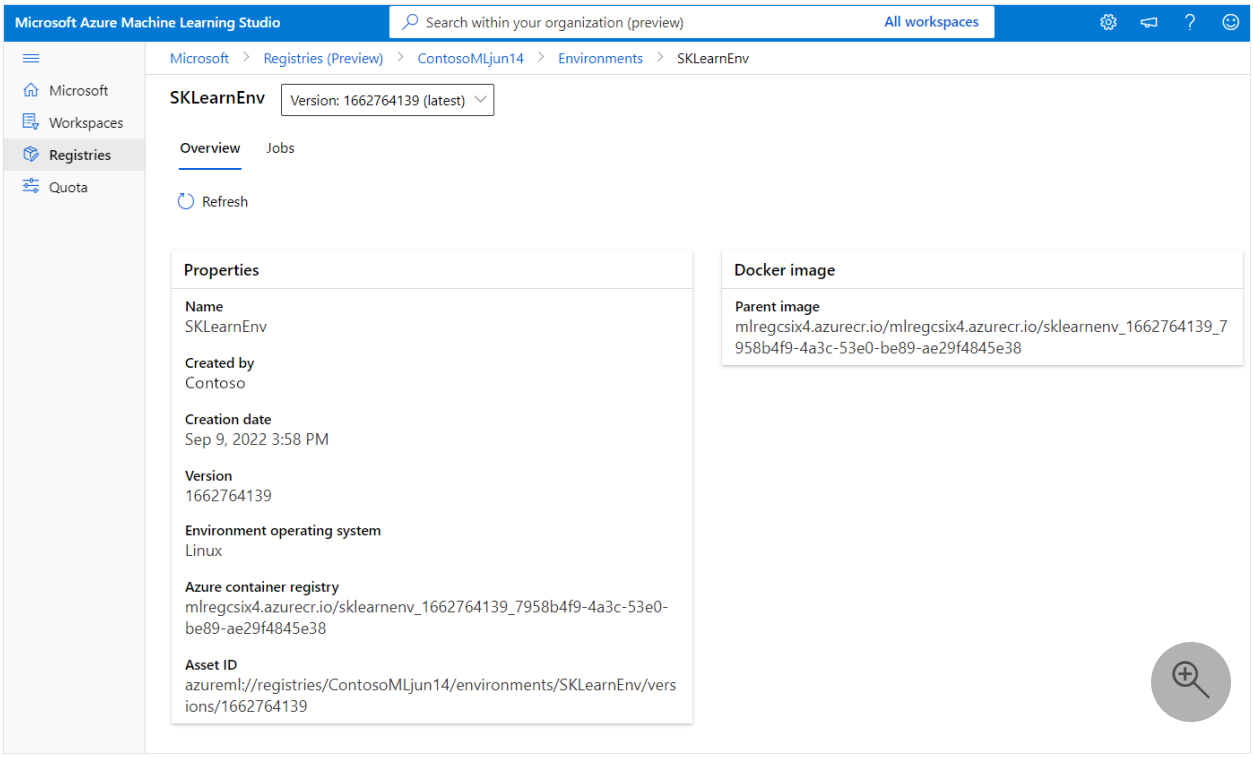
Azure CLI

```
az ml environment show --name SKLearnEnv --version 1 --registry-name <registry-name>
```

> **Tip**
>
> If you used a different environment name or version, replace the `--name` and `--version` parameters accordingly.

You can also use `az ml environment list --registry-name <registry-name>` to list all environments in the registry.

You can browse all environments in the Azure Machine Learning studio. Make sure you navigate to the global UI and look for the **Registries** entry.



Create a component in registry

Components are reusable building blocks of Machine Learning pipelines in Azure Machine Learning. You can package the code, command, environment, input interface and output interface of an individual pipeline step into a component. Then you can reuse the component across multiple pipelines without having to worry about porting dependencies and code each time you write a different pipeline.

Creating a component in a workspace allows you to use the component in any pipeline job within that workspace. Creating a component in a registry allows you to use the component in any pipeline in any workspace within your organization. Creating components in a registry is a great way to build modular reusable utilities or shared training tasks that can be used for experimentation by different teams within your organization.

For more information on components, see the following articles:

- Component concepts
- How to use components in pipelines (CLI)
- How to use components in pipelines (SDK)

Azure CLI

Make sure you are in the folder `cli/jobs/pipelines-with-components/nyc_taxi_data_regression`. You'll find the component definition file `train.yml` that packages a Scikit Learn training script `train_src/train.py` and the curated environment `AzureML-sklearn-0.24-ubuntu18.04-py37-cpu`. We'll use the Scikit Learn environment created in pervious step instead of the curated environment. You can edit `environment` field in the `train.yml` to refer to your Scikit Learn environment. The resulting component definition file `train.yml` will be similar to the following example:

YAML

```yaml
# <component>
$schema:
https://azuremlschemas.azureedge.net/latest/commandComponent.schema.json
name: train_linear_regression_model
display_name: TrainLinearRegressionModel
version: 1
type: command
inputs:
  training_data:
    type: uri_folder
  test_split_ratio:
    type: number
    min: 0
    max: 1
    default: 0.2
outputs:
  model_output:
    type: mlflow_model
  test_data:
    type: uri_folder
code: ./train_src
environment: azureml://registries/<registry-
name>/environments/SKLearnEnv/versions/1`
command: >-
  python train.py
  --training_data ${{inputs.training_data}}
  --test_data ${{outputs.test_data}}
  --model_output ${{outputs.model_output}}
  --test_split_ratio ${{inputs.test_split_ratio}}
```

If you used different name or version, the more generic representation looks like this: `environment: azureml://registries/<registry-name>/environments/<sklearn-environment-name>/versions/<sklearn-environment-version>`, so make sure you replace the `<registry-name>`, `<sklearn-environment-name>` and `<sklearn-`

`environment-version>` accordingly. You then run the `az ml component create` command to create the component as follows.

Azure CLI

```
az ml component create --file train.yml --registry-name <registry-name>
```

> **Tip**
>
> The same the CLI command `az ml component create` can be used to create components in a workspace or registry. Running the command with `--workspace-name` command creates the component in a workspace whereas running the command with `--registry-name` creates the component in the registry.

If you prefer to not edit the `train.yml`, you can override the environment name on the CLI as follows:

Azure CLI

```
az ml component create --file train.yml --registry-name <registry-name>`
--set environment=azureml://registries/<registry-
name>/environments/SKLearnEnv/versions/1
# or if you used a different name or version, replace `<sklearn-
environment-name>` and `<sklearn-environment-version>` accordingly
az ml component create --file train.yml --registry-name <registry-name>`
--set environment=azureml://registries/<registry-
name>/environments/<sklearn-environment-name>/versions/<sklearn-
environment-version>
```

> **Tip**
>
> If you get an error that the name of the component already exists in the registry, you can either edit the version in `train.yml` or override the version on the CLI with a random version.

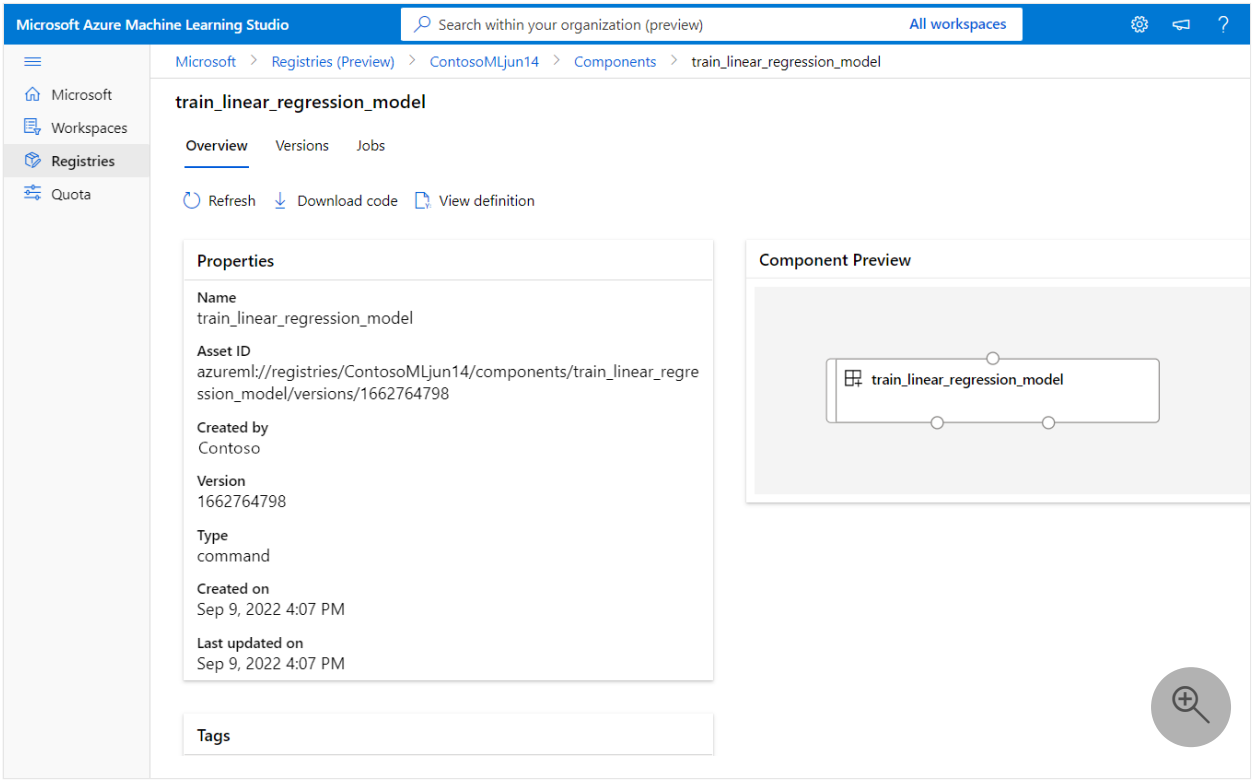
Note down the `name` and `version` of the component from the output of the `az ml component create` command and use them with `az ml component show` commands as follows. You'll need the `name` and `version` in the next section when you create submit a training job in the workspace.

Azure CLI

```
az ml component show --name <component_name> --version
<component_version> --registry-name <registry-name>
```

You can also use `az ml component list --registry-name <registry-name>` to list all components in the registry.

You can browse all components in the Azure Machine Learning studio. Make sure you navigate to the global UI and look for the **Registries** entry.



Run a pipeline job in a workspace using component from registry

When running a pipeline job that uses a component from a registry, the *compute* resources and *training data* are local to the workspace. For more information on running jobs, see the following articles:

- Running jobs (CLI)
- Running jobs (SDK)
- Pipeline jobs with components (CLI)
- Pipeline jobs with components (SDK)

Azure CLI

We'll run a pipeline job with the Scikit Learn training component created in the previous section to train a model. Check that you are in the folder `cli/jobs/pipelines-with-components/nyc_taxi_data_regression`. The training dataset is located in the `data_transformed` folder. Edit the `component` section in under the `train_job` section of the `single-job-pipeline.yml` file to refer to the training component created in the previous section. The resulting `single-job-pipeline.yml` is shown below.

YAML

```yaml
$schema:
https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: nyc_taxi_data_regression_single_job
description: Single job pipeline to train regression model based on nyc
taxi dataset

jobs:
  train_job:
    type: command
    component: azureml://registries/<registry-
name>/component/train_linear_regression_model/versions/1
    compute: azureml:cpu-cluster
    inputs:
      training_data:
        type: uri_folder
        path: ./data_transformed
    outputs:
      model_output:
        type: mlflow_model
      test_data:
```

The key aspect is that this pipeline is going to run in a workspace using a component that isn't in the specific workspace. The component is in a registry that can be used with any workspace in your organization. You can run this training job in any workspace you have access to without having worry about making the training code and environment available in that workspace.

⚠ **Warning**

- Before running the pipeline job, confirm that the workspace in which you will run the job is in a Azure region that is supported by the registry in which you created the component.
- Confirm that the workspace has a compute cluster with the name `cpu-cluster` or edit the `compute` field under `jobs.train_job.compute` with the

name of your compute.

Run the pipeline job with the `az ml job create` command.

Azure CLI

```
az ml job create --file single-job-pipeline.yml
```

Tip

If you have not configured the default workspace and resource group as explained in the prerequisites section, you will need to specify the `--workspace-name` and `--resource-group` parameters for the `az ml job create` to work.

Alternatively, ou can skip editing `single-job-pipeline.yml` and override the component name used by `train_job` in the CLI.

Azure CLI

```
az ml job create --file single-job-pipeline.yml --set
jobs.train_job.component=azureml://registries/<registry-
name>/component/train_linear_regression_model/versions/1
```

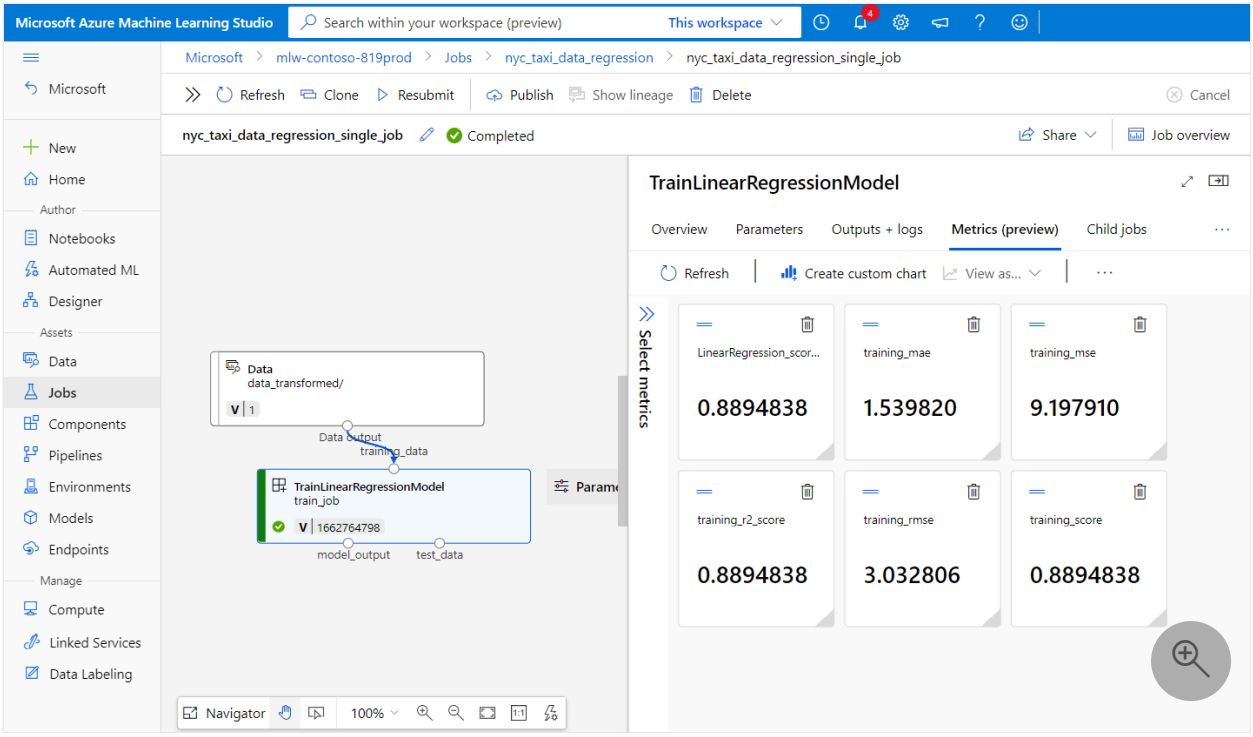
Since the component used in the training job is shared through a registry, you can submit the job to any workspace that you have access to in your organization, even across different subscriptions. For example, if you have `dev-workspace`, `test-workspace` and `prod-workspace`, running the training job in these three workspaces is as easy as running three `az ml job create` commands.

Azure CLI

```
az ml job create --file single-job-pipeline.yml --workspace-name dev-
workspace --resource-group <resource-group-of-dev-workspace>
az ml job create --file single-job-pipeline.yml --workspace-name test-
workspace --resource-group <resource-group-of-test-workspace>
az ml job create --file single-job-pipeline.yml --workspace-name prod-
workspace --resource-group <resource-group-of-prod-workspace>
```

In Azure Machine Learning studio, select the endpoint link in the job output to view the job. Here you can analyze training metrics, verify that the job is using the component

and environment from registry, and review the trained model. Note down the `name` of the job from the output or find the same information from the job overview in Azure Machine Learning studio. You'll need this information to download the trained model in the next section on creating models in registry.



Create a model in registry

You'll learn how to create models in a registry in this section. Review manage models to learn more about model management in Azure Machine Learning. We'll look at two different ways to create a model in a registry. First is from local files. Second, is to copy a model registered in the workspace to a registry.

In both the options, you'll create model with the MLflow format, which will help you to deploy this model for inference without writing any inference code.

Create a model in registry from local files

Azure CLI

Download the model, which is available as output of the `train_job` by replacing `<job-name>` with the name from the job from the previous section. The model along with MLflow metadata files should be available in the `./artifacts/model/`.

Azure CLI

```
# fetch the name of the train_job by listing all child jobs of the
pipeline job
train_job_name=$(az ml job list --parent-job-name <job-name> --query
[0].name | sed 's/\"//g')
# download the default outputs of the train_job
az ml job download --name $train_job_name
# review the model files
ls -l ./artifacts/model/
```

> **Tip**
>
> If you have not configured the default workspace and resource group as explained in the prerequisites section, you will need to specify the `--workspace-name` and `--resource-group` parameters for the `az ml model create` to work.

> **⚠ Warning**
>
> The output of `az ml job list` is passed to `sed`. This works only on Linux shells. If you are on Windows, run `az ml job list --parent-job-name <job-name> --query [0].name` and strip any quotes you see in the train job name.

If you're unable to download the model, you can find sample MLflow model trained by the training job in the previous section in `cli/jobs/pipelines-with-components/nyc_taxi_data_regression/artifacts/model/` folder.

Create the model in the registry:

Azure CLI

```
# create model in registry
az ml model create --name nyc-taxi-model --version 1 --type mlflow_model
--path ./artifacts/model/ --registry-name <registry-name>
```

> **Tip**
>
> - Use a random number for the `version` parameter if you get an error that model name and version exists.
> - The same the CLI command `az ml model create` can be used to create models in a workspace or registry. Running the command with `--`

`workspace-name` command creates the model in a workspace whereas running the command with `--registry-name` creates the model in the registry.

Share a model from workspace to registry

In this workflow, you'll first create the model in the workspace and then share it to the registry. This workflow is useful when you want to test the model in the workspace before sharing it. For example, deploy it to endpoints, try out inference with some test data and then copy the model to a registry if everything looks good. This workflow may also be useful when you're developing a series of models using different techniques, frameworks or parameters and want to promote just one of them to the registry as a production candidate.

Azure CLI

Make sure you have the name of the pipeline job from the previous section and replace that in the command to fetch the training job name below. You'll then register the model from the output of the training job into the workspace. Note how the `--path` parameter refers to the output `train_job` output with the `azureml://jobs/$train_job_name/outputs/artifacts/paths/model` syntax.

Azure CLI

```
# fetch the name of the train_job by listing all child jobs of the
pipeline job
train_job_name=$(az ml job list --parent-job-name <job-name> --
workspace-name <workspace-name> --resource-group <workspace-resource-
group> --query [0].name | sed 's/\"//g')
# create model in workspace
az ml model create --name nyc-taxi-model --version 1 --type mlflow_model
--path azureml://jobs/$train_job_name/outputs/artifacts/paths/model
```

Tip

- Use a random number for the `version` parameter if you get an error that model name and version exists.`
- If you have not configured the default workspace and resource group as explained in the prerequisites section, you will need to specify the `--`

`workspace-name` and `--resource-group` parameters for the `az ml model create` to work.

Note down the model name and version. You can validate if the model is registered in the workspace by browsing it in the Studio UI or using `az ml model show --name nyc-taxi-model --version $model_version` command.

Next, you'll now share the model from the workspace to the registry.

Azure CLI

```
# share model registered in workspace to registry
az ml model share --name nyc-taxi-model --version 1 --registry-name
<registry-name> --share-with-name <new-name> --share-with-version <new-
version>
```

Tip

- Make sure to use the right model name and version if you changed it in the `az ml model create` command.
- The above command has two optional parameters "--share-with-name" and "--share-with-version". If these are not provided the new model will have the same name and version as the model that is being shared. Note down the `name` and `version` of the model from the output of the `az ml model create` command and use them with `az ml model show` commands as follows. You'll need the `name` and `version` in the next section when you deploy the model to an online endpoint for inference.

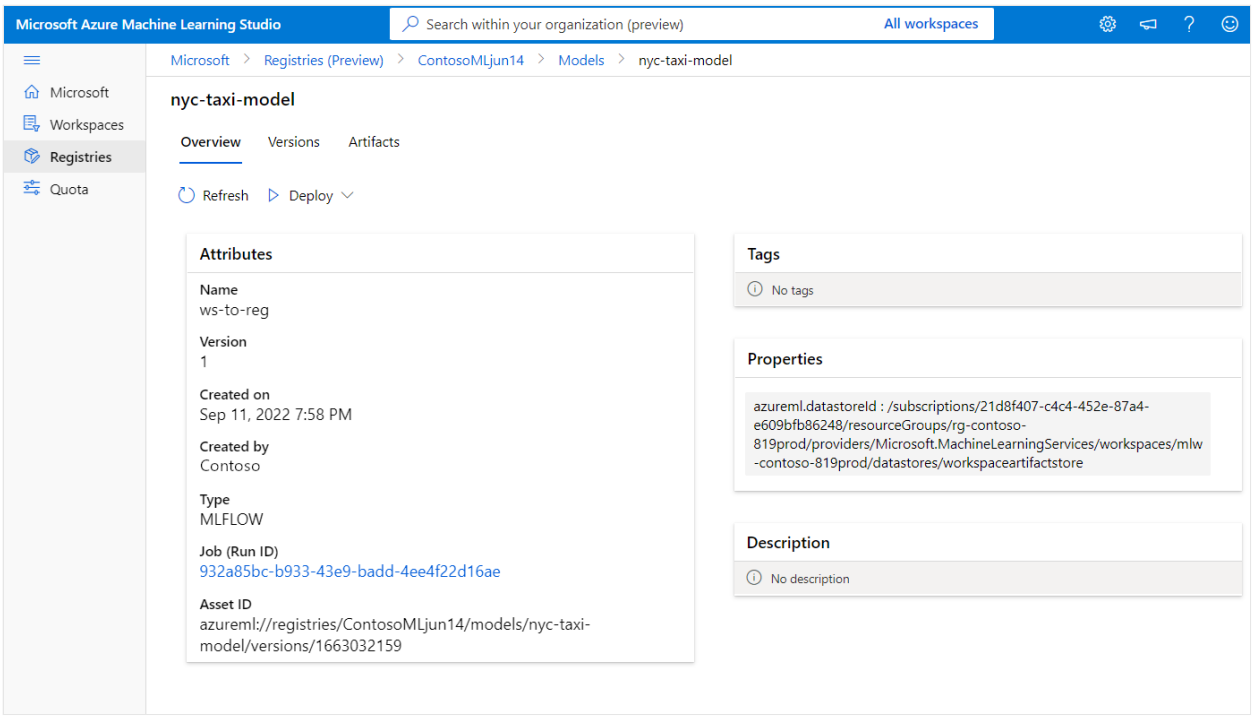
Azure CLI

```
az ml model show --name <model_name> --version <model_version> --
registry-name <registry-name>
```

You can also use `az ml model list --registry-name <registry-name>` to list all models in the registry or browse all components in the Azure Machine Learning studio UI. Make sure you navigate to the global UI and look for the Registries hub.

The following screenshot shows a model in a registry in Azure Machine Learning studio. If you created a model from the job output and then copied the model from the workspace to registry, you'll see that the model has a link to the job that trained the

model. You can use that link to navigate to the training job to review the code, environment and data used to train the model.



Deploy model from registry to online endpoint in workspace

In the last section, you'll deploy a model from registry to an online endpoint in a workspace. You can choose to deploy any workspace you have access to in your organization, provided the location of the workspace is one of the locations supported by the registry. This capability is helpful if you trained a model in a `dev` workspace and now need to deploy the model to `test` or `prod` workspace, while preserving the lineage information around the code, environment and data used to train the model.

Online endpoints let you deploy models and submit inference requests through the REST APIs. For more information, see How to deploy and score a machine learning model by using an online endpoint.

Azure CLI

Create an online endpoint.

Azure CLI

```
az ml online-endpoint create --name reg-ep-1234
```

Update the `model:` line `deploy.yml` available in the `cli/jobs/pipelines-with-components/nyc_taxi_data_regression` folder to refer the model name and version from the pervious step. Create an online deployment to the online endpoint. The `deploy.yml` is shown below for reference.

YAML

```yaml
$schema:
https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.sche
ma.json
name: demo
endpoint_name: reg-ep-1234
model: azureml://registries/<registry-name>/models/nyc-taxi-
model/versions/1
instance_type: Standard_DS2_v2
instance_count: 1
```

Create the online deployment. The deployment takes several minutes to complete.

Azure CLI

```azurecli
az ml online-deployment create --file deploy.yml --all-traffic
```

Fetch the scoring URI and submit a sample scoring request. Sample data for the scoring request is available in the `scoring-data.json` in the `cli/jobs/pipelines-with-components/nyc_taxi_data_regression` folder.

Azure CLI

```azurecli
ENDPOINT_KEY=$(az ml online-endpoint get-credentials -n reg-ep-1234 -o
tsv --query primaryKey)
SCORING_URI=$(az ml online-endpoint show -n $ep_name -o tsv --query
scoring_uri)
curl --request POST "$SCORING_URI" --header "Authorization: Bearer
$ENDPOINT_KEY" --header 'Content-Type: application/json' --data
@./scoring-data.json
```

Tip

- `curl` command works only on Linux.
- If you have not configured the default workspace and resource group as explained in the prerequisites section, you will need to specify the `--workspace-name` and `--resource-group` parameters for the `az ml online-endpoint` and `az ml online-deployment` commands to work.

Clean up resources

If you aren't going use the deployment, you should delete it to reduce costs. The following example deletes the endpoint and all the underlying deployments:

Azure CLI

Azure CLI

```
az ml online-endpoint delete --name reg-ep-1234 --yes --no-wait
```

Next steps

- How to share data assets using registries
- How to create and manage registries
- How to manage environments
- How to train models
- How to create pipelines using components

Share data across workspaces with registries (preview)

Article • 03/31/2023

Azure Machine Learning registry enables you to collaborate across workspaces within your organization. Using registries, you can share models, components, environments and data. Sharing data with registries is currently a preview feature. In this article, you learn how to:

- Create a data asset in the registry.
- Share an existing data asset from workspace to registry
- Use the data asset from registry as input to a model training job in a workspace.

> **Important**
>
> This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see **Supplemental Terms of Use for Microsoft Azure Previews**.

Key scenario addressed by data sharing using Azure Machine Learning registry

You may want to have data shared across multiple teams, projects, or workspaces in a central location. Such data doesn't have sensitive access controls and can be broadly used in the organization.

Examples include:

- A team wants to share a public dataset that is preprocessed and ready to use in experiments.
- Your organization has acquired a particular dataset for a project from an external vendor and wants to make it available to all teams working on a project.
- A team wants to share data assets across workspaces in different regions.

In these scenarios, you can create a data asset in a registry or share an existing data asset from a workspace to a registry. This data asset can then be used across multiple workspaces.

Scenarios NOT addressed by data sharing using Azure Machine Learning registry

- Sharing sensitive data that requires fine grained access control. You can't create a data asset in a registry to share with a small subset of users/workspaces while the registry is accessible by many other users in the org.
- Sharing data that is available in existing storage that must not be copied or is too large or too expensive to be copied. Whenever data assets are created in a registry, a copy of data is ingested into the registry storage so that it can be replicated.

Data asset types supported by Azure Machine Learning registry

> **Tip**
>
> Check out the following **canonical scenarios** when deciding if you want to use `uri_file`, `uri_folder`, or `mltable` for your scenario.

You can create three data asset types:

Type	V2 API	Canonical scenario
File: Reference a single file	`uri_file`	Read/write a single file - the file can have any format.
Folder: Reference a single folder	`uri_folder`	You must read/write a directory of parquet/CSV files into Pandas/Spark. Deep-learning with images, text, audio, video files located in a directory.
Table: Reference a data table	`mltable`	You have a complex schema subject to frequent changes, or you need a subset of large tabular data.

Paths supported by Azure Machine Learning registry

When you create a data asset, you must specify a **path** parameter that points to the data location. Currently, the only supported paths are to locations on your local computer.

> **Tip**

"Local" means the local storage for the computer you are using. For example, if you're using a laptop, the local drive. If an Azure Machine Learning compute instance, the "local" drive of the compute instance.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- Familiarity with Azure Machine Learning registries and Data concepts in Azure Machine Learning.
- An Azure Machine Learning registry to share data. To create a registry, see Learn how to create a registry.
- An Azure Machine Learning workspace. If you don't have one, use the steps in the Quickstart: Create workspace resources article to create one.

  > **ⓘ Important**
  >
  > The Azure region (location) where you create your workspace must be in the list of supported regions for Azure Machine Learning registry.

- The *environment* and *component* created from the How to share models, components, and environments article.
- The Azure CLI and the `ml` extension **or** the Azure Machine Learning Python SDK v2:

  Azure CLI

  To install the Azure CLI and extension, see Install, set up, and use the CLI (v2).

  > **ⓘ Important**
  >
  > - The CLI examples in this article assume that you are using the Bash (or compatible) shell. For example, from a Linux system or **Windows Subsystem for Linux**.
  > - The examples also assume that you have configured defaults for the Azure CLI so that you don't have to specify the parameters for your subscription, workspace, resource group, or location. To set default

settings, use the following commands. Replace the following parameters with the values for your configuration:

- Replace `<subscription>` with your Azure subscription ID.
- Replace `<workspace>` with your Azure Machine Learning workspace name.
- Replace `<resource-group>` with the Azure resource group that contains your workspace.
- Replace `<location>` with the Azure region that contains your workspace.

Azure CLI

```
az account set --subscription <subscription>
az configure --defaults workspace=<workspace> group=
<resource-group> location=<location>
```

You can see what your current defaults are by using the `az configure -l` command.

Clone examples repository

The code examples in this article are based on the `nyc_taxi_data_regression` sample in the examples repository. To use these files on your development environment, use the following commands to clone the repository and change directories to the example:

Bash

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

Azure CLI

For the CLI example, change directories to `cli/jobs/pipelines-with-components/nyc_taxi_data_regression` in your local clone of the examples repository.

Bash

```
cd cli/jobs/pipelines-with-components/nyc_taxi_data_regression
```

Create SDK connection

> **Tip**
>
> This step is only needed when using the Python SDK.

Create a client connection to both the Azure Machine Learning workspace and registry. In the following example, replace the `<...>` placeholder values with the values appropriate for your configuration. For example, your Azure subscription ID, workspace name, registry name, etc.:

Python

```
ml_client_workspace = MLClient( credential=credential,
    subscription_id = "<workspace-subscription>",
    resource_group_name = "<workspace-resource-group",
    workspace_name = "<workspace-name>")
print(ml_client_workspace)

ml_client_registry = MLClient(credential=credential,
                        registry_name="<REGISTRY_NAME>",
                        registry_location="<REGISTRY_REGION>")
print(ml_client_registry)
```

Create data in registry

The data asset created in this step is used later in this article when submitting a training job.

Azure CLI

> **Tip**
>
> The same CLI command `az ml data create` can be used to create data in a workspace or registry. Running the command with `--workspace-name` command creates the data in a workspace whereas running the command with `--registry-name` creates the data in the registry.

The data source is located in the examples repository that you cloned earlier. Under the local clone, go to the following directory path: `cli/jobs/pipelines-with-`

`components/nyc_taxi_data_regression`. In this directory, create a YAML file named `data-registry.yml` and use the following YAML as the contents of the file:

YAML

```yaml
$schema: https://azuremlschemas.azureedge.net/latest/data.schema.json
name: transformed-nyc-taxt-data
description: Transformed NYC Taxi data created from local folder.
version: 1
type: uri_folder
path: data_transformed/
```

The `path` value points to the `data_transformed` subdirectory, which contains the data that is shared using the registry.

To create the data in the registry, use the `az ml data create`. In the following examples, replace `<registry-name>` with the name of your registry.

Azure CLI

```azurecli
az ml data create --file data-registry.yml --registry-name <registry-name>
```

If you get an error that data with this name and version already exists in the registry, you can either edit the `version` field in `data-registry.yml` or specify a different version on the CLI that overrides the version value in `data-registry.yml`.

Azure CLI

```azurecli
# use shell epoch time as the version
version=$(date +%s)
az ml data create --file data-registry.yml --registry-name <registry-name> --set version=$version
```

Tip

If the `version=$(date +%s)` command doesn't set the `$version` variable in your environment, replace `$version` with a random number.

Save the `name` and `version` of the data from the output of the `az ml data create` command and use them with `az ml data show` command to view details for the asset.

Azure CLI

```
az ml data show --name transformed-nyc-taxt-data --version 1 --registry-name <registry-name>
```

> **Tip**
>
> If you used a different data name or version, replace the `--name` and `--version` parameters accordingly.

You can also use `az ml data list --registry-name <registry-name>` to list all data assets in the registry.

Create an environment and component in registry

To create an environment and component in the registry, use the steps in the How to share models, components, and environments article. The environment and component are used in the training job in next section.

> **Tip**
>
> You can use an environment and component from the workspace instead of using ones from the registry.

Run a pipeline job in a workspace using component from registry

When running a pipeline job that uses a component and data from a registry, the *compute* resources are local to the workspace. In the following example, the job uses the Scikit Learn training component and the data asset created in the previous sections to train a model.

> **Note**
>
> The key aspect is that this pipeline is going to run in a workspace using training data that isn't in the specific workspace. The data is in a registry that can be used

with any workspace in your organization. You can run this training job in any workspace you have access to without having worry about making the training data available in that workspace.

Azure CLI

Verify that you are in the `cli/jobs/pipelines-with-components/nyc_taxi_data_regression` directory. Edit the `component` section in under the `train_job` section of the `single-job-pipeline.yml` file to refer to the training component and `path` under `training_data` section to refer to data asset created in the previous sections. The following example shows what the `single-job-pipeline.yml` looks like after editing. Replace the `<registry_name>` with the name for your registry:

YAML

```yaml
$schema:
https://azuremlschemas.azureedge.net/latest/pipelineJob.schema.json
type: pipeline
display_name: nyc_taxi_data_regression_single_job
description: Single job pipeline to train regression model based on nyc
taxi dataset

jobs:
  train_job:
    type: command
    component: azureml://registries/<registry-
name>/component/train_linear_regression_model/versions/1
    compute: azureml:cpu-cluster
    inputs:
      training_data:
        type: uri_folder
        path: azureml://registries/<registry-name>/data/transformed-nyc-
taxt-data/versions/1
    outputs:
      model_output:
        type: mlflow_model
      test_data:
```

⚠ Warning

- Before running the pipeline job, confirm that the workspace in which you will run the job is in a Azure region that is supported by the registry in which you created the data.

- Confirm that the workspace has a compute cluster with the name `cpu-cluster` or edit the `compute` field under `jobs.train_job.compute` with the name of your compute.

Run the pipeline job with the `az ml job create` command.

Azure CLI

```
az ml job create --file single-job-pipeline.yml
```

Tip

If you have not configured the default workspace and resource group as explained in the prerequisites section, you will need to specify the `--workspace-name` and `--resource-group` parameters for the `az ml job create` to work.

For more information on running jobs, see the following articles:

- Running jobs (CLI)
- Pipeline jobs with components (CLI)

Share data from workspace to registry

The following steps show how to share an existing data asset from a workspace to a registry.

Azure CLI

First, create a data asset in the workspace. Make sure that you are in the `cli/assets/data` directory. The `local-folder.yml` located in this directory is used to create a data asset in the workspace. The data specified in this file is available in the `cli/assets/data/sample-data` directory. The following YAML is the contents of the `local-folder.yml` file:

YAML

```
$schema: https://azuremlschemas.azureedge.net/latest/data.schema.json
name: local-folder-example-titanic
description: Dataset created from local folder.
```

```
type: uri_folder
path: sample-data/
```

To create the data asset in the workspace, use the following command:

Azure CLI

```
az ml data create -f local-folder.yml
```

For more information on creating data assets in a workspace, see How to create data assets.

The data asset created in the workspace can be shared to a registry. From the registry, it can be used in multiple workspaces. Note that we are passing `--share_with_name` and `--share_with_version` parameter in share function. These parameters are optional and if you do not pass these data will be shared with same name and version as in workspace.

The following example demonstrates using share command to share a data asset. Replace `<registry-name>` with the name of the registry that the data will be shared to.

Azure CLI

```
az ml data share --name local-folder-example-titanic --version <version-in-workspace> --share-with-name <name-in-registry> --share-with-version <version-in-registry> --registry-name <registry-name>
```

Next steps

- How to create and manage registries
- How to manage environments
- How to train models
- How to create pipelines using components

Endpoints for inference in production

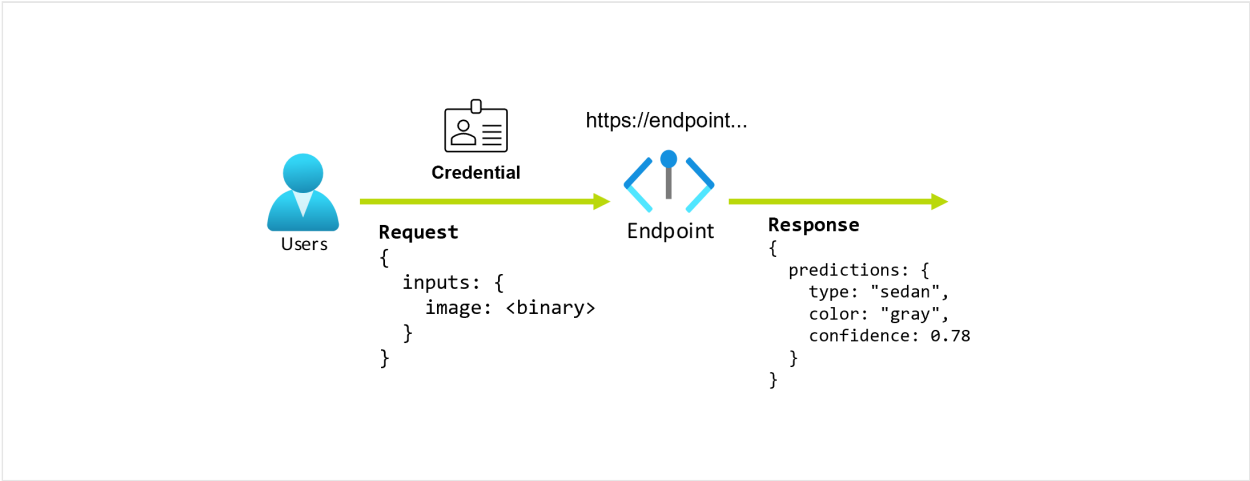
Article • 08/02/2023

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

After you train machine learning models or pipelines, you need to deploy them to production so that others can use them for *inference*. Inference is the process of applying new input data to the machine learning model or pipeline to generate outputs. While these outputs are typically referred to as "predictions," inferencing can be used to generate outputs for other machine learning tasks, such as classification and clustering. In Azure Machine Learning, you perform inferencing by using **endpoints and deployments**. Endpoints and deployments allow you to decouple the interface of your production workload from the implementation that serves it.

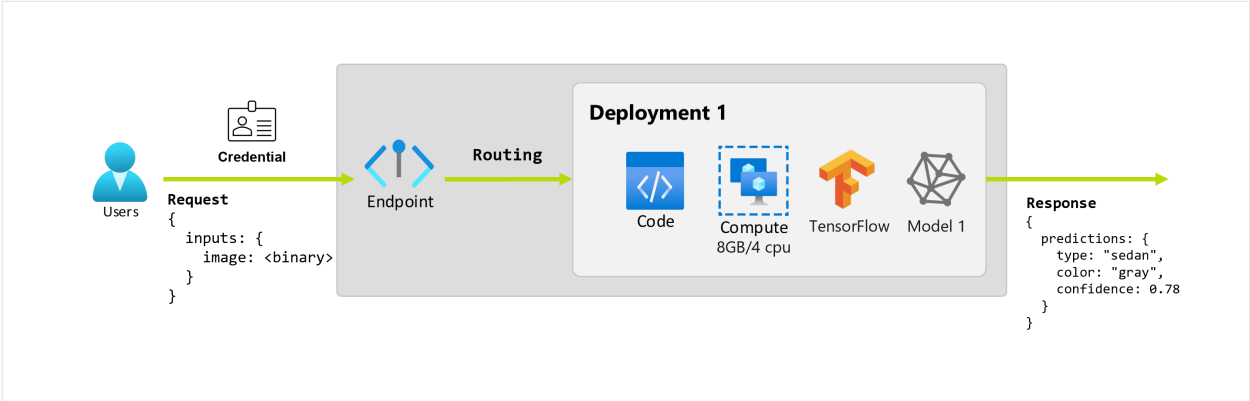
Intuition

Suppose you're working on an application that predicts the type and color of a car, given its photo. For this application, a user with certain credentials makes an HTTP request to a URL and provides a picture of a car as part of the request. In return, the user gets a response that includes the type and color of the car as string values. In this scenario, the URL serves as an **endpoint**.

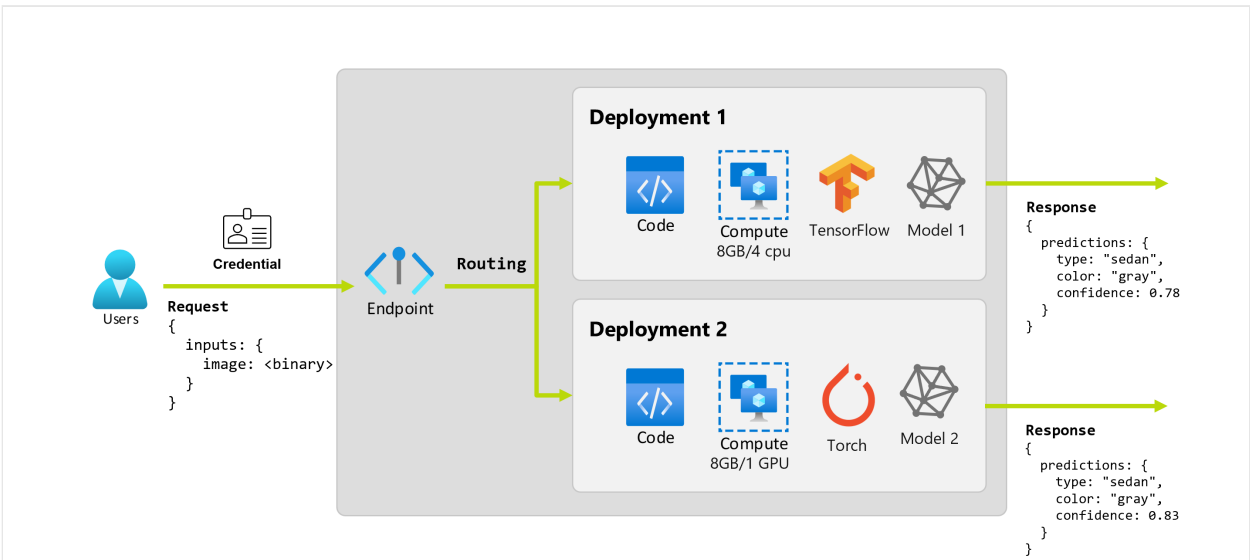


Furthermore, say that a data scientist, Alice, is working on implementing the application. Alice knows a lot about TensorFlow and decides to implement the model using a Keras sequential classifier with a RestNet architecture from the TensorFlow Hub. After testing the model, Alice is happy with its results and decides to use the model to solve the car prediction problem. The model is large in size and requires 8 GB of memory with 4 cores to run. In this scenario, Alice's model and the resources, such as the code and the

compute, that are required to run the model make up a **deployment under the endpoint**.



Finally, let's imagine that after a couple of months, the organization discovers that the application performs poorly on images with less than ideal illumination conditions. Bob, another data scientist, knows a lot about data augmentation techniques that help a model build robustness on that factor. However, Bob feels more comfortable using Torch to implement the model and trains a new model with Torch. Bob wants to try this model in production gradually until the organization is ready to retire the old model. The new model also shows better performance when deployed to GPU, so the deployment needs to include a GPU. In this scenario, Bob's model and the resources, such as the code and the compute, that are required to run the model make up **another deployment under the same endpoint**.



Endpoints and deployments

An **endpoint** is a stable and durable URL that can be used to request or invoke a model. You provide the required inputs to the endpoint and get the outputs back. An endpoint provides:

- a stable and durable URL (like *endpoint-name.region.inference.ml.azure.com*),
- an authentication mechanism, and
- an authorization mechanism.

A **deployment** is a set of resources and computes required for hosting the model or component that does the actual inferencing. A single endpoint can contain multiple deployments. These deployments can host independent assets and consume different resources based on the needs of the assets. Endpoints have a routing mechanism that can direct requests to specific deployments in the endpoint.

To function properly, **each endpoint must have at least one deployment**. Endpoints and deployments are independent Azure Resource Manager resources that appear in the Azure portal.

Online and batch endpoints

Azure Machine Learning allows you to implement online endpoints and batch endpoints. Online endpoints are designed for real-time inference—when you invoke the endpoint, the results are returned in the endpoint's response. Batch endpoints, on the other hand, are designed for long-running batch inference. Each time you invoke a batch endpoint you generate a batch job that performs the actual work.

When to use online vs batch endpoint for your use-case

Use online endpoints to operationalize models for real-time inference in synchronous low-latency requests. We recommend using them when:

- ✓ You have low-latency requirements.
- ✓ Your model can answer the request in a relatively short amount of time.
- ✓ Your model's inputs fit on the HTTP payload of the request.
- ✓ You need to scale up in terms of number of requests.

Use batch endpoints to operationalize models or pipelines (preview) for long-running asynchronous inference. We recommend using them when:

- ✓ You have expensive models or pipelines that require a longer time to run.
- ✓ You want to operationalize machine learning pipelines and reuse components.
- ✓ You need to perform inference over large amounts of data that are distributed in multiple files.
- ✓ You don't have low latency requirements.
- ✓ Your model's inputs are stored in a storage account or in an Azure Machine Learning data asset.

✔ You can take advantage of parallelization.

Comparison of online and batch endpoints

Both online and batch endpoints are based on the idea of endpoints and deployments, which help you transition easily from one to the other. However, when moving from one to another, there are some differences that are important to take into account. Some of these differences are due to the nature of the work:

Endpoints

The following table shows a summary of the different features available to online and batch endpoints.

Feature	Online Endpoints	Batch endpoints
Stable invocation URL	Yes	Yes
Support for multiple deployments	Yes	Yes
Deployment's routing	Traffic split	Switch to default
Mirror traffic for safe rollout	Yes	No
Swagger support	Yes	No
Authentication	Key and token	Azure AD
Private network support	Yes	Yes
Managed network isolation[1]	Yes	No
Customer-managed keys	Yes	No
Cost basis	None	None

[1] *Managed network isolation* allows you to manage the networking configuration of the endpoint independently of the configuration of the Azure Machine Learning workspace.

Deployments

The following table shows a summary of the different features available to online and batch endpoints at the deployment level. These concepts apply to each deployment under the endpoint.

Feature	Online Endpoints	Batch endpoints
Deployment types	Models	Models and Pipeline components (preview)
MLflow model deployment	Yes (requires public networking)	Yes
Custom model deployment	Yes, with scoring script	Yes, with scoring script
Inference server [1]	- Azure Machine Learning Inferencing Server - Triton - Custom (using BYOC)	Batch Inference
Compute resource consumed	Instances or granular resources	Cluster instances
Compute type	Managed compute and Kubernetes	Managed compute and Kubernetes
Low-priority compute	No	Yes
Scaling compute to zero	No	Yes
Autoscaling compute[2]	Yes, based on resources' load	Yes, based on job count
Overcapacity management	Throttling	Queuing
Cost basis[3]	Per deployment: compute instances running	Per job: compute instanced consumed in the job (capped to the maximum number of instances of the cluster).
Local testing of deployments	Yes	No

[1] *Inference server* refers to the serving technology that takes requests, processes them, and creates responses. The inference server also dictates the format of the input and the expected outputs.

[2] *Autoscaling* is the ability to dynamically scale up or scale down the deployment's allocated resources based on its load. Online and batch deployments use different strategies for autoscaling. While online deployments scale up and down based on the resource utilization (like CPU, memory, requests, etc.), batch endpoints scale up or down based on the number of jobs created.

[3] Both online and batch deployments charge by the resources consumed. In online deployments, resources are provisioned at deployment time. However, in batch deployment, no resources are consumed at deployment time but when the job runs. Hence, there is no cost associated with the deployment itself. Notice that queued jobs do not consume resources either.

Developer interfaces

Endpoints are designed to help organizations operationalize production-level workloads in Azure Machine Learning. Endpoints are robust and scalable resources and they provide the best of the capabilities to implement MLOps workflows.

You can create and manage batch and online endpoints with multiple developer tools:

- The Azure CLI and the Python SDK
- Azure Resource Manager/REST API
- Azure Machine Learning studio web portal
- Azure portal (IT/Admin)
- Support for CI/CD MLOps pipelines using the Azure CLI interface & REST/ARM interfaces

Next steps

- How to deploy online endpoints with the Azure CLI and Python SDK
- How to deploy models with batch endpoints
- How to deploy pipelines with batch endpoints (preview)
- How to use online endpoints with the studio
- How to monitor managed online endpoints
- Manage and increase quotas for resources with Azure Machine Learning

Schedule machine learning pipeline jobs

Article • 03/31/2023

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

In this article, you'll learn how to programmatically schedule a pipeline to run on Azure and use the schedule UI to do the same. You can create a schedule based on elapsed time. Time-based schedules can be used to take care of routine tasks, such as retrain models or do batch predictions regularly to keep them up-to-date. After learning how to create schedules, you'll learn how to retrieve, update and deactivate them via CLI, SDK, and studio UI.

Prerequisites

- You must have an Azure subscription to use Azure Machine Learning. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning today.

Azure CLI

- Install the Azure CLI and the `ml` extension. Follow the installation steps in Install, set up, and use the CLI (v2).
- Create an Azure Machine Learning workspace if you don't have one. For workspace creation, see Install, set up, and use the CLI (v2).

Schedule a pipeline job

To run a pipeline job on a recurring basis, you'll need to create a schedule. A `Schedule` associates a job, and a trigger. The trigger can either be `cron` that use cron expression to describe the wait between runs or `recurrence` that specify using what frequency to trigger job. In each case, you need to define a pipeline job first, it can be existing pipeline jobs or a pipeline job define inline, refer to Create a pipeline job in CLI and Create a pipeline job in SDK.

You can schedule a pipeline job yaml in local or an existing pipeline job in workspace.

Create a schedule

Create a time-based schedule with recurrence pattern

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

YAML

```yaml
$schema:
https://azuremlschemas.azureedge.net/latest/schedule.schema.json
name: simple_recurrence_job_schedule
display_name: Simple recurrence job schedule
description: a simple hourly recurrence job schedule

trigger:
  type: recurrence
  frequency: day #can be minute, hour, day, week, month
  interval: 1 #every day
  schedule:
    hours: [4,5,10,11,12]
    minutes: [0,30]
  start_time: "2022-07-10T10:00:00" # optional - default will be
schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

create_job: ./simple-pipeline-job.yml
# create_job: azureml:simple-pipeline-job
```

`trigger` contains the following properties:

- **(Required)** `type` specifies the schedule type is `recurrence`. It can also be `cron`, see details in the next section.

List continues below.

> ⓘ **Note**
>
> The following properties that need to be specified apply for CLI and SDK.

- **(Required)** `frequency` specifies the unit of time that describes how often the schedule fires. Can be `minute`, `hour`, `day`, `week`, `month`.

- **(Required)** `interval` specifies how often the schedule fires based on the frequency, which is the number of time units to wait until the schedule fires again.
- (Optional) `schedule` defines the recurrence pattern, containing `hours`, `minutes`, and `weekdays`.
  - When `frequency` is `day`, pattern can specify `hours` and `minutes`.
  - When `frequency` is `week` and `month`, pattern can specify `hours`, `minutes` and `weekdays`.
  - `hours` should be an integer or a list, from 0 to 23.
  - `minutes` should be an integer or a list, from 0 to 59.
  - `weekdays` can be a string or list from `monday` to `sunday`.
  - If `schedule` is omitted, the job(s) will be triggered according to the logic of `start_time`, `frequency` and `interval`.
- (Optional) `start_time` describes the start date and time with timezone. If `start_time` is omitted, start_time will be equal to the job created time. If the start time is in the past, the first job will run at the next calculated run time.
- (Optional) `end_time` describes the end date and time with timezone. If `end_time` is omitted, the schedule will continue trigger jobs until the schedule is manually disabled.
- (Optional) `time_zone` specifies the time zone of the recurrence. If omitted, by default is UTC. To learn more about timezone values, see appendix for timezone values.

Create a time-based schedule with cron expression

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

YAML

```yaml
$schema:
https://azuremlschemas.azureedge.net/latest/schedule.schema.json
name: simple_cron_job_schedule
display_name: Simple cron job schedule
description: a simple hourly cron job schedule

trigger:
  type: cron
  expression: "0 * * * *"
  start_time: "2022-07-10T10:00:00" # optional - default will be
```

```
schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

# create_job: azureml:simple-pipeline-job
create_job: ./simple-pipeline-job.yml
```

The `trigger` section defines the schedule details and contains following properties:

- **(Required)** `type` specifies the schedule type is `cron`.

List continues below.

- **(Required)** `expression` uses standard crontab expression to express a recurring schedule. A single expression is composed of five space-delimited fields:

  `MINUTES HOURS DAYS MONTHS DAYS-OF-WEEK`

  - A single wildcard (`*`), which covers all values for the field. So a `*` in days means all days of a month (which varies with month and year).
  - The `expression: "15 16 * * 1"` in the sample above means the 16:15PM on every Monday.
  - The table below lists the valid values for each field:

Field	Range	Comment
`MINUTES`	0-59	-
`HOURS`	0-23	-
`DAYS`	-	Not supported. The value will be ignored and treat as `*`.
`MONTHS`	-	Not supported. The value will be ignored and treat as `*`.
`DAYS-OF-WEEK`	0-6	Zero (0) means Sunday. Names of days also accepted.

  - To learn more about how to use crontab expression, see Crontab Expression wiki on GitHub.

  Important

  `DAYS` and `MONTH` are not supported. If you pass a value, it will be ignored and treat as `*`.

- (Optional) `start_time` specifies the start date and time with timezone of the schedule. `start_time: "2022-05-10T10:15:00-04:00"` means the schedule starts from 10:15:00AM on 2022-05-10 in UTC-4 timezone. If `start_time` is omitted, the `start_time` will be equal to schedule creation time. If the start time is in the past, the first job will run at the next calculated run time.
- (Optional) `end_time` describes the end date and time with timezone. If `end_time` is omitted, the schedule will continue trigger jobs until the schedule is manually disabled.
- (Optional) `time_zone` specifies the time zone of the expression. If omitted, by default is UTC. See appendix for timezone values.

Limitations:

- Currently Azure Machine Learning v2 schedule doesn't support event-based trigger.
- You can specify complex recurrence pattern containing multiple trigger timestamps using Azure Machine Learning SDK/CLI v2, while UI only displays the complex pattern and doesn't support editing.
- If you set the recurrence as the 31st day of every month, in months with less than 31 days, the schedule won't trigger jobs.

Change runtime settings when defining schedule

When defining a schedule using an existing job, you can change the runtime settings of the job. Using this approach, you can define multi-schedules using the same job with different inputs.

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

YAML

```yaml
$schema:
https://azuremlschemas.azureedge.net/latest/schedule.schema.json
name: cron_with_settings_job_schedule
display_name: Simple cron job schedule
description: a simple hourly cron job schedule

trigger:
  type: cron
  expression: "0 * * * *"
  start_time: "2022-07-10T10:00:00" # optional - default will be
```

```
schedule creation time
  time_zone: "Pacific Standard Time" # optional - default will be UTC

create_job:
  type: pipeline
  job: ./simple-pipeline-job.yml
  # job: azureml:simple-pipeline-job
  # runtime settings
  settings:
    #default_compute: azureml:cpu-cluster
    continue_on_step_failure: true
  inputs:
    hello_string_top_level_input: ${{name}}
  tags:
    schedule: cron_with_settings_schedule
```

Following properties can be changed when defining schedule:

Property	Description
settings	A dictionary of settings to be used when running the pipeline job.
inputs	A dictionary of inputs to be used when running the pipeline job.
outputs	A dictionary of inputs to be used when running the pipeline job.
experiment_name	Experiment name of triggered job.

> ⓘ Note
>
> Studio UI users can only modify input, output, and runtime settings when creating a schedule. `experiment_name` can only be changed using the CLI or SDK.

Expressions supported in schedule

When define schedule, we support following expression that will be resolved to real value during job runtime.

Expression	Description	Supported properties
`${{creation_context.trigger_time}}`	The time when the schedule is triggered.	String type inputs of pipeline job
`${{name}}`	The name of job.	outputs.path of pipeline job

Manage schedule

Create schedule

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

After you create the schedule yaml, you can use the following command to create a schedule via CLI.

Azure CLI

```
# This action will create related resources for a schedule. It will take dozens of seconds to complete.
az ml schedule create --file cron-schedule.yml --no-wait
```

List schedules in a workspace

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

Azure CLI

```
az ml schedule list
```

Check schedule detail

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

Azure CLI

```
az ml schedule show -n simple_cron_job_schedule
```

Update a schedule

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

Azure CLI

```
az ml schedule update -n simple_cron_job_schedule  --set
description="new description" --no-wait
```

> **Note**
>
> If you would like to update more than just tags/description, it is recomend to use `az ml schedule create --file update_schedule.yml`

Disable a schedule

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

Azure CLI

```
az ml schedule disable -n simple_cron_job_schedule --no-wait
```

Enable a schedule

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

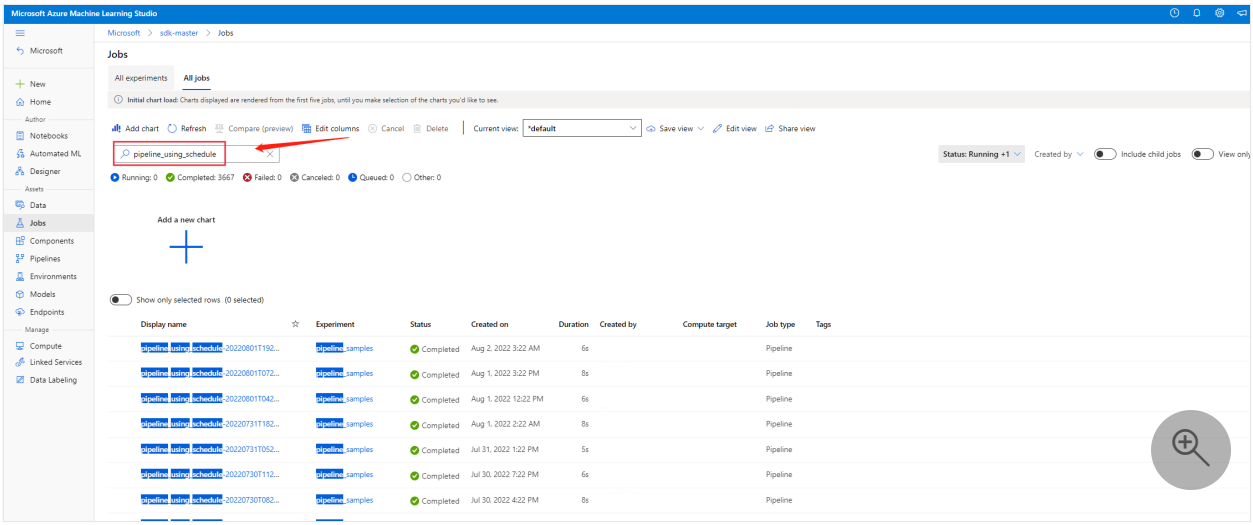
Azure CLI

```
az ml schedule enable -n simple_cron_job_schedule --no-wait
```

Query triggered jobs from a schedule

All the display name of jobs triggered by schedule will have the display name as <schedule_name>-YYYYMMDDThhmmssZ. For example, if a schedule with a name of named-schedule is created with a scheduled run every 12 hours starting at 6 AM on Jan 1 2021, then the display names of the jobs created will be as follows:

- named-schedule-20210101T060000Z
- named-schedule-20210101T180000Z
- named-schedule-20210102T060000Z
- named-schedule-20210102T180000Z, and so on



You can also apply Azure CLI JMESPath query to query the jobs triggered by a schedule name.

Azure CLI

```
# query triggered jobs from schedule, please replace the
simple_cron_job_schedule to your schedule name
az ml job list --query "[?contains(display_name,'simple_cron_schedule')]"
```

> **Note**
>
> For a simpler way to find all jobs triggered by a schedule, see the *Jobs history* on the *schedule detail page* using the studio UI.

Delete a schedule

> **Important**
>
> A schedule must be disabled to be deleted. Delete is an unrecoverable action. After a schedule is deleted, you can never access or recover it.

Azure CLI

APPLIES TO: Azure CLI ml extension **v2 (current)**

Azure CLI

```
az ml schedule delete -n simple_cron_job_schedule
```

RBAC (Role-based-access-control) support

Since schedules are usually used for production, to reduce impact of misoperation, workspace admins may want to restrict access to creating and managing schedules within a workspace.

Currently there are three action rules related to schedules and you can configure in Azure portal. You can learn more details about how to manage access to an Azure Machine Learning workspace.

Action	Description	Rule
Read	Get and list schedules in Machine Learning workspace	Microsoft.MachineLearningServices/workspaces/schedules/read
Write	Create, update, disable and enable schedules in Machine Learning workspace	Microsoft.MachineLearningServices/workspaces/schedules/write
Delete	Delete a schedule in Machine Learning workspace	Microsoft.MachineLearningServices/workspaces/schedules/delete

Frequently asked questions

- Why my schedules created by SDK aren't listed in UI?

  The schedules UI is for v2 schedules. Hence, your v1 schedules won't be listed or accessed via UI.

However, v2 schedules also support v1 pipeline jobs. You don't have to publish pipeline first, and you can directly set up schedules for a pipeline job.

- Why my schedules don't trigger job at the time I set before?
  - By default schedules will use UTC timezone to calculate trigger time. You can specify timezone in the creation wizard, or update timezone in schedule detail page.
  - If you set the recurrence as the 31st day of every month, in months with less than 31 days, the schedule won't trigger jobs.
  - If you're using cron expressions, MONTH isn't supported. If you pass a value, it will be ignored and treated as *. This is a known limitation.

- Are event-based schedules supported?
  - No, V2 schedule does not support event-based schedules.

Next steps

- Learn more about the CLI (v2) schedule YAML schema.
- Learn how to create pipeline job in CLI v2.
- Learn how to create pipeline job in SDK v2.
- Learn more about CLI (v2) core YAML syntax.
- Learn more about Pipelines.
- Learn more about Component.

Use Azure Pipelines with Azure Machine Learning

Article • 09/29/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

You can use an Azure DevOps pipeline to automate the machine learning lifecycle. Some of the operations you can automate are:

- Data preparation (extract, transform, load operations)
- Training machine learning models with on-demand scale-out and scale-up
- Deployment of machine learning models as public or private web services
- Monitoring deployed machine learning models (such as for performance or data-drift analysis)

This article teaches you how to create an Azure Pipeline that builds and deploys a machine learning model to Azure Machine Learning.

This tutorial uses Azure Machine Learning Python SDK v2 and Azure CLI ML extension v2.

Prerequisites

- Complete the Create resources to get started to:
  - Create a workspace
- Create a cloud-based compute cluster to use for training your model
- Azure Machine Learning extension for Azure Pipelines. This extension can be installed from the Visual Studio marketplace at https://marketplace.visualstudio.com/items?itemName=ms-air-aiagility.azureml-v2 .

Step 1: Get the code

Fork the following repo at GitHub:

```
https://github.com/azure/azureml-examples
```

Step 2: Sign in to Azure Pipelines

Sign-in to Azure Pipelines. After you sign in, your browser goes to `https://dev.azure.com/my-organization-name` and displays your Azure DevOps dashboard.

Within your selected organization, create a *project*. If you don't have any projects in your organization, you see a **Create a project to get started** screen. Otherwise, select the **New Project** button in the upper-right corner of the dashboard.

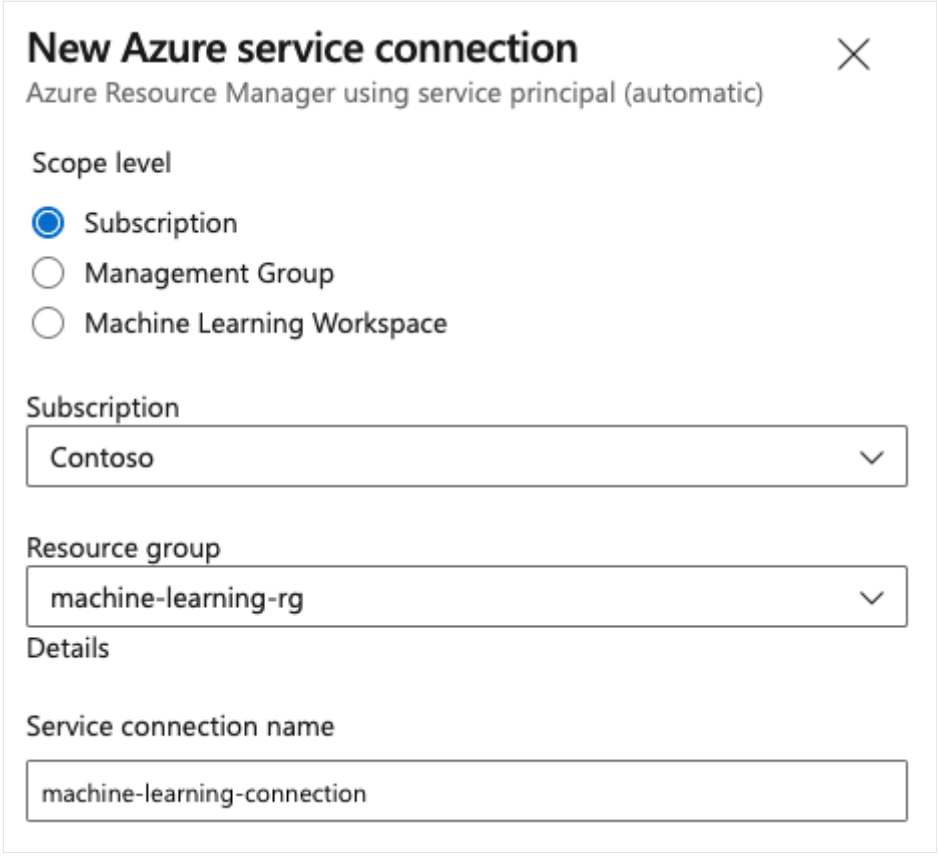
Step 3: Create a service connection

You can use an existing service connection.

Azure Resource Manager

You need an Azure Resource Manager connection to authenticate with Azure portal.

1. In Azure DevOps, select **Project Settings** and open the **Service connections** page.
2. Choose **+ New service connection** and select **Azure Resource Manager**.
3. Select the default authentication method, **Service principal (automatic)**.
4. Create your service connection. Set your preferred scope level, subscription, resource group, and connection name.



Step 4: Create a pipeline

1. Go to **Pipelines**, and then select **New pipeline**.
2. Do the steps of the wizard by first selecting **GitHub** as the location of your source code.
3. You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.
4. When you see the list of repositories, select your repository.
5. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.
6. Select the **Starter pipeline**. You'll update the starter pipeline template.

Step 5: Build your YAML pipeline to submit the Azure Machine Learning job

Delete the starter pipeline and replace it with the following YAML code. In this pipeline, you'll:

- Use the Python version task to set up Python 3.8 and install the SDK requirements.
- Use the Bash task to run bash scripts for the Azure Machine Learning SDK and CLI.
- Use the Azure CLI task to submit an Azure Machine Learning job.

Select the following tabs depending on whether you're using an Azure Resource Manager service connection or a generic service connection. In the pipeline YAML, replace the value of variables with your resources.

Using Azure Resource Manager service connection

YAML

```yaml
name: submit-azure-machine-learning-job

trigger:
- none

variables:
  service-connection: 'machine-learning-connection' # replace with your
service connection name
  resource-group: 'machinelearning-rg' # replace with your resource
group name
  workspace: 'docs-ws' # replace with your workspace name

jobs:
- job: SubmitAzureMLJob
  displayName: Submit AzureML Job
  timeoutInMinutes: 300
  pool:
    vmImage: ubuntu-latest
  steps:
  - checkout: none
  - task: UsePythonVersion@0
    displayName: Use Python >=3.8
    inputs:
      versionSpec: '>=3.8'

  - bash: |
      set -ex

      az version
      az extension add -n ml
    displayName: 'Add AzureML Extension'

  - task: AzureCLI@2
    name: submit_azureml_job_task
    displayName: Submit AzureML Job Task
    inputs:
      azureSubscription: $(service-connection)
      workingDirectory: 'cli/jobs/pipelines-with-
components/nyc_taxi_data_regression'
```

```
      scriptLocation: inlineScript
      scriptType: bash
      inlineScript: |

      # submit component job and get the run name
      job_name=$(az ml job create --file single-job-pipeline.yml -g
$(resource-group) -w $(workspace) --query name --output tsv)

      # Set output variable for next task
      echo "##vso[task.setvariable
variable=JOB_NAME;isOutput=true;]$job_name"
```

Step 6: Wait for Azure Machine Learning job to complete

In step 5, you added a job to submit an Azure Machine Learning job. In this step, you add another job that waits for the Azure Machine Learning job to complete.

Using Azure Resource Manager service connection

If you're using an Azure Resource Manager service connection, you can use the "Machine Learning" extension. You can search this extension in the Azure DevOps extensions Marketplace or go directly to the extension. Install the "Machine Learning" extension.

> **Important**
>
> Don't install the **Machine Learning (classic)** extension by mistake; it's an older extension that doesn't provide the same functionality.

In the Pipeline review window, add a Server Job. In the steps part of the job, select **Show assistant** and search for **AzureML**. Select the **AzureML Job Wait** task and fill in the information for the job.

The task has four inputs: `Service Connection`, `Azure Resource Group Name`, `AzureML Workspace Name` and `AzureML Job Name`. Fill these inputs. The resulting YAML for these steps is similar to the following example:

> **Note**

- The Azure Machine Learning job wait task runs on a **server job**, which doesn't use up expensive agent pool resources and requires no additional charges. Server jobs (indicated by `pool: server`) run on the same machine as your pipeline. For more information, see **Server jobs**.
- One Azure Machine Learning job wait task can only wait on one job. You'll need to set up a separate task for each job that you want to wait on.
- The Azure Machine Learning job wait task can wait for a maximum of 2 days. This is a hard limit set by Azure DevOps Pipelines.

yml

```yml
- job: WaitForAzureMLJobCompletion
  displayName: Wait for AzureML Job Completion
  pool: server
  timeoutInMinutes: 0
  dependsOn: SubmitAzureMLJob
  variables:
    # We are saving the name of azureMl job submitted in previous step
to a variable and it will be used as an inut to the AzureML Job Wait
task
    azureml_job_name_from_submit_job: $[
dependencies.SubmitAzureMLJob.outputs['submit_azureml_job_task.AZUREML_J
OB_NAME'] ]
  steps:
  - task: AzureMLJobWaitTask@1
    inputs:
      serviceConnection: $(service-connection)
      resourceGroupName: $(resource-group)
      azureMLWorkspaceName: $(workspace)
      azureMLJobName: $(azureml_job_name_from_submit_job)
```

Step 7: Submit pipeline and verify your pipeline run

Select **Save and run**. The pipeline will wait for the Azure Machine Learning job to complete, and end the task under `WaitForJobCompletion` with the same status as the Azure Machine Learning job. For example: Azure Machine Learning job `Succeeded` == Azure DevOps Task under `WaitForJobCompletion` job `Succeeded` Azure Machine Learning job `Failed` == Azure DevOps Task under `WaitForJobCompletion` job `Failed` Azure

Machine Learning job `Cancelled` == Azure DevOps Task under `WaitForJobCompletion` job `Cancelled`

> **Tip**
>
> You can view the complete Azure Machine Learning job in **Azure Machine Learning studio**.

Clean up resources

If you're not going to continue to use your pipeline, delete your Azure DevOps project. In Azure portal, delete your resource group and Azure Machine Learning instance.

Use GitHub Actions with Azure Machine Learning

Article • 02/24/2023

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

Get started with GitHub Actions to train a model on Azure Machine Learning.

This article will teach you how to create a GitHub Actions workflow that builds and deploys a machine learning model to Azure Machine Learning. You'll train a scikit-learn linear regression model on the NYC Taxi dataset.

GitHub Actions uses a workflow YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

Prerequisites

Before following the steps in this article, make sure you have the following prerequisites:

- An Azure Machine Learning workspace. If you don't have one, use the steps in the Quickstart: Create workspace resources article to create one.
- To install the Python SDK v2, use the following command:

  ```
  Bash

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

  To update an existing installation of the SDK to the latest version, use the following command:

  ```
  Bash

  pip install --upgrade azure-ai-ml azure-identity
  ```

  For more information, see Install the Python SDK v2 for Azure Machine Learning.

- A GitHub account. If you don't have one, sign up for free.

Step 1. Get the code

Fork the following repo at GitHub:

```
https://github.com/azure/azureml-examples
```

Step 2. Authenticate with Azure

You'll need to first define how to authenticate with Azure. You can use a service principal or OpenID Connect.

Generate deployment credentials

Service principal

Create a service principal with the az ad sp create-for-rbac command in the Azure CLI. Run this command with Azure Cloud Shell in the Azure portal or by selecting the **Try it** button.

Azure CLI

```
az ad sp create-for-rbac --name "myML" --role contributor \
                         --scopes /subscriptions/<subscription-
id>/resourceGroups/<group-name> \
                         --sdk-auth
```

In the example above, replace the placeholders with your subscription ID, resource group name, and app name. The output is a JSON object with the role assignment credentials that provide access to your App Service app similar to below. Copy this JSON object for later.

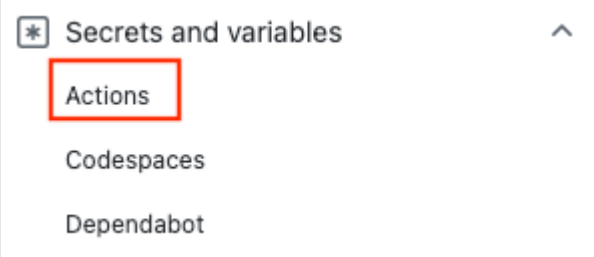
Output

```
{
  "clientId": "<GUID>",
  "clientSecret": "<GUID>",
  "subscriptionId": "<GUID>",
  "tenantId": "<GUID>",
  (...)
}
```

Create secrets

Service principal

1. In GitHub, go to your repository.
2. Go to **Settings** in the navigation menu.
3. Select **Security** > **Secrets and variables** > **Actions**.



4. Select **New repository secret**.
5. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.
6. Select **Add secret**.

Step 3. Update `setup.sh` to connect to your Azure Machine Learning workspace

You'll need to update the CLI setup file variables to match your workspace.

1. In your cloned repository, go to `azureml-examples/cli/`.
2. Edit `setup.sh` and update these variables in the file.

Variable	Description
GROUP	Name of resource group
LOCATION	Location of your workspace (example: `eastus2`)
WORKSPACE	Name of Azure Machine Learning workspace

Step 4. Update `pipeline.yml` with your compute cluster name

You'll use a `pipeline.yml` file to deploy your Azure Machine Learning pipeline. This is a machine learning pipeline and not a DevOps pipeline. You only need to make this update if you're using a name other than `cpu-cluster` for your computer cluster name.

1. In your cloned repository, go to `azureml-examples/cli/jobs/pipelines/nyc-taxi/pipeline.yml`.
2. Each time you see `compute: azureml:cpu-cluster`, update the value of `cpu-cluster` with your compute cluster name. For example, if your cluster is named `my-cluster`, your new value would be `azureml:my-cluster`. There are five updates.

Step 5: Run your GitHub Actions workflow

Your workflow authenticates with Azure, sets up the Azure Machine Learning CLI, and uses the CLI to train a model in Azure Machine Learning.

Service principal

Your workflow file is made up of a trigger section and jobs:

- A trigger starts the workflow in the `on` section. The workflow runs by default on a cron schedule and when a pull request is made from matching branches and paths. Learn more about events that trigger workflows.
- In the jobs section of the workflow, you checkout code and log into Azure with your service principal secret.
- The jobs section also includes a setup action that installs and sets up the Machine Learning CLI (v2). Once the CLI is installed, the run job action runs your Azure Machine Learning `pipeline.yml` file to train a model with NYC taxi data.

Enable your workflow

1. In your cloned repository, open `.github/workflows/cli-jobs-pipelines-nyc-taxi-pipeline.yml` and verify that your workflow looks like this.

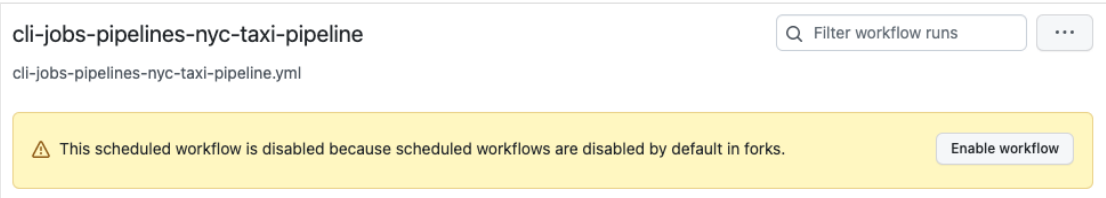
   YAML

```yaml
name: cli-jobs-pipelines-nyc-taxi-pipeline
on:
  workflow_dispatch:
  schedule:
    - cron: "0 0/4 * * *"
  pull_request:
    branches:
      - main
      - sdk-preview
    paths:
      - cli/jobs/pipelines/nyc-taxi/**
      - .github/workflows/cli-jobs-pipelines-nyc-taxi-pipeline.yml
      - cli/run-pipeline-jobs.sh
      - cli/setup.sh
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: check out repo
      uses: actions/checkout@v2
    - name: azure login
      uses: azure/login@v1
      with:
        creds: ${{secrets.AZURE_CREDENTIALS}}
    - name: setup
      run: bash setup.sh
      working-directory: cli
      continue-on-error: true
    - name: run job
      run: bash -x ../../../run-job.sh pipeline.yml
      working-directory: cli/jobs/pipelines/nyc-taxi
```

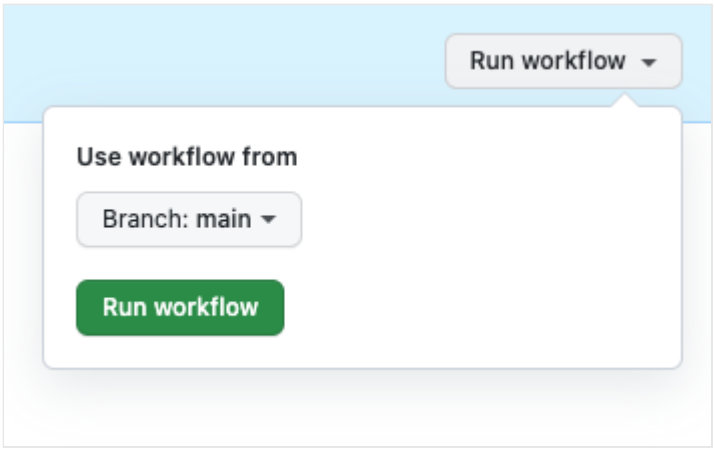
2. Select **View runs**.

3. Enable workflows by selecting **I understand my workflows, go ahead and enable them**.

4. Select the **cli-jobs-pipelines-nyc-taxi-pipeline workflow** and choose to **Enable workflow**.



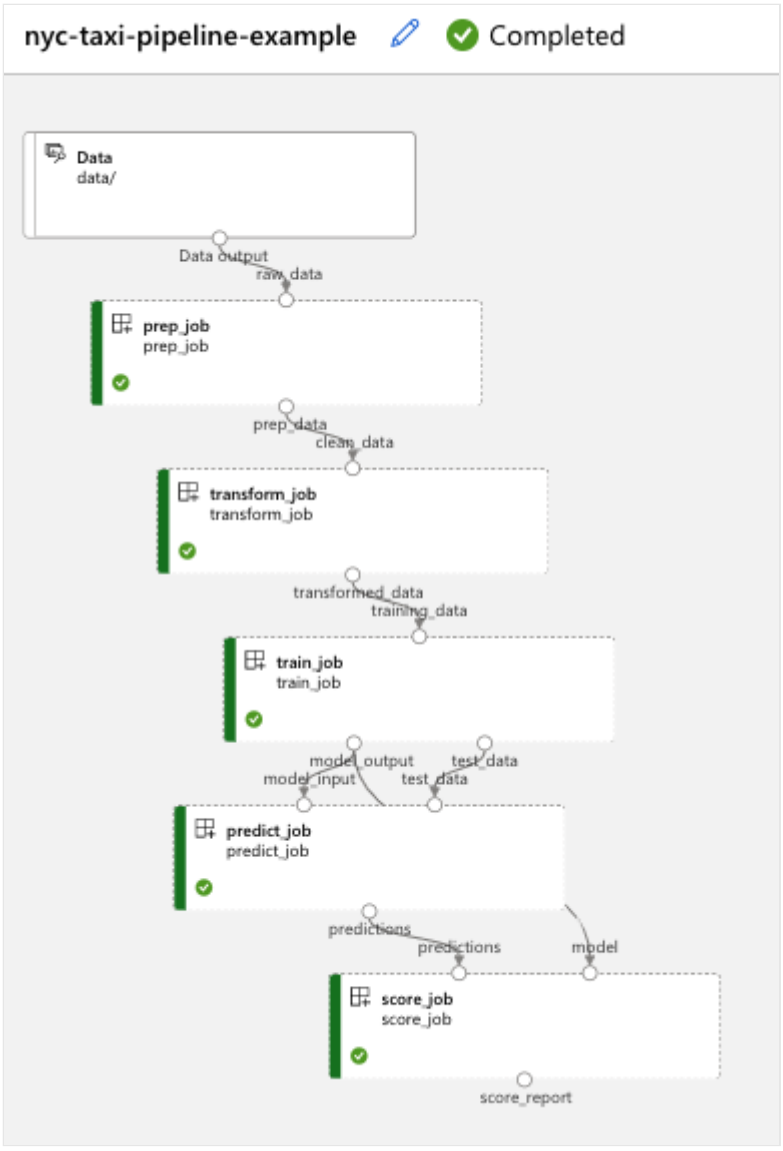
5. Select **Run workflow** and choose the option to **Run workflow** now.



Step 6: Verify your workflow run

1. Open your completed workflow run and verify that the build job ran successfully. You'll see a green checkmark next to the job.

2. Open Azure Machine Learning studio and navigate to the **nyc-taxi-pipeline-example**. Verify that each part of your job (prep, transform, train, predict, score) completed and that you see a green checkmark.



Clean up resources

When your resource group and repository are no longer needed, clean up the resources you deployed by deleting the resource group and your GitHub repository.

Next steps

Create production ML pipelines with Python SDK

Trigger applications, processes, or CI/CD workflows based on Azure Machine Learning events (preview)

Article • 04/04/2023

In this article, you learn how to set up event-driven applications, processes, or CI/CD workflows based on Azure Machine Learning events, such as failure notification emails or ML pipeline runs, when certain conditions are detected by Azure Event Grid.

Azure Machine Learning manages the entire lifecycle of machine learning process, including model training, model deployment, and monitoring. You can use Event Grid to react to Azure Machine Learning events, such as the completion of training runs, the registration and deployment of models, and the detection of data drift, by using modern serverless architectures. You can then subscribe and consume events such as run status changed, run completion, model registration, model deployment, and data drift detection within a workspace.

When to use Event Grid for event driven actions:

- Send emails on run failure and run completion
- Use an Azure function after a model is registered
- Streaming events from Azure Machine Learning to various of endpoints
- Trigger an ML pipeline when drift is detected

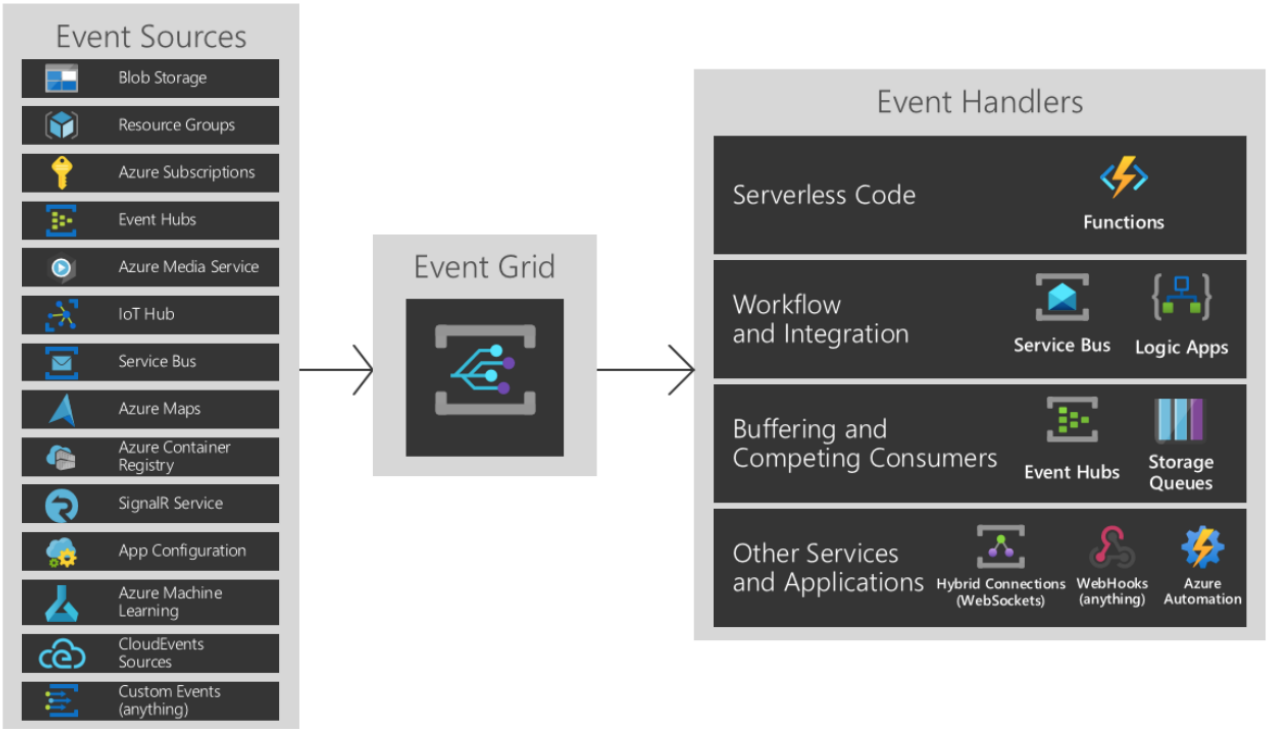
> **Important**
>
> This feature is currently in public preview. This preview version is provided without a service-level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see **Supplemental Terms of Use for Microsoft Azure Previews**.

Prerequisites

To use Event Grid, you need contributor or owner access to the Azure Machine Learning workspace you will create events for.

The event model & types

Azure Event Grid reads events from sources, such as Azure Machine Learning and other Azure services. These events are then sent to event handlers such as Azure Event Hubs, Azure Functions, Logic Apps, and others. The following diagram shows how Event Grid connects sources and handlers, but is not a comprehensive list of supported integrations.



For more information on event sources and event handlers, see What is Event Grid?

Event types for Azure Machine Learning

Azure Machine Learning provides events in the various points of machine learning lifecycle:

Event type	Description
`Microsoft.MachineLearningServices.RunCompleted`	Raised when a machine learning experiment run is completed
`Microsoft.MachineLearningServices.ModelRegistered`	Raised when a machine learning model is registered in the workspace
`Microsoft.MachineLearningServices.ModelDeployed`	Raised when a deployment of inference service with one or more models is completed
`Microsoft.MachineLearningServices.DatasetDriftDetected`	Raised when a data drift detection job for two datasets is completed
`Microsoft.MachineLearningServices.RunStatusChanged`	Raised when a run status is changed

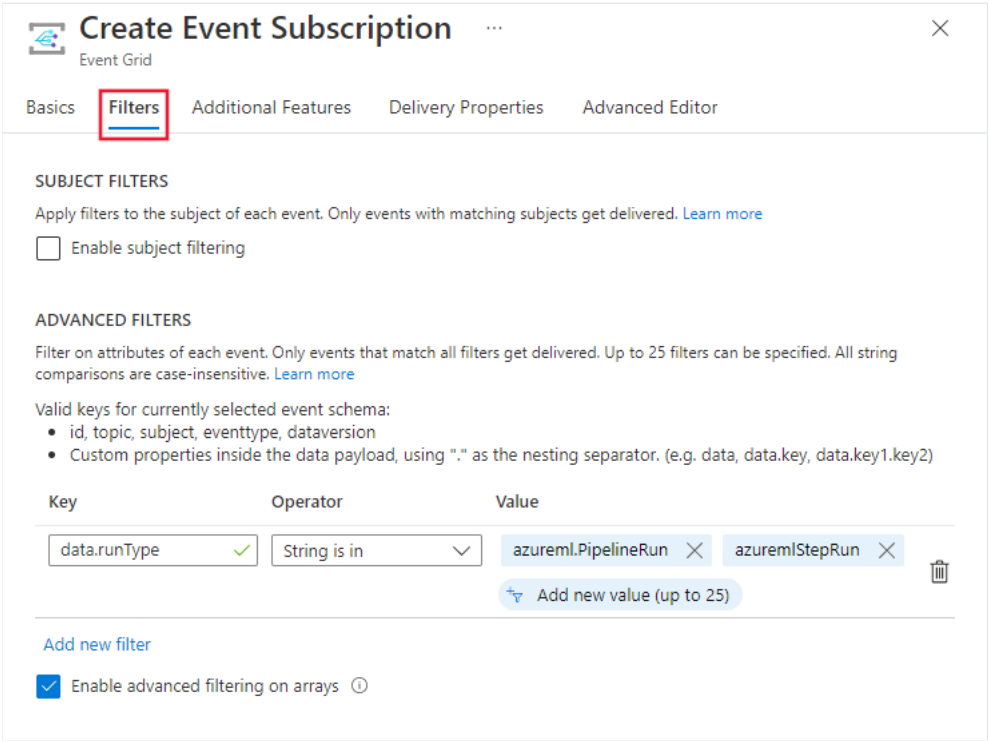
Filter & subscribe to events

These events are published through Azure Event Grid. Using Azure portal, PowerShell or Azure CLI, customers can easily subscribe to events by specifying one or more event types, and filtering conditions.

When setting up your events, you can apply filters to only trigger on specific event data. In the example below, for run status changed events, you can filter by run types. The event only triggers when the criteria is met. Refer to the Azure Machine Learning event grid schema to learn about event data you can filter by.

Subscriptions for Azure Machine Learning events are protected by Azure role-based access control (Azure RBAC). Only contributor or owner of a workspace can create, update, and delete event subscriptions. Filters can be applied to event subscriptions either during the creation of the event subscription or at a later time.

1. Go to the Azure portal, select a new subscription or an existing one.
2. Select the Events entry from the left navigation area, and then select + **Event subscription**.
3. Select the filters tab and scroll down to Advanced filters. For the **Key** and **Value**, provide the property types you want to filter by. Here you can see the event will only trigger when the run type is a pipeline run or pipeline step run.



- **Filter by event type:** An event subscription can specify one or more Azure Machine Learning event types.
- **Filter by event subject:** Azure Event Grid supports subject filters based on **begins with** and **ends with** matches, so that events with a matching subject are delivered to the subscriber. Different machine learning events have different subject format.

Event type	Subject format	Sample subject

Event type	Subject format	Sample subject
`Microsoft.MachineLearningServices.RunCompleted`	`experiments/{ExperimentId}/runs/{RunId}`	`experiments/b1d7966c-f73a-4c68-b846-992ace89551f/runs/my_exp1_1554835758_38dbaa94`
`Microsoft.MachineLearningServices.ModelRegistered`	`models/{modelName}:{modelVersion}`	`models/sklearn_regression_model:3`
`Microsoft.MachineLearningServices.ModelDeployed`	`endpoints/{serviceId}`	`endpoints/my_sklearn_aks`
`Microsoft.MachineLearningServices.DatasetDriftDetected`	`datadrift/{data.DataDriftId}/run/{data.RunId}`	`datadrift/4e694bf5-712e-4e40-b06a-d2a2755212d4/run/my_driftrun1_1550564444_fbbc`
`Microsoft.MachineLearningServices.RunStatusChanged`	`experiments/{ExperimentId}/runs/{RunId}`	`experiments/b1d7966c-f73a-4c68-b846-992ace89551f/runs/my_exp1_1554835758_38dbaa94`

- **Advanced filtering**: Azure Event Grid also supports advanced filtering based on published event schema. Azure Machine Learning event schema details can be found in Azure Event Grid event schema for Azure Machine Learning. Some sample advanced filterings you can perform include:

  For `Microsoft.MachineLearningServices.ModelRegistered` event, to filter model's tag value:

  ```
  --advanced-filter data.ModelTags.key1 StringIn ('value1')
  ```

  To learn more about how to apply filters, see Filter events for Event Grid.

Consume Machine Learning events

Applications that handle Machine Learning events should follow a few recommended practices:

- ✓ As multiple subscriptions can be configured to route events to the same event handler, it is important not to assume events are from a particular source, but to check the topic of the message to ensure that it comes from the machine learning workspace you are expecting.
- ✓ Similarly, check that the eventType is one you are prepared to process, and do not assume that all events you receive will be the types you expect.
- ✓ As messages can arrive out of order and after some delay, use the etag fields to understand if your information about objects is still up-to-date. Also, use the sequencer fields to understand the order of events on any particular object.
- ✓ Ignore fields you don't understand. This practice will help keep you resilient to new features that might be added in the future.
- ✓ Failed or cancelled Azure Machine Learning operations will not trigger an event. For example, if a model deployment fails Microsoft.MachineLearningServices.ModelDeployed won't be triggered. Consider such failure mode when design your applications. You can always use Azure Machine Learning SDK, CLI or portal to check the status of an operation and understand the detailed failure reasons.

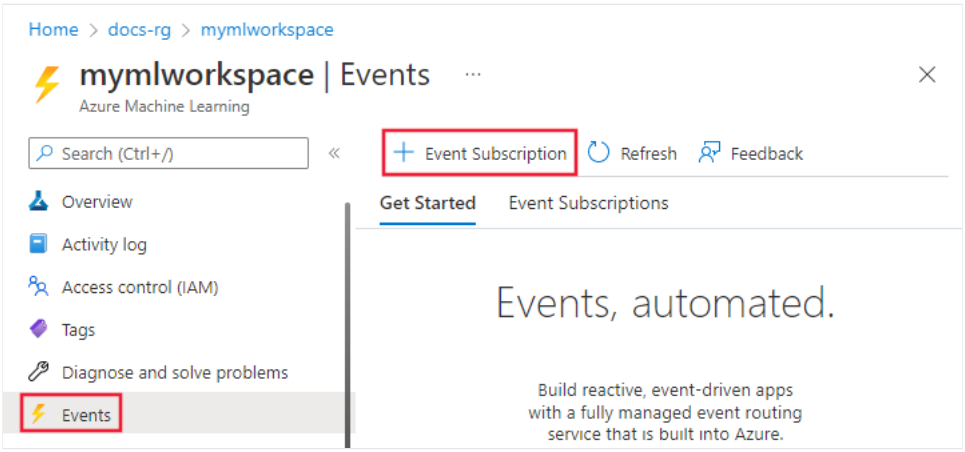
Azure Event Grid allows customers to build de-coupled message handlers, which can be triggered by Azure Machine Learning events. Some notable examples of message handlers are:

- Azure Functions
- Azure Logic Apps
- Azure Event Hubs
- Azure Data Factory Pipeline
- Generic webhooks, which may be hosted on the Azure platform or elsewhere

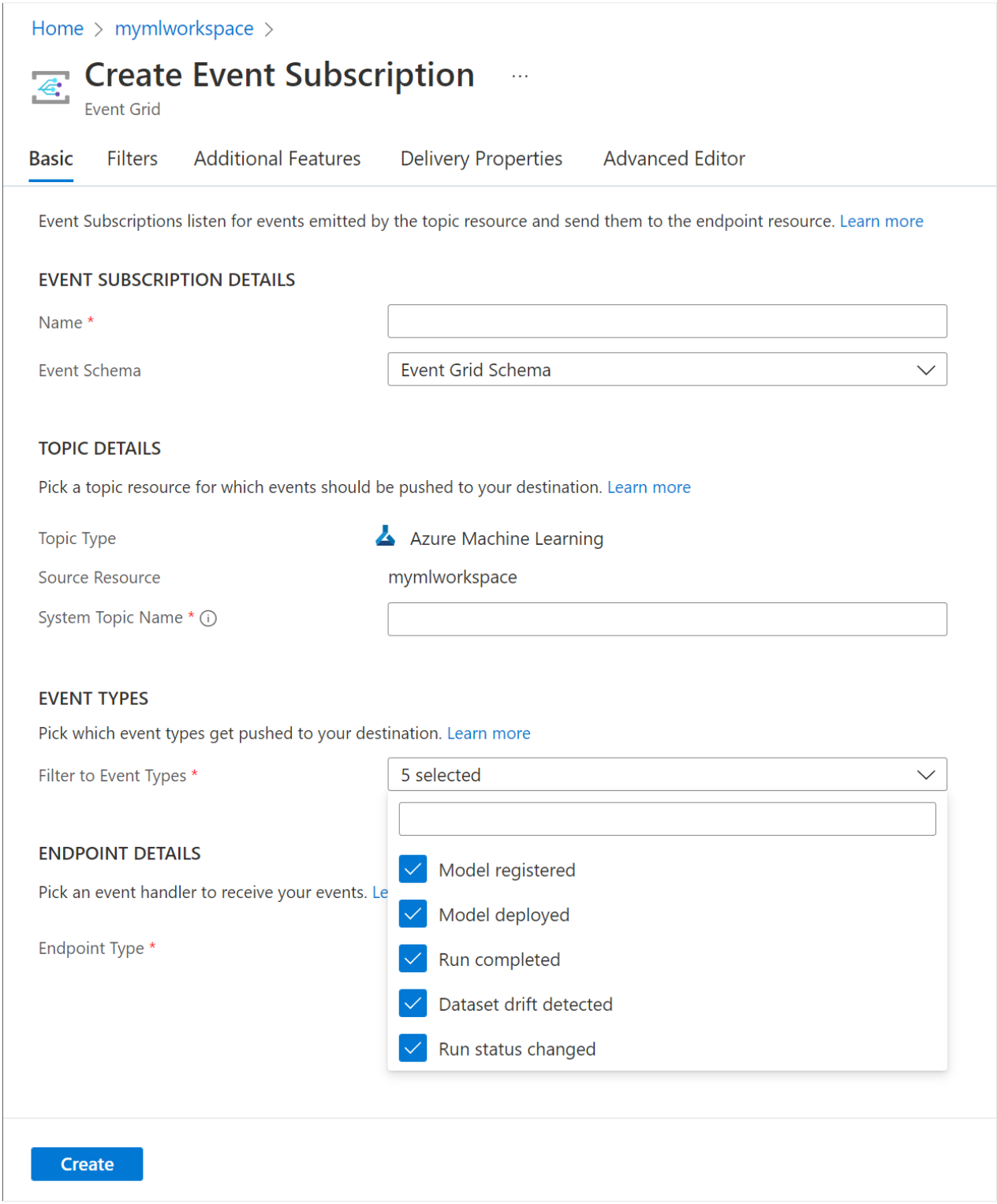
Set up in Azure portal

1. Open the Azure portal and go to your Azure Machine Learning workspace.

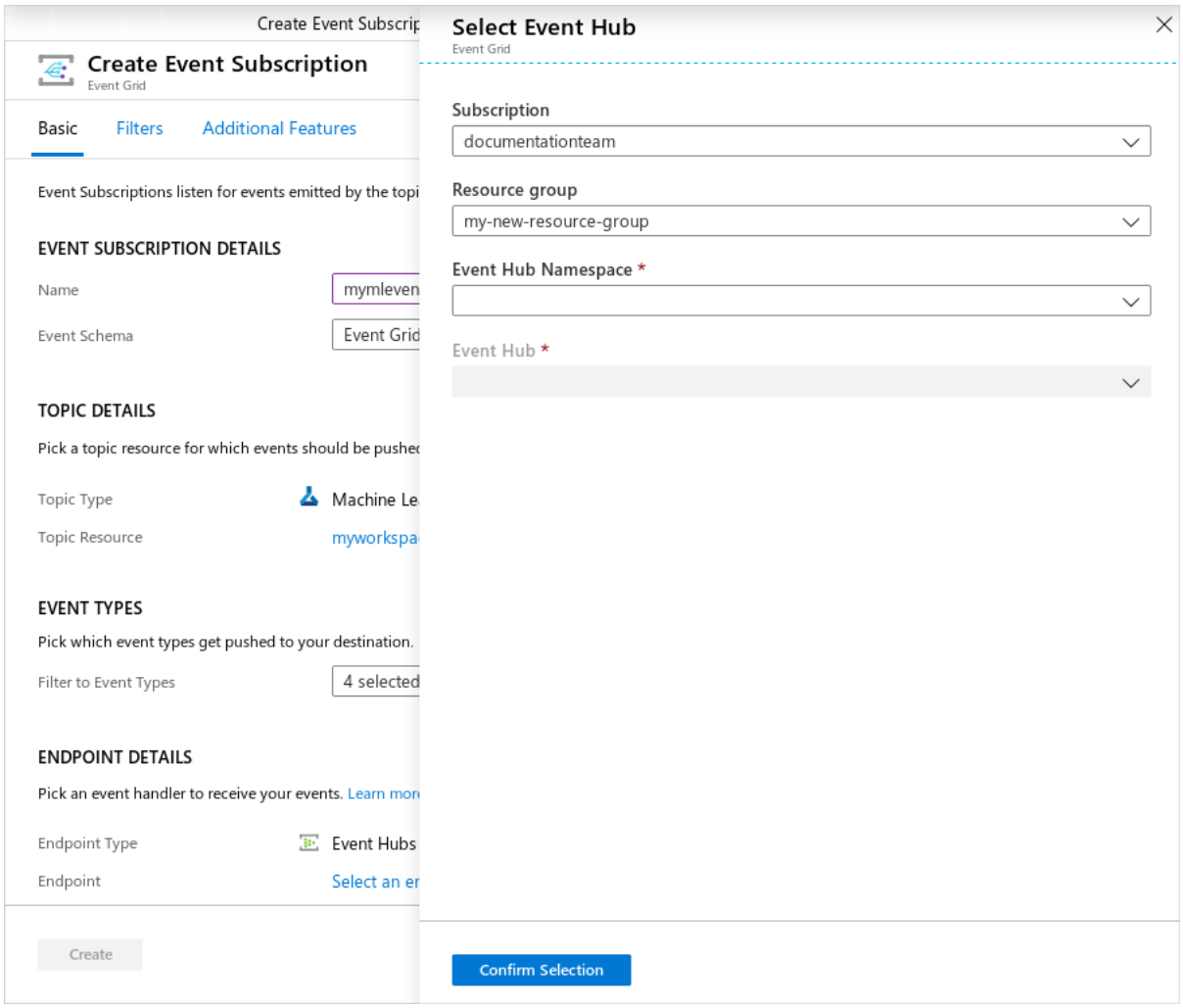
2. From the left bar, select **Events** and then select **Event Subscriptions**.



3. Select the event type to consume. For example, the following screenshot has selected **Model registered**, **Model deployed**, **Run completed**, and **Dataset drift detected**:



4. Select the endpoint to publish the event to. In the following screenshot, **Event hub** is the selected endpoint:



Once you have confirmed your selection, click **Create**. After configuration, these events will be pushed to your endpoint.

Set up with the CLI

You can either install the latest Azure CLI, or use the Azure Cloud Shell that is provided as part of your Azure subscription.

To install the Event Grid extension, use the following command from the CLI:

Azure CLI

```
az extension add --name eventgrid
```

The following example demonstrates how to select an Azure subscription and creates e a new event subscription for Azure Machine Learning:

Azure CLI

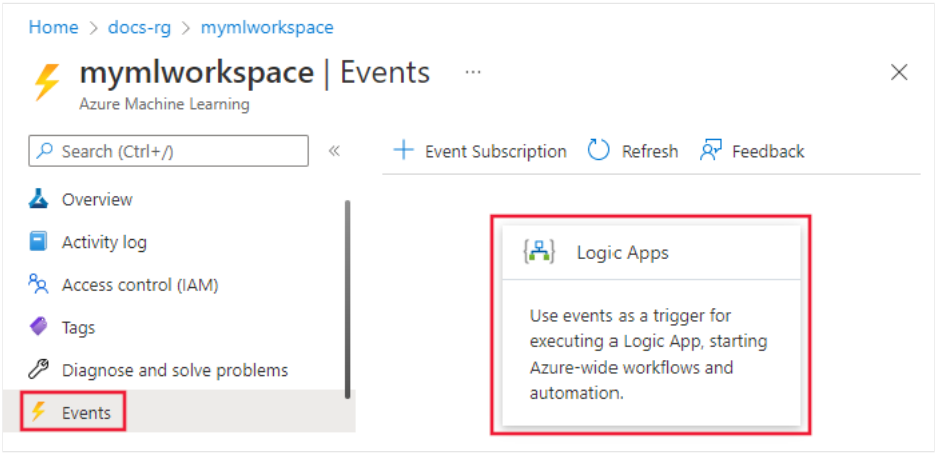
```
# Select the Azure subscription that contains the workspace
az account set --subscription "<name or ID of the subscription>"

# Subscribe to the machine learning workspace. This example uses EventHub as a destination.
az eventgrid event-subscription create --name {eventGridFilterName} \
  --source-resource-id
/subscriptions/{subId}/resourceGroups/{RG}/providers/Microsoft.MachineLearningServices/workspaces/{wsName} \
  --endpoint-type eventhub \
  --endpoint /subscriptions/{SubID}/resourceGroups/TestRG/providers/Microsoft.EventHub/namespaces/n1/eventhubs/EH1 \
  --included-event-types Microsoft.MachineLearningServices.ModelRegistered \
  --subject-begins-with "models/mymodelname"
```

Examples

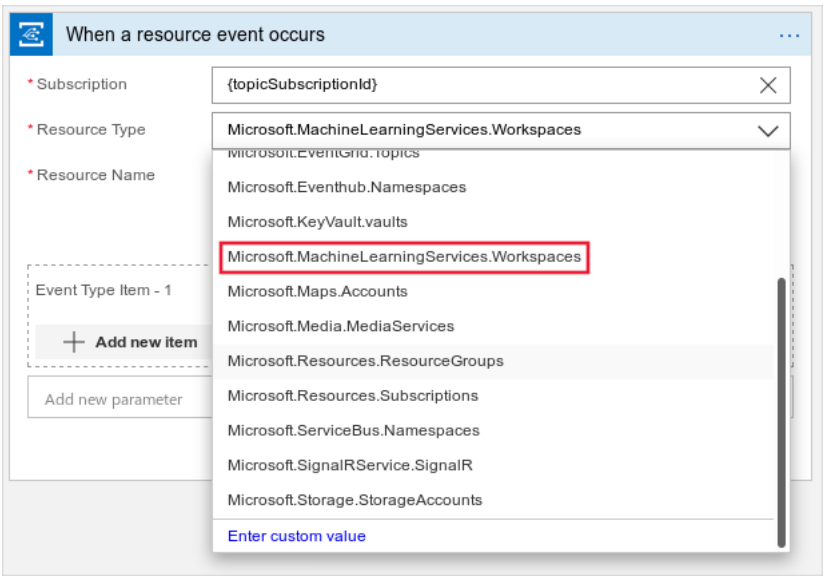
Example: Send email alerts

Use Azure Logic Apps to configure emails for all your events. Customize with conditions and specify recipients to enable collaboration and awareness across teams working together.

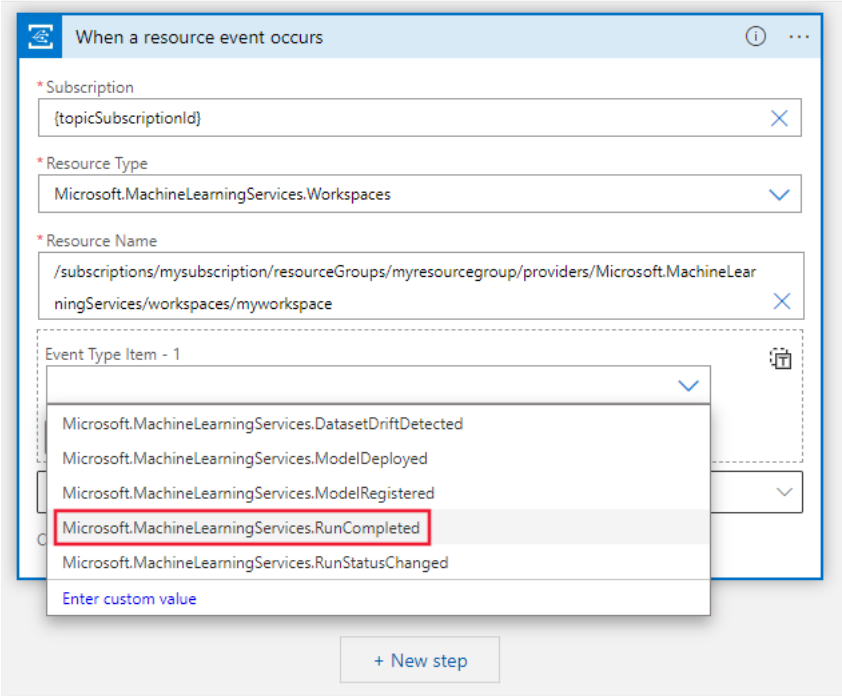
1. In the Azure portal, go to your Azure Machine Learning workspace and select the events tab from the left bar. From here, select **Logic apps**.



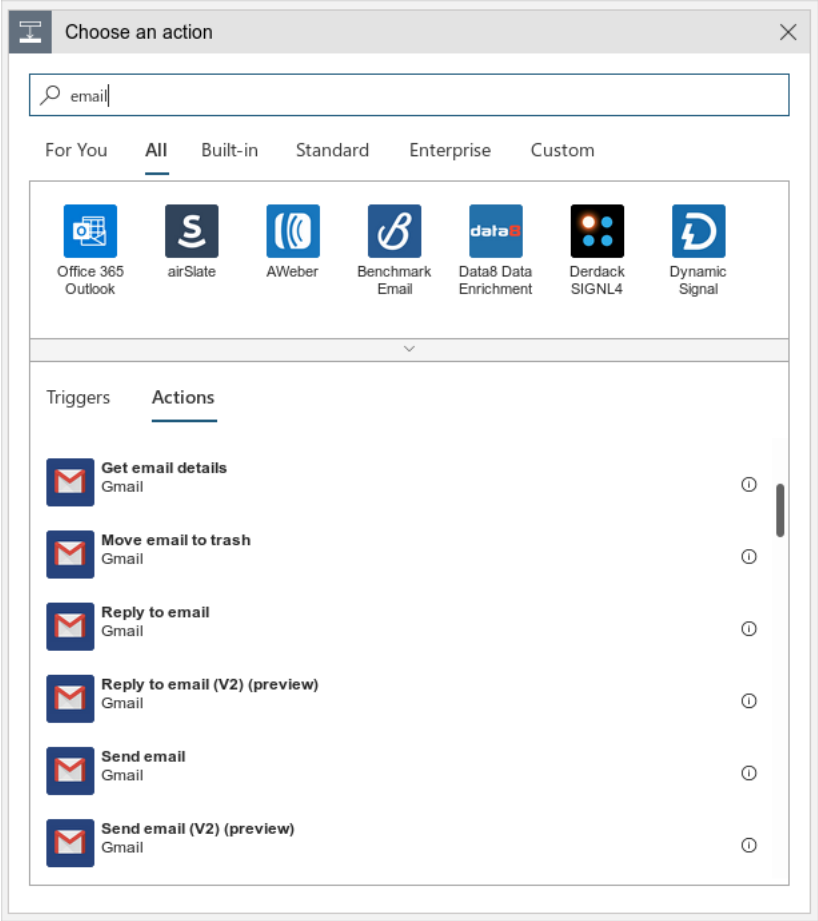
2. Sign into the Logic App UI and select Machine Learning service as the topic type.



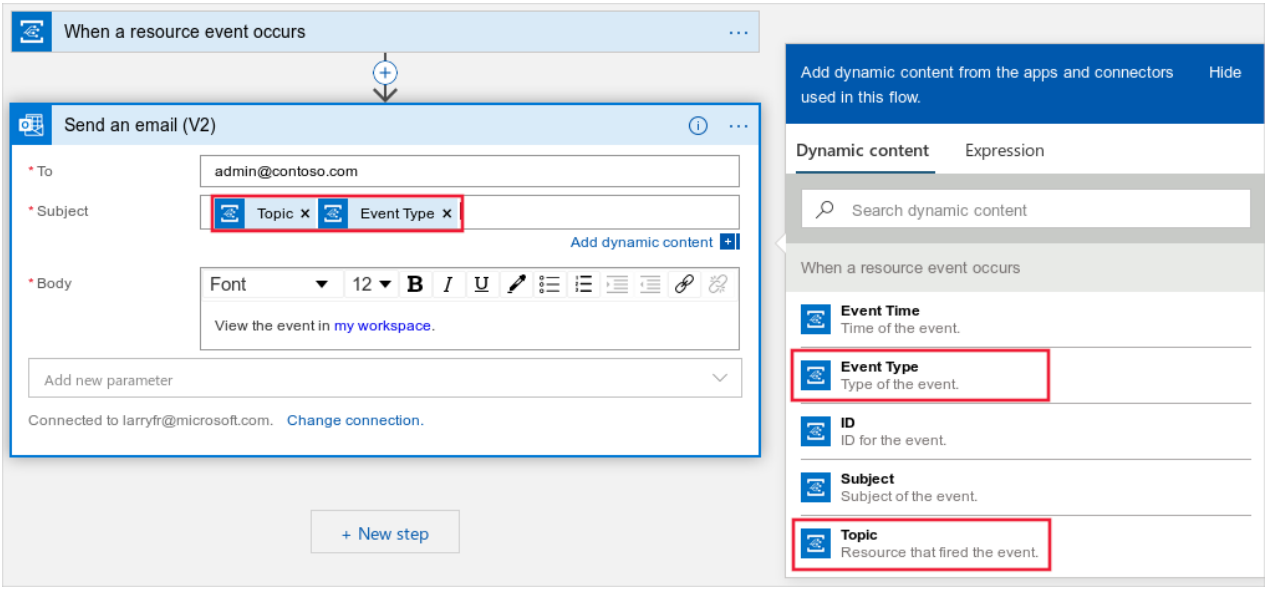
3. Select which event(s) to be notified for. For example, the following screenshot **RunCompleted**.



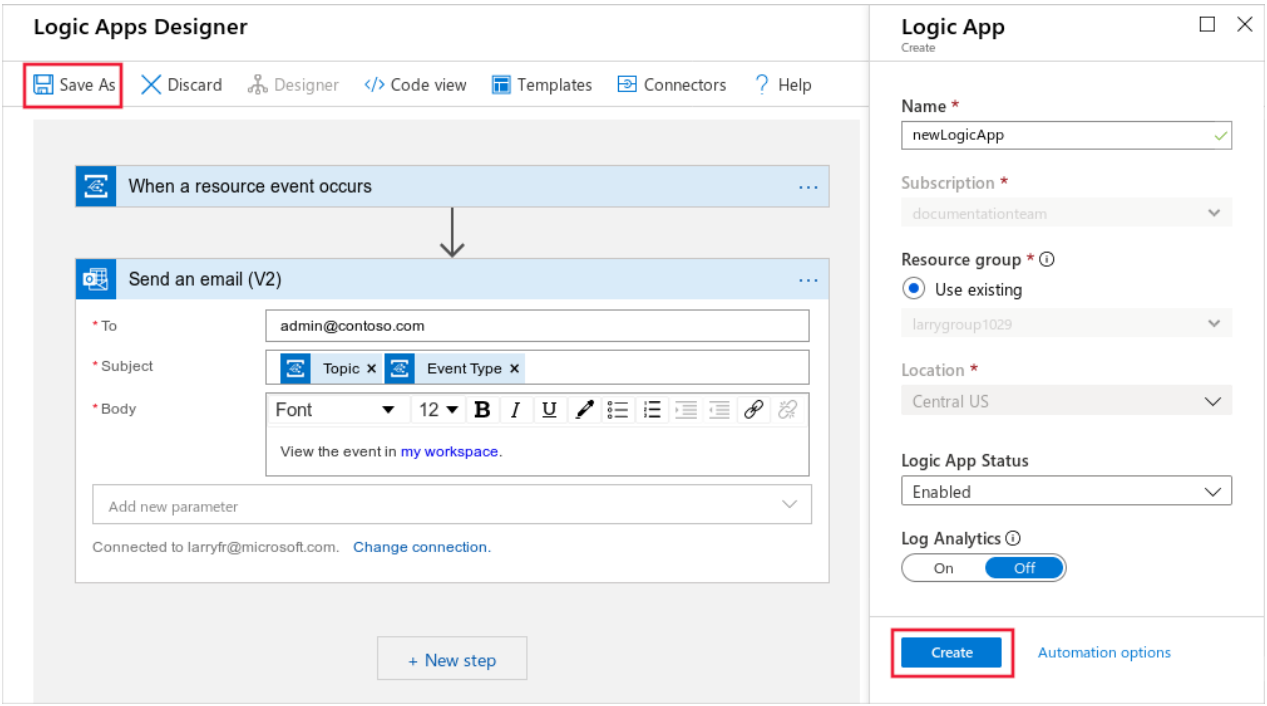
4. Next, add a step to consume this event and search for email. There are several different mail accounts you can use to receive events. You can also configure conditions on when to send an email alert.



5. Select **Send an email** and fill in the parameters. In the subject, you can include the **Event Type** and **Topic** to help filter events. You can also include a link to the workspace page for runs in the message body.



6. To save this action, select **Save As** on the left corner of the page. From the right bar that appears, confirm creation of this action.



Next steps

Learn more about Event Grid and give Azure Machine Learning events a try:

- About Event Grid
- Event schema for Azure Machine Learning

Set up MLOps with Azure DevOps

Article • 02/24/2023

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

Azure Machine Learning allows you to integrate with Azure DevOps pipeline to automate the machine learning lifecycle. Some of the operations you can automate are:

- Deployment of Azure Machine Learning infrastructure
- Data preparation (extract, transform, load operations)
- Training machine learning models with on-demand scale-out and scale-up
- Deployment of machine learning models as public or private web services
- Monitoring deployed machine learning models (such as for performance analysis)

In this article, you learn about using Azure Machine Learning to set up an end-to-end MLOps pipeline that runs a linear regression to predict taxi fares in NYC. The pipeline is made up of components, each serving different functions, which can be registered with the workspace, versioned, and reused with various inputs and outputs. you're going to be using the recommended Azure architecture for MLOps and AzureMLOps (v2) solution accelerator to quickly setup an MLOps project in Azure Machine Learning.

> **Tip**
>
> We recommend you understand some of the **recommended Azure architectures** for MLOps before implementing any solution. You'll need to pick the best architecture for your given Machine learning project.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning.
- An Azure Machine Learning workspace.
- Git running on your local machine.
- An organization in Azure DevOps.
- Azure DevOps project that will host the source repositories and pipelines.
- The Terraform extension for Azure DevOps if you're using Azure DevOps + Terraform to spin up infrastructure

> **Note**
>
> Git version 2.27 or newer is required. For more information on installing the Git command, see **https://git-scm.com/downloads** and select your operating system

> **Important**
>
> The CLI commands in this article were tested using Bash. If you use a different shell, you may encounter errors.

Set up authentication with Azure and DevOps

Before you can set up an MLOps project with Azure Machine Learning, you need to set up authentication for Azure DevOps.

Create service principal

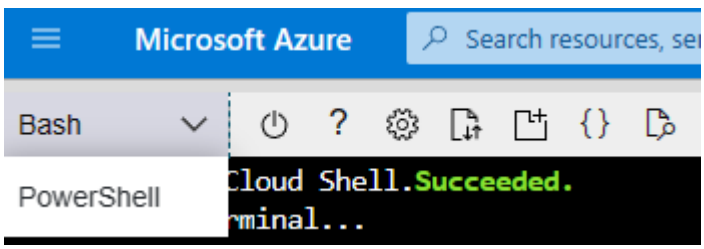
For the use of the demo, the creation of one or two service principles is required, depending on how many environments, you want to work on (Dev or Prod or Both). These principles can be created using one of the following methods:

Create from Azure Cloud Shell

1. Launch the Azure Cloud Shell.

   > **Tip**
   >
   > The first time you've launched the Cloud Shell, you'll be prompted to create a storage account for the Cloud Shell.

2. If prompted, choose **Bash** as the environment used in the Cloud Shell. You can also change environments in the drop-down on the top navigation bar



3. Copy the following bash commands to your computer and update the **projectName**, **subscriptionId**, and **environment** variables with the values for your project. If you're creating both a Dev and Prod environment, you'll need to run this script once for each environment, creating a service principal for each. This command will also grant the **Contributor** role to the service principal in the subscription provided. This is required for Azure DevOps to properly use resources in that subscription.

```bash
projectName="<your project name>"
roleName="Contributor"
subscriptionId="<subscription Id>"
environment="<Dev|Prod>" #First letter should be capitalized
servicePrincipalName="Azure-ARM-${environment}-${projectName}"
# Verify the ID of the active subscription
echo "Using subscription ID $subscriptionID"
echo "Creating SP for RBAC with name $servicePrincipalName, with
role $roleName and in scopes     /subscriptions/$subscriptionId"
az ad sp create-for-rbac --name $servicePrincipalName --role
$roleName --scopes /subscriptions/$subscriptionId
echo "Please ensure that the information created here is properly
save for future use."
```

4. Copy your edited commands into the Azure Shell and run them (**Ctrl** + **Shift** + **v**).

5. After running these commands, you'll be presented with information related to the service principal. Save this information to a safe location, it will be use later in the demo to configure Azure DevOps.

```json
{
   "appId": "<application id>",
   "displayName": "Azure-ARM-dev-Sample_Project_Name",
   "password": "<password>",
   "tenant": "<tenant id>"
}
```

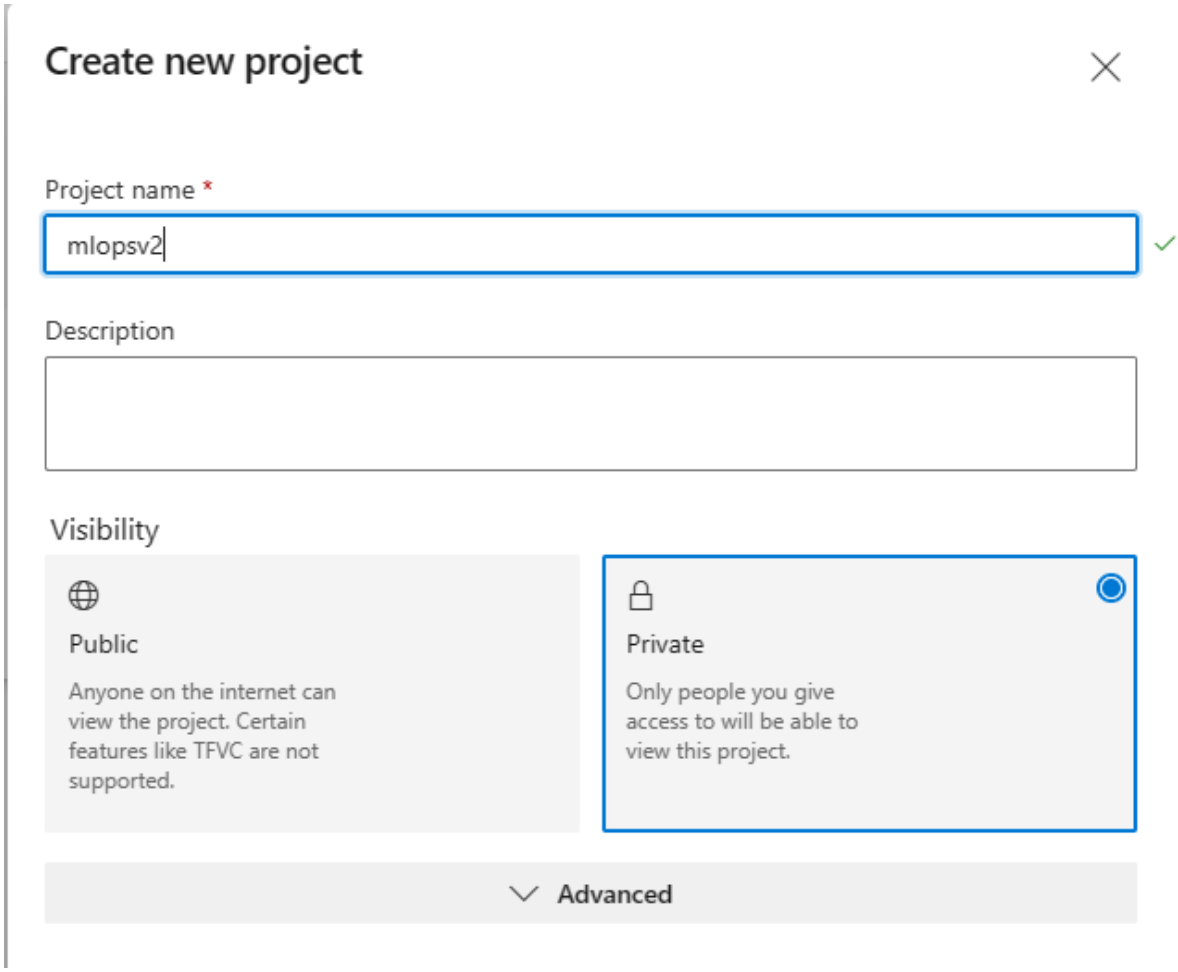
6. Repeat **Step 3.** if you're creating service principals for Dev and Prod environments. For this demo, we'll be creating only one environment, which is Prod.

7. Close the Cloud Shell once the service principals are created.

Set up Azure DevOps

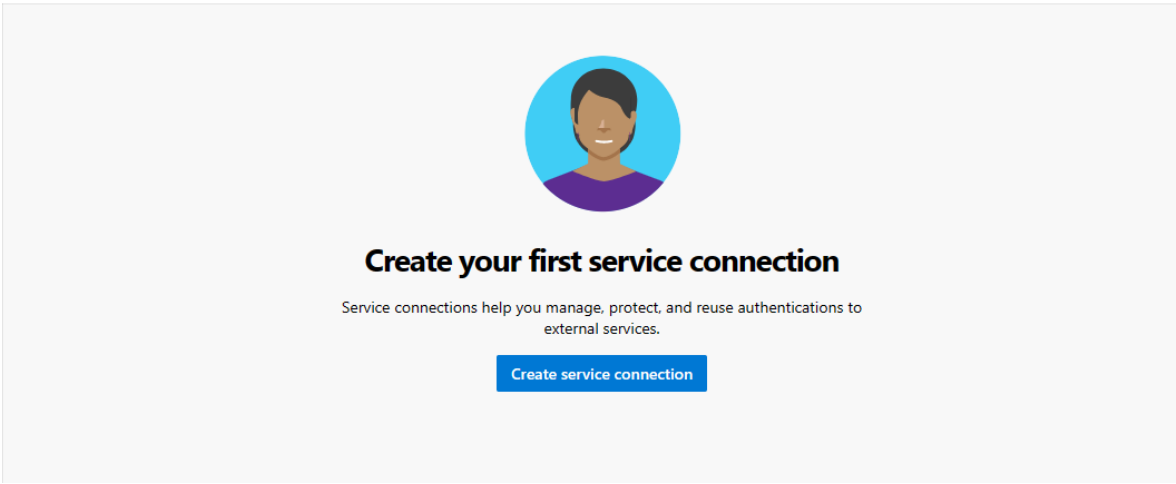
1. Navigate to Azure DevOps.

2. Select **create a new project** (Name the project `mlopsv2` for this tutorial).



3. In the project under **Project Settings** (at the bottom left of the project page) select **Service Connections**.

4. Select **Create Service Connection**.



5. Select **Azure Resource Manager**, select **Next**, select **Service principal (manual)**, select **Next** and select the Scope Level **Subscription**.

    - **Subscription Name** - Use the name of the subscription where your service principal is stored.
    - **Subscription Id** - Use the `subscriptionId` you used in **Step 1.** input as the Subscription ID
    - **Service Principal Id** - Use the `appId` from **Step 1.** output as the Service Principal ID
    - **Service principal key** - Use the `password` from **Step 1.** output as the Service Principal Key
    - **Tenant ID** - Use the `tenant` from **Step 1.** output as the Tenant ID

6. Name the service connection **Azure-ARM-Prod**.

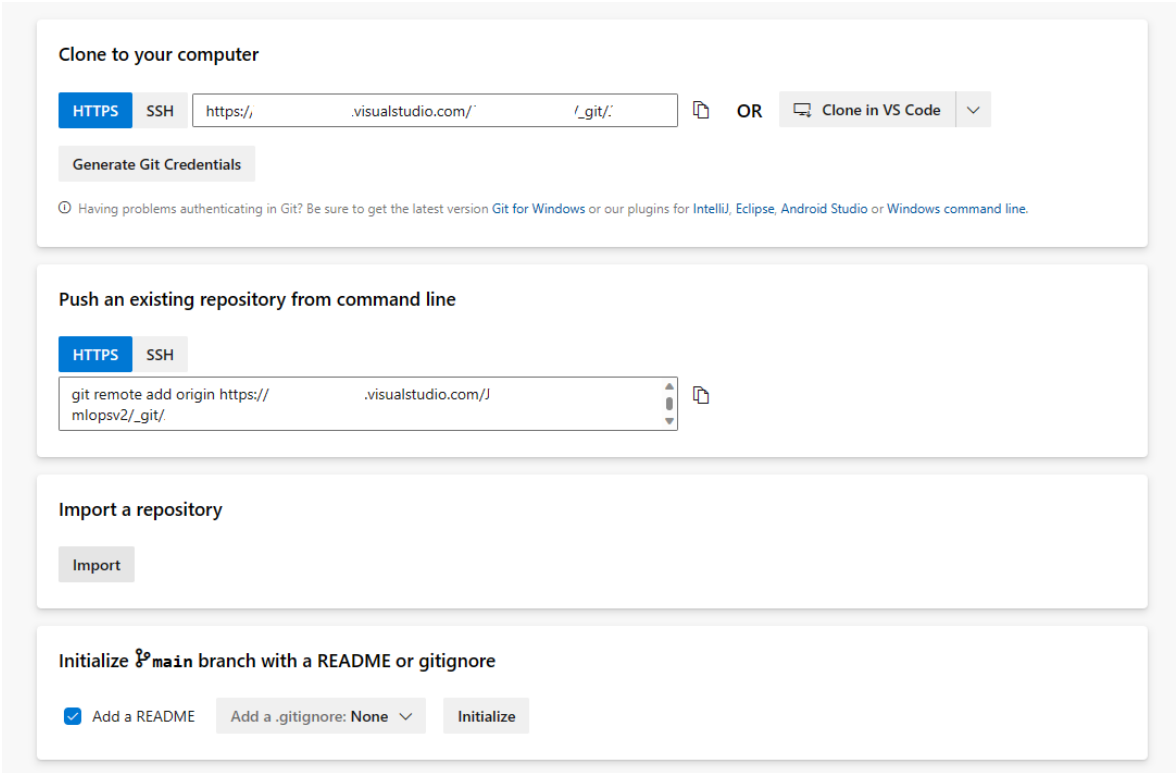
7. Select **Grant access permission to all pipelines**, then select **Verify and Save**.

The Azure DevOps setup is successfully finished.

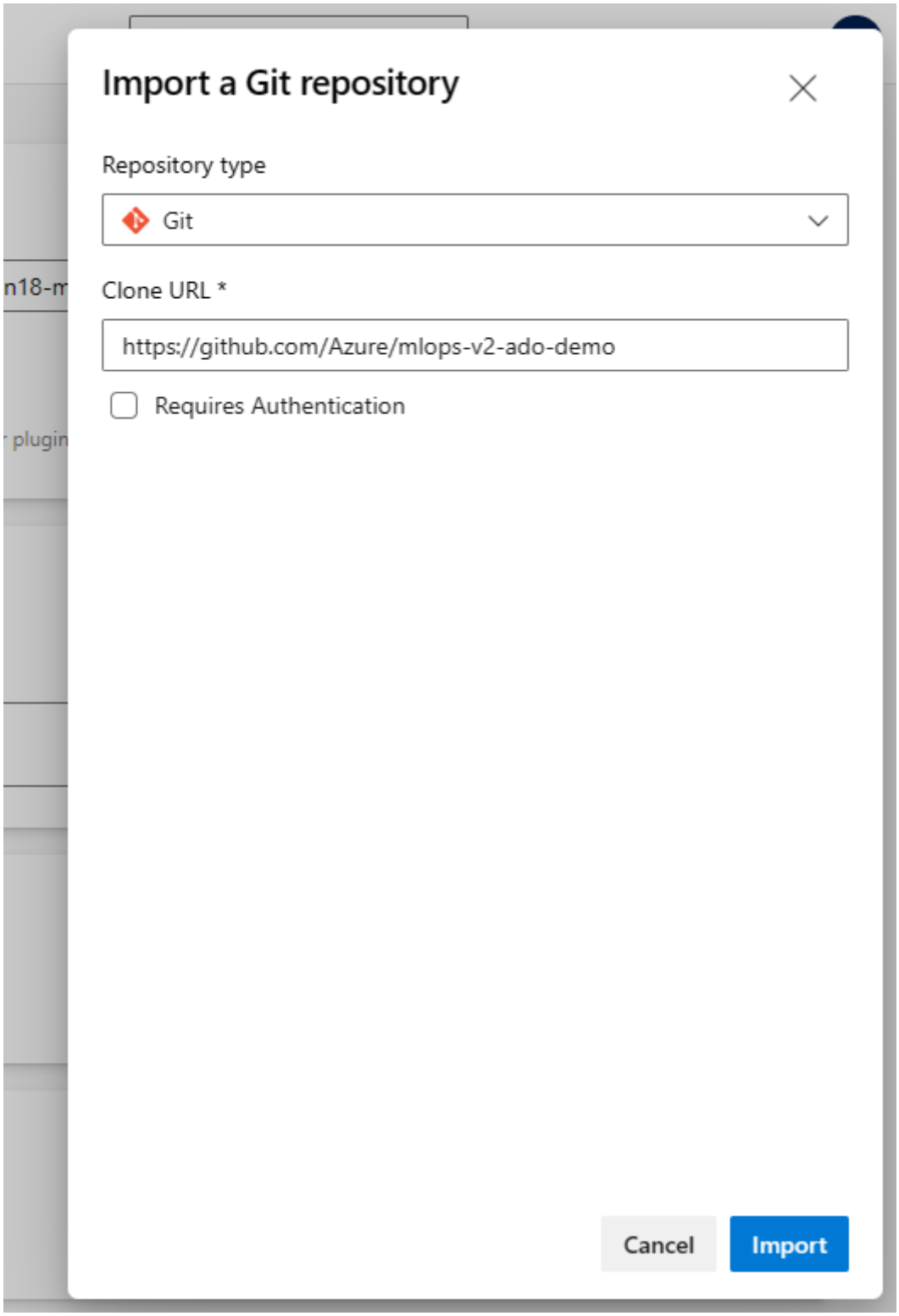
Set up source repository with Azure DevOps

1. Open the project you created in Azure DevOps

2. Open the Repos section and select **Import Repository**



3. Enter https://github.com/Azure/mlops-v2-ado-demo into the Clone URL field. Select import at the bottom of the page

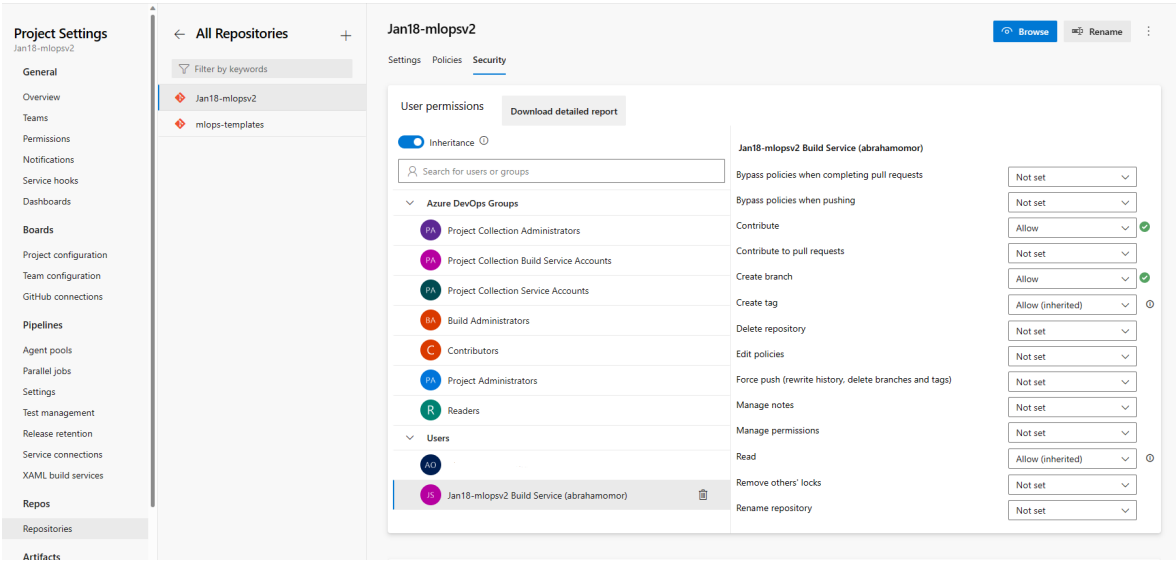


4. Open the **Project settings** at the bottom of the left hand navigation pane

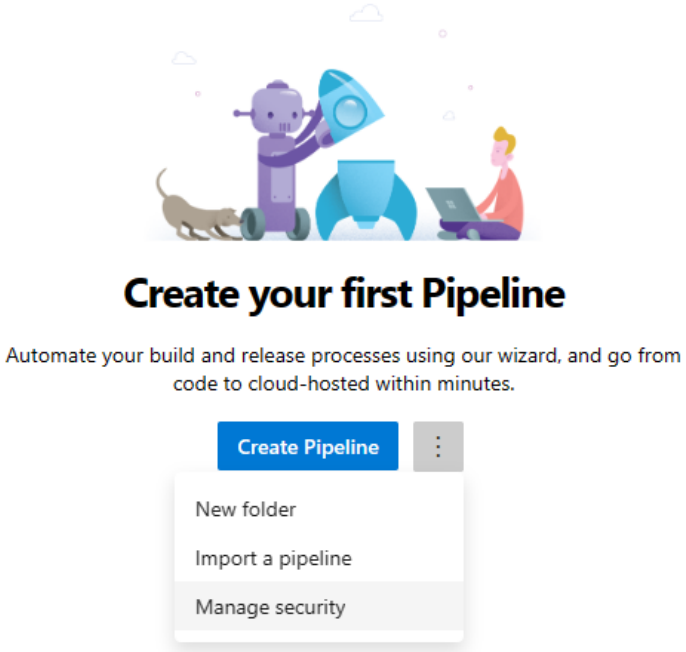
5. Under the Repos section, select **Repositories**. Select the repository you created in previous step Select the **Security** tab

6. Under the User permissions section, select the **mlopsv2 Build Service** user. Change the permission **Contribute** permission to **Allow** and the **Create branch** permission

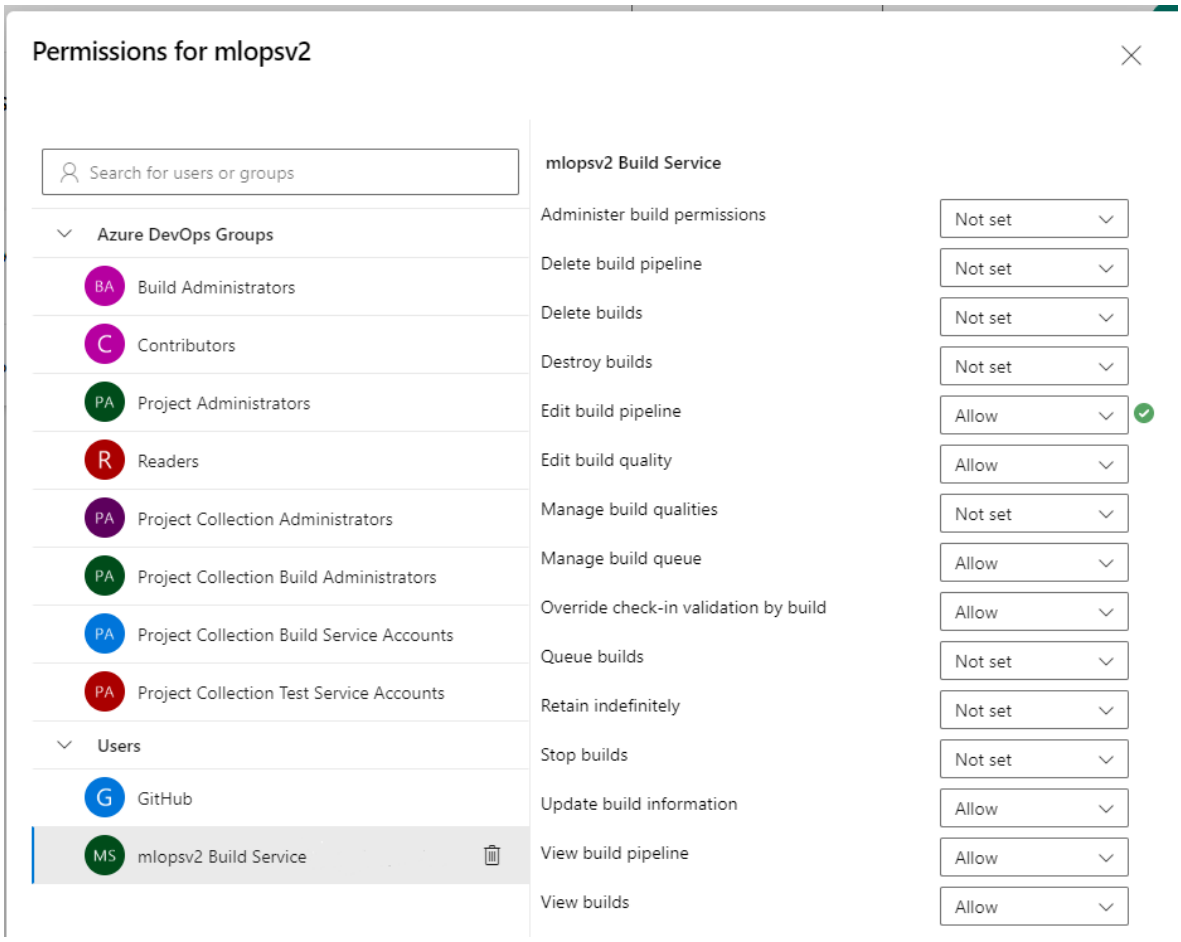
to **Allow**.



7. Open the **Pipelines** section in the left hand navigation pane and select on the 3 vertical dots next to the **Create Pipelines** button. Select **Manage Security**



8. Select the **mlopsv2 Build Service** account for your project under the Users section. Change the permission **Edit build pipeline** to **Allow**



> **Note**
>
> This finishes the prerequisite section and the deployment of the solution accelerator can happen accordingly.

Deploying infrastructure via Azure DevOps

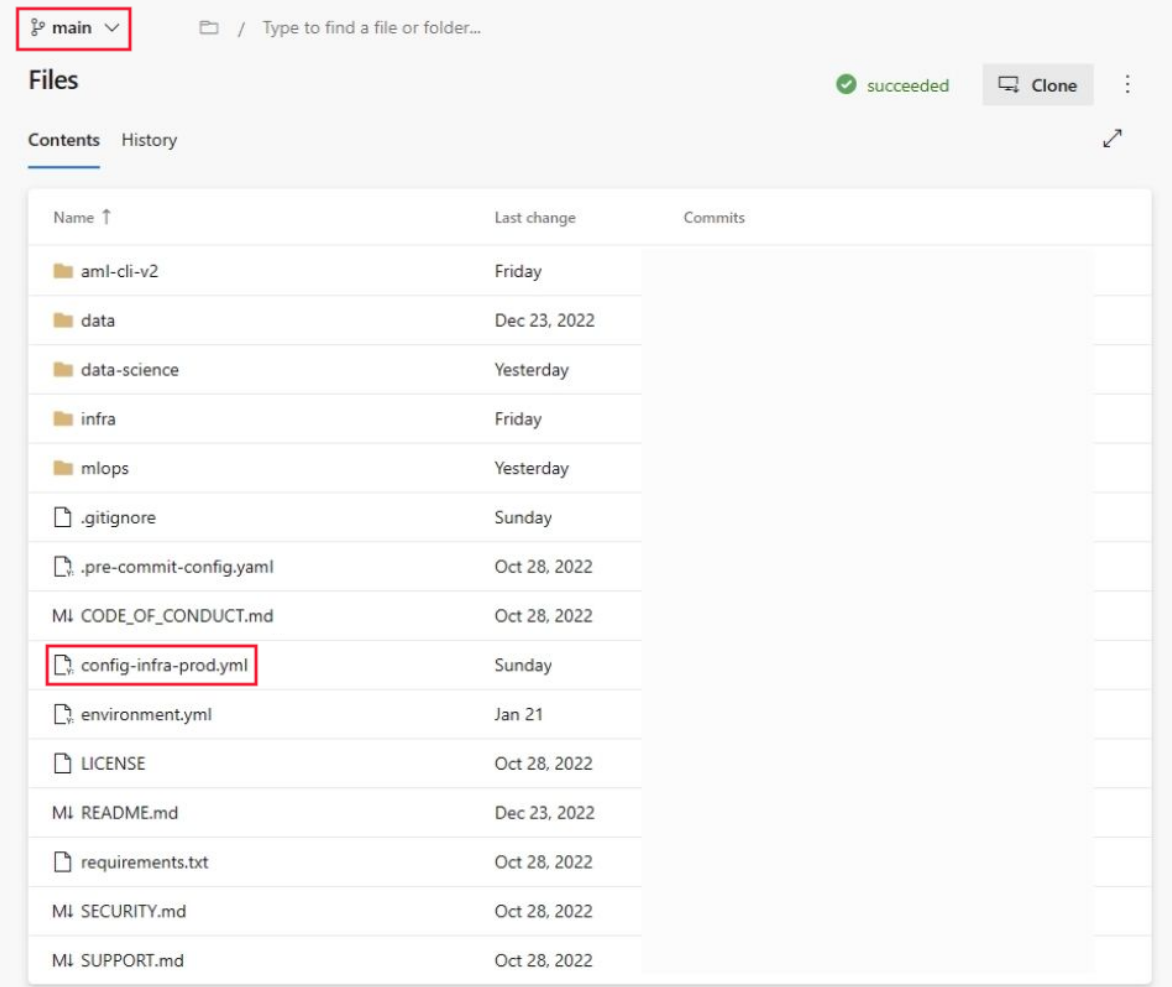
This step deploys the training pipeline to the Azure Machine Learning workspace created in the previous steps.

> **Tip**
>
> Make sure you understand the **Architectural Patterns** of the solution accelerator before you checkout the MLOps v2 repo and deploy the infrastructure. In examples you'll use the **classical ML project type**.

Run Azure infrastructure pipeline

1. Go to your repository, `mlops-v2-ado-demo`, and select the **config-infra-prod.yml** file.

> **ⓘ Important**
>
> Make sure you've selected the **main** branch of the repo.



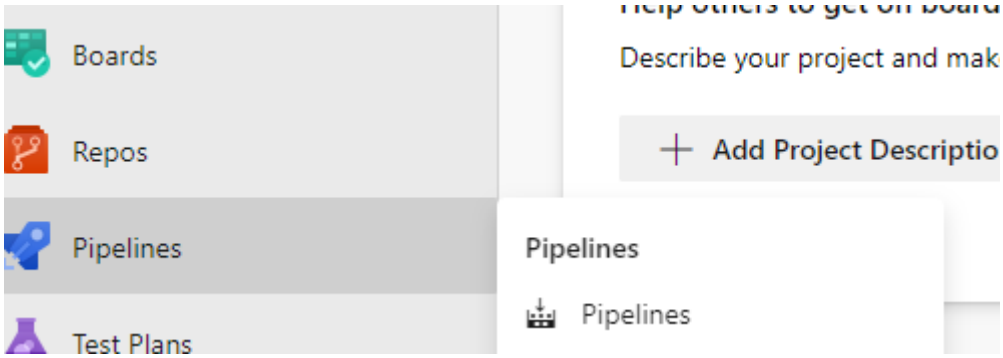
This config file uses the namespace and postfix values the names of the artifacts to ensure uniqueness. Update the following section in the config to your liking.

```
namespace: [5 max random new letters]
postfix: [4 max random new digits]
location: eastus
```

> **ⓘ Note**
>
> If you are running a Deep Learning workload such as CV or NLP, ensure your GPU compute is available in your deployment zone.

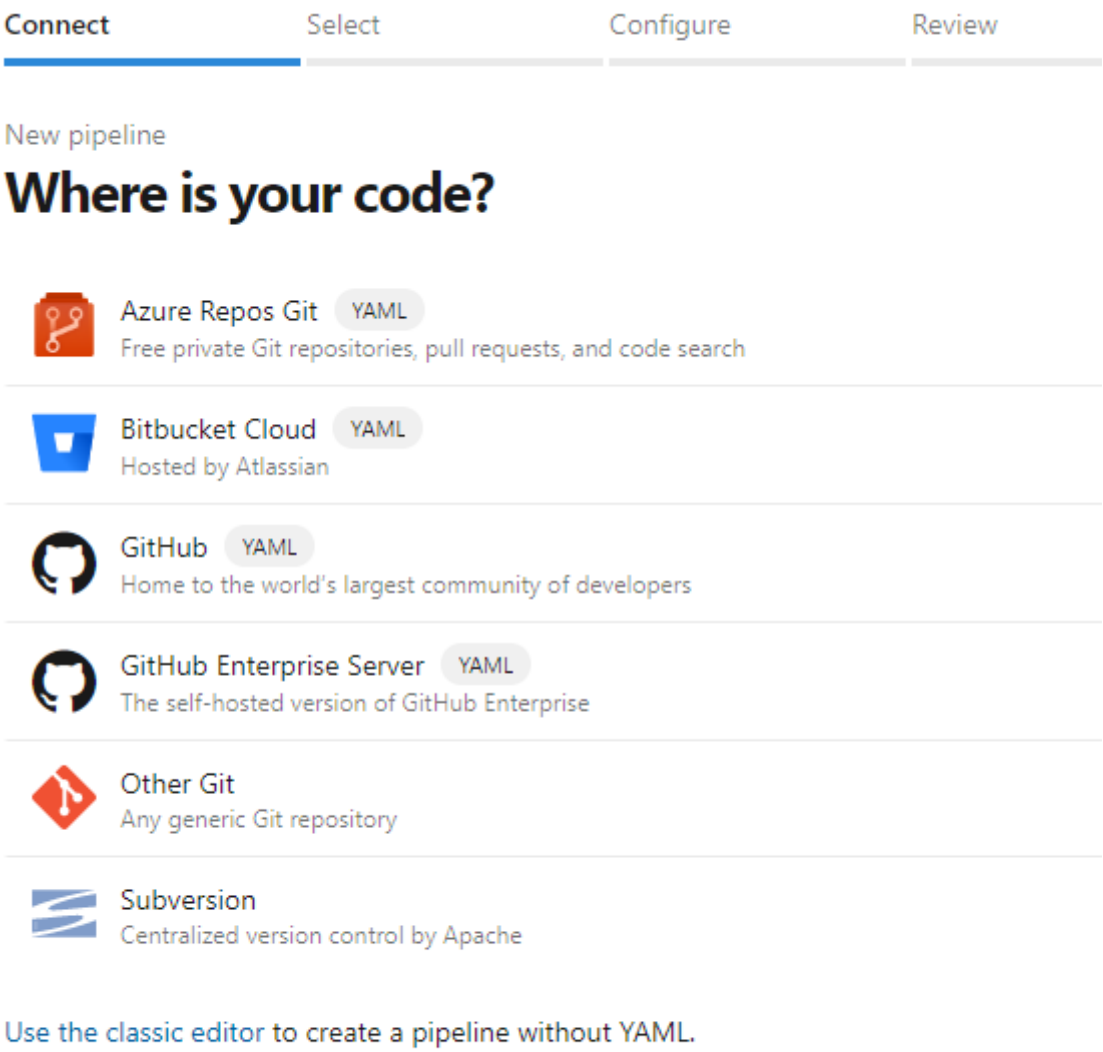
2. Select Commit and push code to get these values into the pipeline.

3. Go to Pipelines section



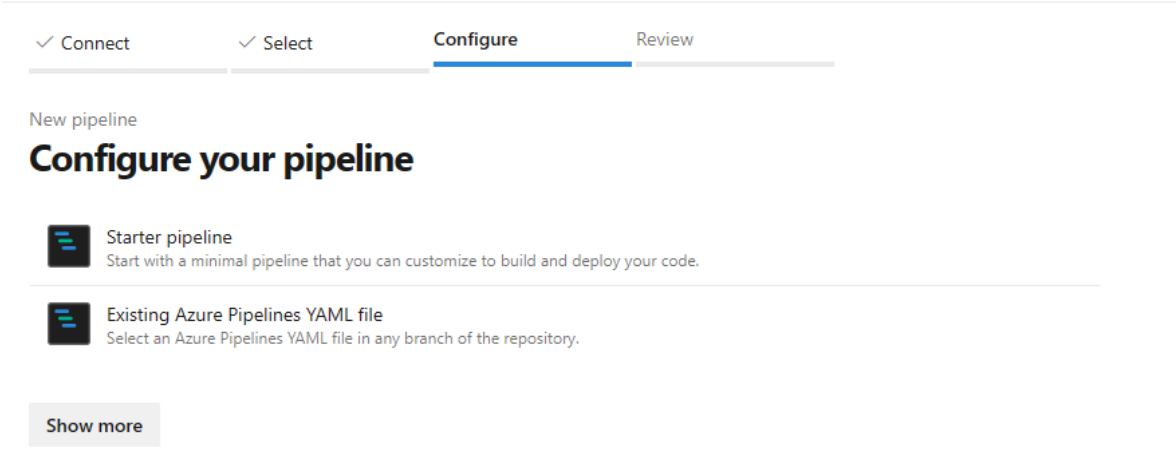
4. Select **Create Pipeline**.

5. Select **Azure Repos Git**.



6. Select the repository that you cloned in from the previous section `mlops-v2-ado-demo`

7. Select **Existing Azure Pipeline YAML File**

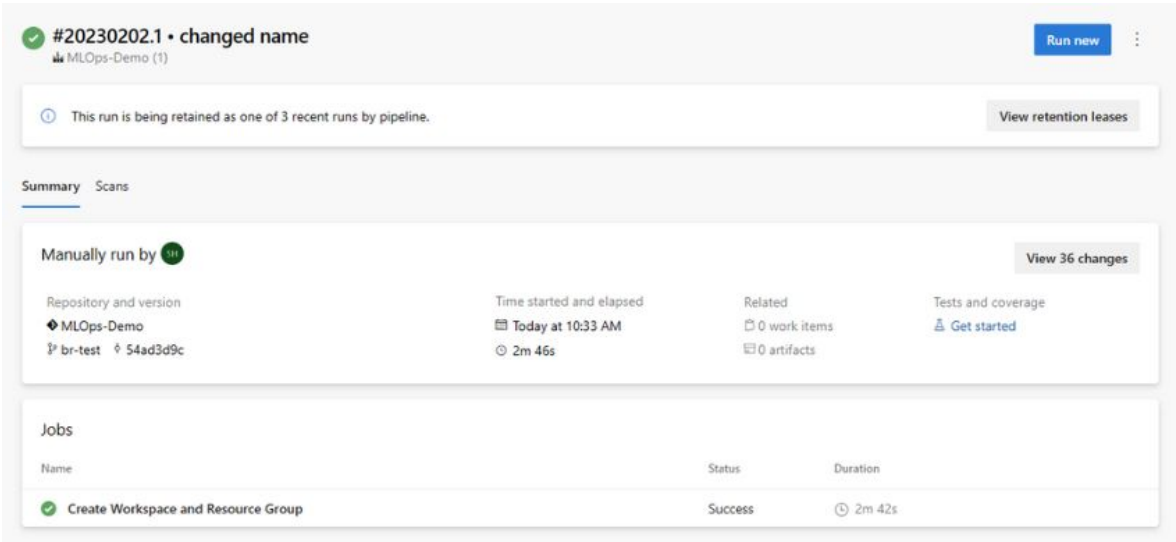


8. Select the `main` branch and choose `mlops/devops-pipelines/cli-ado-deploy-infra.yml`, then select **Continue**.

9. Run the pipeline; it will take a few minutes to finish. The pipeline should create the following artifacts:

    - Resource Group for your Workspace including Storage Account, Container Registry, Application Insights, Keyvault and the Azure Machine Learning Workspace itself.
    - In the workspace, there's also a compute cluster created.

10. Now the infrastructure for your MLOps project is deployed.



> ⓘ **Note**

> The **Unable move and reuse existing repository to required location** warnings may be ignored.

Sample Training and Deployment Scenario

The solution accelerator includes code and data for a sample end-to-end machine learning pipeline which runs a linear regression to predict taxi fares in NYC. The pipeline is made up of components, each serving different functions, which can be registered with the workspace, versioned, and reused with various inputs and outputs. Sample pipelines and workflows for the Computer Vision and NLP scenarios will have different steps and deployment steps.

This training pipeline contains the following steps:

Prepare Data

- This component takes multiple taxi datasets (yellow and green) and merges/filters the data, and prepare the train/val and evaluation datasets.
- Input: Local data under ./data/ (multiple .csv files)
- Output: Single prepared dataset (.csv) and train/val/test datasets.

Train Model

- This component trains a Linear Regressor with the training set.
- Input: Training dataset
- Output: Trained model (pickle format)

Evaluate Model

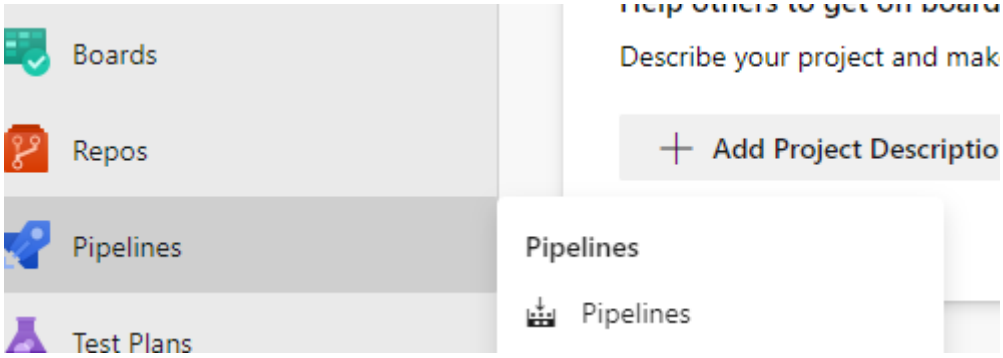
- This component uses the trained model to predict taxi fares on the test set.
- Input: ML model and Test dataset
- Output: Performance of model and a deploy flag whether to deploy or not.
- This component compares the performance of the model with all previous deployed models on the new test dataset and decides whether to promote or not model into production. Promoting model into production happens by registering the model in AML workspace.

Register Model

- This component scores the model based on how accurate the predictions are in the test set.
- Input: Trained model and the deploy flag.
- Output: Registered model in Azure Machine Learning.

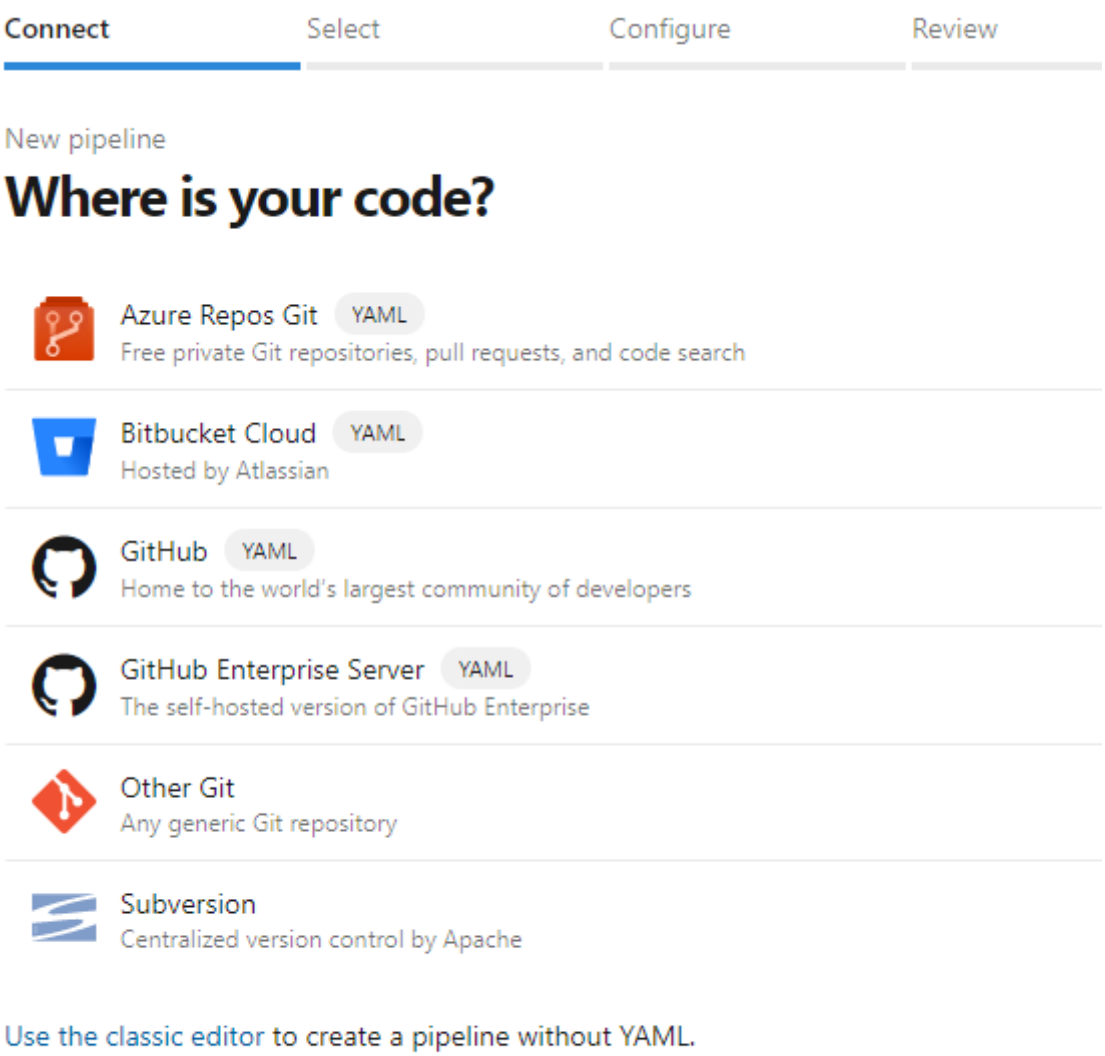
Deploying model training pipeline

1. Go to ADO pipelines



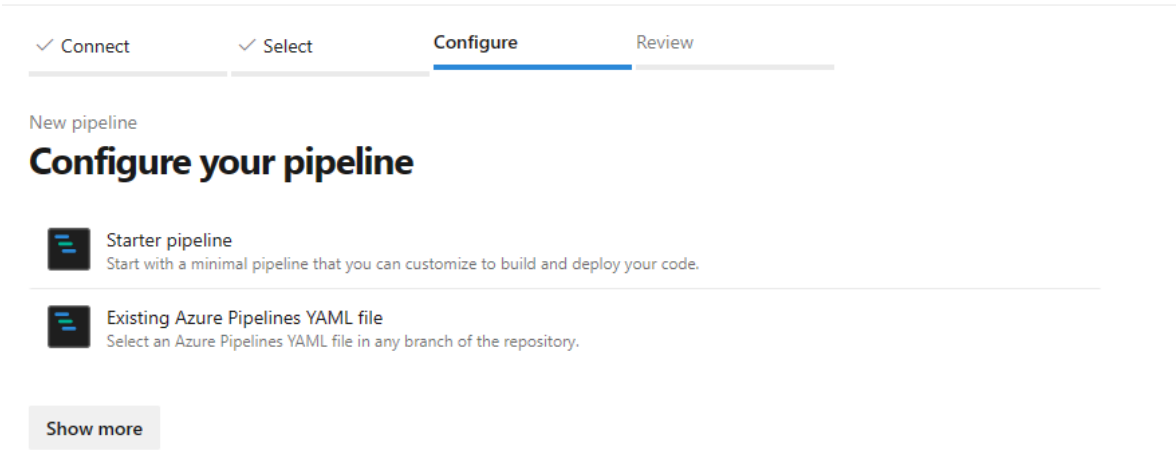
2. Select **New Pipeline**.

3. Select **Azure Repos Git**.



4. Select the repository that you cloned in from the previous section `mlopsv2`

5. Select **Existing Azure Pipeline YAML File**



6. Select `main` as a branch and choose `/mlops/devops-pipelines/deploy-model-training-pipeline.yml`, then select **Continue**.

7. **Save and Run** the pipeline

> **Note**
>
> At this point, the infrastructure is configured and the Prototyping Loop of the MLOps Architecture is deployed. you're ready to move to our trained model to production.

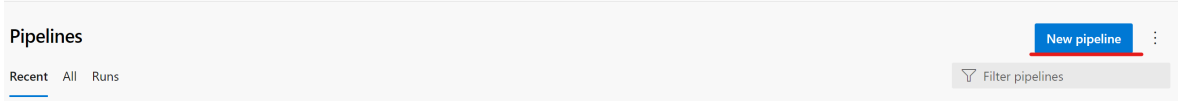
Deploying the Trained model

This scenario includes prebuilt workflows for two approaches to deploying a trained model, batch scoring or a deploying a model to an endpoint for real-time scoring. You may run either or both of these workflows to test the performance of the model in your Azure ML workspace. IN this example we will be using real-time scoring.

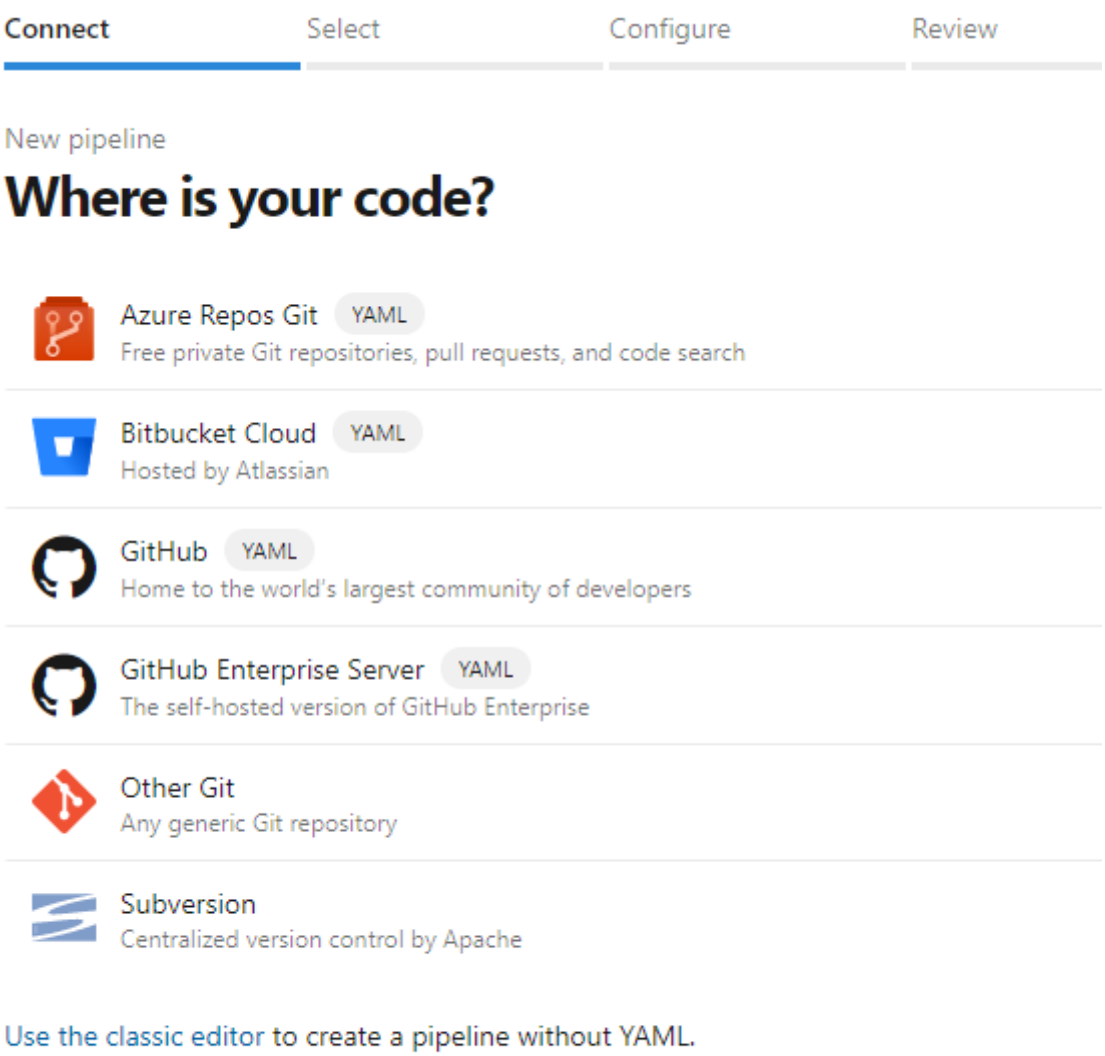
Deploy ML model endpoint

1. Go to ADO pipelines

2. Select **New Pipeline**.

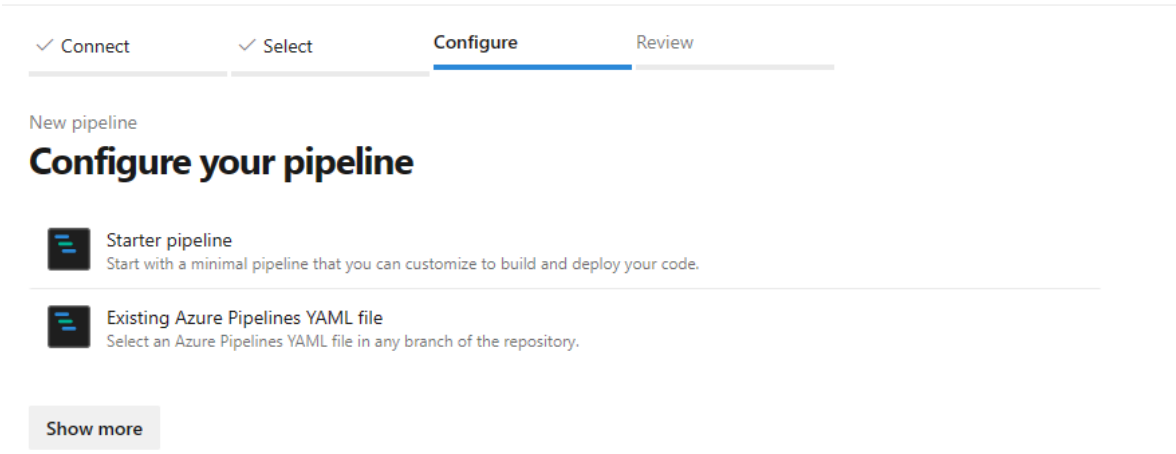


3. Select **Azure Repos Git**.



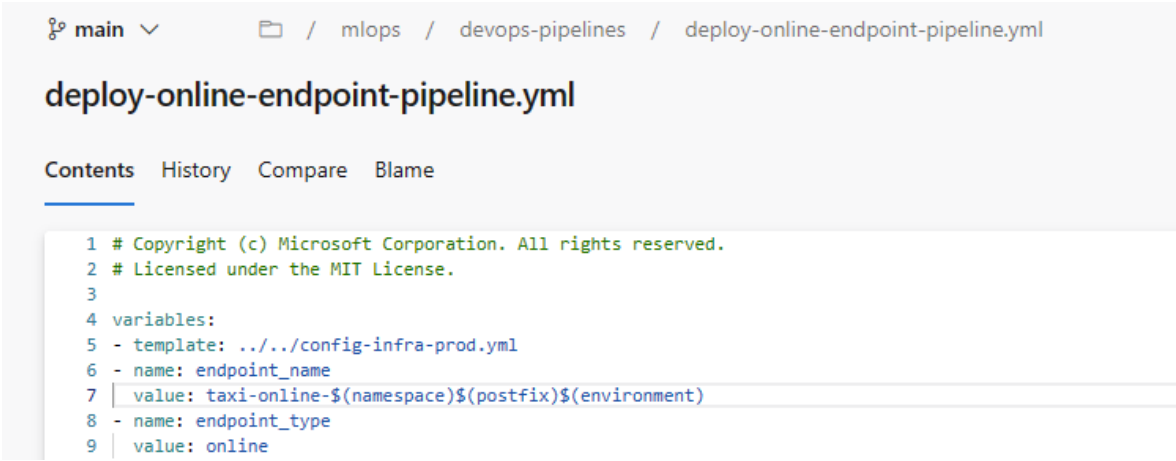
4. Select the repository that you cloned in from the previous section `mlopsv2`

5. Select **Existing Azure Pipeline YAML File**



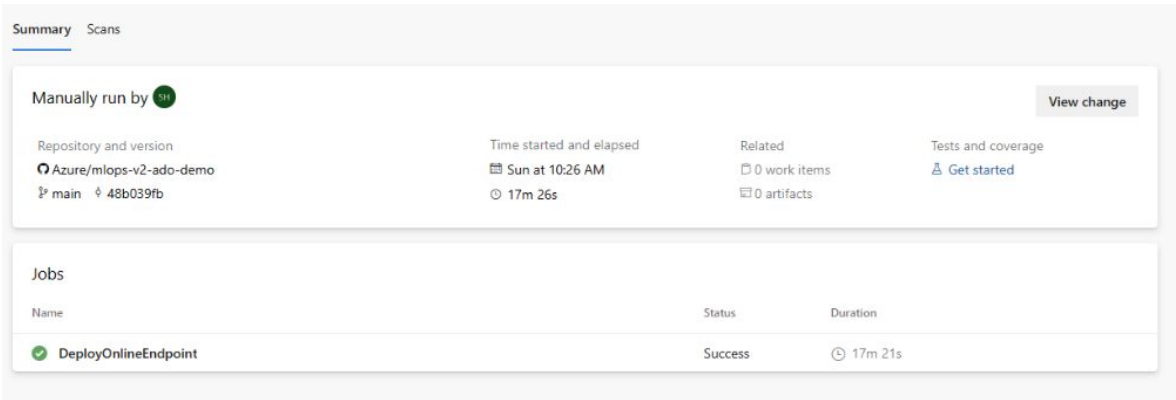
6. Select `main` as a branch and choose Managed Online Endpoint `/mlops/devops-pipelines/deploy-online-endpoint-pipeline.yml` then select **Continue**.

7. Online endpoint names need to be unique, so change `taxi-online-$(namespace)$(postfix)$(environment)` to another unique name and then select **Run**. No need to change the default if it doesn't fail.



> **ⓘ Important**
>
> If the run fails due to an existing online endpoint name, recreate the pipeline as described previously and change **[your endpoint-name]** to **[your endpoint-name (random number)]**

8. When the run completes, you'll see output similar to the following image:



9. To test this deployment, go to the **Endpoints** tab in your AzureML workspace, select the endpoint and click the **Test** Tab. You can use the sample input data located in the cloned repo at `/data/taxi-request.json` to test the endpoint.

Clean up resources

1. If you're not going to continue to use your pipeline, delete your Azure DevOps project.
2. In Azure portal, delete your resource group and Azure Machine Learning instance.

Next steps

- Install and set up Python SDK v2
- Install and set up Python CLI v2
- Azure MLOps (v2) solution accelerator on GitHub
- Learn more about Azure Pipelines with Azure Machine Learning
- Learn more about GitHub Actions with Azure Machine Learning
- Deploy MLOps on Azure in Less Than an Hour - Community MLOps V2 Accelerator video

Set up MLOps with GitHub

Article • 03/10/2023

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

Azure Machine Learning allows you to integrate with GitHub Actions to automate the machine learning lifecycle. Some of the operations you can automate are:

- Deployment of Azure Machine Learning infrastructure
- Data preparation (extract, transform, load operations)
- Training machine learning models with on-demand scale-out and scale-up
- Deployment of machine learning models as public or private web services
- Monitoring deployed machine learning models (such as for performance analysis)

In this article, you learn about using Azure Machine Learning to set up an end-to-end MLOps pipeline that runs a linear regression to predict taxi fares in NYC. The pipeline is made up of components, each serving different functions, which can be registered with the workspace, versioned, and reused with various inputs and outputs. you're going to be using the recommended Azure architecture for MLOps and Azure MLOps (v2) solution accelerator to quickly setup an MLOps project in Azure Machine Learning.

> **Tip**
>
> We recommend you understand some of the **recommended Azure architectures** for MLOps before implementing any solution. You'll need to pick the best architecture for your given Machine learning project.

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 Machine Learning.
- A Machine Learning workspace.
- Git running on your local machine.
- GitHub as the source control repository

> **Note**
>
> Git version 2.27 or newer is required. For more information on installing the Git command, see **https://git-scm.com/downloads** and select your operating system

> **Important**
>
> The CLI commands in this article were tested using Bash. If you use a different shell, you may encounter errors.

Set up authentication with Azure and DevOps

Before you can set up an MLOps project with Machine Learning, you need to set up authentication for Azure DevOps.

Create service principal

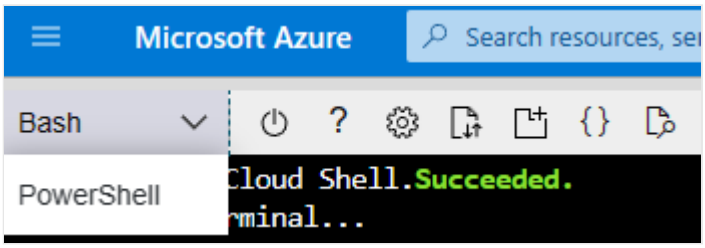
Create one Prod service principal for this demo. You can add more depending on how many environments, you want to work on (Dev or Prod or Both). Service principals can be created using one of the following methods:

Create from Azure Cloud Shell

1. Launch the Azure Cloud Shell.

   > **Tip**
   >
   > The first time you've launched the Cloud Shell, you'll be prompted to create a storage account for the Cloud Shell.

2. If prompted, choose **Bash** as the environment used in the Cloud Shell. You can also change environments in the drop-down on the top navigation bar



3. Copy the following bash commands to your computer and update the **projectName**, **subscriptionId**, and **environment** variables with the values for your project. This command will also grant the **Contributor** role to the service

principal in the subscription provided. This is required for GitHub Actions to properly use resources in that subscription.

Bash

```
projectName="<your project name>"
roleName="Contributor"
subscriptionId="<subscription Id>"
environment="<Prod>" #First letter should be capitalized
servicePrincipalName="Azure-ARM-${environment}-${projectName}"
# Verify the ID of the active subscription
echo "Using subscription ID $subscriptionID"
echo "Creating SP for RBAC with name $servicePrincipalName, with
role $roleName and in scopes     /subscriptions/$subscriptionId"
az ad sp create-for-rbac --name $servicePrincipalName --role
$roleName --scopes /subscriptions/$subscriptionId --sdk-auth
echo "Please ensure that the information created here is properly
save for future use."
```

4. Copy your edited commands into the Azure Shell and run them (**Ctrl** + **Shift** + **v**).

5. After running these commands, you'll be presented with information related to the service principal. Save this information to a safe location, you'll use it later in the demo to configure Azure DevOps.

JSON

```
  {
  "clientId": "<service principal client id>",
  "clientSecret": "<service principal client secret>",
  "subscriptionId": "<Azure subscription id>",
  "tenantId": "<Azure tenant id>",
  "activeDirectoryEndpointUrl":
"https://login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl":
"https://management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/"
  }
```

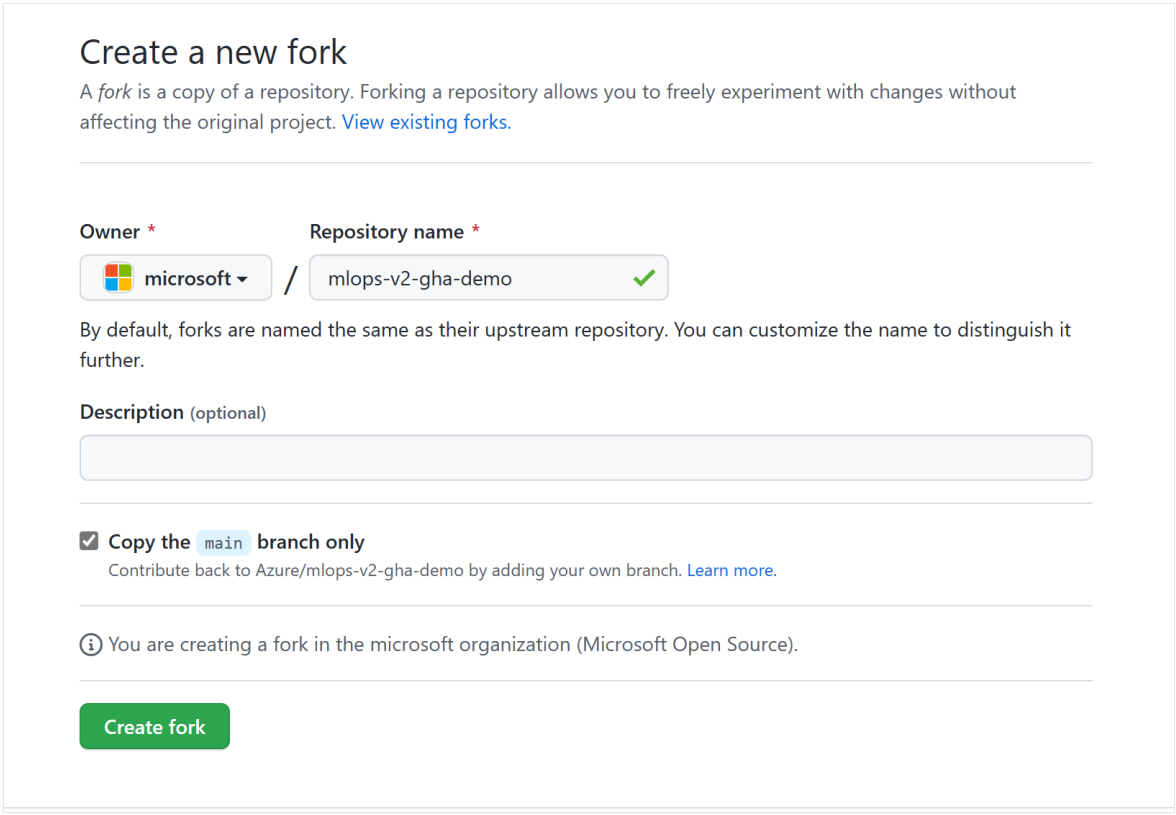
6. Copy all of this output, braces included. Save this information to a safe location, it will be use later in the demo to configure GitHub Repo.

7. Close the Cloud Shell once the service principals are created.

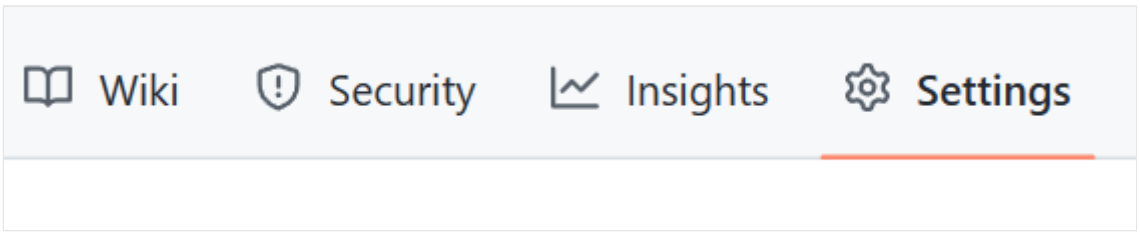
Set up GitHub repo

1. Fork the MLOps v2 Demo Template Repo in your GitHub organization

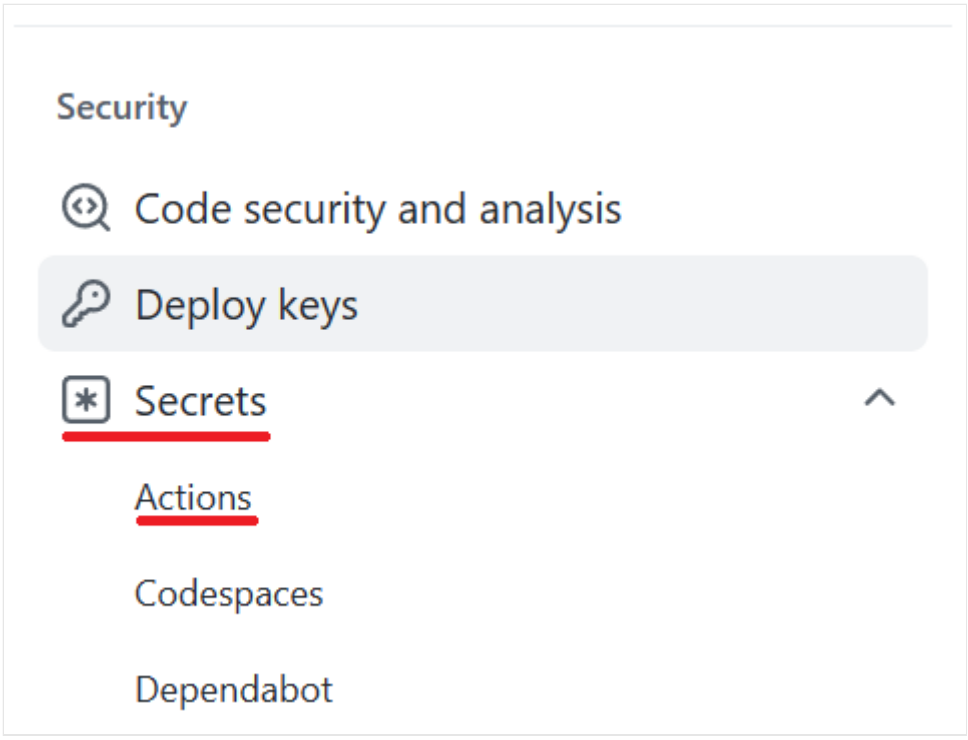
2. Go to https://github.com/Azure/mlops-v2-gha-demo/fork to fork the MLOps v2 demo repo into your GitHub org. This repo has reusable MLOps code that can be used across multiple projects.



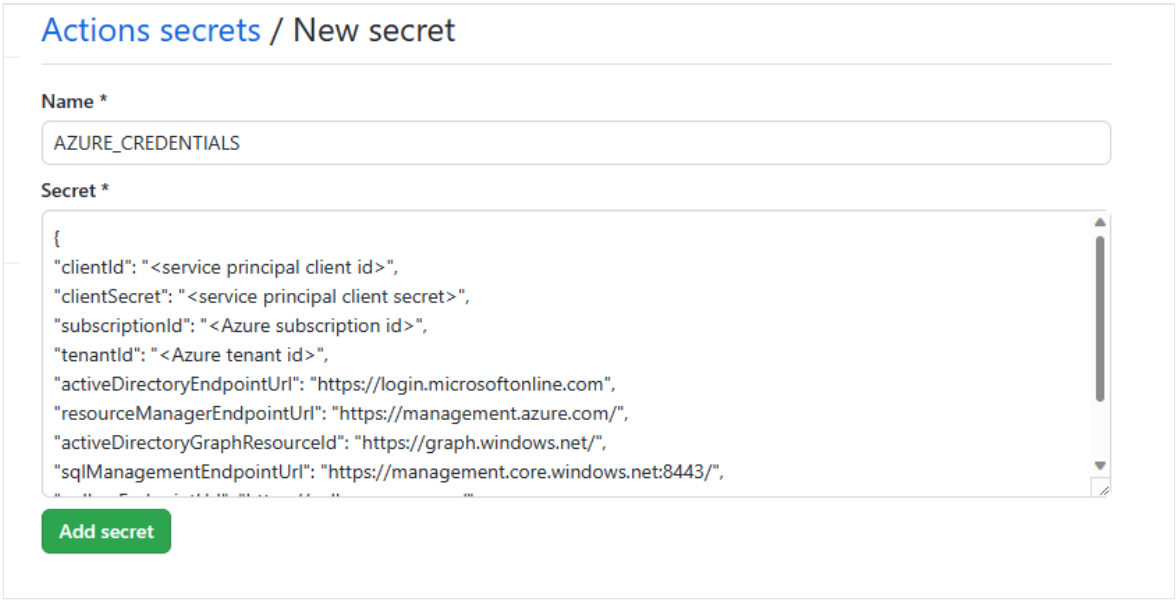
3. From your GitHub project, select **Settings**:



4. Then select **Secrets**, then **Actions**:

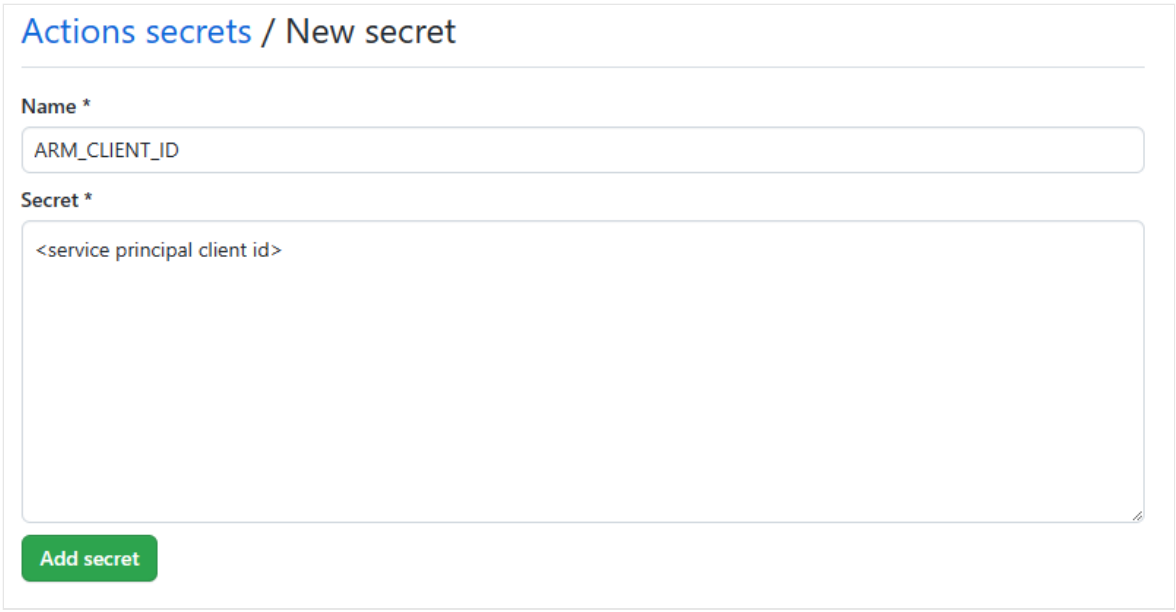


5. Select **New repository secret**. Name this secret **AZURE_CREDENTIALS** and paste the service principal output as the content of the secret. Select **Add secret**.



6. Add each of the following additional GitHub secrets using the corresponding values from the service principal output as the content of the secret:

   - **ARM_CLIENT_ID**
   - **ARM_CLIENT_SECRET**
   - **ARM_SUBSCRIPTION_ID**
   - **ARM_TENANT_ID**



> **Note**
>
> This finishes the prerequisite section and the deployment of the solution accelerator can happen accordingly.

Deploy machine learning project infrastructure with GitHub Actions

This step deploys the training pipeline to the Machine Learning workspace created in the previous steps.

> **Tip**
>
> Make sure you understand the **Architectural Patterns** of the solution accelerator before you checkout the MLOps v2 repo and deploy the infrastructure. In examples you'll use the **classical ML project type**.

Configure Machine Learning environment parameters

Go to your repository and select the `config-infra-prod.yml` file in the root. Change the following parameters to your liking, and then **commit** the changes.

This config file uses the namespace and postfix values the names of the artifacts to ensure uniqueness. Update the following section in the config to your liking. Default values and settings in the files are show below:

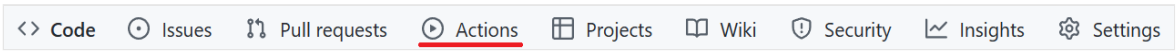
Bash

```
   namespace: mlopslite #Note: A namespace with many characters will cause
storage account creation to fail due to storage account names having a limit
of 24 characters.
   postfix: ao04
   location: westus

   environment: prod
   enable_aml_computecluster: true
   enable_aml_secure_workspace: true
   enable_monitoring: false
```

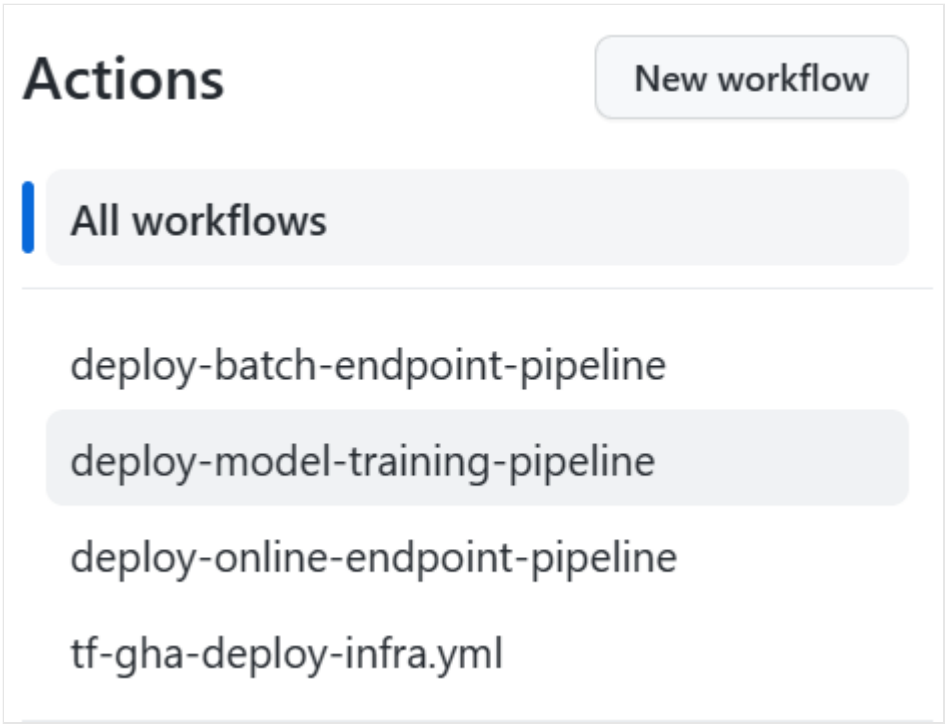
> **Note**
>
> If you are running a Deep Learning workload such as CV or NLP, ensure your GPU compute is available in your deployment zone. The enable_monitoring flag in these files defaults to False. Enabling this flag will add additional elements to the deployment to support Azure Machine Learning monitoring based on **https://github.com/microsoft/AzureML-Observability**. This will include an ADX cluster and increase the deployment time and cost of the MLOps solution.

Deploy Machine Learning infrastructure

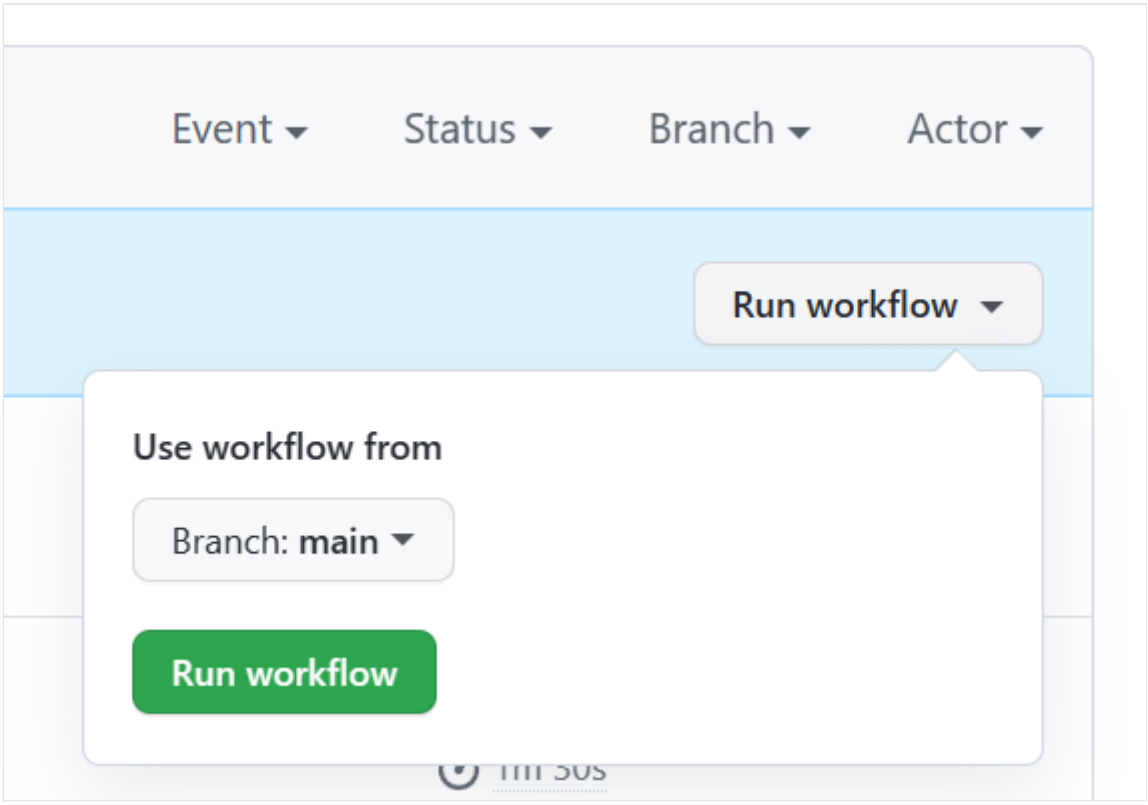
1. In your GitHub project repository (ex: taxi-fare-regression), select **Actions**



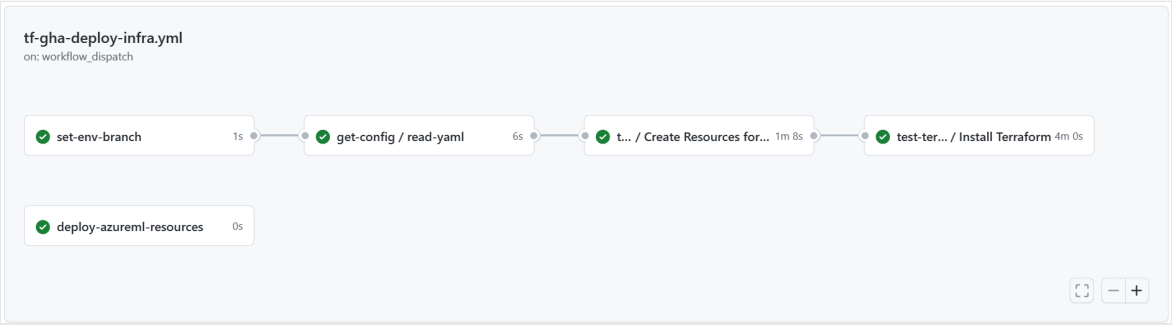
   This displays the pre-defined GitHub workflows associated with your project. For a classical machine learning project, the available workflows look similar to this:



2. Select would be **tf-gha-deploy-infra.yml**. This would deploy the Machine Learning infrastructure using GitHub Actions and Terraform.



3. On the right side of the page, select **Run workflow** and select the branch to run the workflow on. This may deploy Dev Infrastructure if you've created a dev branch or Prod infrastructure if deploying from main. Monitor the workflow for successful completion.



4. When the pipeline has complete successfully, you can find your Azure Machine Learning Workspace and associated resources by logging in to the Azure Portal. Next, a model training and scoring pipelines will be deployed into the new Machine Learning environment.

Sample Training and Deployment Scenario

The solution accelerator includes code and data for a sample end-to-end machine learning pipeline which runs a linear regression to predict taxi fares in NYC. The pipeline is made up of components, each serving different functions, which can be registered with the workspace, versioned, and reused with various inputs and outputs. Sample pipelines and workflows for the Computer Vision and NLP scenarios will have different steps and deployment steps.

This training pipeline contains the following steps:

Prepare Data

- This component takes multiple taxi datasets (yellow and green) and merges/filters the data, and prepare the train/val and evaluation datasets.
- Input: Local data under `./data/` (multiple .csv files)
- Output: Single prepared dataset (.csv) and train/val/test datasets.

Train Model

- This component trains a Linear Regressor with the training set.
- Input: Training dataset
- Output: Trained model (pickle format)

Evaluate Model

- This component uses the trained model to predict taxi fares on the test set.
- Input: ML model and Test dataset
- Output: Performance of model and a deploy flag whether to deploy or not.

- This component compares the performance of the model with all previous deployed models on the new test dataset and decides whether to promote or not model into production. Promoting model into production happens by registering the model in AML workspace.

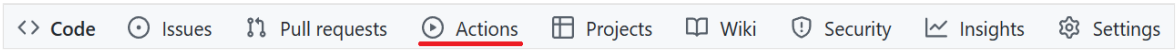
Register Model

- This component scores the model based on how accurate the predictions are in the test set.
- Input: Trained model and the deploy flag.
- Output: Registered model in Machine Learning.

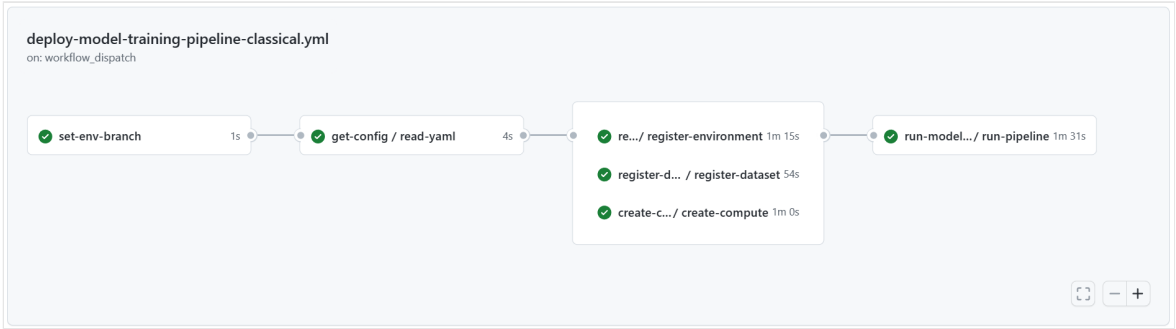
Deploying the Model Training Pipeline

Next, you will deploy the model training pipeline to your new Machine Learning workspace. This pipeline will create a compute cluster instance, register a training environment defining the necessary Docker image and python packages, register a training dataset, then start the training pipeline described in the last section. When the job is complete, the trained model will be registered in the Azure Machine Learning workspace and be available for deployment.

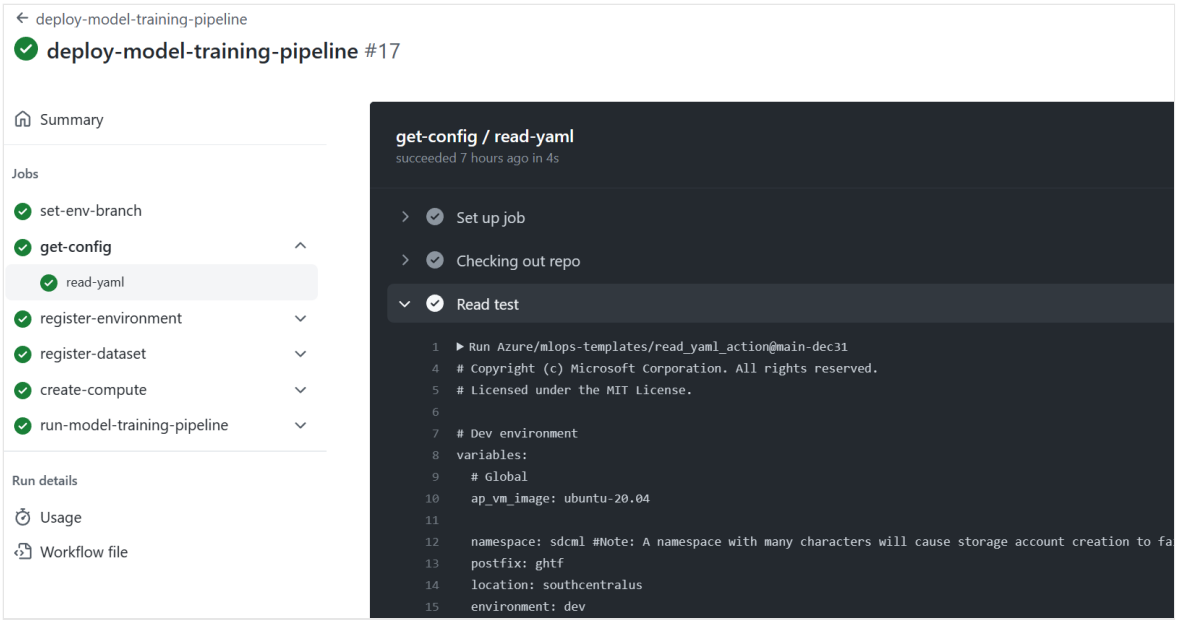
1. In your GitHub project repository (example: taxi-fare-regression), select **Actions**



2. Select the **deploy-model-training-pipeline** from the workflows listed on the left and the click **Run Workflow** to execute the model training workflow. This will take several minutes to run, depending on the compute size.



3. Once completed, a successful run will register the model in the Machine Learning workspace.



> **Note**
>
> If you want to check the output of each individual step, for example to view output of a failed run, click a job output, and then click each step in the job to view any output of that step.

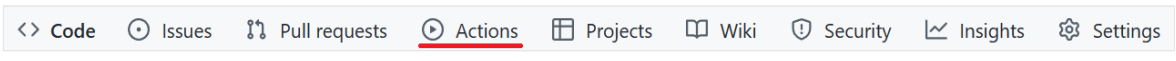
With the trained model registered in the Machine learning workspace, you are ready to deploy the model for scoring.

Deploying the Trained Model

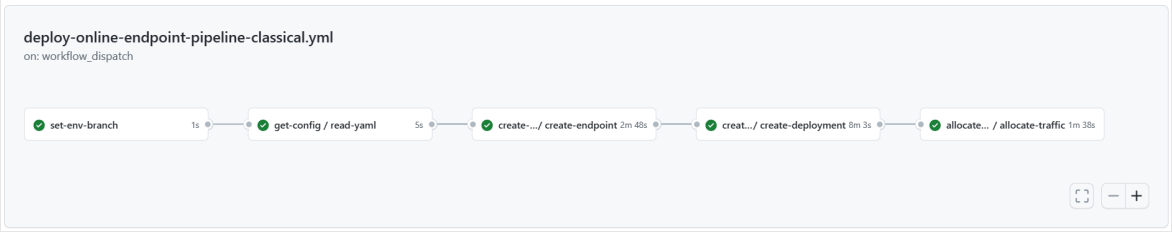
This scenario includes prebuilt workflows for two approaches to deploying a trained model, batch scoring or a deploying a model to an endpoint for real-time scoring. You may run either or both of these workflows to test the performance of the model in your Azure Machine Learning workspace.

Online Endpoint

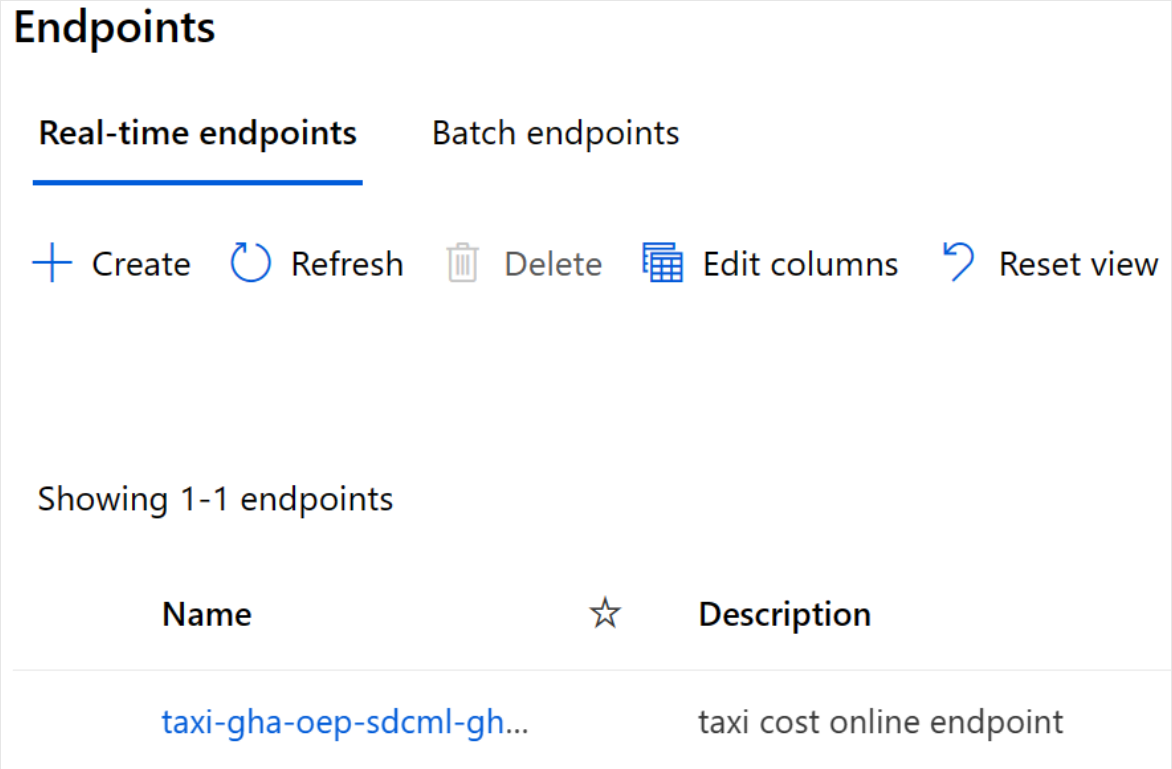
1. In your GitHub project repository (ex: taxi-fare-regression), select **Actions**



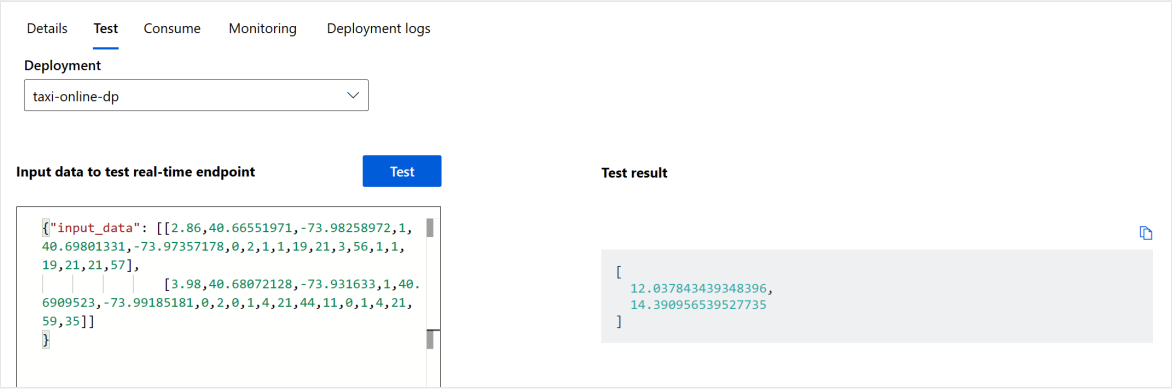
2. Select the **deploy-online-endpoint-pipeline** from the workflows listed on the left and click **Run workflow** to execute the online endpoint deployment pipeline workflow. The steps in this pipeline will create an online endpoint in your Machine Learning workspace, create a deployment of your model to this endpoint, then allocate traffic to the endpoint.



Once completed, you will find the online endpoint deployed in the Azure Machine Learning workspace and available for testing.

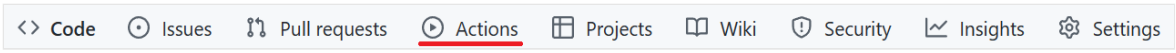


3. To test this deployment, go to the **Endpoints** tab in your Machine Learning workspace, select the endpoint and click the **Test** Tab. You can use the sample input data located in the cloned repo at `/data/taxi-request.json` to test the endpoint.

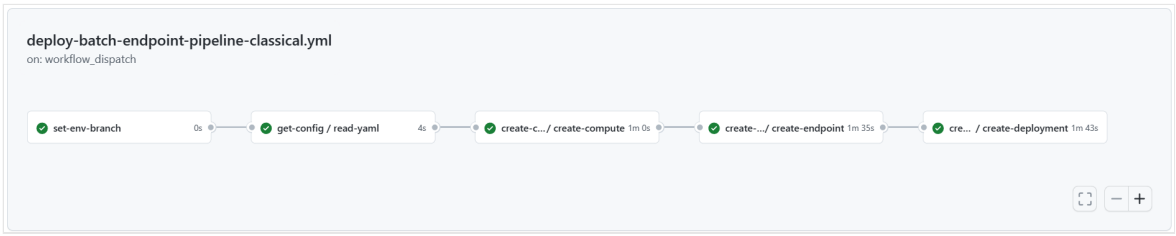


Batch Endpoint

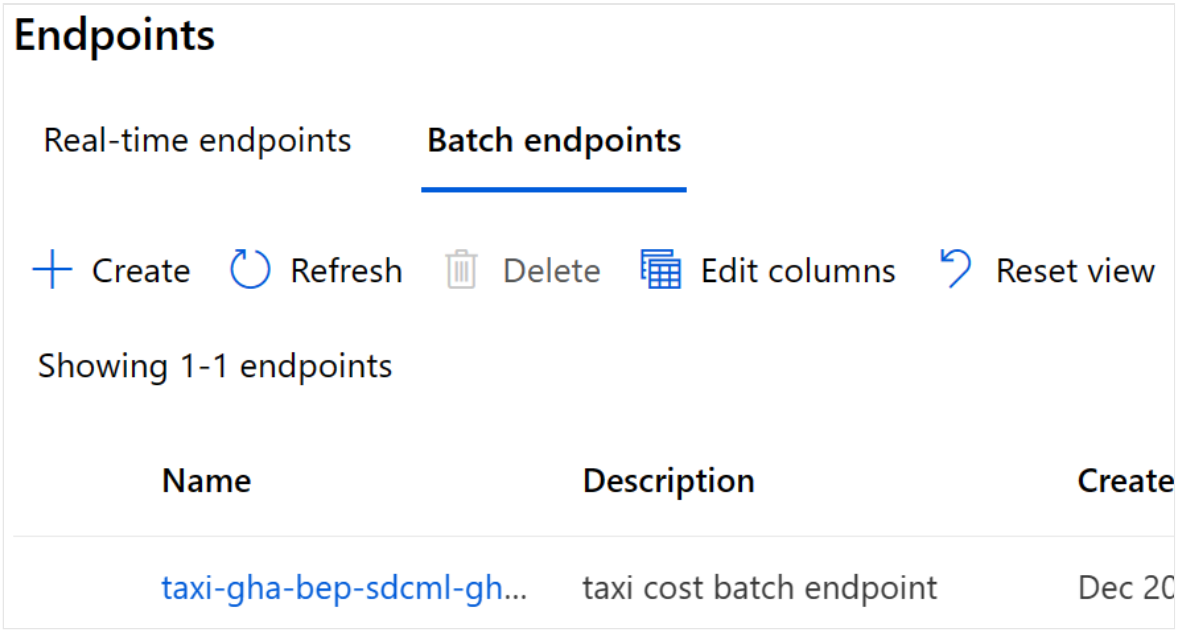
1. In your GitHub project repository (ex: taxi-fare-regression), select **Actions**



2. Select the **deploy-batch-endpoint-pipeline** from the workflows and click **Run workflow** to execute the batch endpoint deployment pipeline workflow. The steps in this pipeline will create a new AmlCompute cluster on which to execute batch scoring, create the batch endpoint in your Machine Learning workspace, then create a deployment of your model to this endpoint.



3. Once completed, you will find the batch endpoint deployed in the Azure Machine Learning workspace and available for testing.



Moving to production

Example scenarios can be trained and deployed both for Dev and Prod branches and environments. When you are satisfied with the performance of the model training pipeline, model, and deployment in Testing, Dev pipelines and models can be replicated and deployed in the Production environment.

The sample training and deployment Machine Learning pipelines and GitHub workflows can be used as a starting point to adapt your own modeling code and data.

Clean up resources

1. If you're not going to continue to use your pipeline, delete your Azure DevOps project.
2. In Azure portal, delete your resource group and Machine Learning instance.

Next steps

- Install and set up Python SDK v2
- Install and set up Python CLI v2
- Azure MLOps (v2) solution accelerator on GitHub
- Learn more about Azure Pipelines with Machine Learning
- Learn more about GitHub Actions with Machine Learning
- Deploy MLOps on Azure in Less Than an Hour - Community MLOps V2 Accelerator video

Set up end-to-end LLMOps with prompt flow and GitHub (preview)

Article • 09/13/2023

Azure Machine Learning allows you to integrate with GitHub Actions to automate the machine learning lifecycle. Some of the operations you can automate are:

- Running prompt flow after a pull request
- Running prompt flow evaluation to ensure results are high quality
- Registering of prompt flow models
- Deployment of prompt flow models

In this article, you learn about using Azure Machine Learning to set up an end-to-end LLMOps pipeline that runs a web classification flow that classifies a website based on a given URL. The flow is made up of multiple LLM calls and components, each serving different functions. All the LLMs used are managed and store in your Azure Machine Learning workspace in your Prompt flow connections.

> **Tip**
>
> We recommend you understand how we integrate **LLMOps with Prompt flow**.

> **Important**
>
> Prompt flow is currently in public preview. This preview is provided without a service-level agreement, and are not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see **Supplemental Terms of Use for Microsoft Azure Previews**.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Machine Learning.
- An Azure Machine Learning workspace.
- Git running on your local machine.
- GitHub as the source control repository.

> **Note**
>
> Git version 2.27 or newer is required. For more information on installing the Git command, see **https://git-scm.com/downloads** and select your operating system.

> **Important**
>
> The CLI commands in this article were tested using Bash. If you use a different shell, you may encounter errors.

Set up authentication with Azure and GitHub

Before you can set up a Prompt flow project with Azure Machine Learning, you need to set up authentication for Azure GitHub.

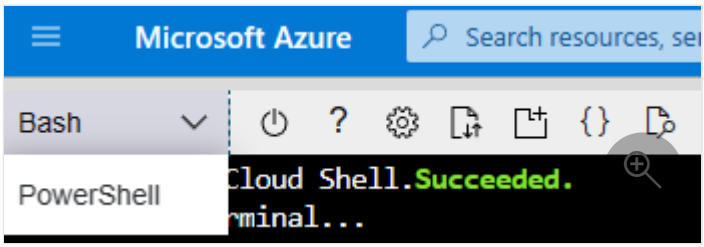
Create service principal

Create one production service principal for this demo. You can add more depending on how many environments, you want to work on (development, production or both). Service principals can be created using one of the following methods:

1. Launch the Azure Cloud Shell.

   > **Tip**
   >
   > The first time you've launched the Cloud Shell, you'll be prompted to create a storage account for the Cloud Shell.

2. If prompted, choose **Bash** as the environment used in the Cloud Shell. You can also change environments in the drop-down on the top navigation bar

3. Copy the following bash commands to your computer and update the **projectName**, **subscriptionId**, and **environment** variables with the values for your

project. This command will also grant the **Contributor** role to the service principal in the subscription provided. This is required for GitHub Actions to properly use resources in that subscription.

Bash

```
projectName="<your project name>"
roleName="Contributor"
subscriptionId="<subscription Id>"
environment="<Prod>" #First letter should be capitalized
servicePrincipalName="Azure-ARM-${environment}-${projectName}"
# Verify the ID of the active subscription
echo "Using subscription ID $subscriptionID"
echo "Creating SP for RBAC with name $servicePrincipalName, with role
$roleName and in scopes     /subscriptions/$subscriptionId"
az ad sp create-for-rbac --name $servicePrincipalName --role $roleName
--scopes /subscriptions/$subscriptionId --sdk-auth
echo "Please ensure that the information created here is properly save
for future use."
```

4. Copy your edited commands into the Azure Shell and run them (**Ctrl** + **Shift** + **v**).

5. After running these commands, you'll be presented with information related to the service principal. Save this information to a safe location, you'll use it later in the demo to configure GitHub.

JSON

```
  {
  "clientId": "<service principal client id>",
  "clientSecret": "<service principal client secret>",
  "subscriptionId": "<Azure subscription id>",
  "tenantId": "<Azure tenant id>",
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl":
"https://management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/"
  }
```

6. Copy all of this output, braces included. Save this information to a safe location, it will be use later in the demo to configure GitHub repo.

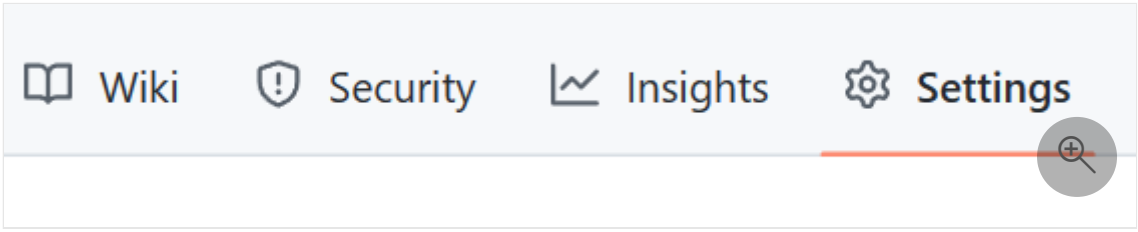
7. Close the Cloud Shell once the service principals are created.

Setup GitHub repo

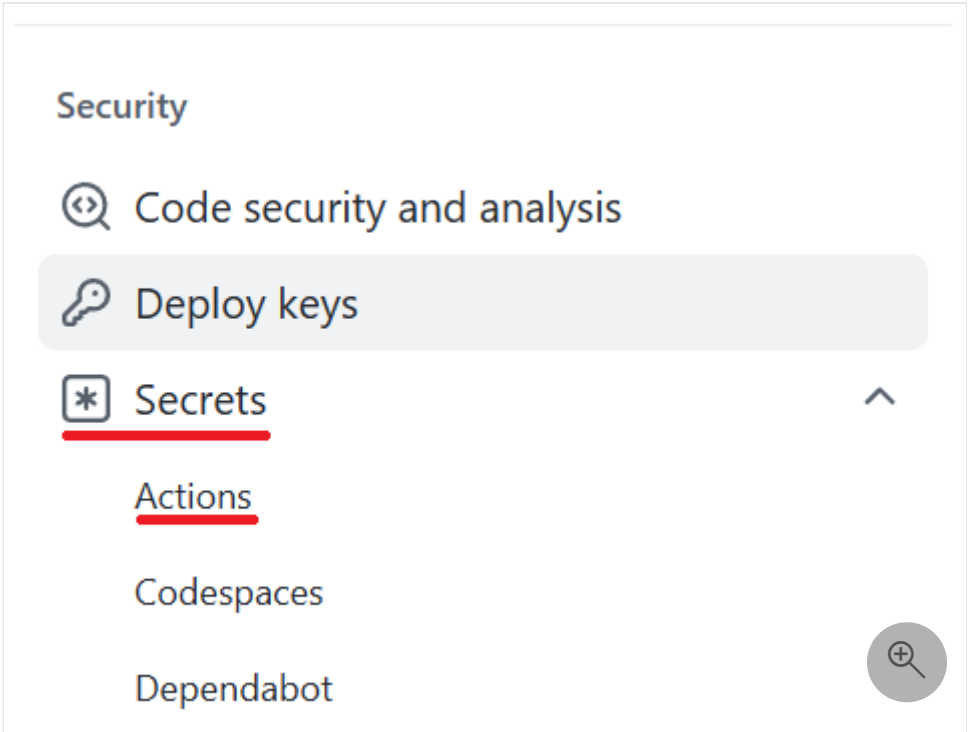
1. Fork example repo. LLMOps Demo Template Repo in your GitHub organization. This repo has reusable LLMOps code that can be used across multiple projects.

Add secret to GitHub repo

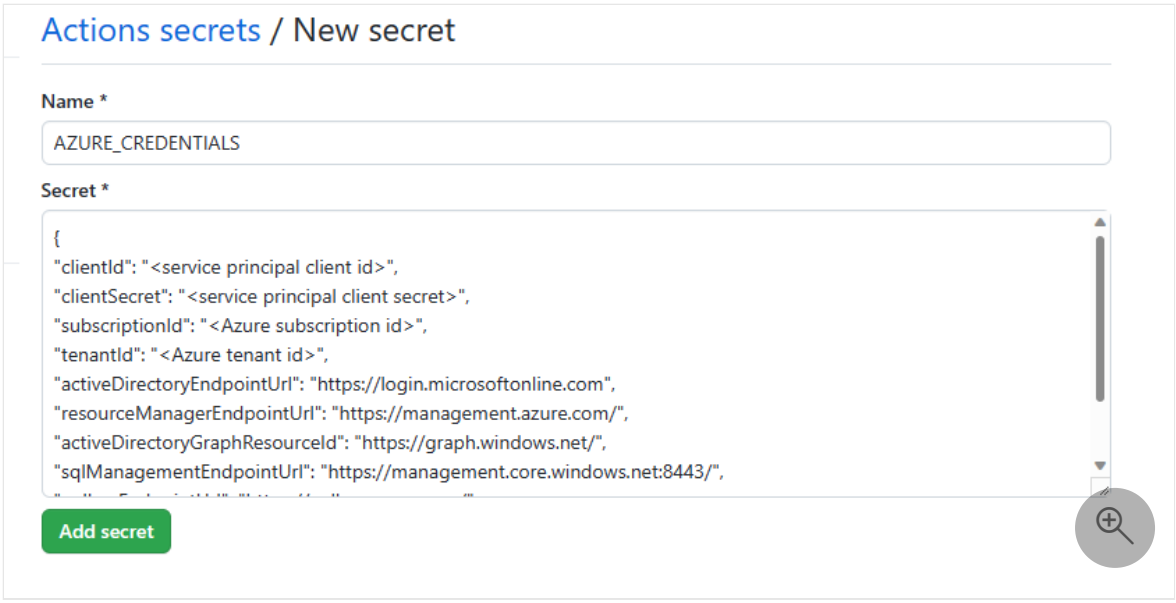
1. From your GitHub project, select **Settings**:



2. Then select **Secrets and variables**, then **Actions**:



3. Select **New repository secret**. Name this secret **AZURE_CREDENTIALS** and paste the service principal output as the content of the secret. Select **Add secret**.



4. Add each of the following additional GitHub secrets using the corresponding values from the service principal output as the content of the secret:

   - **GROUP**: <Resource Group Name>
   - **WORKSPACE**: <Azure ML Workspace Name>
   - **SUBSCRIPTION**: <Subscription ID>

Variable	Description
GROUP	Name of resource group
SUBSCRIPTION	Subscription ID of your workspace
WORKSPACE	Name of Azure Machine Learning workspace

> ⓘ **Note**
>
> This finishes the prerequisite section and the deployment of the solution accelerator can happen accordingly.

Setup connections for prompt flow

Connection helps securely store and manage secret keys or other sensitive credentials required for interacting with LLM and other external tools, for example, Azure Content Safety.

In this guide, we'll use flow `web-classification`, which uses connection `azure_open_ai_connection` inside, we need to set up the connection if we haven't added it before.

Go to workspace portal, select `Prompt flow` -> `Connections` -> `Create` -> `Azure OpenAI`, then follow the instruction to create your own connections. To learn more, see connections.

Setup runtime for Prompt flow

Prompt flow's runtime provides the computing resources required for the application to run, including a Docker image that contains all necessary dependency packages.

In this guide, we will use a runtime to run your prompt flow. You need to create your own Prompt flow runtime

Go to workspace portal, select **Prompt flow** -> **Runtime** -> **Add**, then follow the instruction to create your own connections

Setup variables with for prompt flow and GitHub Actions

Clone repo to your local machine.

```Bash
git clone https://github.com/<user-name>/llmops-gha-demo
```

Update workflow to connect to your Azure Machine Learning workspace

1. Update `run-eval-pf-pipeline.yml` and `deploy-pf-online-endpoint-pipeline.yml` to connect to your Azure Machine Learning workspace. You'll need to update the CLI setup file variables to match your workspace.

2. In your cloned repository, go to `.github/workflow/`.

3. Verify `env` section in the `run-eval-pf-pipeline.yml` and `deploy-pf-online-endpoint-pipeline.yml` refers to the workspace secrets you added in the previous step.

Update run.yml with your connections and runtime

You'll use a `run.yml` file to deploy your Azure Machine Learning pipeline. This is a flow run definition. You only need to make this update if you're using a name other than `pf-runtime` for your prompt flow runtime. You'll also need to update all the `connections` to match the connections in your Azure Machine Learning workspace and `deployment_name` to match the name of your GPT 3.5 Turbo deployment associate with that connection.

1. In your cloned repository, open `web-classification/run.yml` and `web-classification/run_evaluation.yml`
2. Each time you see `runtime: <runtime-name>`, update the value of `<runtime-name>` with your runtime name.
3. Each time you see `connection: Default_AzureOpenAI`, update the value of `Default_AzureOpenAI` to match the connection name in your Azure Machine Learning workspace.
4. Each time you see `deployment_name: gpt-35-turbo-0301`, update the value of `gpt-35-turbo-0301` with the name of your GPT 3.5 Turbo deployment associate with that connection.

Sample prompt run, evaluation and deployment scenario

This is a flow demonstrating multi-class classification with LLM. Given an url, it will classify the url into one web category with just a few shots, simple summarization and classification prompts.

This training pipeline contains the following steps:

Run prompts in flow:

- Compose a classification flow with LLM.
- Feed few shots to LLM classifier.
- Upload prompt test dataset.
- Bulk run prompt flow based on dataset.

Evaluate results:

- Upload ground test dataset
- Evaluation of the bulk run result and new uploaded ground test dataset

Register prompt flow LLM app:

- Check in logic, Customer defined logic (accuracy rate, if >=90% you can deploy)

Deploy and test LLM app:

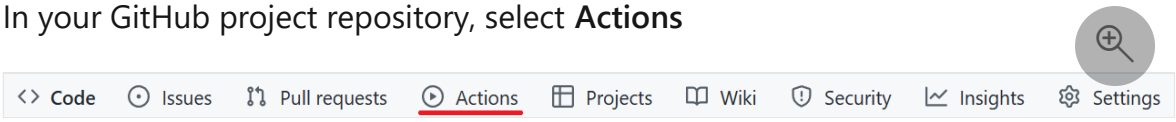
- Deploy the PF as a model to production
- Test the model/prompt flow real-time endpoint.

Run and evaluate prompt flow in Azure Machine Learning with GitHub Actions

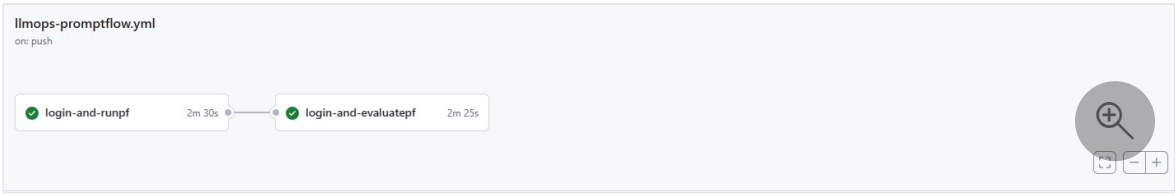
Using a GitHub Action workflow we'll trigger actions to run a Prompt Flow job in Azure Machine Learning.

This pipeline will start the prompt flow run and evaluate the results. When the job is complete, the prompt flow model will be registered in the Azure Machine Learning workspace and be available for deployment.

1. In your GitHub project repository, select **Actions**



2. Select the `run-eval-pf-pipeline.yml` from the workflows listed on the left and the select **Run Workflow** to execute the Prompt flow run and evaluate workflow. This will take several minutes to run.

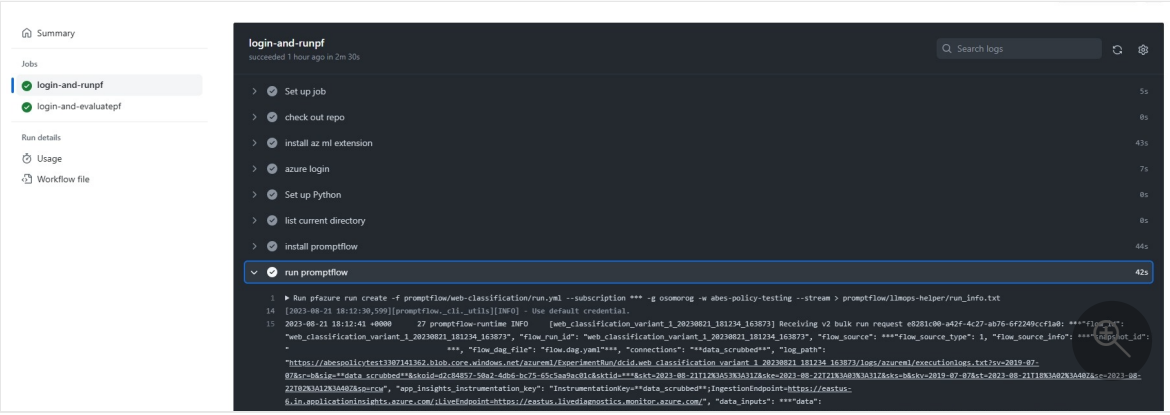


3. The workflow will only register the model for deployment, if the accuracy of the classification is greater than 60%. You can adjust the accuracy threshold in the `run-eval-pf-pipeline.yml` file in the `jobMetricAssert` section of the workflow file. The section should look like:

```yaml
id: jobMetricAssert
run: |
    export ASSERT=$(python promptflow/llmops-helper/assert.py
result.json 0.6)
```

You can update the current `0.6` number to fit your preferred threshold.

4. Once completed, a successful run and all test were passed, it will register the Prompt Flow model in the Azure Machine Learning workspace.



> **Note**
>
> If you want to check the output of each individual step, for example to view output of a failed run, select a job output, and then select each step in the job to view any output of that step.

With the Prompt flow model registered in the Azure Machine Learning workspace, you're ready to deploy the model for scoring.

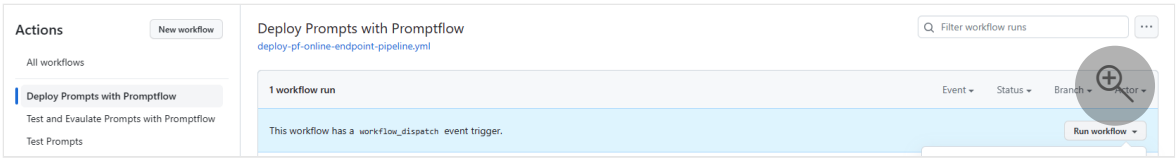
Deploy prompt flow in Azure Machine Learning with GitHub Actions

This scenario includes prebuilt workflows for deploying a model to an endpoint for real-time scoring. You may run the workflow to test the performance of the model in your Azure Machine Learning workspace.

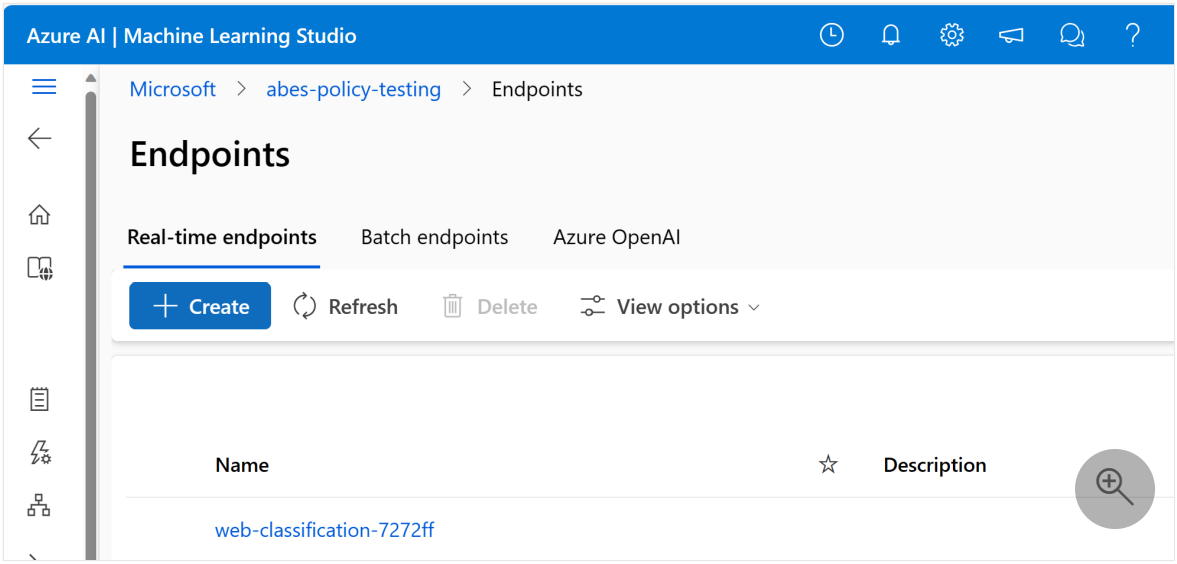
Online endpoint

1. In your GitHub project repository, select **Actions**.

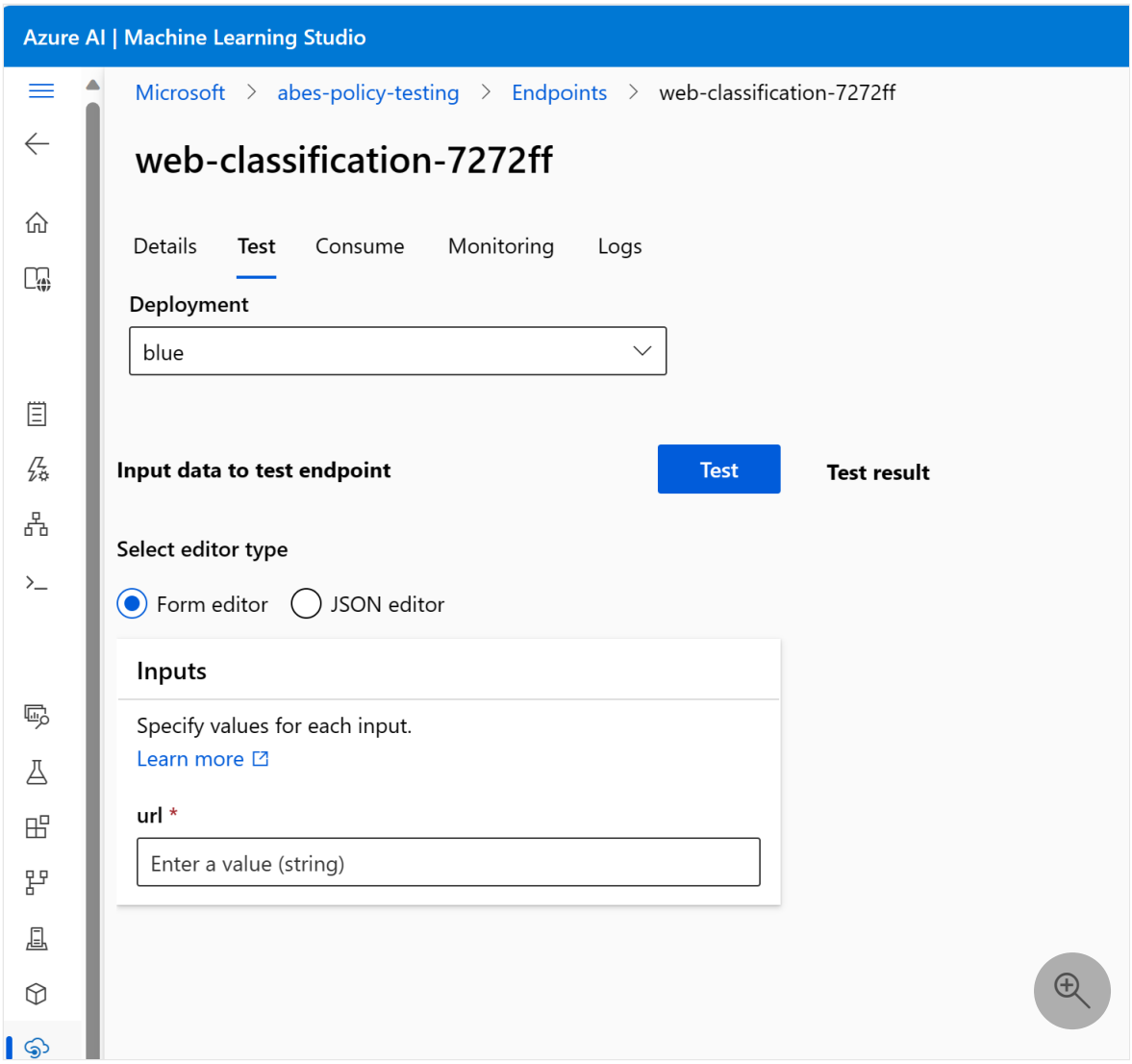
2. Select the **deploy-pf-online-endpoint-pipeline** from the workflows listed on the left and select **Run workflow** to execute the online endpoint deployment pipeline workflow. The steps in this pipeline will create an online endpoint in your Azure Machine Learning workspace, create a deployment of your model to this endpoint, then allocate traffic to the endpoint.



3. Once completed, you'll find the online endpoint deployed in the Azure Machine Learning workspace and available for testing.



4. To test this deployment, go to the **Endpoints** tab in your Azure Machine Learning workspace, select the endpoint and select the **Test** tab. You can use the sample input data located in the cloned repo at `/deployment/sample-request.json` to test the endpoint.



> **Note**
>
> Make sure you have already **granted permissions to the endpoint** before you test or consume the endpoint.

Moving to production

This example scenario can be run and deployed both for Dev and production branches and environments. When you're satisfied with the performance of the prompt evaluation pipeline, Prompt Flow model, deployment in testing, development pipelines, and models can be replicated and deployed in the production environment.

The sample Prompt flow run and evaluation and GitHub workflows can be used as a starting point to adapt your own prompt engineering code and data.

Clean up resources

1. If you're not going to continue to use your pipeline, delete your GitHub project.
2. In Azure portal, delete your resource group and Azure Machine Learning instance.

Next steps

- Install and set up Python SDK v2
- Install and set up Python CLI v2