Manage Azure Machine Learning environments with the CLI & SDK (v2)

Article • 10/17/2022 • 10 minutes to read

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

Select the version of Azure Machine Learning SDK or CLI extension you are using:

Azure Machine Learning environments define the execution environments for your jobs or deployments and encapsulate the dependencies for your code. Azure ML uses the environment specification to create the Docker container that your training or scoring code runs in on the specified compute target. You can define an environment from a conda specification, Docker image, or Docker build context.

In this article, learn how to create and manage Azure ML environments using the SDK & CLI (v2).

Prerequisites

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

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning .
- An Azure Machine Learning workspace. If you don't have one, use the steps in the Quickstart: Create workspace resources article to create one.
- The Azure CLI and the `ml` extension **or** the Azure Machine Learning Python SDK v2:
  - 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**.

- To install the Python SDK v2, use the following command:

```
Bash

pip install azure-ai-ml
```

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

> **Tip**
>
> For a full-featured development environment, use Visual Studio Code and the **Azure Machine Learning extension** to **manage Azure Machine Learning resources** and **train machine learning models**.

Clone examples repository

To run the training examples, first clone the examples repository. For the CLI examples, change into the `cli` directory. For the SDK examples, change into the `SDK` directory:

```
Azure CLI

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

Note that `--depth 1` clones only the latest commit to the repository, which reduces time to complete the operation.

Connect to the workspace

> **Tip**

Use the tabs below to select the method you want to use to work with environments. Selecting a tab will automatically switch all the tabs in this article to the same tab. You can select another tab at any time.

Azure CLI

When using the Azure CLI, you need identifier parameters - a subscription, resource group, and workspace name. While you can specify these parameters for each command, you can also set defaults that will be used for all the commands. Use the following commands to set default values. Replace `<subscription ID>`, `<AzureML workspace name>`, and `<resource group>` with the values for your configuration:

Azure CLI

```
az account set --subscription <subscription ID>
az configure --defaults workspace=<AzureML workspace name> group=<re-
source group>
```

Curated environments

There are two types of environments in Azure ML: curated and custom environments. Curated environments are predefined environments containing popular ML frameworks and tooling. Custom environments are user-defined and can be created via `az ml environment create`.

Curated environments are provided by Azure ML and are available in your workspace by default. Azure ML routinely updates these environments with the latest framework version releases and maintains them for bug fixes and security patches. They're backed by cached Docker images, which reduce job preparation cost and model deployment time.

You can use these curated environments out of the box for training or deployment by referencing a specific environment using the `azureml:<curated-environment-name>:<version>` or `azureml:<curated-environment-name>@latest` syntax. You can also use them as reference for your own custom environments by modifying the Dockerfiles that back these curated environments.

You can see the set of available curated environments in the Azure ML studio UI, or by using the CLI (v2) via `az ml environments list`.

Create an environment

You can define an environment from a Docker image, a Docker build context, and a conda specification with Docker image.

Create an environment from a Docker image

To define an environment from a Docker image, provide the image URI of the image hosted in a registry such as Docker Hub or Azure Container Registry.

Azure CLI

The following example is a YAML specification file for an environment defined from a Docker image. An image from the official PyTorch repository on Docker Hub is specified via the `image` property in the YAML file.

YAML

```yaml
$schema: https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: docker-image-example
image: pytorch/pytorch:latest
description: Environment created from a Docker image.
```

To create the environment:

cli

```
az ml environment create --file assets/environment/docker-image.yml
```

Tip

Azure ML maintains a set of CPU and GPU Ubuntu Linux-based base images with common system dependencies. For example, the GPU images contain Miniconda, OpenMPI, CUDA, cuDNN, and NCCL. You can use these images for your

environments, or use their corresponding Dockerfiles as reference when building your own custom images.

For the set of base images and their corresponding Dockerfiles, see the **AzureML-Containers repo** .

Create an environment from a Docker build context

Instead of defining an environment from a prebuilt image, you can also define one from a Docker build context . To do so, specify the directory that will serve as the build context. This directory should contain a Dockerfile and any other files needed to build the image.

Azure CLI

The following example is a YAML specification file for an environment defined from a build context. The local path to the build context folder is specified in the `build.path` field, and the relative path to the Dockerfile within that build context folder is specified in the `build.dockerfile_path` field. If `build.dockerfile_path` is omitted in the YAML file, Azure ML will look for a Dockerfile named `Dockerfile` at the root of the build context.

In this example, the build context contains a Dockerfile named `Dockerfile` and a `requirements.txt` file that is referenced within the Dockerfile for installing Python packages.

YAML

```yaml
$schema: https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: docker-context-example
build:
  path: docker-contexts/python-and-pip
```

To create the environment:

cli

```
az ml environment create --file assets/environment/docker-context.yml
```

Azure ML will start building the image from the build context when the environment is created. You can monitor the status of the build and view the build logs in the studio UI.

Create an environment from a conda specification

You can define an environment using a standard conda YAML configuration file that includes the dependencies for the conda environment. See Creating an environment manually for information on this standard format.

You must also specify a base Docker image for this environment. Azure ML will build the conda environment on top of the Docker image provided. If you install some Python dependencies in your Docker image, those packages won't exist in the execution environment thus causing runtime failures. By default, Azure ML will build a Conda environment with dependencies you specified, and will execute the job in that environment instead of using any Python libraries that you installed on the base image.

Azure CLI

The following example is a YAML specification file for an environment defined from a conda specification. Here the relative path to the conda file from the Azure ML environment YAML file is specified via the `conda_file` property. You can alternatively define the conda specification inline using the `conda_file` property, rather than defining it in a separate file.

YAML

```
$schema: https://azuremlschemas.azureedge.net/latest/environment.schema.json
name: docker-image-plus-conda-example
image: mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04
conda_file: conda-yamls/pydata.yml
description: Environment created from a Docker image plus Conda environment.
```

To create the environment:

cli

```
az ml environment create --file assets/environment/docker-image-plus-conda.yml
```

Azure ML will build the final Docker image from this environment specification when the environment is used in a job or deployment. You can also manually trigger a build of the environment in the studio UI.

Manage environments

The SDK and CLI (v2) also allow you to manage the lifecycle of your Azure ML environment assets.

List

List all the environments in your workspace:

Azure CLI

cli

```
az ml environment list
```

List all the environment versions under a given name:

Azure CLI

cli

```
az ml environment list --name docker-image-example
```

Show

Get the details of a specific environment:

Azure CLI

```cli
az ml environment list --name docker-image-example --version 1
```

Update

Update mutable properties of a specific environment:

Azure CLI

```cli
az ml environment update --name docker-image-example --version 1 --set
description="This is an updated description."
```

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

Archive

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

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

Archive all versions of an environment:

Azure CLI

```cli
az ml environment archive --name docker-image-example
```

Archive a specific environment version:

Azure CLI

```cli
az ml environment archive --name docker-image-example --version 1
```

Use environments for training

Azure CLI

To use an environment for a training job, specify the `environment` field of the job YAML configuration. You can either reference an existing registered Azure ML environment via `environment: azureml:<environment-name>:<environment-version>` or `environment: azureml:<environment-name>@latest` (to reference the latest version of an environment), or define an environment specification inline. If defining an environment inline, don't specify the `name` and `version` fields, as these environments are treated as "unregistered" environments and aren't tracked in your environment asset registry.

When you submit a training job, the building of a new environment can take several minutes. The duration depends on the size of the required dependencies. The environments are cached by the service. So as long as the environment definition remains unchanged, you incur the full setup time only once.

For more information on how to use environments in jobs, see Train models.

Use environments for model deployments

Azure CLI

You can also use environments for your model deployments for both online and batch scoring. To do so, specify the `environment` field in the deployment YAML configuration.

For more information on how to use environments in deployments, see Deploy and score a machine learning model by using a managed online endpoint.

Next steps

- Train models (create jobs)
- Deploy and score a machine learning model by using a managed online endpoint
- Environment YAML schema reference