Configure and submit training jobs

Article • 10/12/2022 • 10 minutes to read

APPLIES TO: Python SDK azureml **v1**

In this article, you learn how to configure and submit Azure Machine Learning jobs to train your models. Snippets of code explain the key parts of configuration and submission of a training script. Then use one of the example notebooks to find the full end-to-end working examples.

When training, it is common to start on your local computer, and then later scale out to a cloud-based cluster. With Azure Machine Learning, you can run your script on various compute targets without having to change your training script.

All you need to do is define the environment for each compute target within a **script job configuration**. Then, when you want to run your training experiment on a different compute target, specify the job configuration for that compute.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning today
- The Azure Machine Learning SDK for Python (>= 1.13.0)
- An Azure Machine Learning workspace, `ws`
- A compute target, `my_compute_target`. Create a compute target

What's a script run configuration?

A ScriptRunConfig is used to configure the information necessary for submitting a training job as part of an experiment.

You submit your training experiment with a ScriptRunConfig object. This object includes the:

- **source_directory**: The source directory that contains your training script
- **script**: The training script to run
- **compute_target**: The compute target to run on

- **environment**: The environment to use when running the script
- and some additional configurable options (see the reference documentation for more information)

Train your model

The code pattern to submit a training job is the same for all types of compute targets:

1. Create an experiment to run
2. Create an environment where the script will run
3. Create a ScriptRunConfig, which specifies the compute target and environment
4. Submit the job
5. Wait for the job to complete

Or you can:

- Submit a HyperDrive run for hyperparameter tuning.
- Submit an experiment via the VS Code extension.

Create an experiment

Create an experiment in your workspace. An experiment is a light-weight container that helps to organize job submissions and keep track of code.

Python

```
from azureml.core import Experiment

experiment_name = 'my_experiment'
experiment = Experiment(workspace=ws, name=experiment_name)
```

Select a compute target

Select the compute target where your training script will run on. If no compute target is specified in the ScriptRunConfig, or if `compute_target='local'`, Azure ML will execute your script locally.

The example code in this article assumes that you have already created a compute

target `my_compute_target` from the "Prerequisites" section.

> **Note**
>
> Azure Databricks is not supported as a compute target for model training. You can use Azure Databricks for data preparation and deployment tasks.

> **Note**
>
> To create and attach a compute target for training on Azure Arc-enabled Kubernetes cluster, see **Configure Azure Arc-enabled Machine Learning**

Create an environment

Azure Machine Learning environments are an encapsulation of the environment where your machine learning training happens. They specify the Python packages, Docker image, environment variables, and software settings around your training and scoring scripts. They also specify runtimes (Python, Spark, or Docker).

You can either define your own environment, or use an Azure ML curated environment. Curated environments are predefined environments that are available in your workspace by default. These environments are backed by cached Docker images which reduce the job preparation cost. See Azure Machine Learning Curated Environments for the full list of available curated environments.

For a remote compute target, you can use one of these popular curated environments to start with:

Python

```
from azureml.core import Workspace, Environment

ws = Workspace.from_config()
myenv = Environment.get(workspace=ws, name="AzureML-Minimal")
```

For more information and details about environments, see Create & use software environments in Azure Machine Learning.

Local compute target

If your compute target is your **local machine**, you are responsible for ensuring that all the necessary packages are available in the Python environment where the script runs. Use `python.user_managed_dependencies` to use your current Python environment (or the Python on the path you specify).

Python

```
from azureml.core import Environment

myenv = Environment("user-managed-env")
myenv.python.user_managed_dependencies = True

# You can choose a specific Python environment by pointing to a Python path
# myenv.python.interpreter_path = '/home/johndoe/miniconda3/envs/myenv
/bin/python'
```

Create the script job configuration

Now that you have a compute target (`my_compute_target`, see Prerequisites and environment (`myenv`, see Create an environment), create a script job configuration that runs your training script (`train.py`) located in your `project_folder` directory:

Python

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory=project_folder,
                      script='train.py',
                      compute_target=my_compute_target,
                      environment=myenv)

# Set compute target
# Skip this if you are running on your local computer
script_run_config.run_config.target = my_compute_target
```

If you do not specify an environment, a default environment will be created for you.

If you have command-line arguments you want to pass to your training script, you can specify them via the **`arguments`** parameter of the ScriptRunConfig constructor, e.g.

`arguments=['--arg1', arg1_val, '--arg2', arg2_val]`.

If you want to override the default maximum time allowed for the job, you can do so via the **`max_run_duration_seconds`** parameter. The system will attempt to automatically cancel the job if it takes longer than this value.

Specify a distributed job configuration

If you want to run a distributed training job, provide the distributed job-specific config to the **`distributed_job_config`** parameter. Supported config types include MpiConfiguration, TensorflowConfiguration, and PyTorchConfiguration.

For more information and examples on running distributed Horovod, TensorFlow and PyTorch jobs, see:

- Distributed training of deep learning models on Azure

Submit the experiment

Python

```
run = experiment.submit(config=src)
run.wait_for_completion(show_output=True)
```

Important

When you submit the training job, a snapshot of the directory that contains your training scripts is created and sent to the compute target. It is also stored as part of the experiment in your workspace. If you change files and submit the job again, only the changed files will be uploaded.

To prevent unnecessary files from being included in the snapshot, make an ignore file (`.gitignore` or `.amlignore`) in the directory. Add the files and directories to exclude to this file. For more information on the syntax to use inside this file, see **syntax and patterns** for `.gitignore`. The `.amlignore` file uses the same syntax. *If both files exist, the `.amlignore` file is used and the `.gitignore` file is unused.*

For more information about snapshots, see **Snapshots**.

Important

Special Folders Two folders, *outputs* and *logs*, receive special treatment by Azure Machine Learning. During training, when you write files to folders named *outputs* and *logs* that are relative to the root directory (`./outputs` and `./logs`, respectively), the files will automatically upload to your job history so that you have access to them once your job is finished.

To create artifacts during training (such as model files, checkpoints, data files, or plotted images) write these to the `./outputs` folder.

Similarly, you can write any logs from your training job to the `./logs` folder. To utilize Azure Machine Learning's **TensorBoard integration** make sure you write your TensorBoard logs to this folder. While your job is in progress, you will be able to launch TensorBoard and stream these logs. Later, you will also be able to restore the logs from any of your previous jobs.

For example, to download a file written to the *outputs* folder to your local machine after your remote training job: `run.download_file(name='outputs/my_output_file', output_file_path='my_destination_path')`

Git tracking and integration

When you start a training job where the source directory is a local Git repository, information about the repository is stored in the job history. For more information, see Git integration for Azure Machine Learning.

Notebook examples

See these notebooks for examples of configuring jobs for various training scenarios:

- Training on various compute targets
- Training with ML frameworks
- tutorials/img-classification-part1-training.ipynb

Learn how to run notebooks by following the article Use Jupyter notebooks to explore this service.

Troubleshooting

- **AttributeError: 'RoundTripLoader' object has no attribute 'comment_handling'**: This error comes from the new version (v0.17.5) of `ruamel-yaml`, an `azureml-core` dependency, that introduces a breaking change to `azureml-core`. In order to fix this error, please uninstall `ruamel-yaml` by running `pip uninstall ruamel-yaml` and installing a different version of `ruamel-yaml`; the supported versions are v0.15.35 to v0.17.4 (inclusive). You can do this by running `pip install "ruamel-yaml>=0.15.35,<0.17.5"`.

- **Job fails with `jwt.exceptions.DecodeError`**: Exact error message:

  ```
  jwt.exceptions.DecodeError: It is required that you pass in a value for the "algorithms" argument when calling decode().
  ```

  Consider upgrading to the latest version of azureml-core: `pip install -U azureml-core`.

  If you are running into this issue for local jobs, check the version of PyJWT installed in your environment where you are starting jobs. The supported versions of PyJWT are < 2.0.0. Uninstall PyJWT from the environment if the version is >= 2.0.0. You

may check the version of PyJWT, uninstall and install the right version as follows:

1. Start a command shell, activate conda environment where azureml-core is installed.
2. Enter `pip freeze` and look for `PyJWT`, if found, the version listed should be < 2.0.0
3. If the listed version is not a supported version, `pip uninstall PyJWT` in the command shell and enter y for confirmation.
4. Install using `pip install 'PyJWT<2.0.0'`

If you are submitting a user-created environment with your job, consider using the latest version of azureml-core in that environment. Versions >= 1.18.0 of azureml-core already pin PyJWT < 2.0.0. If you need to use a version of azureml-core < 1.18.0 in the environment you submit, make sure to specify PyJWT < 2.0.0 in your pip dependencies.

- **ModuleErrors (No module named)**: If you are running into ModuleErrors while submitting experiments in Azure ML, the training script is expecting a package to be installed but it isn't added. Once you provide the package name, Azure ML installs the package in the environment used for your training job.

  If you are using Estimators to submit experiments, you can specify a package name via `pip_packages` or `conda_packages` parameter in the estimator based on from which source you want to install the package. You can also specify a yml file with all your dependencies using `conda_dependencies_file` or list all your pip requirements in a txt file using `pip_requirements_file` parameter. If you have your own Azure ML Environment object that you want to override the default image used by the estimator, you can specify that environment via the `environment` parameter of the estimator constructor.

  Azure ML maintained docker images and their contents can be seen in AzureML Containers . Framework-specific dependencies are listed in the respective framework documentation:
  - Chainer
  - PyTorch
  - TensorFlow
  - SKLearn

> **Note**
>
> If you think a particular package is common enough to be added in Azure ML maintained images and environments please raise a GitHub issue in **AzureML Containers** .

- **NameError (Name not defined), AttributeError (Object has no attribute)**: This exception should come from your training scripts. You can look at the log files from Azure portal to get more information about the specific name not defined or attribute error. From the SDK, you can use `run.get_details()` to look at the error message. This will also list all the log files generated for your job. Please make sure to take a look at your training script and fix the error before resubmitting your job.

- **Job or experiment deletion**: Experiments can be archived by using the Experiment.archive method, or from the Experiment tab view in Azure Machine Learning studio client via the "Archive experiment" button. This action hides the experiment from list queries and views, but does not delete it.

  Permanent deletion of individual experiments or jobs is not currently supported. For more information on deleting Workspace assets, see Export or delete your Machine Learning service workspace data.

- **Metric Document is too large**: Azure Machine Learning has internal limits on the size of metric objects that can be logged at once from a training job. If you encounter a "Metric Document is too large" error when logging a list-valued metric, try splitting the list into smaller chunks, for example:

  Python

  ```
  run.log_list("my metric name", my_metric[:N])
  run.log_list("my metric name", my_metric[N:])
  ```

  Internally, Azure ML concatenates the blocks with the same metric name into a contiguous list.

- **Compute target takes a long time to start**: The Docker images for compute targets are loaded from Azure Container Registry (ACR). By default, Azure Machine Learning creates an ACR that uses the *basic* service tier. Changing the ACR for your workspace to standard or premium tier may reduce the time it takes to build and

load images. For more information, see Azure Container Registry service tiers.

Next steps

- Tutorial: Train and deploy a model uses a managed compute target to train a model.
- See how to train models with specific ML frameworks, such as Scikit-learn, TensorFlow, and PyTorch.
- Learn how to efficiently tune hyperparameters to build better models.
- Once you have a trained model, learn how and where to deploy models.
- View the ScriptRunConfig class SDK reference.
- Use Azure Machine Learning with Azure Virtual Networks