Curated Env - Comes with certain frequently used packages

Custom Env - Used if the curated env doesn't suit your needs.

Work with environments

When you run a script as an Azure Machine Learning job, you need to define the execution context for the job run. One key configuration is the compute target on which the script will be run. This could be the local workstation (in this case the compute instance), or a remote compute target such as the Azure Machine Learning managed compute cluster that is provisioned on-demand.

In this notebook, you'll create a compute cluster and explore compute targets for jobs.

Before you start

You'll need the latest version of the **azureml-ai-ml** package to run the code in this notebook. Run the cell below to verify that it is installed.

> **Note**: If the **azure-ai-ml** package is not installed, run `pip install azure-ai-ml` to install it.

```
pip show azure-ai-ml
```

Connect to your workspace

With the required SDK packages installed, now you're ready to connect to your workspace.

To connect to a workspace, we need identifier parameters - a subscription ID, resource group name, and workspace name. Since you're working with a compute instance, managed by Azure Machine Learning, you can use the default values to connect to the workspace.

```
from azure.identity import DefaultAzureCredential, 
InteractiveBrowserCredential
from azure.ai.ml import MLClient

try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case 
DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()

# Get a handle to workspace
ml_client = MLClient.from_config(credential=credential)
```

Run a script as a job

To train a model, you'll first create the **diabetes_training.py** script in the **src** folder. The script uses the **diabetes.csv** file in the same folder as the training data.

Note that you import libraries at the beginning of the script. Functions from these libraries are used to process the data and train the model. Whatever compute you use to run the script must have these libraries installed.

```
%%writefile src/diabetes-training.py
# import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

# load the diabetes dataset
print("Loading Data...")
diabetes = pd.read_csv('diabetes.csv')

# separate features and labels
X, y =
diabetes[['Pregnancies','PlasmaGlucose','DiastolicBloodPressure','TricepsThickness','SerumInsulin','BMI','DiabetesPedigree','Age']].values,
diabetes['Diabetic'].values

# split data into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=0)

# set regularization hyperparameter
reg = 0.01

# train a logistic regression model
print('Training a logistic regression model with regularization rate
of', reg)
model = LogisticRegression(C=1/reg, solver="liblinear").fit(X_train,
y_train)

# calculate accuracy
y_hat = model.predict(X_test)
acc = np.average(y_hat == y_test)
print('Accuracy:', acc)

# calculate AUC
y_scores = model.predict_proba(X_test)
auc = roc_auc_score(y_test,y_scores[:,1])
print('AUC: ' + str(auc))
```

After you create the script, you can run the script as a job. The script uses common libraries. So you can use a curated environment that includes pandas, numpy, and scikit-learn, among others.

The job uses the latest version of the curated environment: `AzureML-sklearn-0.24-ubuntu18.04-py37-cpu`.

```
from azure.ai.ml import command

# configure job
job = command(
    code="./src",
    command="python diabetes-training.py",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="diabetes-train-curated-env",
    experiment_name="diabetes-training"
)

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

While the job is running, you can already run the next cells.

List environments

Let's explore the environments within the workspace.

In the previous job, you used one of the curated environments. To explore all environments that already exist in the workspace, you can list the environments:

```
envs = ml_client.environments.list()
for env in envs:
    print(env.name)
```

Note that all curated environments have names that begin **AzureML-** (you can't use this prefix for your own environments).

To review a specific environment, you can retrieve an environment by its name and version. For example, you can retrieve the *description* and *tags* of the curated environment you used for the previous job:

```
env = ml_client.environments.get("AzureML-sklearn-0.24-ubuntu18.04-
py37-cpu", version=44)
print(env. description, env.tags)
```

Create and use a custom environment

If a curated environment doesn't include all the Python packages you need to run your script, you can create your own custom environment. By listing all necessary packages in an environment, you can easily re-run your scripts. All the dependencies are stored in the environment which you can then specify in the job configuration, independent of the compute you use.

For example, you can create an environment simply from a Docker image. Certain frameworks like PyTorch will have a public Docker image that already includes everything you need.

Let's create an environment from a Docker image:

```python
from azure.ai.ml.entities import Environment

env_docker_image = Environment(
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    name="docker-image-example",
    description="Environment created from a Docker image.",
)
ml_client.environments.create_or_update(env_docker_image)
```

The environment is now registered in your workspace and you can reference it when you run a script as a job:

```python
from azure.ai.ml import command

# configure job
job = command(
    code="./src",
    command="python diabetes-training.py",
    environment="docker-image-example:1",
    compute="aml-cluster",
    display_name="diabetes-train-custom-env",
    experiment_name="diabetes-training"
)

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

The error message will tell you that there is no module named pandas. There are two possible causes for such an error:

- The script uses pandas but didn't import the library (`import pandas as pd`).
- The script does import the library at the top of the script but the compute didn't have the library installed (`pip install pandas`).

After reviewing the `diabetes-training.py` script you can observe the script is correct, which means the library wasn't installed. In other words, the environment didn't include the necessary packages.

Let's create a new environment, using the base Docker image used in the previous job. Now, you'll add a conda specification to ensure the necessary packages will be installed. First, run the following cell to create the conda specification file:

TO BUILD CONDA ENV ON TOP OF DOCKER IMAGE
If the docker images misses any particular package and you need it in the next run of your job,
you can add a conda specification to ensure necessary package is installed.

```
%%writefile src/conda-env.yml
name: basic-env-cpu
channels:
  - conda-forge
dependencies:
  - python=3.7
  - scikit-learn
  - pandas
  - numpy
  - matplotlib
```

Note that all necessary dependencies are included in the conda specification file for the script to run successfully.

Create a new environment using the base Docker image **and** the conda specification file to add the necessary dependencies. Azure Machine Learning will build the conda environment on top of the Docker image you provided.

```
from azure.ai.ml.entities import Environment

env_docker_conda = Environment(
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    conda_file="./src/conda-env.yml",
    name="docker-image-plus-conda-example",
    description="Environment created from a Docker image plus Conda 
environment.",
)
ml_client.environments.create_or_update(env_docker_conda)
```

Now, you can submit a job with the new environment to run the script:

```
from azure.ai.ml import command

# configure job
job = command(
    code="./src",
    command="python diabetes-training.py",
    environment="docker-image-plus-conda-example:1",
    compute="aml-cluster",
    display_name="diabetes-train-custom-env",
    experiment_name="diabetes-training"
)

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

Submitting the job with the new custom environment triggers the build of the environment. The first time you use a newly created environment, it can take 10-15 minutes to build the environment, which also means your job will take longer to complete.

You can also choose to manually trigger the build of the environment before you submit a job. The environment only needs to be built the first time you use it.

When the job triggers the build of a new environment, you can review the logs of the build in the **Outputs + logs** tab of the job. Open **azureml-logs/20_image_build_log.txt** to inspect the logs of the environment build.

Screenshot build logs