Work with Compute

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)
```

Create a compute cluster

In many cases, your local compute resources may not be sufficient to process a complex or long-running experiment that needs to process a large volume of data; and you may want to take advantage of the ability to dynamically create and use compute resources in the

cloud. Azure Machine Learning supports a range of compute targets, which you can define in your workpace and use to run jobs; paying for the resources only when using them.

You can create a compute cluster in Azure Machine Learning studio, by using the Python SDK, or the Azure CLI. The following code cell checks your workspace for the existence of a compute cluster names `aml-cluster`, and if it doesn't exist, creates it.

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

# Name assigned to the compute cluster
cpu_compute_target = "aml-cluster"

try:
    # let's see if the compute target already exists
    cpu_cluster = ml_client.compute.get(cpu_compute_target)
    print(
        f"You already have a cluster named {cpu_compute_target}, we'll
reuse it as is."
    )

except Exception:
    print("Creating a new cpu compute target...")

    # Let's create the Azure ML compute object with the intended
parameters
    cpu_cluster = AmlCompute(
        name=cpu_compute_target,
        # Azure ML Compute is the on-demand VM service
        type="amlcompute",
        # VM Family
        size="STANDARD_DS11_V2",
        # Minimum running nodes when there is no job running
        min_instances=0,
        # Nodes in cluster
        max_instances=2,
        # How many seconds will the node running after the job
termination
        idle_time_before_scale_down=120,
        # Dedicated or LowPriority. The latter is cheaper but there is
a chance of job termination
        tier="Dedicated",
    )

    # Now, we pass the object to MLClient's create_or_update method
    cpu_cluster =
ml_client.compute.begin_create_or_update(cpu_cluster)
```

After you've created a compute cluster, you can only change the configuration for:

- `min_instances`: Minimum number of nodes

- `max_instances`: Maximum number of nodes
- `idle_time_before_scale_down`: Idle time before scale down

Currently, your compute cluster `aml-cluster` can only scale do a maximum of one node. Let's change that to two, to allow for parallel compute.

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

cluster_scale = AmlCompute(
    name="aml-cluster",
    max_instances=2,
)
ml_client.begin_create_or_update(cluster_scale)
```

When the compute cluster is updated, you can verify its configuration by printing its attributes.

```
cpu_cluster = ml_client.compute.get("aml-cluster")

print (
        f"AMLCompute with name {cpu_cluster.name} has a maximum of
{cpu_cluster.max_instances} nodes"
    )
```

Create a script to train a model

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.

```
%%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','Tric
epsThickness','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))
```

Run a job on a compute cluster

Now, you're ready to run the job on the compute cluster you created.

> **Note**: The job will take some time to start as the compute cluster will need to scale from zero to one node. Once the compute cluster is ready, the script will be run. When the job has finished, the compute cluster will scale back down to zero nodes. You can review the compute cluster's status in the **Compute** page.

Autoscale up-down

```
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-cluster",
    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)
```