Deploy to an batch endpoint

Imagine a health clinic takes patient measurements all day, saving the details for each patient in a separate file. Then overnight, the diabetes prediction model can be used to process all of the day's patient data as a batch, generating predictions that will be waiting the following morning so that the clinic can follow up with patients who are predicted to be at risk of diabetes. With Azure Machine Learning, you can accomplish this by creating a batch endpoint; and that's what you'll implement in this exercise.

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.**

In [ ]:

```
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.

In [ ]:

```
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()
```

In [ ]:

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

Register the model

Batch deployments can only deploy models registered in the workspace. You'll register an MLflow model, which is stored in the local `model` folder.

In [ ]:

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

model_name = 'diabetes-mlflow'
model = ml_client.models.create_or_update(
    Model(name=model_name, path='./model', type=AssetTypes.MLFLOW_MODEL)
```

```
)
```

Create a batch endpoint

A batch endpoint is an HTTPS endpoint that applications can call to trigger a batch scoring job. A batch endpoint name needs to be unique within an Azure region. You'll use the `datetime` function to generate a unique name based on the current date and time.

In [ ]:

```
import datetime

endpoint_name = "batch-" + datetime.datetime.now().strftime("%m%d%H%M%f")
endpoint_name
```

To create an endpoint with the `BatchEndpoint` class, you need to specify the name and optionally a description. After creating an endpoint, you'll deploy a model to the endpoint.

In [ ]:

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

# create a batch endpoint
endpoint = BatchEndpoint(
    name=endpoint_name,
    description="A batch endpoint for classifying diabetes in patients",
)

ml_client.batch_endpoints.begin_create_or_update(endpoint)
```

Create the deployment

A deployment is a set of resources required for hosting the model that does the actual inferencing. We will create a deployment for our endpoint using the `BatchDeployment` class.

Since you're deploying an MLflow model, you don't need a scoring script or define the environment. Azure Machine Learning will automatically create those assets for you. The `MLmodel` file in the `model` folder is used to understand what the expected inputs and outputs are of the model.

You'll deploy a model with the following parameters:

- `name`: Name of the deployment.
- `description`: Optional description to further clarify what the deployment represents.
- `endpoint_name`: Name of the previously created endpoint the model should be deployed to.
- `model`: Name of the registered model.
- `compute`: Compute to be used when invoking the deployed model to generate predictions.
- `instance_count`: Count of compute nodes to use for generating predictions.
- `max_concurrency_per_instance`: Maximum number of parallel scoring script runs per compute node.
- `mini_batch_size`: Number of files passed per scoring script run.
- `output_action`: Each new prediction will be appended as a new row to the output file.
- `output_file_name`: File to which predictions will be appended.
- `retry_settings`: Settings for a mini-batch fails.
- `logging_level`: The log verbosity level. Allowed values are `warning`, `info`, and `debug`.

Running the following cell will configure and create the deployment.

In [ ]:

```
from azure.ai.ml.entities import BatchDeployment, BatchRetrySettings
```

```
from azure.ai.ml.constants import BatchDeploymentOutputAction

deployment = BatchDeployment(
    name="classifier-diabetes-mlflow",
    description="A diabetes classifier",
    endpoint_name=endpoint.name,
    model=model,
    compute="aml-cluster",
    instance_count=2,
    max_concurrency_per_instance=2,
    mini_batch_size=2,
    output_action=BatchDeploymentOutputAction.APPEND_ROW,
    output_file_name="predictions.csv",
    retry_settings=BatchRetrySettings(max_retries=3, timeout=300),
    logging_level="info",
)
ml_client.batch_deployments.begin_create_or_update(deployment)
```

You can deploy multiple models to an endpoint. You can set the default deployment to specify which model should be used by default when calling a batch endpoint.

In [ ]:

```
endpoint.defaults = {}

endpoint.defaults["deployment_name"] = deployment.name

ml_client.batch_endpoints.begin_create_or_update(endpoint)
```

Prepare the data for batch predictions

In the `data` folder you'll find CSV files with unlabeled data. You'll create a data asset that points to the files in the `data` folder, which you'll use as input for the batch job.

In [ ]:

```
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

data_path = "./data"
dataset_name = "patient-data-unlabeled"

patient_dataset_unlabeled = Data(
    path=data_path,
    type=AssetTypes.URI_FOLDER,
    description="An unlabeled dataset for diabetes classification",
    name=dataset_name,
)
ml_client.data.create_or_update(patient_dataset_unlabeled)
```

In [ ]:

```
patient_dataset_unlabeled = ml_client.data.get(
    name="patient-data-unlabeled", label="latest"
)
```

Submit the job

Now that you have deployed a model to a batch endpoint, and have an unlabeled data asset, you're ready to invoke the endpoint to generate predictions on the unlabeled data.

First, you'll define the input by referring to the registered data asset. Then, you'll invoke the endpoint, which will submit a pipeline job. You can use the job URL to monitor it in the Studio. The job will contain a child job that represents the running of the (generated) scoring script to get the predictions.

In [ ]:

```
from azure.ai.ml import Input
from azure.ai.ml.constants import AssetTypes

input = Input(type=AssetTypes.URI_FOLDER, path=patient_dataset_unlabeled.id)
```

In [ ]:

```
job = ml_client.batch_endpoints.invoke(
    endpoint_name=endpoint.name,
    input=input)

ml_client.jobs.get(job.name)
```

Get the results

When the pipeline job that invokes the batch endpoint is completed, you can view the results. All predictions are collected in the `predictions.csv` file that is stored in the default datastore. You can download the file and visualize the data by running the following cells.

In [ ]:

```
ml_client.jobs.download(name=job.name, download_path=".", output_name="score")
```

In [ ]:

```
with open("predictions.csv", "r") as f:
    data = f.read()
```

In [ ]:

```
from ast import literal_eval
import pandas as pd

score = pd.DataFrame(
    literal_eval(data.replace("\n", ",")), columns=["file", "prediction"]
)
score
```

Registries in Azure Machine Learning are organization wide repositories of machine learning assets such as models, environments, and components.

As we define "registering the model," the PROCESS OF UPLOADING A MODEL SEPARATE FROM CODE is referred to as that.

You register a model, and it is uploaded to the cloud (in the default account for your workspace) and then mounted to the same compute where your web service is running.