Highlight Note

Deploying a Real-Time Inferencing Service

You can deploy a model as a real-time web service to several kinds of compute target, including local compute, an Azure Machine Learning compute instance, an Azure Container Instance (ACI), an Azure Kubernetes Service (AKS) cluster, an Azure Function, or an Internet of Things (IoT) module. Azure Machine Learning uses *containers* as a deployment mechanism, packaging the model and the code to use it as an image that can be deployed to a container in your chosen compute target.

Note: Deployment to a local service, a compute instance, or an ACI is a good choice for testing and development. For production, you should deploy to a target that meets the specific performance, scalability, and security needs of your application architecture.

To deploy a model as a real-time inferencing service, you must perform the following tasks:

1. Register a trained model

After successfully training a model, you must register it in your Azure Machine Learning workspace. Your real-time service will then be able to load the model when required.

To register a model from a local file, you can use the **register** method of the **Model** object as shown here:

```
from azureml.core import Model

classification_model = Model.register(workspace=ws,
                       model_name='classification_model',
                       model_path='model.pkl', # local path
                       description='A classification model')
```

Alternatively, if you have a reference to the **Run** used to train the model, you can use its **register_model** method as shown here:

```
run.register_model( model_name='classification_model',
                    model_path='outputs/model.pkl', # run outputs path
                    description='A classification model')
```

2. Define an Inference Configuration

The model will be deployed as a service that consist of:

- A script to load the model and return predictions for submitted data.
- An environment in which the script will be run.

You must therefore define the script and environment for the service.

Creating an Entry Script

Create the *entry script* (sometimes referred to as the *scoring script*) for the service as a Python (.py) file. It must include two functions:

- **init()**: Called when the service is initialized.
- **run(raw_data)**: Called when new data is submitted to the service.

Typically, you use the **init** function to load the model from the model registry, and use the **run** function to generate predictions from the input data. The following example script shows this pattern:

```
import json
import joblib
import numpy as np
from azureml.core.model import Model

# Called when the service is loaded
def init():
    global model
    # Get the path to the registered model file and load it
    model_path = Model.get_model_path('classification_model')
    model = joblib.load(model_path)

# Called when a request is received
def run(raw_data):
    # Get the input data as a numpy array
    data = np.array(json.loads(raw_data)['data'])
    # Get a prediction from the model
    predictions = model.predict(data)
    # Return the predictions as any JSON serializable format
    return predictions.tolist()
```

Creating an Environment

Your service requires a Python environment in which to run the entry script, which you can configure using Conda configuration file. An easy way to create this file is to use a **CondaDependencies** class to create a default environment (which includes the **azureml-defaults** package and commonly-used packages like **numpy** and **pandas**), add any other required packages, and then serialize the environment to a string and save it:

```
from azureml.core.conda_dependencies import CondaDependencies

# Add the dependencies for your model
myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")

# Save the environment config as a .yml file
env_file = 'service_files/env.yml'
with open(env_file,"w") as f:
    f.write(myenv.serialize_to_string())
print("Saved dependency info in", env_file)
```

Combining the Script and Environment in an InferenceConfig

After creating the entry script and environment configuration file, you can combine them in an **InferenceConfig** for the service like this:

```
from azureml.core.model import InferenceConfig
```

```
classifier_inference_config = InferenceConfig(runtime= "python",
                                              source_directory = 'service_files',
                                              entry_script="score.py",
                                              conda_file="env.yml")
```

3. Define a Deployment Configuration

Now that you have the entry script and environment, you need to configure the compute to which the service will be deployed. If you are deploying to an AKS cluster, you must create the cluster and a compute target for it before deploying:

```
from azureml.core.compute import ComputeTarget, AksCompute

cluster_name = 'aks-cluster'
compute_config = AksCompute.provisioning_configuration(location='eastus')
production_cluster = ComputeTarget.create(ws, cluster_name, compute_config)
production_cluster.wait_for_completion(show_output=True)
```

With the compute target created, you can now define the deployment configuration, which sets the target-specific compute specification for the containerized deployment:

```
from azureml.core.webservice import AksWebservice

classifier_deploy_config = AksWebservice.deploy_configuration(cpu_cores = 1,
                                                              memory_gb = 1)
```

The code to configure an ACI deployment is similar, except that you do not need to explicitly create an ACI compute target, and you must use the **deploy_configuration** class from the **azureml.core.webservice.AciWebservice** namespace. Similarly, you can use the **azureml.core.webservice.LocalWebservice** namespace to configure a local Docker-based service.

Note: To deploy a model to an Azure Function, you do not need to create a deployment configuration. Instead, you need to package the model based on the type of function trigger you want to use. This functionality is in preview at the time of writing. For more details, see **Deploy a machine learning model to Azure Functions** in the Azure Machine Learning documentation.

4. Deploy the Model

After all of the configuration is prepared, you can deploy the model. The easiest way to do this is to call the **deploy** method of the **Model** class, like this:

```
from azureml.core.model import Model

service = Model.deploy(workspace=ws,
                       name = 'classifier-service',
                       models = [classification_model],
                       inference_config = classifier_inference_config,
                       deployment_config = classifier_deploy_config,
                       deployment_target = production_cluster)
service.wait_for_deployment(show_output = True)
```

For ACI or local services, you can omit the **deployment_target** parameter (or set it to **None**).

More Information: For more information about deploying models with Azure Machine Learning, see **Deploy models with Azure Machine Learning** in the documentation.