Managed Services for Machine Learning

Wednesday, July 29, 2020 11:57 AM

- Managed Services for Azure ML
  - Services you use to enhance ML processes
  - Managed Services
    - Conventional ML
      - Lengthy installation and setup process
        - Library installation, IDE setup, resource acquisition and setup
      - Expertise to configure hardware
        - GPUs configurations for DL
      - Fair amount of troubleshooting
    - Managed Services Approach
      - Pre-made, pre-optimized environment for ML development
      - Very little setup, no patching
      - Easy configuration for any needed hardware
      - Can run anywhere because it is all cloud-based
      - Offers support for datastore/dataset management and other specialized tools meant to increase efficiency and time spent focusing on main research tasks
      - Compute Resources
        - Training clusters
        - Inferencing clusters
          - Operationalizing model
        - Compute Instances
          - Contain notebook environment to run notebook-based code
        - Attached Compute
          - Individual VMs
        - Local Compute
          - Use local computer as compute resource

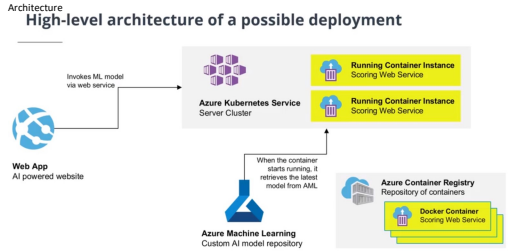
Examples of other services

- Notebooks gallery
- Automated Machine Learning configurator
- Pipeline designer
- Datasets and datastores managers
- Experiments manager
- Pipelines manager
- Models registry
- Endpoints manager

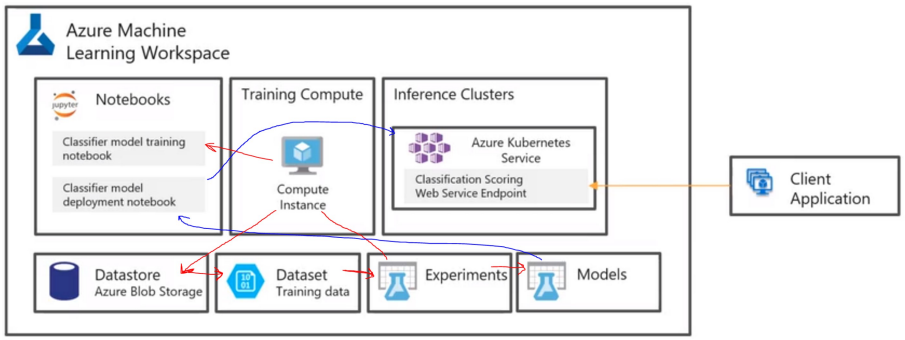
Compute Resources / Notebooks

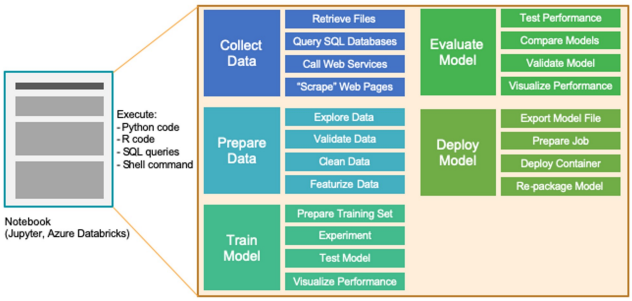
Wednesday, July 29, 2020 1:40 PM

- Compute Resources
  - Compute target: designated compute resource or environment where you run training scripts of host your service deployment
    - Training compute targets
      - Training clusters
        - Used for training and batch inferencing
        - Can be used for generic purposes of running ML python code
        - Single or multi-node cluster
          - Can provision multi-node cluster with GPU support to run distributed training jobs
        - Automatic cluster management and job scheduling, no manual provisioning needed
        - Supports CPU and GPU resources
      - Compute instances
        - Intended to be used as a notebook environment
      - Local Compute
    - Inferencing compute targets
      - Inferencing
        - Real-time: make inferences for each row of data in real-time -> package model as web services is desired
          - Can deployment to iot device as well
        - Batch: inferences on multiple rows of data (batches) -> can use any compute resource that supports any environment that lets you load and train model
          - Resources required for this can be significant
      - Inferencing compute
        - After model is trained, deploy to web hosting env or IoT device
        - When model is used, it infers things about new data based on its training
        - Inferencing cluster -> real time inferencing
        - Any compute resource -> batch inferencing
      - Deployment for production workloads
        - Real-Time: Azure Kubernetes (AKS)
        - Azure allows packaging trained models in a container for multiple avenues of deployment
          - Azure Functions
          - Azure IoT Edge
          - Azure Data Box Edge
        - Architecture



- Managed Notebook Environments
  - Traditionally provided by Matlab and Wolfram Mathematica
  - Self documenting notebooks used to reproduce experiments
  - Made up of one or more cells that allow for the execution of the code snippets or commands within those cells
    - Stores commands and the outputs of those commands
  - Notebooks for Classification
    - Jupyter notebook created within compute instance in workspace
      - Uses Python SDK to retrieve data from dataset as a set of flat files
      - Dataset pulled with credentials from datastore
      - Supervised model training executes within the experiment run
      - Results and related metadata are uploaded to model registry
    - New notebook used to deploy model as web service in Azure Kuberneted Service (AKS)
      - Defines the web service logic to retrieve model, load model into memory and then use loaded model for inferencing against every request for classification
      - Scoring web service handles classification of input samples arriving from client application
        - Samples arrive in form of HTTP request





Lab: Managing Compute

Wednesday, July 29, 2020 2:11 PM

Managing a compute instance

Machine learning requires several tools to prepare data, and train and deploy models. Most of the work usually takes place within web-based, interactive notebooks, such as Jupyter notebooks. Although notebooks are lightweight and easily run in a web browser, you still need a server to to host them.

So, the setup process for most users is to install several applications and libraries on a machine, configure the environment settings, then loaed any additional resources to begin working within notebooks or integrated development environments (IDEs). All this setup takes time, and there is sometimes a fair amount of troubleshooting involved to make sure you have the right combination of software versions that are compatible with one another.

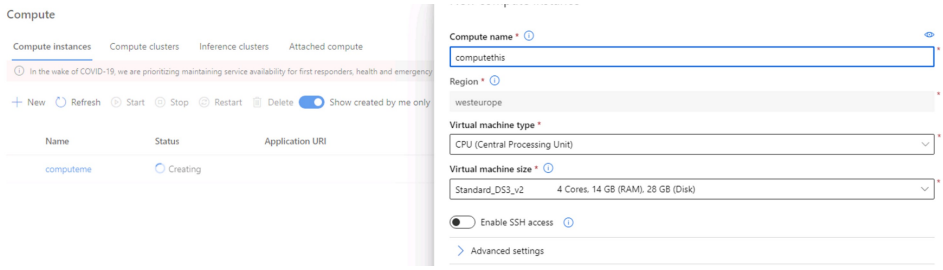
What if you could use a ready-made environment that is pre-optimized for your machine learning development?

Azure Machine Learning compute instance provides this type of environment for you, and is fully managed, meaning you don't have to worry about setup and applying patches and updates to the underlying virtual machine. Plus, since it is cloud-based, you can run it from anywhere and from any machine. All you need to do is specify the type of virtual machine, including GPUs and I/O-optimized options, then you have what you need to start working.

The managed services, such as computer instance and compute cluster, can be used as a training compute target to scale out training resources to handle larger data sets. When you are ready to run your experiments and build your models, you need to specify a compute target. Compute targets are compute resources where you run your experiments or host your service deployment. The target may be your local machine or a cloud-based resource. This is another example of where managed services like compute instance and computer cluster really shine.

A managed compute resource is created and managed by Azure Machine Learning. This compute is optimized for machine learning workloads.

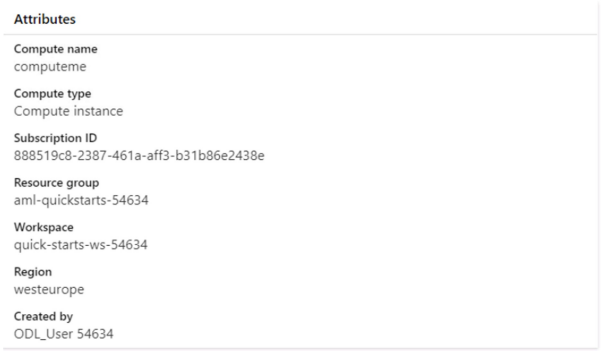
> Azure Machine Learning compute clusters and compute instances are the only managed computes. Additional managed compute resources may be added in the future.

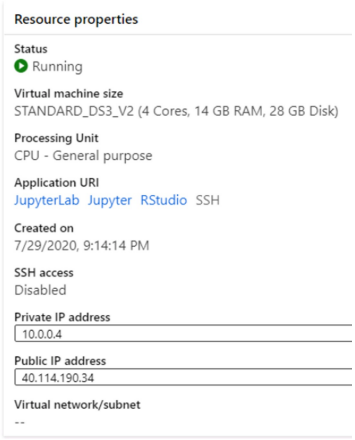


- **Stop**: Since the compute instance runs on a virtual machine (VM), you pay for the instance as long as it is running. Naturally, it needs to run to perform compute tasks, but when you are done using it, be sure to stop it with this option to prevent unnecessary costs.
- **Restart**: Restarting an instance is sometimes necessary after installing certain libraries or extensions. There may be times, however, when the compute instance stops functioning as expected. When this happens, try restarting it before taking further action.
- **Delete**: You can create and delete instances as you see fit. The good news is, all notebooks and R scripts are stored in the default storage account of your workspace in Azure file share, within the "User files" directory. This central storage allows all compute instances *in the same workspace* to access the same files so you don't lose them when you delete an instance you no longer need.

The **Attributes** describe the resource details of the compute instance, including the name, type, Azure subscription, the resource group to which it belongs, the Azure Machine Learning workspace that manages it, and the Azure region to which it is deployed. If you need to execute scripts that require details about your compute instance, this is where you can find most of what you need.

The **Resource properties** show the status and configuration of the compute instance, including links to its applications and public and private endpoints. In this screenshot, you will see that SSH access is disabled. You cannot enable SSH access after creating a compute instance. You can only enable this option at the time of creation. SSH access allows you to securely connect to the VM from a terminal or command window. Use the public IP address to connect via SSH or an integrated development environment (IDE) like Visual Studio Code.





Lab: Managed Notebook Environments

Wednesday, July 29, 2020 2:22 PM

git clone https://github.com/solliancenet/udacity-intro-to-ml-labs.git LAB-19

Compute Resources

Train a machine learning model from a managed notebook environment

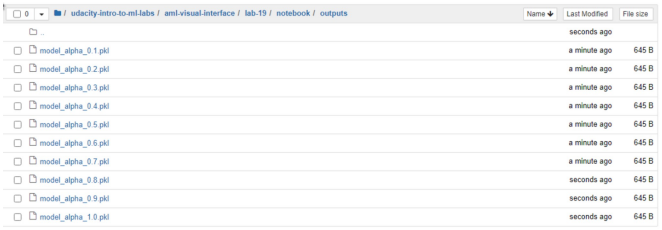
So far, the Managed Services for Azure Machine Learning lesson has covered **compute instance** and the benefits it provides through its fully managed environment containing everythng you need to run Azure Machine Learning. Now it is time to gain some hands-on experience by putting a compute instance to work.

Overview

In this lab, you learn the foundational design patterns in Azure Machine Learning, and train a simple scikit-learn model based on the diabetes data set. After completing this lab, you will have the practical knowledge of the SDK to scale up to developing more-complex experiments and workflows.

In this tutorial, you learn the following tasks:

> - Connect your workspace and create an experiment
> - Load data and train a scikit-learn model



Tutorial: Train your first model

In this tutorial, you learn the foundational design patterns in Azure Machine Learning, and train a simple scikit-learn model based on the diabetes data set. After completing this tutorial, you will have the practical knowledge of the SDK to scale up to developing more-complex experiments and workflows.

In this tutorial, you learn the following tasks:

> - Connect your workspace and create an experiment
> - Load data and train a scikit-learn model

Connect workspace and create experiment

Import the `Workspace` class, and load your subscription information from the file `config.json` using the function `from_config()`. This looks for the JSON file in the current directory by default, but you can also specify a path parameter to point to the file using `from_config(path="your/file/path")`. If you are running this notebook in a cloud notebook server in your workspace, the file is automatically in the root directory.

If the following code asks for additional authentication, simply **paste the link in a browser and enter the authentication token**.

In [3]:
```
from azureml.core import Workspace
ws = Workspace.from_config()
```

Now create an experiment in your workspace. An experiment is another foundational cloud resource that represents a collection of trials (individual model runs). In this tutorial you use the experiment to create runs and track your model training in the Azure Portal. Parameters include your workspace reference, and a string name for the experiment.

In [4]:
```
from azureml.core import Experiment
experiment = Experiment(workspace=ws, name="diabetes-experiment")
```

Load data and prepare for training

For this tutorial, you use the diabetes data set, which uses features like age, gender, and BMI to predict diabetes disease progression. Load the data from the Azure Open Datasets class, and split it into training and test sets using `train_test_split()`. This function segregates the data so the model has unseen data to use for testing following training.

In [6]:
```
from azureml.opendatasets import Diabetes
from sklearn.model_selection import train_test_split

x_df = Diabetes.get_tabular_dataset().to_pandas_dataframe().dropna()
y_df = x_df.pop("Y")

X_train, X_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_state=66)
```

Train a model

Training a simple scikit-learn model can easily be done locally for small-scale training, but when training many iterations with dozens of different feature permutations and hyperparameter settings, it is easy to lose track of what models you've trained and how you trained them. The following design pattern shows how to leverage the SDK to easily keep track of your training in the cloud.

Build a script that trains ridge models in a loop through different hyperparameter alpha values.

In [7]:
```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.externals import joblib
import math

alphas = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

for alpha in alphas:
    run = experiment.start_logging()
    run.log("alpha_value", alpha)

    model = Ridge(alpha=alpha)
    model.fit(X=X_train, y=y_train)
    y_pred = model.predict(X=X_test)
    rmse = math.sqrt(mean_squared_error(y_true=y_test, y_pred=y_pred))
    run.log("rmse", rmse)

    model_name = "model_alpha_" + str(alpha) + ".pkl"
    filename = "outputs/" + model_name

    joblib.dump(value=model, filename=filename)
    run.upload_file(name=model_name, path_or_stream=filename)
    run.complete()
```

The above code accomplishes the following:

1. For each alpha hyperparameter value in the `alphas` array, a new run is created within the experiment. The alpha value is logged to differentiate between each run.
2. In each run, a Ridge model is instantiated, trained, and used to run predictions. The root-mean-squared-error is calculated for the actual versus predicted values, and then logged to the run. At this point the run has metadata attached for both the alpha value and the rmse accuracy.
3. Next, the model for each run is serialized and uploaded to the run. This allows you to download the model file from the run in the portal.
4. At the end of each iteration the run is completed by calling `run.complete()`.

Basic Modeling

Wednesday, July 29, 2020 4:13 PM

- Modeling for Machine Learning
  - Training a ML model
    - Process through which a mathematical model is built from data that contains both inputs and expected outputs (supervised) or just inputs (unsupervised)
  - Components of Modeling
    - Experiment
      - Generic context of handling runs (folder for organizing artifacts of model training process
      - Organizational tool for grouping runs together and viewing results pertinent to experiment runs
      - Provides metrics and charts for keeping track of relevant and important metrics over long periods of time and numerous runs
      - Can download model files instead of retraining models
    - Model Training Runs
      - Used to train model, contains all resources associated with the training process
        - Output, metrics, logs, snapshots of scripts
      - Single execution of a training script
      - Stores metadata about run, metrics, logs, snapshot of directory that contains scripts
      - Produces run when you submit script to train model, can have 0 or multiple child runs
      - Run Configuration define how a script should be run in a specified compute target
        - Can be persisted into file or an in-memory object
        - Includes wide set of behavior definitions
          - Use existing Python environments or Conda environments
    - Models
      - Piece of code that takes an input and produces outputs
      - Model = Algorithm + Data + Hyperparameters
      - Iterative process, produced by runs
      - ScikitLearn
    - Model Registry
      - Keeps track of all models in Azure ML workspace
      - Registered model: logical container for one or more files that make up the model
      - Models identified by name and version
        - Name conflicts -> system assumes new version
        - Can provide additional metadata tags which can identify model
        - Can't delete models being use in active deployment
      - Using Designer registers model automatically

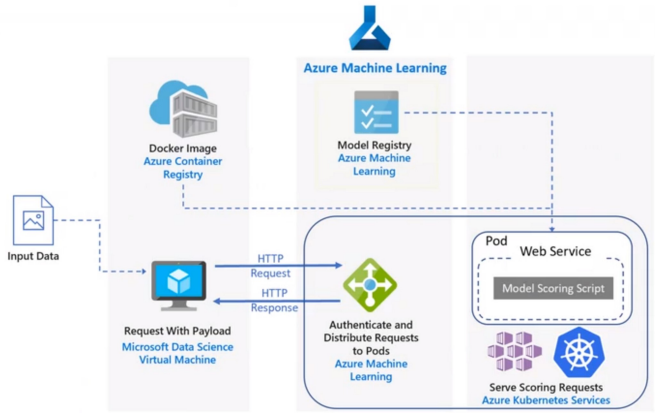
        How to register a model in the workspace with the Azure Machine Learning Python SDK:

        - ```
          # register model model =
          run.register_model(model_name='sklearn_mnist',
          model_path='outputs/sklearn_mnist_model.pkl')
          print(model.name, model.id, model.version, sep='\t')
          ```

      - Models are in PKL (pickle) format
      - Model files also available in workspace

How to deploy a custom ML model as a web service hosted in AKS

Real-time scoring of Python scikit-learn and deep learning models on Azure



- Application Flow
  - Trained model registered in model registry
  - Model registered as scoring web service docker image in Azure container registry
  - Deploys to AKS as web service
  - Client sends HTTP POST request with encoded question data
  - Web service created by Azure ML extracts data from request and is sent to model for scoring
  - Matching data with associated scores returned to client

- Azure ML
  - Cloud service used to train/deploy/automate/manage ML models
  - Manage model deployment, authentication, routing, load balancing web service
- VM
  - Example of a device that can send HTTP request
- AKS
  - Deploy application on kubernetes cluster
  - Simplifies deployment and operations of kubernetes
  - Can be configured using CPU only VMs for classical ML models or GPU VMs for DL models
- Azure Container Registry
  - Contains docker images of models
  - Enables storage of images for all sorts of Docker deployments
  - Scoring images deployed as containers and used to run scoring script
          ```

Lab: Explore Experiment and Runs

Wednesday, July 29, 2020 4:36 PM

git clone https://github.com/solliancenet/udacity-intro-to-ml-labs.git LAB-20

Compute Resources

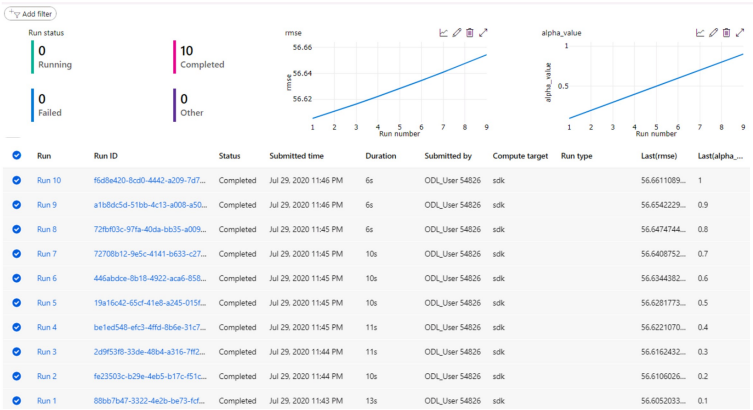
Explore experiments and runs

In the previous lab (19), you executed a Jupyter notebook that trained a model through a series of 10 different runs, each with a different alpha hyperparameter applied. These runs were created within the experiment you created at the beginning of the notebook. Because of this, Azure Machine Learning logged the details so you can review the result of each run and see how the alpha value is different between the them.

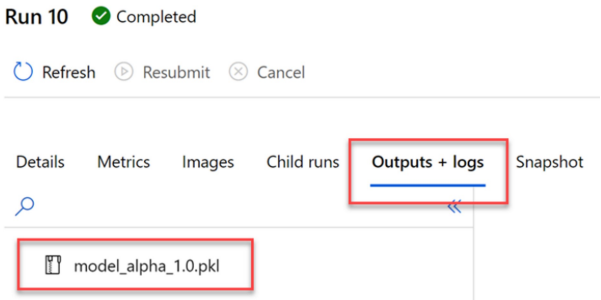
Overview

In this lab, you view the experiments and runs executed by a notebook. In the first part of the lab, you will use a notebook to create and run the experiments. In the second part of the lab, you will navigate to the **Experiments** blade in Azure Machine Learning Studio. Here you see all the individual runs in the experiment. Any custom-logged values (alpha_value and rmse, in this case) become fields for each run, and also become available for the charts and tiles at the top of the experiment page. To add a logged metric to a chart or tile, hover over it, click the edit button, and find your custom-logged metric.

When training models at scale over hundreds and thousands of separate runs, this page makes it easy to see every model you trained, specifically how they were trained, and how your unique metrics have changed over time.



6. Select the **Outputs + logs** tab. You see the `.pkl` file for the model that was uploaded to the run during each training iteration. This lets you download the model file rather than having to retrain it manually.



Connect workspace and create experiment

Import the `Workspace` class, and load your subscription information from the file `config.json` using the function `from_config()`. This looks for the JSON file in the current directory by default, but you can also specify a path parameter to point to the file using `from_config(path="your/file/path")`. If you are running this notebook in a cloud notebook server in your workspace, the file is automatically in the root directory.

If the following code asks for additional authentication, simply **paste the link in a browser and enter the authentication token**.

In [3]:
```
from azureml.core import Workspace
ws = Workspace.from_config("/Users/odl_user_54826")
```

Now create an experiment in your workspace. An experiment is another foundational cloud resource that represents a collection of trials (individual model runs). In this tutorial you use the experiment to create runs and track your model training in the Azure Portal. Parameters include your workspace reference, and a string name for the experiment.

In [4]:
```
from azureml.core import Experiment
experiment = Experiment(workspace=ws, name="diabetes-experiment")
```

Load data and prepare for training

For this tutorial, you use the diabetes data set, which uses features like age, gender, and BMI to predict diabetes disease progression. Load the data from the Azure Open Datasets class, and split it into training and test sets using `train_test_split()`. This function segregates the data so the model has unseen data to use for testing following training.

In [*]:
```
from azureml.opendatasets import Diabetes
from sklearn.model_selection import train_test_split

x_df = Diabetes.get_tabular_dataset().to_pandas_dataframe().dropna()
y_df = x_df.pop("Y")

X_train, X_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_state=66)
```

Load data and prepare for training

For this tutorial, you use the diabetes data set, which uses features like age, gender, and BMI to predict diabetes disease progression. Load the data from the Azure Open Datasets class, and split it into training and test sets using `train_test_split()`. This function segregates the data so the model has unseen data to use for testing following training.

In [5]:
```
from azureml.opendatasets import Diabetes
from sklearn.model_selection import train_test_split

x_df = Diabetes.get_tabular_dataset().to_pandas_dataframe().dropna()
y_df = x_df.pop("Y")

X_train, X_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.2, random_state=66)
```

Train a model

Training a simple scikit-learn model can easily be done locally for small-scale training, but when training many iterations with dozens of different feature permutations and hyperparameter settings, it is easy to lose track of what models you've trained and how you trained them. The following design pattern shows how to leverage the SDK to easily keep track of your training in the cloud.

Build a script that trains ridge models in a loop through different hyperparameter alpha values.

In [ ]:
```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.externals import joblib
import math

alphas = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

for alpha in alphas:
    run = experiment.start_logging()
    run.log("alpha_value", alpha)

    model = Ridge(alpha=alpha)
    model.fit(X=X_train, y=y_train)
    y_pred = model.predict(X=X_test)
    rmse = math.sqrt(mean_squared_error(y_true=y_test, y_pred=y_pred))
    run.log("rmse", rmse)

    model_name = "model_alpha_" + str(alpha) + ".pkl"
    filename = "outputs/" + model_name

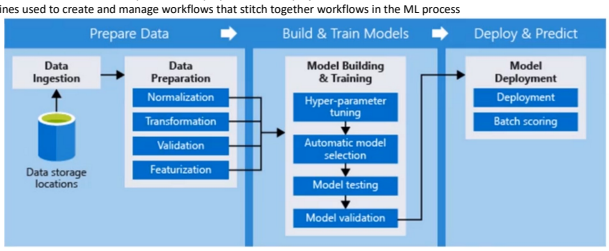
    joblib.dump(value=model, filename=filename)
    run.upload_file(name=model_name, path_or_stream=filename)
    run.complete()
```

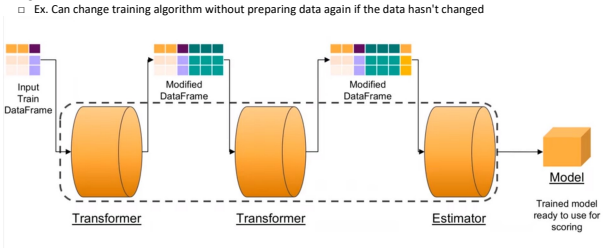
The above code accomplishes the following:

1. For each alpha hyperparameter value in the `alphas` array, a new run is created within the experiment. The alpha value is logged to differentiate between each run.
2. In each run, a Ridge model is instantiated, trained, and used to run predictions. The root-mean-squared-error is calculated for the actual versus predicted values, and then logged to the run. At this point the run has metadata attached for both the alpha value and the rmse accuracy.
3. Next, the model for each run is serialized and uploaded to the run. This allows you to download the model file from the run in the portal.
4. At the end of each iteration the run is completed by calling `run.complete()`.

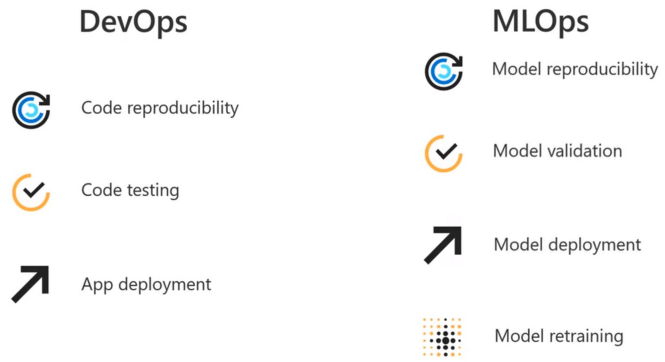
Advanced Modeling

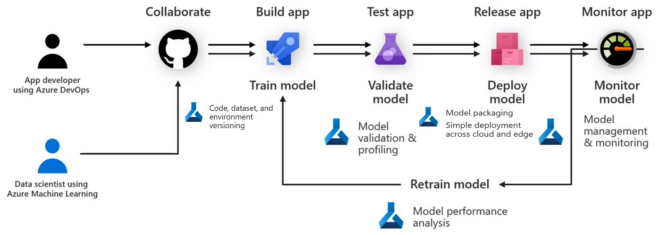
Wednesday, July 29, 2020 4:49 PM

- Advanced Modeling
  - Steps organized into pipelines
    - Data Ingestion and preparation
    - Model building and training
    - Model deployment
  - DevOps
    - Process automation applied to classical software development
  - MLOps
    - Applying DevOps principles to ML pipelines
    - Azure ML and Azure DevOps used to deploy data science projects
  - Pipelines used to create and manage workflows that stitch together workflows in the ML process
    - 

    - Ensures you have a repeatable process when you need to retrain models
    - Modular, can repeat and re-run certain steps without redundancy if the steps prior to it hasn't changed
      - Ex. Can change training algorithm without preparing data again if the data hasn't changed
    - 

      - Pipelines allow data scientists to collaborate while working on different areas of the workflow
      - Different scientists can work on different transformer steps concurrently
      - Data ingestion is the input, first transformer is usually data preparation
      - Modified dataframe output from first transformer sent to second transformer which validates and applies featurization to the data
      - Estimator processes modified data frame using selected algorithm to train model

- DevOps and MLOps
  - Not enough to version of the code, need the version of the data as well
  - What does a build / release mean?
  - MLOps applies classical DevOps principles to AIML
  - MLOps
    - Automate the end-to-end ML lifecycle
    - Monitor ML processes for operational issues
    - Capture traceability data
  - 

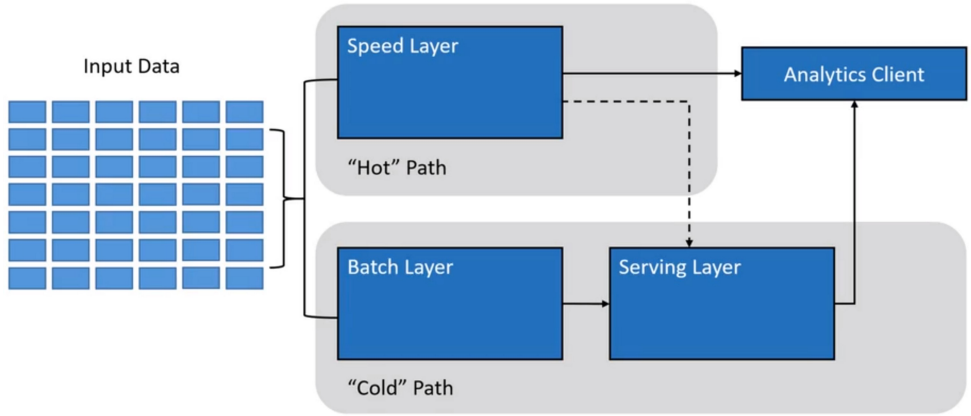
  - 

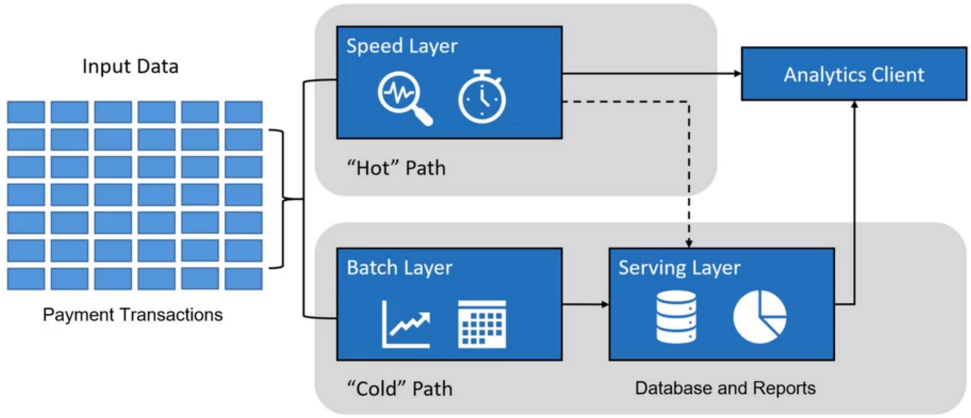
  - Model reproducibility Model validation Model deployment Model retraining
    - Data scientists obtain model reproducibility through code, dataset and environment versioning
      - Resulting pipeline allows data scientists to define reusable and repeatable steps for data preparation, training and scoring processes
    - MLOps supports model validation and profiling within AzureML experiments
    - MLOps supports packaging and deployment of model across cloud and edge devices
    - MLOps enables monitoring the model in production, can kick off retraining automatically

Operationalizing Models

Thursday, July 30, 2020 8:26 PM

- Operationalizing Models
  - Operationalization: the process of deploying a trained-and-evaluated model outside personal deployments and test environments
  - Typical Model Deployment
    - Get model file (any format)
    - Create a scoring script (.py)
    - Create a schema file describing web service input (.json)
    - Create a real-time scoring web service
    - Call web service from applications
    - Repeat the process every time you retrain the model
  - Azure ML Deployment
    - Trained model is stored as a docker image
    - Azure ML deploys to Azure Kubernetes Service or Azure Container Instance
      - AKS good for high scale production deployments
        - Fast response time
        - Auto-scaling of deployed service
        - Hardware acceleration options->GPUs or FPGAs
      - Azure Container Instances
        - Need quick deploy and validation of model
        - Testing model under development
  - Models are deployed for either real-time or batch inferencing
    - Real-time inferencing means web service hosting a model is called to score against a small set of data
    - Large volume scoring usually means model is used in a batch process
  - Real-Time Inferencing
    - Trained model used to make decisions on new data in real time
    - Create manually from Azure ML Studio UI
    - Create programmatically using Azure ML Python SDK
  - Batch Inferencing
    - Using ML to make predictions on large quantities of data, no real-time
    - Run on a reoccurring schedule on data stored in a DB or datastore
    - Predictions written to datastore for later use, persisted
    - Post processing or analysis on predictions can be done if required
    - Better fit if inferencing is complex or compute intensive
      - scale out compute resources for parallel processing
    - Typically involves latency requirements of hours or days so it doesn't need trained models deployed to REST web services
    - Leverages high throughput and scalable compute targets
      - Cost effective
      - Large volume of data asynchronously
  - Lambda Architecture
    - Predictions are needed for both newly arriving and stored data
    - Lambda architecture takes ingested data and processes it in two different speeds
      - Hot path: Speed layer attempts to make real-time inferencing predictions
        - Results are made available to both analytics client and stored in serving layer (DB)
      - Cold path: Batch layer stores all incoming data in raw form and performs batch inference on a predefined schedule
        - Results stored into serving layer (DB) for later use
      - Both paths converge at analytics client
    - Ex. Fraud detection in banking
      - Batch scoring used to predict total number of fraudulent transactions within a period of time (month)
        - Consider historical data and predict at a certain layer of granularity
      - Real Time Scoring used to flag potentially fraudulent transactions based on properties of the transaction / customer
      - Both results are sent to analytics client where fraud-related decisions are made based on both sets of predictions
      - When daily number of fraudulent transactions are higher than predicted for multiple days, auditing action could be triggered to reevaluate the algorithm





Lab: Deploy a Model as a Webservice

Thursday, July 30, 2020 9:59 PM

Compute Resources

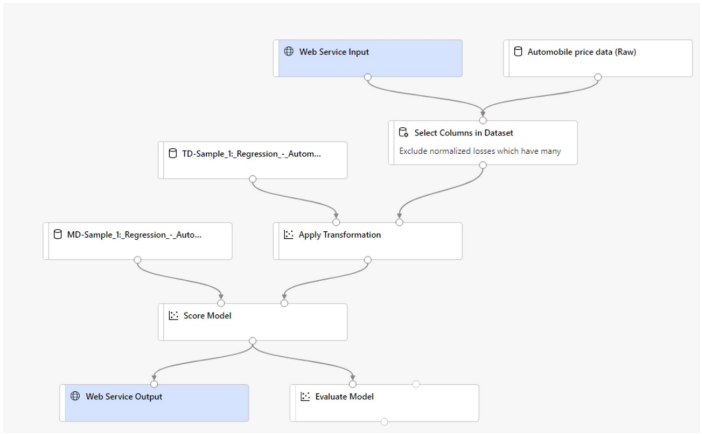
Deploy a trained model as a webservice

In previous lessons, we spent much time talking about training a machine learning model, which is a multi-step process involving data preparation, feature engineering, training, evaluation, and model selection. The model training process can be very compute-intensive, with training times spanning across many hours, days, or weeks depending on the amount of data, type of algorithm used, and other factors. A trained model, on the other hand, is used to make decisions on new data quickly. In other words, it infers things about new data it is given based on its training. Making these decisions on new data on-demand is called real-time inferencing.

Overview

In this lab, you learn how to deploy a trained model that can be used as a webservice, hosted on an Azure Kubernetes Service (AKS) cluster. This process is what enables you to use your model for real-time inferencing.

The Azure Machine Learning designer simplifies the process by enabling you to train and deploy your model without writing any code.



sample-1-regression---automobile

Details Test Consume

Attributes

Service ID
sample-1-regression---automobile

Description
--

Deployment state
Healthy

Compute type
AKS

Created by
ODL_User 56986

Model ID
amlstudio-sample-1-regression:1

Created on
7/31/2020 5:29:50 AM

Last updated on
7/31/2020 5:29:55 AM

Compute target
nyc-taxi-service

Image ID
--

REST endpoint
http://23.100.86.167:80/api/v1/service/sample-1-regression---automobile/score

Key-based authentication enabled
true

Token-based authentication enabled
false

Tags

CreatedByAMLStudio
true

Properties

LinkedPipelineDraftId
a56900c0-dc7d-48eb-ba8f-2ec98835daa7

LinkedPipelineRunId
2dc3856d-df05-485e-9c06-1b436b4f07c4

hasInferenceSchema
True

hasHttps
False

Swagger URI
http://23.100.86.167/api/v1/service/sample-1-regression---automobile/swagger.json

Scoring timeout
60000 ms

CPU
0.1

Memory
0.5 GB

Application Insights enabled
true

Application Insights url
https://portal.azure.com#@microsoft.onmicrosoft.com/resource/subscriptions/de47103e-2da6-4f5e-88fc-d18b27fd249b/resourcegroups/aml-quickstarts-56986/providers/microsoft.insights/components/mlappinsight56986

Event Hubs enabled
false

Storage enabled
false

Autoscale enabled
true

Min replicas
1

Max replicas
10

Target utilization
70%

Refresh period
1 s

```python
import urllib.request
import json
import os
import ssl

def allowSelfSignedHttps(allowed):
    # bypass the server certificate verification on client side
    if allowed and not os.environ.get('PYTHONHTTPSVERIFY', '') and getattr(ssl, '_create_unverified_context', None):
        ssl._create_default_https_context = ssl._create_unverified_context

allowSelfSignedHttps(True) # this line is needed if you use self-signed certificate in your scoring service.

data = {
    "Inputs": {
        "WebServiceInput0":
        [
            {
                'symboling': "3",
                'normalized-losses': "1",
                'make': "alfa-romero",
                'fuel-type': "gas",
                'aspiration': "std",
                'num-of-doors': "two",
                'body-style': "convertible",
                'drive-wheels': "rwd",
                'engine-location': "front",
                'wheel-base': "88.6",
                'length': "168.8",
                'width': "64.1",
                'height': "48.8",
                'curb-weight': "2548",
                'engine-type': "dohc",
                'num-of-cylinders': "four",
                'engine-size': "130",
                'fuel-system': "mpfi",
                'bore': "3.47",
                'stroke': "2.68",
                'compression-ratio': "9",
                'horsepower': "111",
                'peak-rpm': "5000",
                'city-mpg': "21",
                'highway-mpg': "27",
                'price': "13495",
            },
        ],
    },
    "GlobalParameters":  {
    }
}

body = str.encode(json.dumps(data))

url = 'http://23.100.86.167:80/api/v1/service/sample-1-regression---automobile/score'
api_key = 'pmazMvSrEbNukbdEysI4LjC4gD0jPoy4' # Replace this with the API key for the web service
headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ api_key)}

req = urllib.request.Request(url, body, headers)

try:
    response = urllib.request.urlopen(req)

    result = response.read()
    print(result)
except urllib.error.HTTPError as error:
    print("The request failed with status code: " + str(error.code))

    # Print the headers - they include the requert ID and the timestamp, which are useful for debugging the failure
    print(error.info())
    print(json.loads(error.read().decode("utf8", 'ignore')))

```

Basic consumption info

REST endpoint
http://23.100.86.167:80/api/v1/service/sample-1-regression-...

Authentication types

Using key Using token

Primary key
pmazMvSrEbNukbdEysI4LjC4gD0jPoy4 Regenerate

Secondary key
Ro9a5JXm3iOZg96A9EzyyM4d4oO9UTrW Regenerate

Programmatically Accessing Managed Services

Friday, July 31, 2020 12:15 AM

- Azure ML SDK for Python
  - Can build models and workflows the same as with designer but this is the code-first experience
  - Can train models on Azure ML then scale out and use compute resources from Azure
  - Supports Scikit-learn, Tensowflow, PyTorch, Keras
  - Key areas of the SDK
    - Manage datasets
    - Organize and monitor experiments
    - Model training
    - Automated ML
    - Model deployment
  - Azure ML lets you manage model training runs from within Python code
    - Lets you query run details, cancel runs, mark runs as complete etc.
    - Lets you monitor training script output from within Python notebooks
      - run.wait_for_completion(show_output=True)
      - Jupyter notebooks -> RunDetails(run).show()

Option 1: Wait for run completion with **show_output = True**

```
run.wait_for_completion(show_output = True)
```

```
run.wait_for_completion(show_output = True)

Streaming log file azureml-logs/60_control_log.txt
Running: ['python', 'azureml-setup/run_script.py', 'python', 'azureml-setup/context_manager_injector.py', '-i', 'Proj
ectPythonPath:context_managers.ProjectPythonPath', '-i', 'OutputCollection:context_managers.RunHistory', '-i', 'Track
UserError:context_managers.TrackUserError', 'train_sklearn.py']
Logging experiment running status in history service.
Streaming log file azureml-logs/70_driver_log.txt

Streaming azureml-logs/70_driver_log.txt
========================================

max_depth: 5 RMSE score: 4.4301443628697585
max_depth: 7 RMSE score: 4.599890967633183
max_depth: 8 RMSE score: 4.624470628723321
max_depth: 10 RMSE score: 5.007093165019506
max_depth: 12 RMSE score: 5.253510197973364
max_depth: 15 RMSE score: 5.493736380263984
max_depth: 20 RMSE score: 5.679438803175692
max_depth: 25 RMSE score: 5.993765777521834
Best max_depth: 5 Best RMSE score: 4.4301443628697585
```

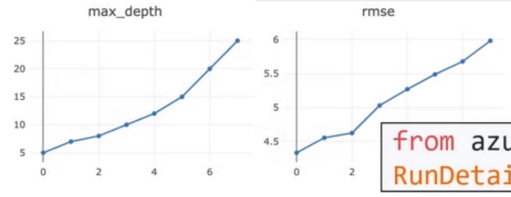
```
RunDetails(run).show()
```

Run Properties

Status	Finalizing
Start Time	7/26/2019 3:13:05 PM
Duration	0:02:12
Run Id	model-monitoring_1564168384_3af4e0ed
Arguments	N/A

Output Logs azureml-logs/70_driver_log.txt Auto-switch

```
max_depth: 5 RMSE score: 4.335601704334848
max_depth: 7 RMSE score: 4.555593228619814
max_depth: 8 RMSE score: 4.625442223680089
max_depth: 10 RMSE score: 5.030268838325685
max_depth: 12 RMSE score: 5.26917740823404
max_depth: 15 RMSE score: 5.487615080424344
max_depth: 20 RMSE score: 5.677821912090937
max_depth: 25 RMSE score: 5.981724374409508
Best max_depth: 5 Best RMSE score: 4.335601704334848
Registering best model!
```




```
from azureml.widgets import RunDetails
RunDetails(run).show()
```

Lab: Train/Deploy from a Compute Instance

Friday, July 31, 2020 1:11 AM

git clone https://github.com/solliancenet/udacity-intro-to-ml-labs.git LAB 22

Compute Resources

Training and deploying a model from a notebook running in a Compute Instance

So far, the Managed Services for Azure Machine Learning lesson has covered **compute instance** and the benefits it provides through its fully managed environment containing everything you need to run Azure Machine Learning.

The compute instance provides a comprehensive set of a capabilities that you can use directly within a python notebook or python code including:

- Creating a **Workspace** that acts as the root object to organize all artifacts and resources used by Azure Machine Learning.
- Creating **Experiments** in your Workspace that capture versions of the trained model along with any desired model performance telemetry. Each time you train a model and evaluate its results, you can capture that run (model and telemetry) within an Experiment.
- Creating **Compute** resources that can be used to scale out model training, so that while your notebook may be running in a lightweight container in Azure Notebooks, your model training can actually occur on a powerful cluster that can provide large amounts of memory, CPU or GPU.
- Using **Automated Machine Learning (AutoML)** to automatically train multiple versions of a model using a mix of different ways to prepare the data and different algorithms and hyperparameters (algorithm settings) in search of the model that performs best according to a performance metric that you specify.
- Packaging a Docker **Image** that contains everything your trained model needs for scoring (prediction) in order to run as a web service.
- Deploying your Image to either Azure Kubernetes or Azure Container Instances, effectively hosting the **Web Service**.

Overview

In this lab, you start with a model that was trained using Automated Machine Learning. Learn how to use the Azure ML Python SDK to register, package, and deploy the trained model to Azure Container Instances (ACI) as a scoring web service. Finally, test the deployed model (1) by make direct calls on service object, (2) by calling the service end point (Scoring URI) over http.

Deployment of Automated Machine Learning Model

Lab Overview

In this lab, you will start with a model that was trained using Automated Machine Learning. Learn how to use the Azure ML Python SDK to register, package, and deploy the trained model to Azure Container Instance as a scoring web service. Finally, test the deployed model (1) by make direct calls on service object, (2) by calling the service end point (Scoring URI) over http.

Because you will be using the Azure Machine Learning SDK, you will be able to provision all your required Azure resources directly from this notebook, without having to use the Azure Portal to create any resources.

Setup

To begin, you will need to provide the following information about your Azure Subscription.

If you are using your own Azure subscription, please provide names for subscription_id, resource_group, workspace_name and workspace_region to use. Note that the workspace needs to be of type Machine Learning Workspace.

If an environment is provided to you be sure to replace XXXXX in the values below with your unique identifier. `` In the following cell, be sure to set the values for `subscription_id`, `resource_group`, `workspace_name` and `workspace_region` as directed by the comments (*these values can be acquired from the Azure Portal*).

To get these values, do the following:

1. Navigate to the Azure Portal and login with the credentials provided.
2. From the left hand menu, under Favorites, select `Resource Groups`.
3. In the list, select the resource group provided to you for this lab.
4. From the Overview tab, capture the desired values.

Execute the following cell by selecting the `>|Run` button in the command bar above.

In [1]:
```
#Provide the Subscription ID of your existing Azure subscription
subscription_id = "b19c9c81-5f59-4537-b3c0-c1beb163ec22" # <- needs to be the subscription within the Azure resource group for th

#Provide values for the existing Resource Group
resource_group = "aml-quickstarts-57170" # <- enter the name of your Azure Resource Group

#Provide the Workspace Name and Azure Region of the Azure Machine Learning Workspace
workspace_name = "quick-starts-ws-57170" # <- enter the name of the Azure Machine Learning workspace
workspace_region = "SouthCentralUS" # <- region of your Azure Machine Learning workspace
```

Download the model that was trained using Automated Machine Learning

In [2]:
```
import urllib.request
import os

model_folder = './automl-model'
model_file_name = 'model.pkl'
model_path = os.path.join(model_folder, model_file_name)

# this is the URL to download a model that was trained using Automated Machine Learning
model_url = ('https://quickstartsws9073123377.blob.core.windows.net/'
             'azureml-blobstore-0d1c4218-a5f9-418b-bf55-902b65277b85/'
             'quickstarts/automl-model/v2/model.pkl')

# Download the model to your local disk in the model_folder
os.makedirs(model_folder, exist_ok=True)
urllib.request.urlretrieve(model_url, model_path)
```

Out[2]:
```
('./automl-model/model.pkl', <http.client.HTTPMessage at 0x7fe2af4d46d8>)
```

Import required packages

The Azure Machine Learning SDK provides a comprehensive set of a capabilities that you can use directly within a notebook including:

- Creating a **Workspace** that acts as the root object to organize all artifacts and resources used by Azure Machine Learning.
- Creating **Experiments** in your Workspace that capture versions of the trained model along with any desired model performance telemetry. Each time you train a model and evaluate its results, you can capture that run (model and telemetry) within an Experiment.
- Creating **Compute** resources that can be used to scale out model training, so that while your notebook may be running in a lightweight container in Azure Notebooks, your model training can actually occur on a powerful cluster that can provide large amounts of memory, CPU or GPU.
- Using **Automated Machine Learning (AutoML)** to automatically train multiple versions of a model using a mix of different ways to prepare the data and different algorithms and hyperparameters (algorithm settings) in search of the model that performs best according to a performance metric that you specify.
- Packaging a Docker **Image** that contains everything your trained model needs for scoring (prediction) in order to run as a web service.
- Deploying your Image to either Azure Kubernetes or Azure Container Instances, effectively hosting the **Web Service**.

In Azure Notebooks, all of the libraries needed for Azure Machine Learning are pre-installed. To use them, you just need to import them. Run the following cell to do so:

In [3]:
```
import azureml.core
from azureml.core import Workspace
from azureml.core.webservice import Webservice, AksWebservice
from azureml.core.image import Image
from azureml.core.model import Model

print("Azure ML SDK version:", azureml.core.VERSION)
```

```
Azure ML SDK version: 1.9.0
```

Create and connect to an Azure Machine Learning Workspace

Run the following cell to create a new Azure Machine Learning **Workspace**.

Important Note: You will be prompted to login in the text that is output below the cell. Be sure to navigate to the URL displayed and enter the code that is provided. Once you have entered the code, return to this notebook and wait for the output to read `Workspace configuration succeeded`.

In [5]:
```
ws = Workspace.create(
```

Deploy the Model as a Web Service

Create the Scoring Script

Azure Machine Learning SDK gives you control over the logic of the web service, so that you can define how it retrieves the model and how the model is used for scoring. This is an important bit of flexibility. For example, you often have to prepare any input data before sending it to your model for scoring. You can define this data preparation logic (as well as the model loading approach) in the scoring file.

Run the following cell to create a scoring file that will be included in the Docker Image that contains your deployed web service.

Important Please update the `model_name` variable in the script below. The model name should be the same as the `Model registered` printed above.

In [8]:
```
%%writefile scoring_service.py

import json
import numpy as np
import pandas as pd
import azureml.train.automl

columns = ['vendorID', 'passengerCount', 'tripDistance', 'hour_of_day', 'day_of_week', 'day_of_month',
           'month_num', 'normalizeHolidayName', 'isPaidTimeOff', 'snowDepth', 'precipTime',
           'precipDepth', 'temperature']

def init():
    try:
        # One-time initialization of predictive model and scaler
        from azureml.core.model import Model
        from sklearn.externals import joblib
        global model

        model_name = 'nyc-taxi-automl-predictor'
        print('Looking for model path for model: ', model_name)
        model_path = Model.get_model_path(model_name=model_name)
        print('Looking for model in: ', model_path)
        model = joblib.load(model_path)
        print('Model loaded...')

    except Exception as e:
        print('Exception during init: ', str(e))

def run(input_json):
    try:
        inputs = json.loads(input_json)
        data_df = pd.DataFrame(np.array(inputs).reshape(-1, len(columns)), columns = columns)
        # Get the predictions...
        prediction = model.predict(data_df)
        prediction = json.dumps(prediction.tolist())
    except Exception as e:
        prediction = str(e)
    return prediction
```

```
Writing scoring_service.py
```

Package Model

Run the next two cells to create the deployment **Image**

WARNING: to install, build-essential needs to be available on the Docker image and is not by default. Thus, we will create a custom dockerfile with build-essential installed.

In [9]:
```
%%writefile dockerfile
RUN apt-get update && apt-get install -y build-essential
```

```
Writing dockerfile
```

In [10]:
```
conda_file = 'automl_dependencies.yml'
runtime = 'python'

# create container image configuration
print("Creating container image configuration...")
from azureml.core.image import ContainerImage
image_config = ContainerImage.image_configuration(execution_script = 'scoring_service.py',
                                                  runtime = runtime,
                                                  conda_file = conda_file,
                                                  docker_file = 'dockerfile')

# create the image
image_name = 'nyc-taxi-automl-image'

from azureml.core import Image
image = Image.create(name=image_name, models=[model], image_config=image_config, workspace=ws)

# wait for image creation to finish
image.wait_for_creation(show_output=True)
```

```
Creating container image configuration...
/anaconda/envs/azureml_py36/lib/python3.6/site-packages/ipykernel_launcher.py:10: DeprecationWarning: ContainerImage class has been deprecated and will be removed in a future release. Please migrate to using Environments. https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-environments
  # Remove the CWD from sys.path while we load stuff.
/anaconda/envs/azureml_py36/lib/python3.6/site-packages/ipykernel_launcher.py:16: DeprecationWarning: Image class has been deprecated and will be removed in a future release. Please migrate to using Environments. https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-environments
  app.launch_new_instance()

Creating image
Running.....................................................................
Succeeded
Image creation operation finished for image nyc-taxi-automl-image:1, operation "Succeeded"
```

Deploy Model to Azure Container Instance (ACI) as a Web Service

Create and connect to an Azure Machine Learning Workspace

Run the following cell to create a new Azure Machine Learning **Workspace**.

Important Note: You will be prompted to login in the text that is output below the cell. Be sure to navigate to the URL displayed and enter the code that is provided. Once you have entered the code, return to this notebook and wait for the output to read `Workspace configuration succeeded`.

In [5]:
```
ws = Workspace.create(
    name = workspace_name,
    subscription_id = subscription_id,
    resource_group = resource_group,
    location = workspace_region,
    exist_ok = True)

ws.write_config()

print('Workspace configuration succeeded')
```

```
Workspace configuration succeeded
```

In [6]:
```
# Display a summary of the current environment
import pandas as pd
output = {}
output['SDK version'] = azureml.core.VERSION
output['Workspace'] = ws.name
output['Resource Group'] = ws.resource_group
output['Location'] = ws.location
pd.set_option('display.max_colwidth', -1)
pd.DataFrame(data=output, index=['']).T
```

Out[6]:

SDK version	1.9.0
Workspace	quick-starts-ws-57170
Resource Group	aml-quickstarts-57170
Location	southcentralus

Register Model

Azure Machine Learning provides a Model Registry that acts like a version controlled repository for each of your trained models. To version a model, you use the SDK as follows. Run the following cell to register the best model with Azure Machine Learning.

In [7]:
```
# register the model for deployment
model = Model.register(model_path = model_path, # this points to a local file
                       model_name = "nyc-taxi-automl-predictor", # name the model is registered as
                       tags = {'area': "auto", 'type': "regression"},
                       description = "NYC Taxi Fare Predictor",
                       workspace = ws)

print()
print("Model registered: {} \nModel Description: {} \nModel Version: {}".format(model.name,
                                                                                model.description, model.version))
```

```
Registering model nyc-taxi-automl-predictor

Model registered: nyc-taxi-automl-predictor
Model Description: NYC Taxi Fare Predictor
Model Version: 1
```

```
ERROR - Service deployment polling reached non-successful terminal state, current service state: Unhealthy
Operation ID: af336a30-b9f2-4fb9-84ed-77670d4a70c7
More information can be found using '.get_logs()'
Error:
{
  "code": "DeploymentTimedOut",
  "statusCode": 504,
  "message": "The deployment operation polling has TimedOut. The service creation is taking longer than our normal time. We a
re still trying to achieve the desired state for the web service. Please check the webservice state for the current webservic
e health. You can run print(service.state) from the python SDK to retrieve the current state of the webservice."
}

ERROR - Service deployment polling reached non-successful terminal state, current service state: Unhealthy
Operation ID: af336a30-b9f2-4fb9-84ed-77670d4a70c7
More information can be found using '.get_logs()'
Error:
{
  "code": "DeploymentTimedOut",
  "statusCode": 504,
  "message": "The deployment operation polling has TimedOut. The service creation is taking longer than our normal time. We a
re still trying to achieve the desired state for the web service. Please check the webservice state for the current webservic
e health. You can run print(service.state) from the python SDK to retrieve the current state of the webservice."
}
```

```
Creating image
Running.....................................................................
Succeeded
Image creation operation finished for image nyc-taxi-automl-image:1, operation "Succeeded"
```

Deploy Model to Azure Container Instance (ACI) as a Web Service

In [*]:
```
from azureml.core.webservice import AciWebservice, Webservice

aci_name = 'aci-cluster'

aci_config = AciWebservice.deploy_configuration(
    cpu_cores = 1,
    memory_gb = 1,
    tags = {'name': aci_name},
    description = 'NYC Taxi Fare Predictor Web Service')

service_name = 'nyc-taxi-srv'

aci_service = Webservice.deploy_from_image(deployment_config=aci_config,
                                           image=image,
                                           name=service_name,
                                           workspace=ws)

aci_service.wait_for_deployment(show_output=True)
```

```
/anaconda/envs/azureml_py36/lib/python3.6/site-packages/ipykernel_launcher.py:16: DeprecationWarning: deploy_from_image has bee
n deprecated and will be removed in a future release. Please migrate to using Environments. https://docs.microsoft.com/en-us/az
ure/machine-learning/how-to-use-environments
  app.launch_new_instance()

Runnin
g.....................................................................................................................
.....................................................................................................................
.....................................................................................
TimedOut
```

Test the deployed web service

Make direct calls on the service object

In [ ]:
```
import json

data1 = [1, 2, 5, 9, 4, 27, 5, 'Memorial Day', True, 0, 0.0, 0.0, 65]

data2 = [[1, 3, 10, 15, 4, 27, 7, 'None', False, 0, 2.0, 1.0, 80],
         [1, 2, 5, 9, 4, 27, 5, 'Memorial Day', True, 0, 0.0, 0.0, 65]]

result = aci_service.run(json.dumps(data1))
print('Predictions for data1')
print(result)

result = aci_service.run(json.dumps(data2))
print('Predictions for data2')
print(result)
```

Consume the Deployed Web Service

Execute the code below to consume the deployed webservice over HTTP.

In [ ]:
```
import requests

url = aci_service.scoring_uri
print('ACI Service: {} scoring URI is: {}'.format(service_name, url))
headers = {'Content-Type':'application/json'}

response = requests.post(url, json.dumps(data1), headers=headers)
print('Predictions for data1')
print(response.text)
response = requests.post(url, json.dumps(data2), headers=headers)
print('Predictions for data2')
print(response.text)
```

Scoring Service

In [ ]:
```
```

In [ ]:
```
import json
import numpy as np
import pandas as pd
import azureml.train.automl
```

In [ ]:
```
columns = ['vendorID', 'passengerCount', 'tripDistance', 'hour_of_day', 'day_of_week', 'day_of_month',
           'month_num', 'normalizeHolidayName', 'isPaidTimeOff', 'snowDepth', 'precipTime',
           'precipDepth', 'temperature']
```

In [ ]:
```
def init():
    try:
        # One-time initialization of predictive model and scaler
        from azureml.core.model import Model
        from sklearn.externals import joblib
        global model

        model_name = 'nyc-taxi-automl-predictor'
        print('Looking for model path for model: ', model_name)
        model_path = Model.get_model_path(model_name=model_name)
        print('Looking for model in: ', model_path)
        model = joblib.load(model_path)
        print('Model loaded...')

    except Exception as e:
        print('Exception during init: ', str(e))
```

In [ ]:
```
def run(input_json):
    try:
        inputs = json.loads(input_json)
        data_df = pd.DataFrame(np.array(inputs).reshape(-1, len(columns)), columns = columns)
        # Get the predictions...
        prediction = model.predict(data_df)
        prediction = json.dumps(prediction.tolist())
    except Exception as e:
        prediction = str(e)
    return prediction
```

nyc-taxi-srv

Service ID
nyc-taxi-srv

Description
NYC Taxi Fare Predictor Web Service

Deployment state
Unhealthy

Compute type
ACI

Created by
ODL_User 62865

Model ID
--

Created on
8/3/2020 10:27:30 PM

Last updated on
8/3/2020 10:27:30 PM

Image ID
nyc-taxi-automl-image:1

REST endpoint
http://f00885be-7ef8-4b80-a5a8-04548631da0e.centralus.azurecontainer.io/score

Key-based authentication enabled
false

Swagger URI
--

CPU
1

Memory
1 GB

Application Insights enabled
false

name
aci-cluster

Properties

azureml.git.repository_uri
https://github.com/solliancenet/udacity-intro-to-ml-labs.git

mlflow.source.git.repoURL
https://github.com/solliancenet/udacity-intro-to-ml-labs.git

azureml.git.branch
master

mlflow.source.git.branch
master

azureml.git.commit
3e314d14b42cca7367f169dccfb0524d12948021

mlflow.source.git.commit
3e314d14b42cca7367f169dccfb0524d12948021

azureml.git.dirty
True

hasInferenceSchema
False

hasHttps
False

nyc-taxi-srv

Service ID
nyc-taxi-srv

Description
NYC Taxi Fare Predictor Web Service

Deployment state
Unhealthy

Compute type
ACI

Created by
ODL_User 57170

Model ID
--

Created on
7/31/2020 8:46:59 AM

Last updated on
7/31/2020 8:46:59 AM

Image ID
nyc-taxi-automl-image:1

REST endpoint
http://b853a618-143f-4285-bbf0-5da84060e977.southcentralus.azurecontainer.io/score

Key-based authentication enabled
false

Swagger URI
--

CPU
1

Memory
1 GB

Application Insights enabled
false

name
aci-cluster

Properties

azureml.git.repository_uri
https://github.com/solliancenet/udacity-intro-to-ml-labs.git

mlflow.source.git.repoURL
https://github.com/solliancenet/udacity-intro-to-ml-labs.git

azureml.git.branch
master

mlflow.source.git.branch
master

azureml.git.commit
3e314d14b42cca7367f169dccfb0524d12948021

mlflow.source.git.commit
3e314d14b42cca7367f169dccfb0524d12948021

azureml.git.dirty
True