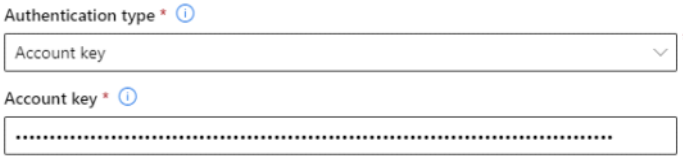
Azure Machine Learning, AzureML, Exam DP-100: Designing and Implementing a Data Science Solution, 4 End-to-End Projects

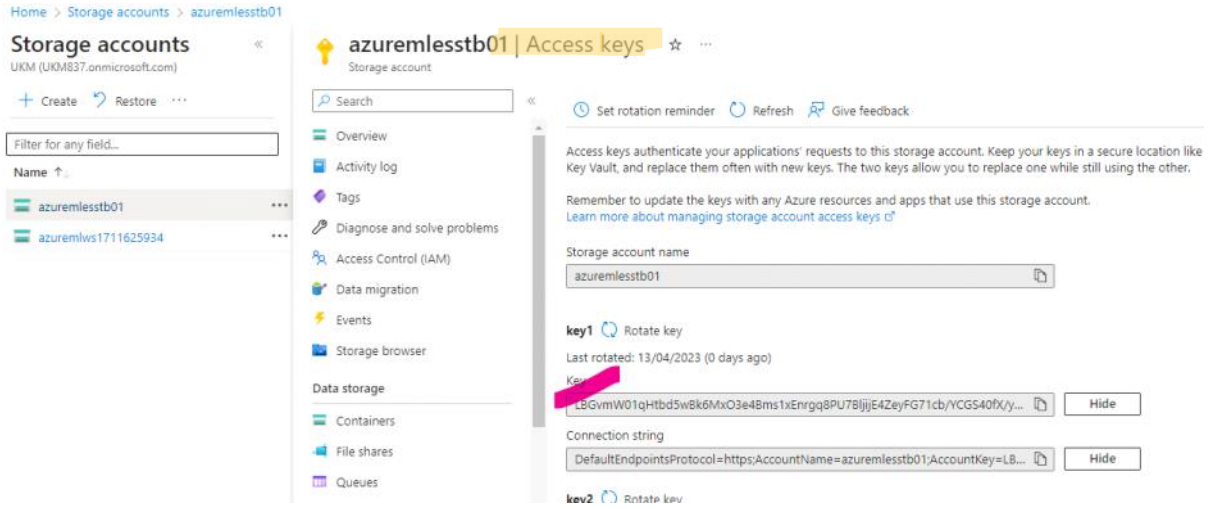
Thursday, 13 April, 2023 1:49 PM

Section 2: Setting up Azure ML Workspace and Resources

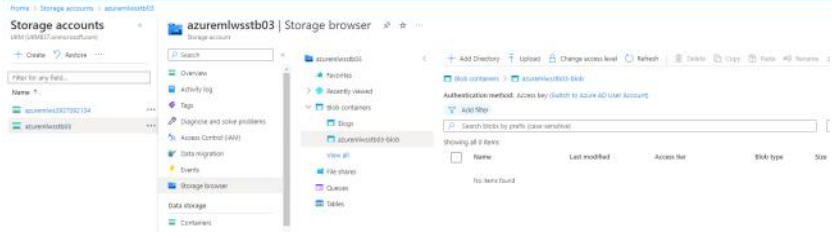
Thursday, 13 April, 2023 1:50 PM

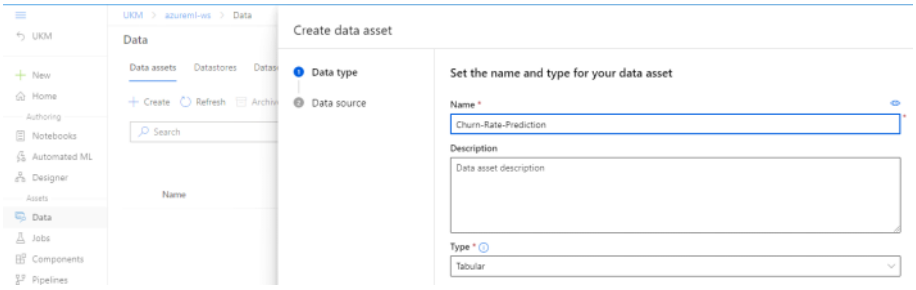
1. Create a Resource Group
2. Create Azure ML Workspace
3. Create Compute Instance/Cluster
4. Creating a Datastore
   (The default datastore is tied to the workspace, so if the workspace is deleted, the datastore will be deleted as well. Therefore, we need to create another datastore to store the data to prevent data loss.)
   Steps to create a Dataset:
   a. Create a new storage account(with a new resource group) & upload data
   b. Create a container (to dump data into the blob container)
      - Created a blob container
   c. Create datastore
      - To get the account key:

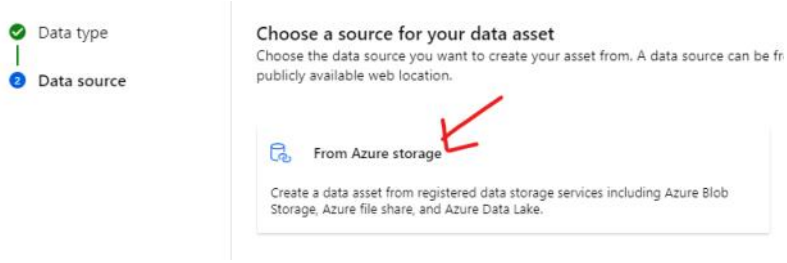




   d. Create Data assets/Datasets (from Azure Storage)







1. *Remove resource groups when it is not in use anymore

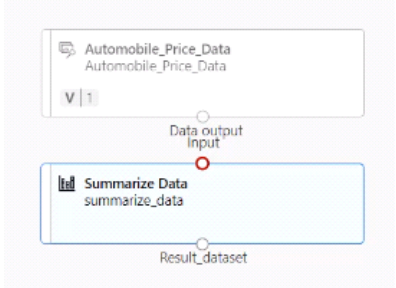
Section 3: Running Experiments and Training Models

Thursday, 13 April, 2023 2:57 PM

New Section (More detailed explanation, similar to Section 6):

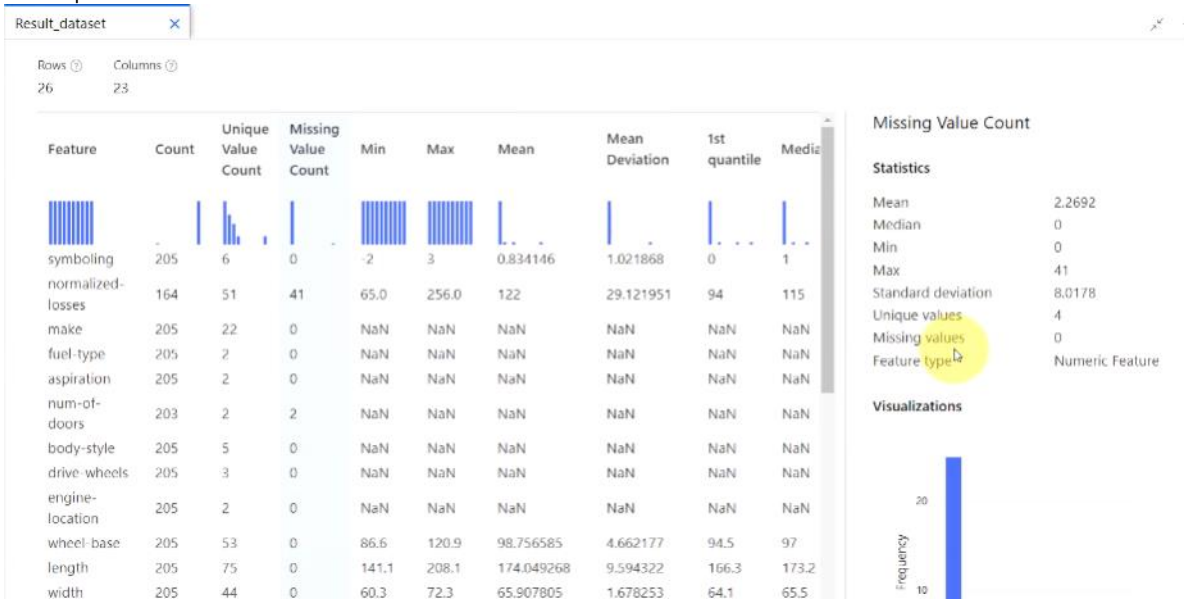
1. Creating Resources (as shown below)

2. Identify the Business problem
3. Summarize Data (Analyse Data)

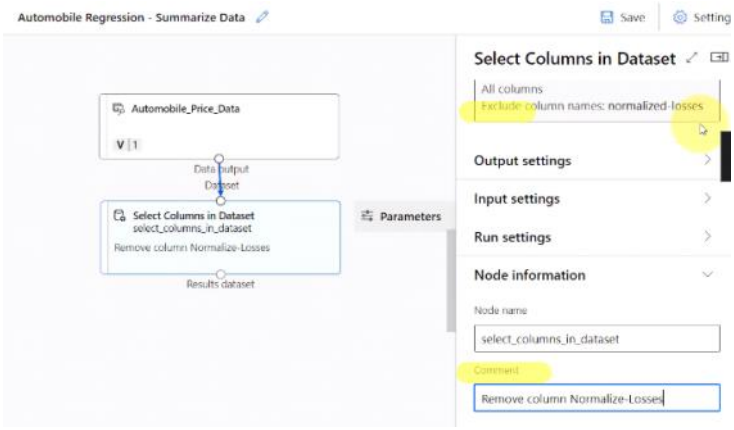


To run the pipeline, an experiment/job is created

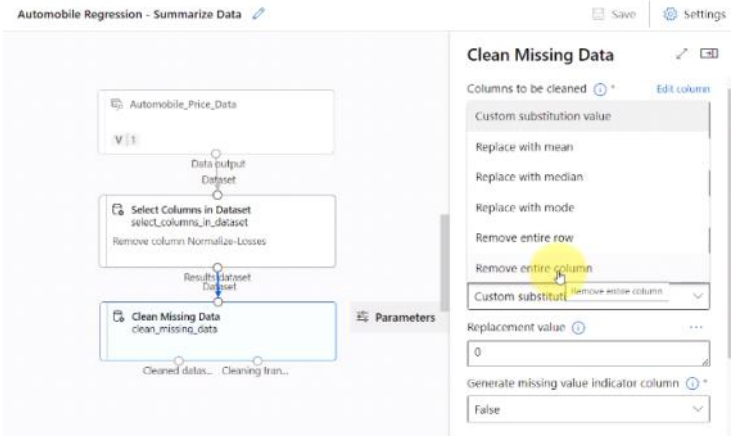
The outputs:



4. Data Pre-processing
   a. Select columns/Exclude Columns



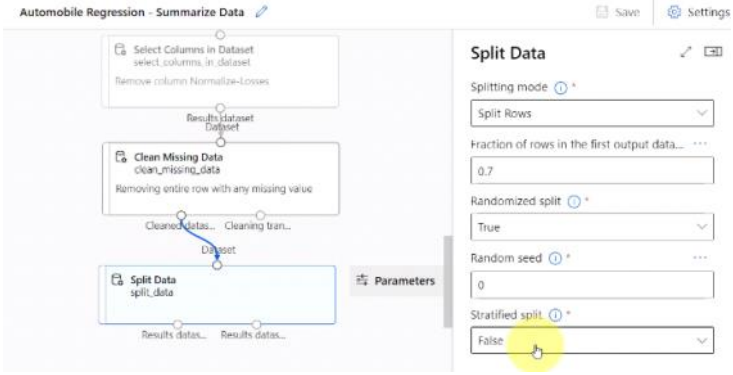
b. Clean missing data (remove entire row)



There are 2 outputs (Cleaned dataset & Cleaning transformation)

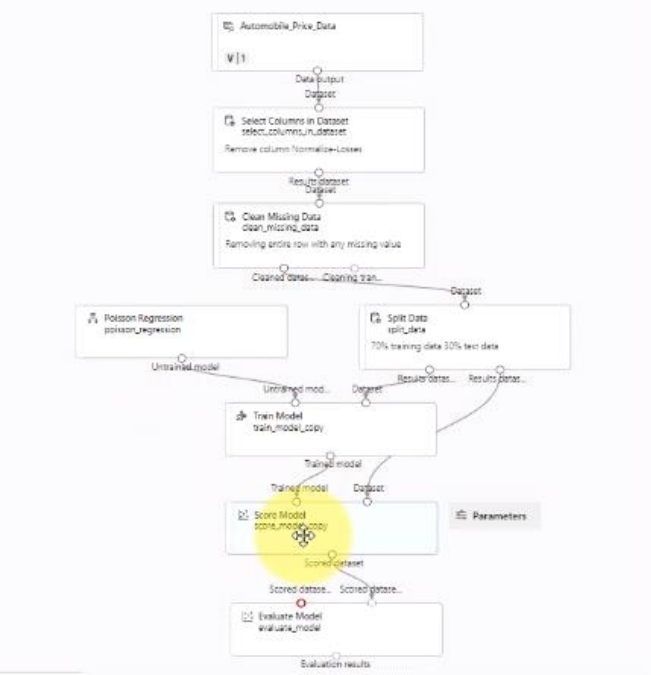
- Cleaning transformation (Apply the same cleaning transformation to another dataset)

c. Split data



Regression (Stratified Split-False)

5. Training ML Model with Linear Regression (Online Gradient Descent), Boosted DT, Decision Forest Regression, Poisson Regression
6. Evaluating the Results (Feature Importance, Scores etc.)
7. Finalizing ML model (best score - Poisson Regression)

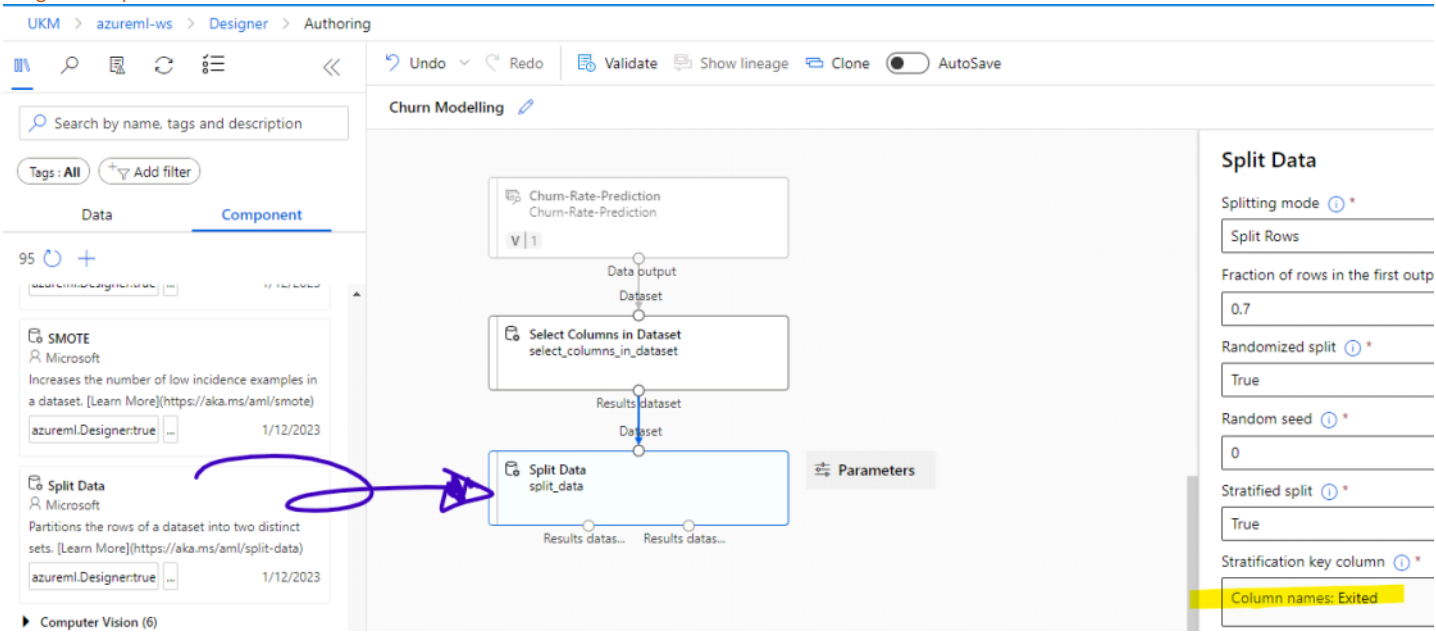


8. Creating Real-Time Inference Pipeline

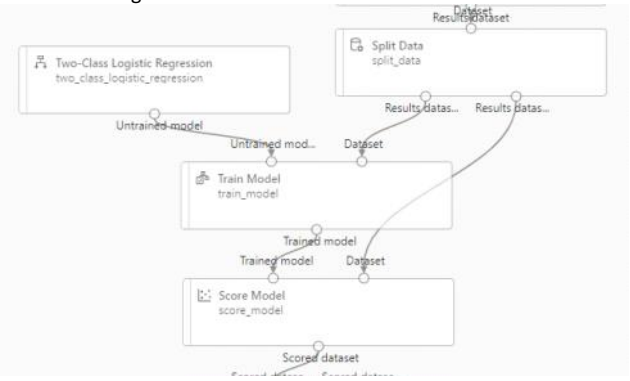
Churn Modeling

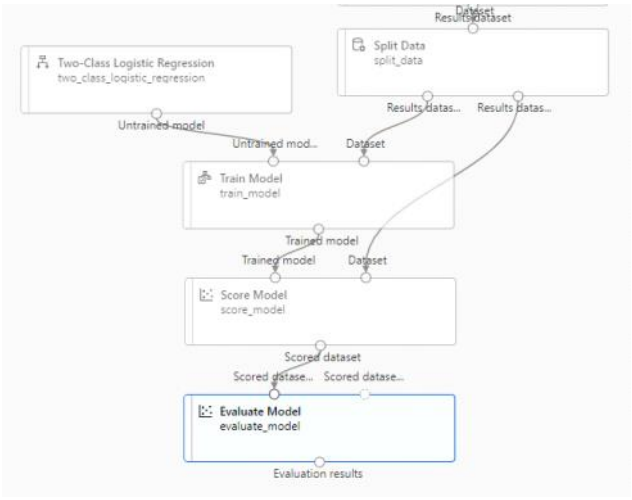
- **The Components**
  a. Data Pre-processing

Drag and drop



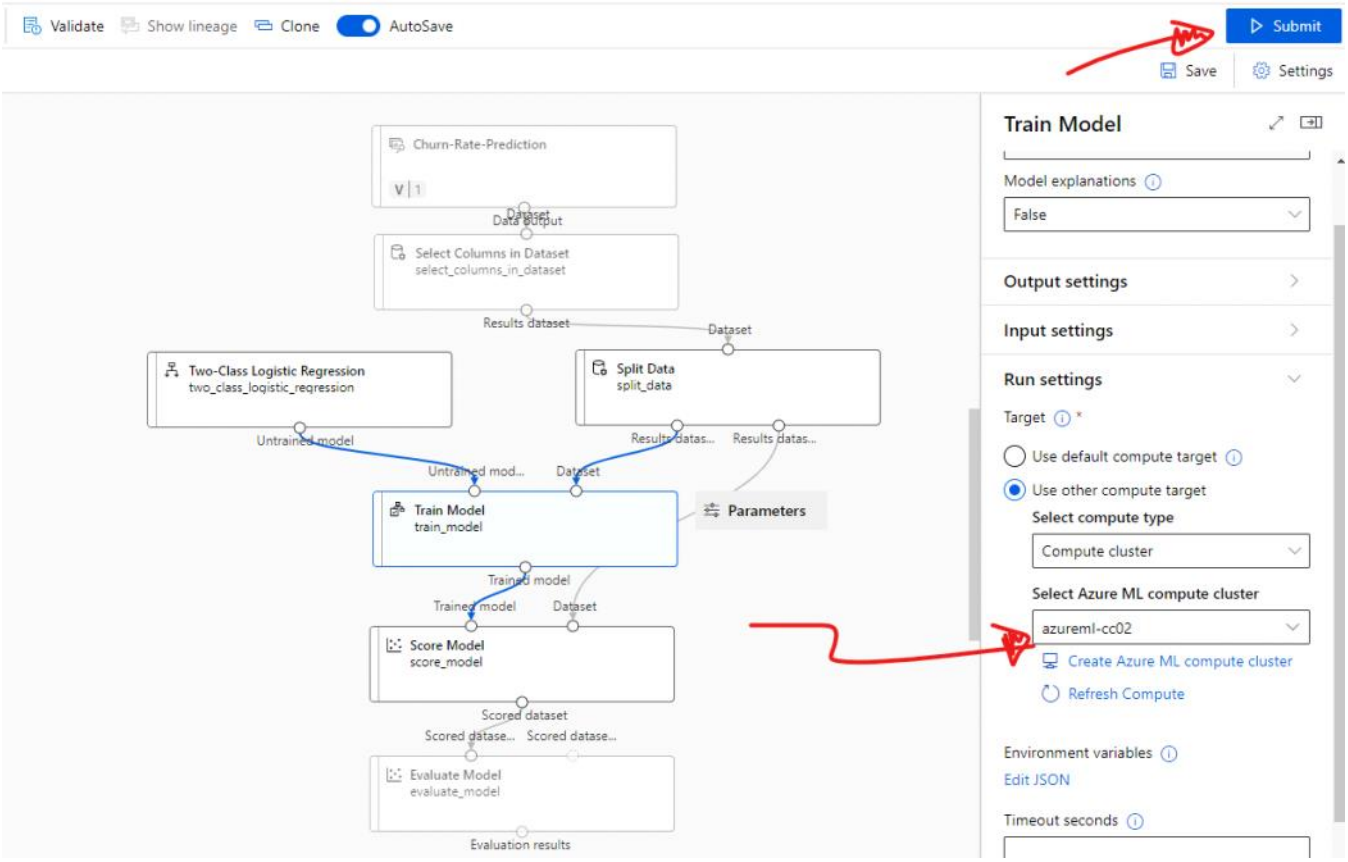
a. Modeling





- **Run pipeline**

(Can change different compute cluster to run diff components)



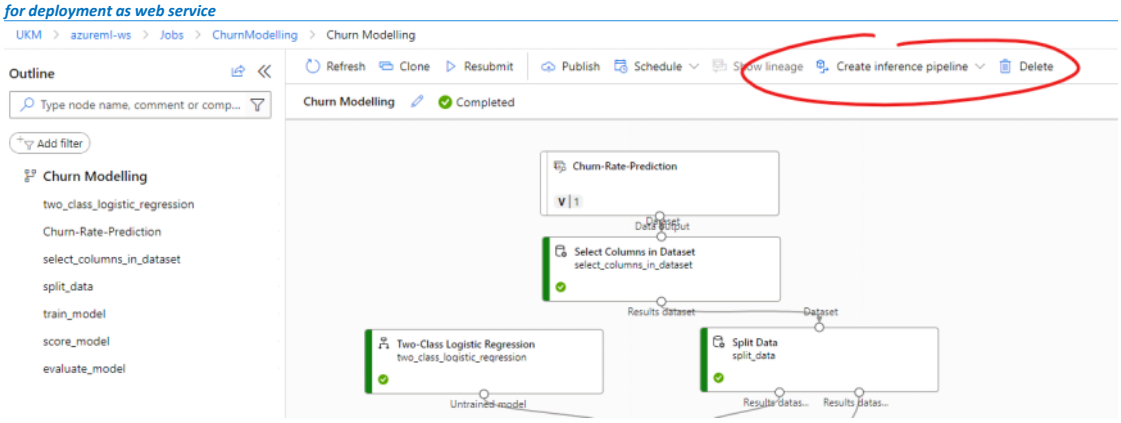
Section 4: Deploying the Models

Saturday, 15 April, 2023 6:23 PM

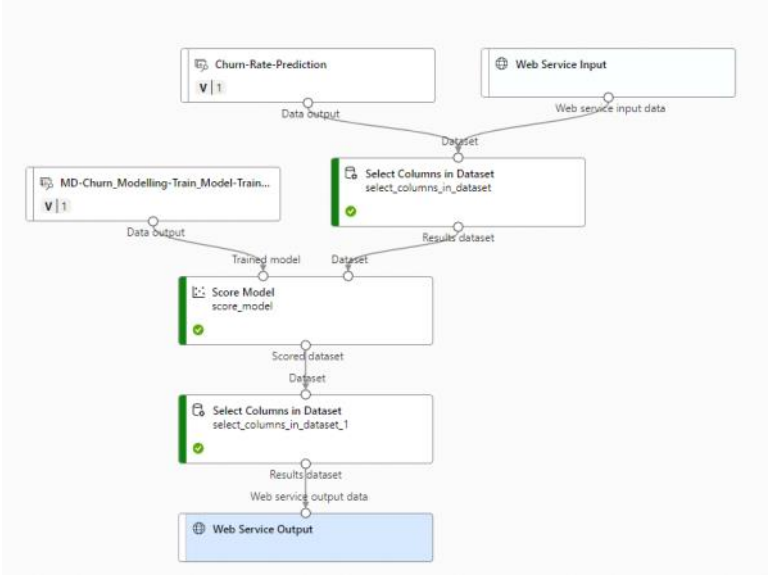
Churn Modeling

- **Create pipeline - Real Time Inference**

for deployment as web service

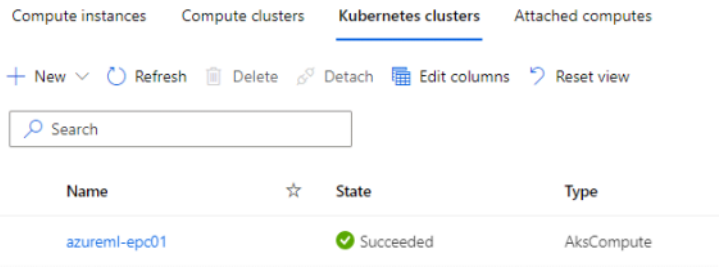


- Removed target variable "Exited" from "Select Columns in Dataset" component
- Removed "Evaluate Model" component
- Added "Select Columns in Dataset" after "score model" component - to select **Scored Labels, Scored Probabilities only**
- Added Web Service Input and Output

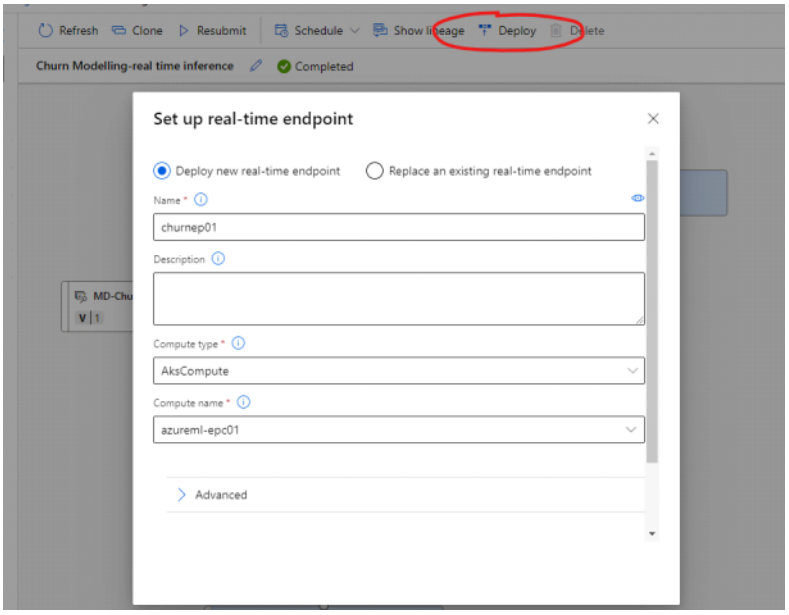


Deployment

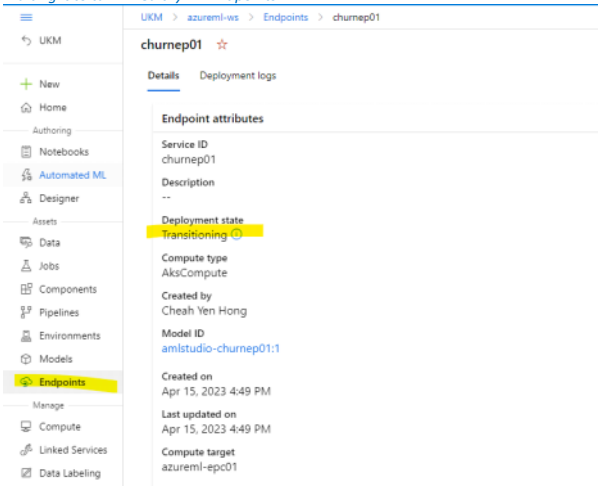
a. Create Kubernetes Clusters



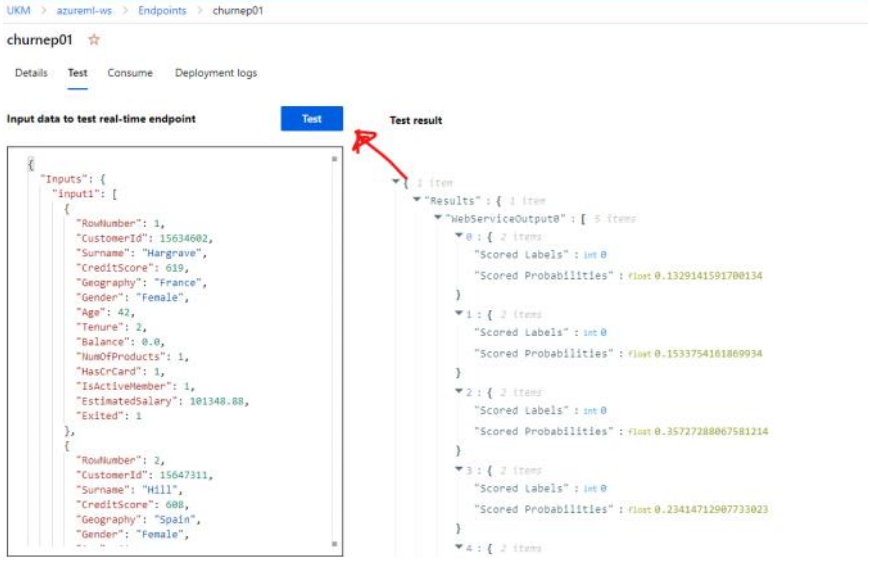
b. Deploy



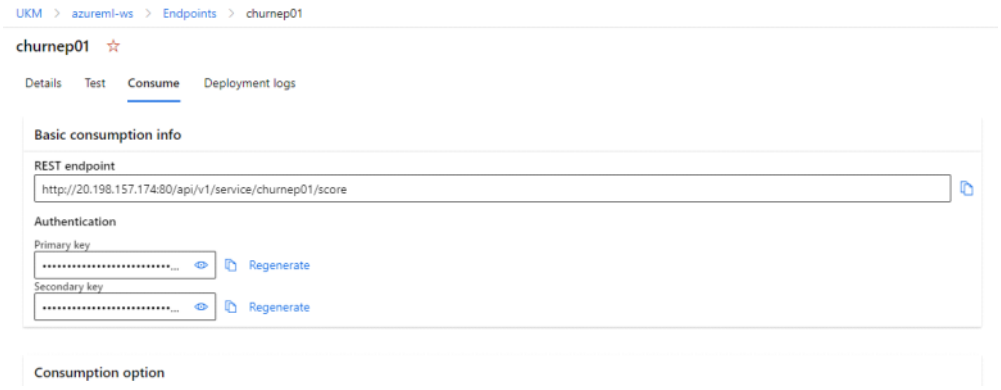
Waiting it to turn "Healthy" in Endpoints ...



After it turned "Healthy"

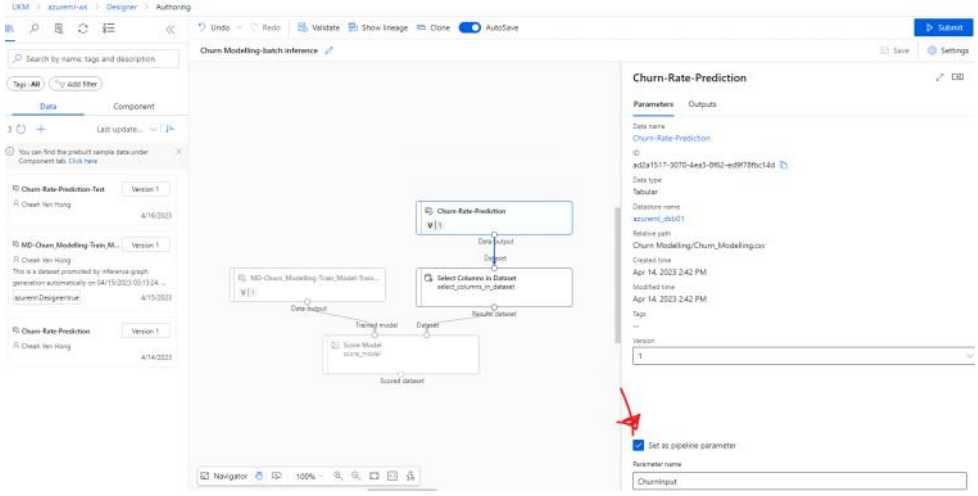


Done 😊

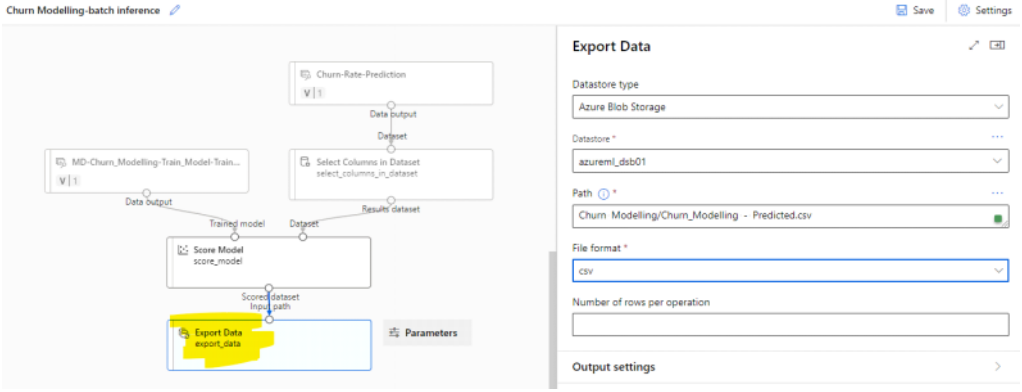


- **Create pipeline - Batch Inference**

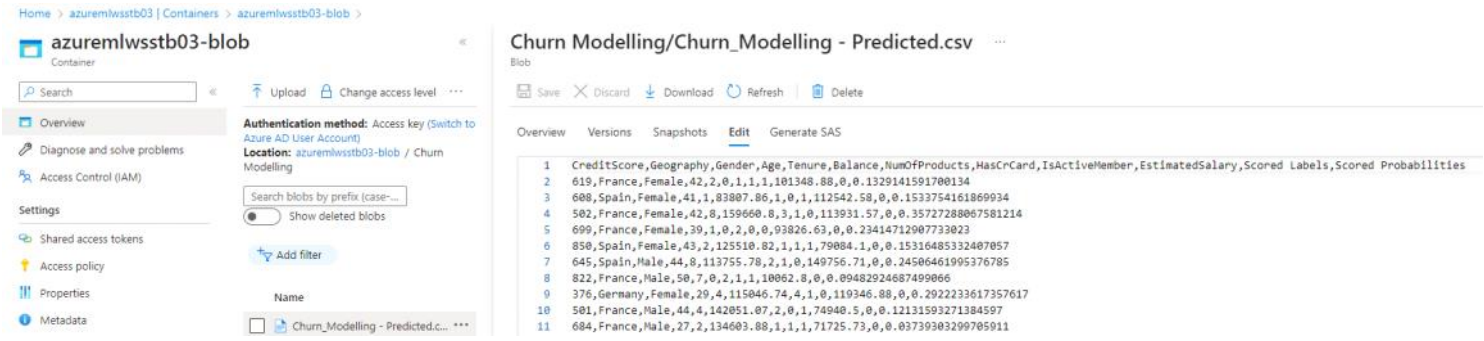
  No Webservice input/output, ***batch inference is expecting a dataset as an input param***
  - **Set dataset as pipeline parameter** (after deploying the ML model, we will have to make predictions on a new dataset. This new dataset will need to be specified as a parameter. Batch inference pipeline will expect a dataset as an input param)
  - ~~Removed target variable "XXX" from "Select Columns in Dataset" component~~
  - Removed "Evaluate Model" component
  - ~~Added "Select Columns in Dataset" after "score model" component - to select Scored Labels, Scored Probabilities only~~ **(We need the final dataset)**
  - ~~Added Web Service Input and Output~~



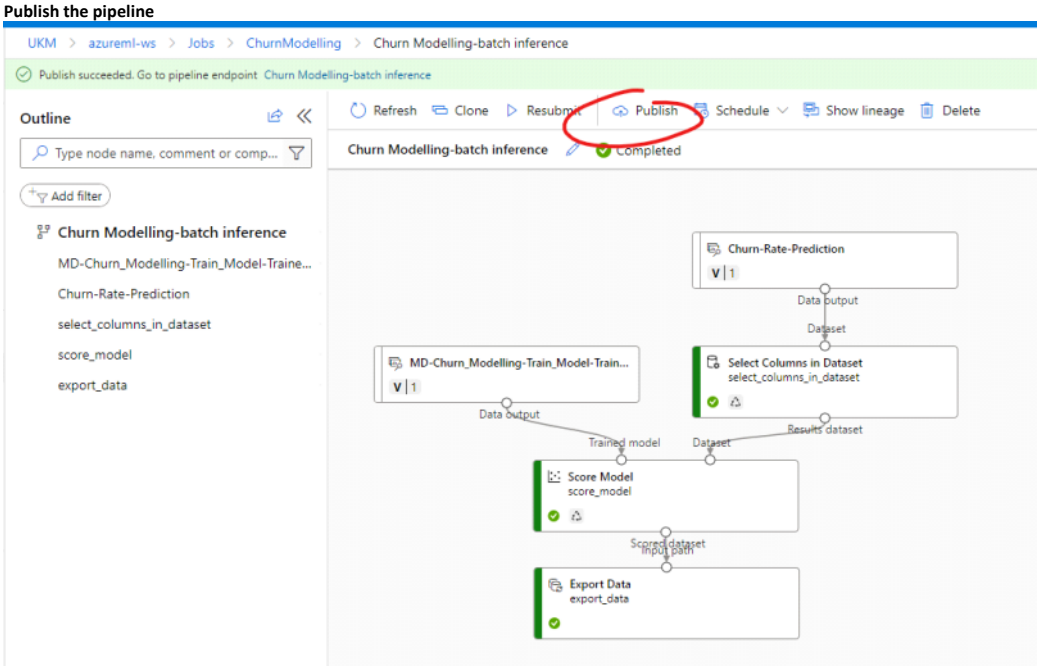
  - Added "Export Data" component, to export the output data (check "Outputs + logs") from "Score Model" to blob storage



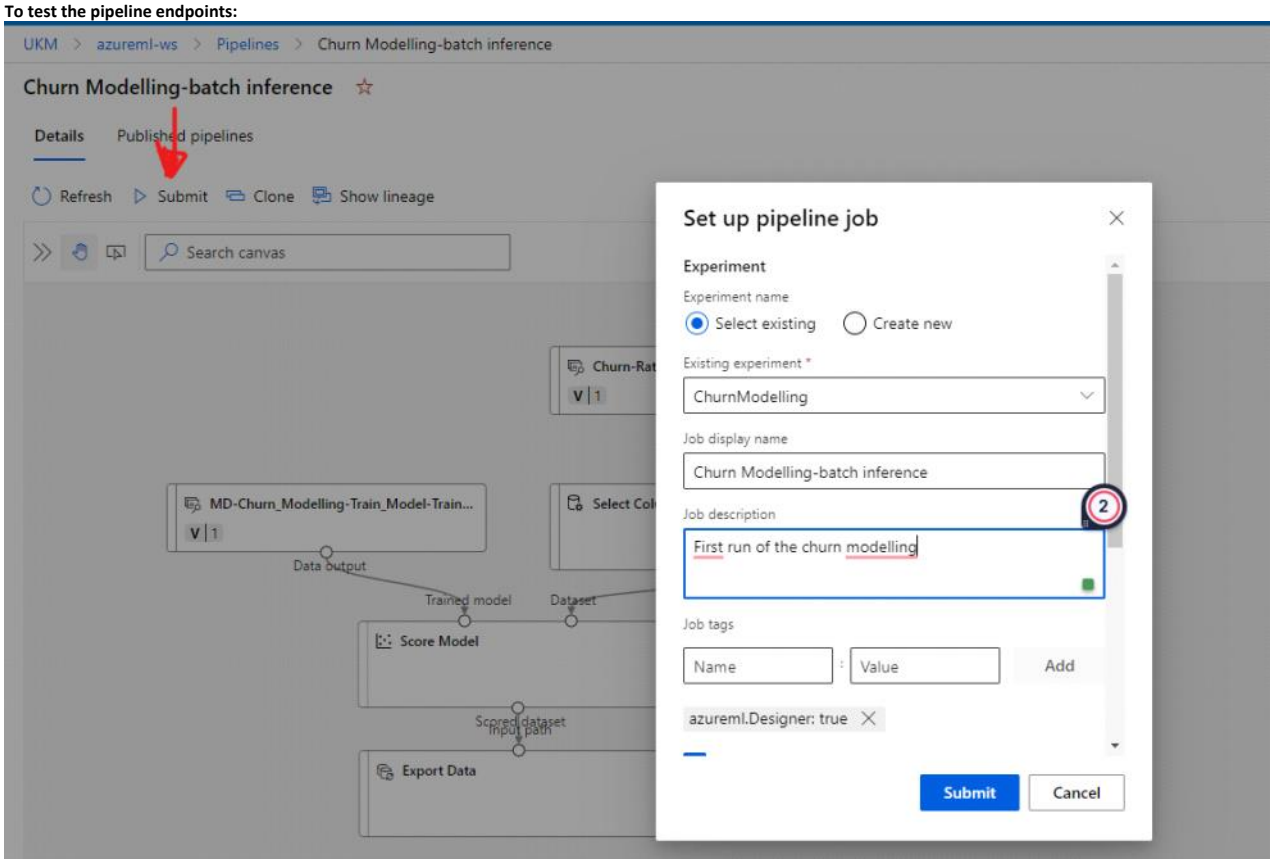
Submit



Publish the pipeline



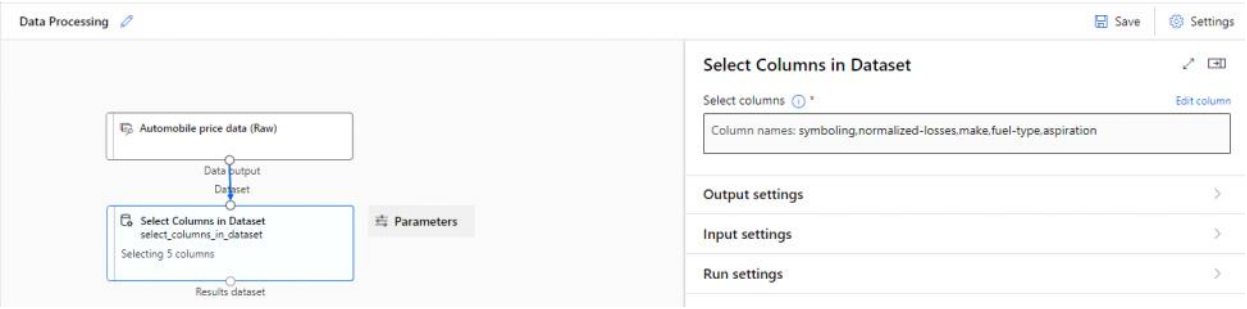
To test the pipeline endpoints:



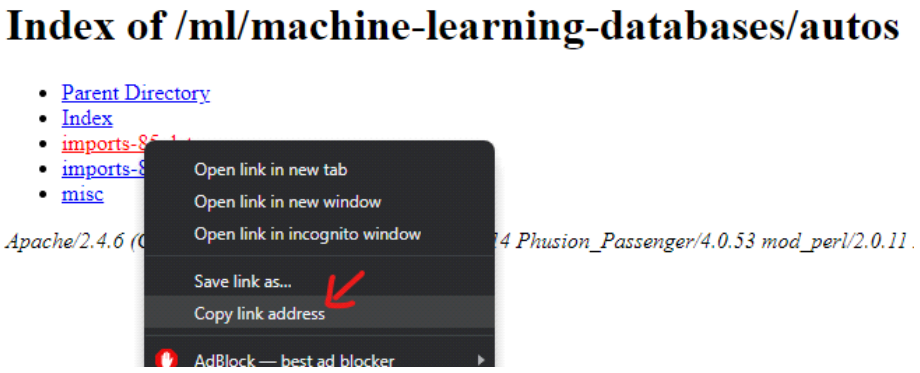
Section 5: AzureML Designer: Data Preprocessing

Sunday, 16 April, 2023 2:50 PM

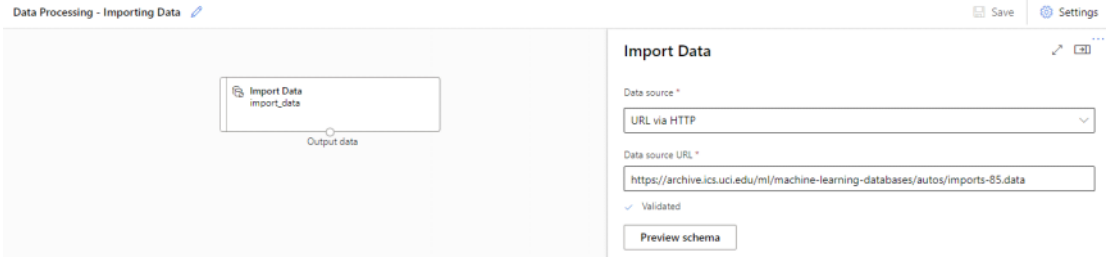
- **Select Columns in Sample Dataset**
  Pipeline - Data Processing



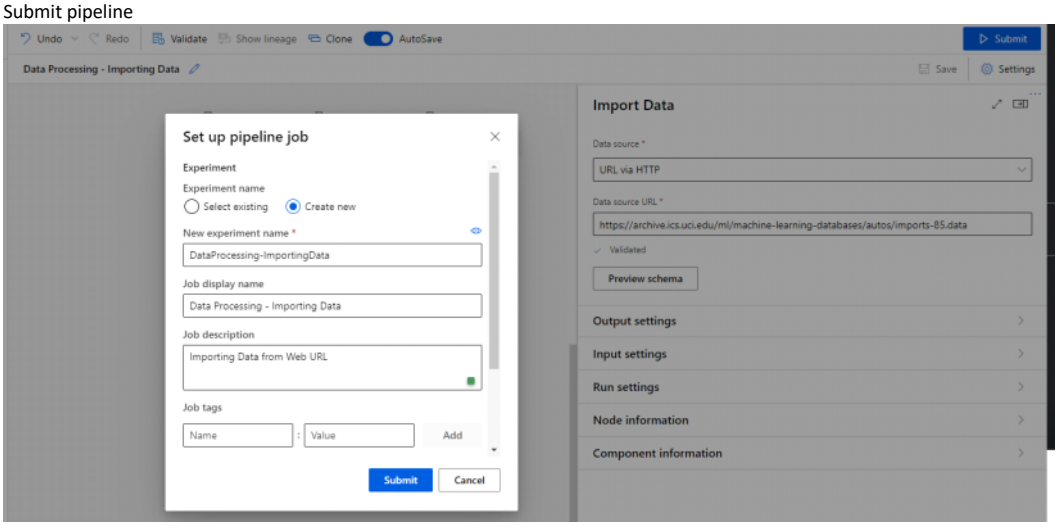
- **To import dataset from web URL**
  Pipeline - Data Processing - Importing Data
  - https://archive.ics.uci.edu/ml/datasets/Automobile
    Click on Data Folder



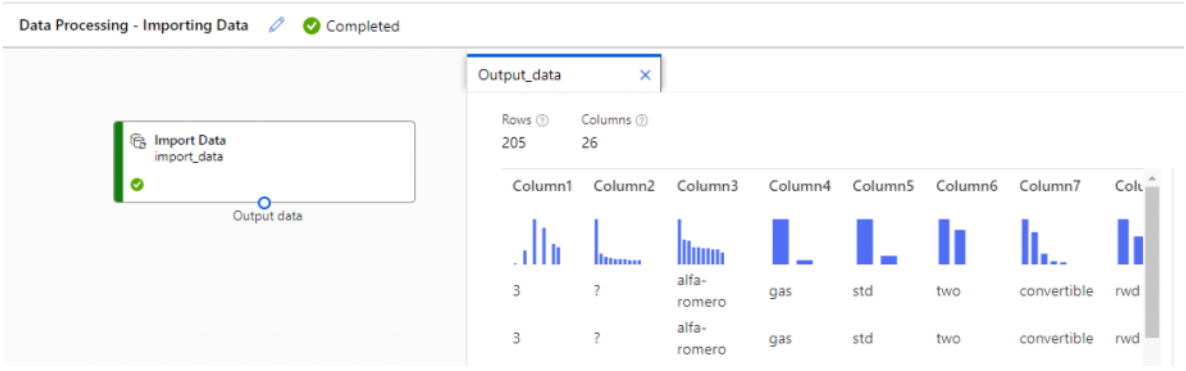
  - Import Data



  - Submit pipeline
  -

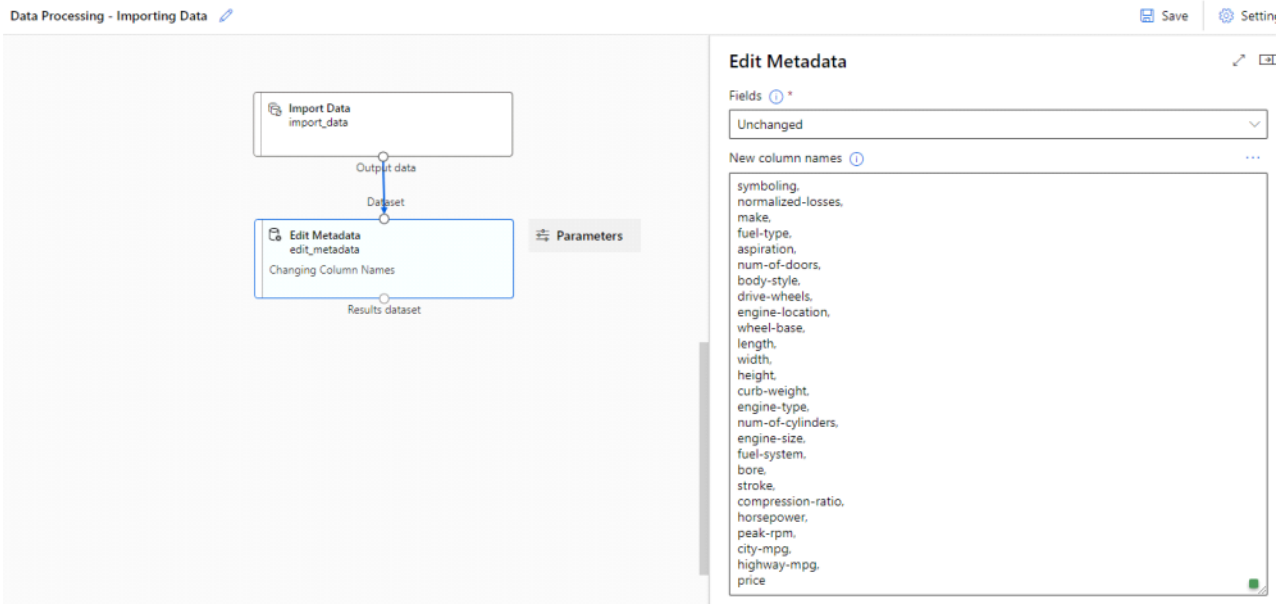


    The outputs:



- **Edit metadata - Column Names**

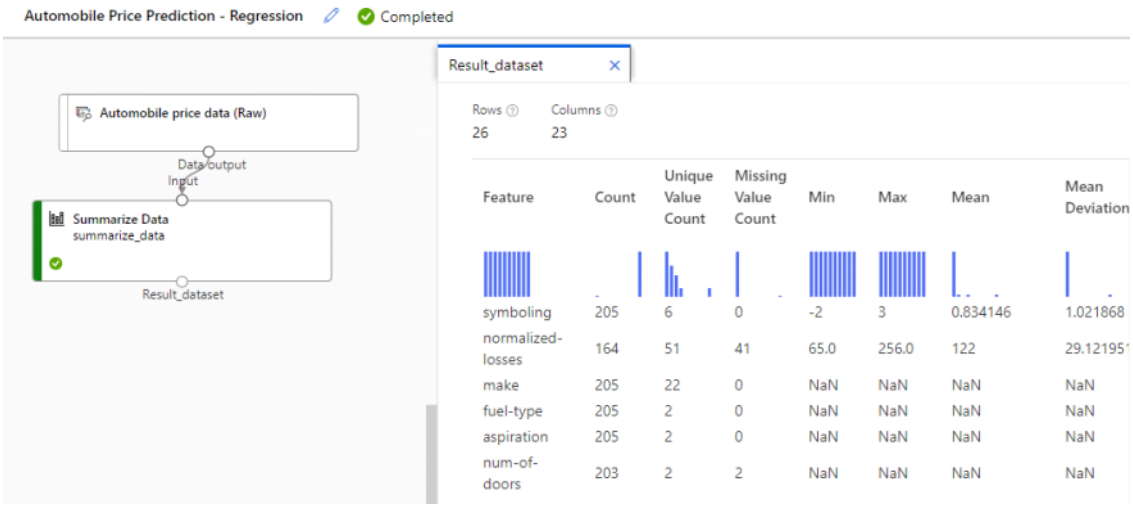
symboling,normalized-losses,make,fuel-type,aspiration,num-of-doors,body-style,drive-wheels,engine-location,wheel-base,length,width,height,curb-weight,engine-type,num-of-cylinders,engine-size,fuel-system,bore,stroke,compression-ratio,horsepower,peak-rpm,city-mpg,highway-mpg,price



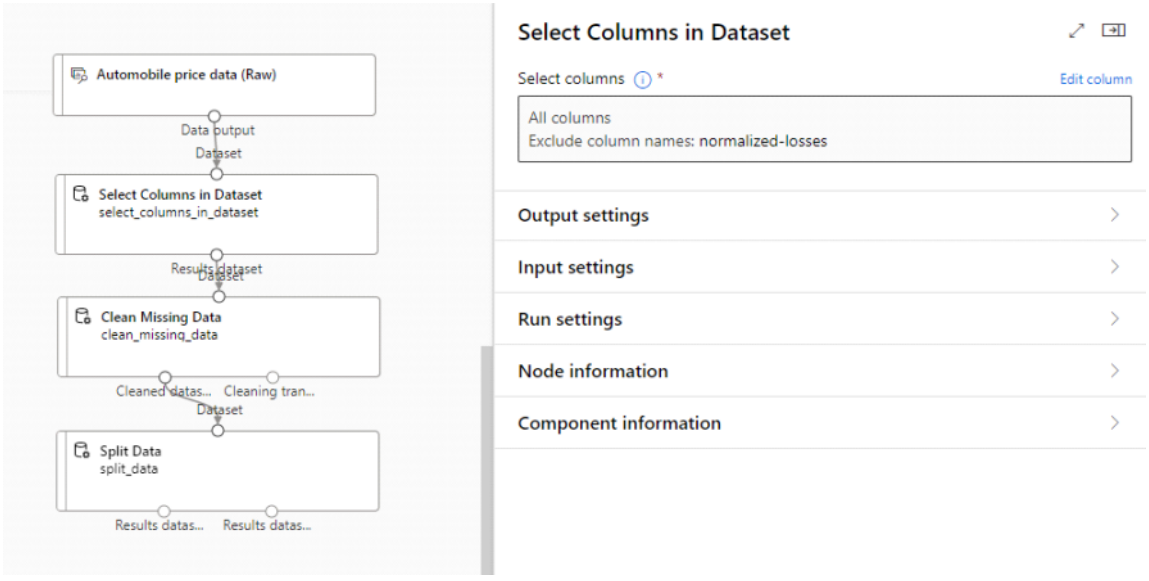
Section 6: Project 1: Regression Using AzureML Designer

Tuesday, 25 April, 2023 10:12 AM

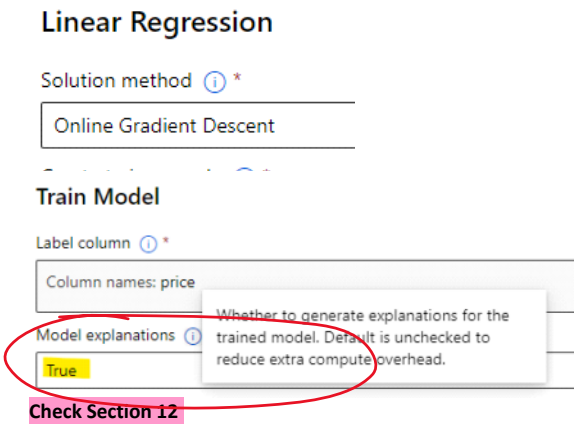
- **Analyzing the dataset**



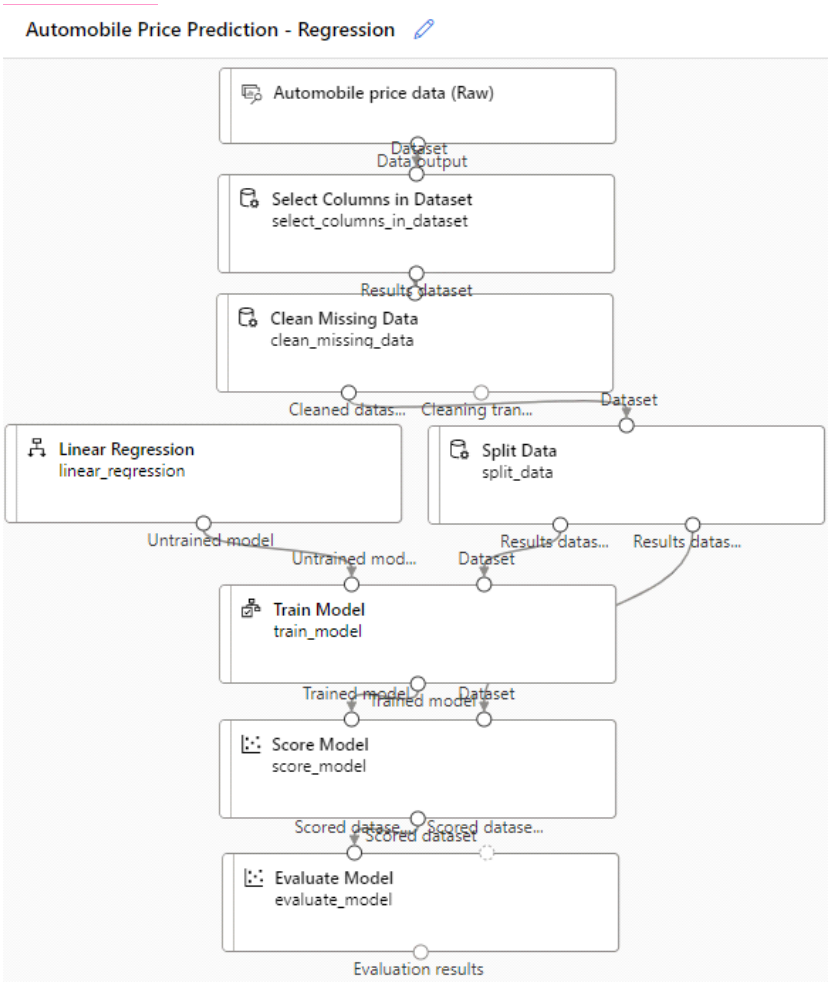
- **Data Preprocessing**



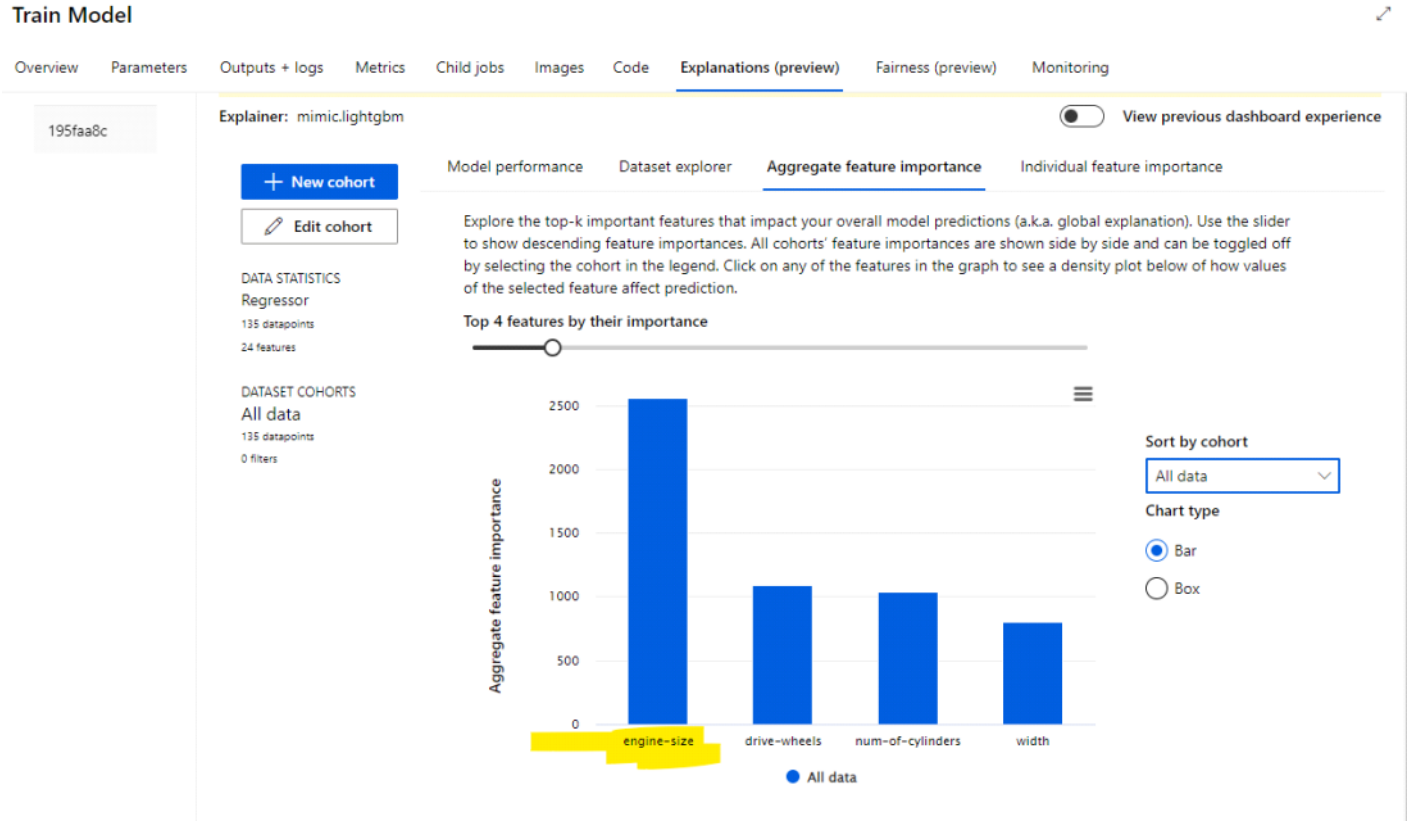
- **Training ML Model with Linear Regression (Online Gradient Descent)**



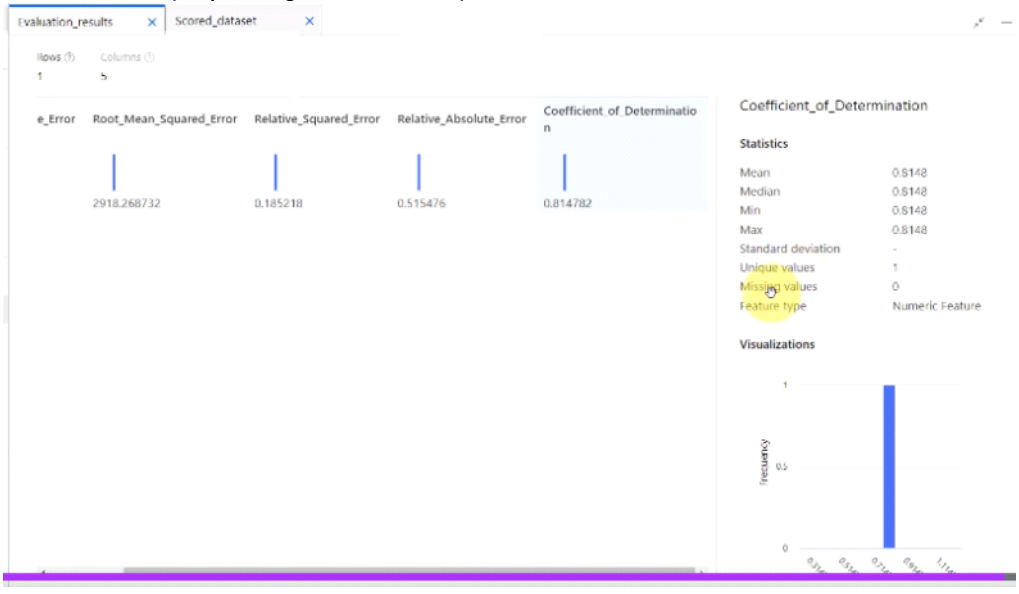
Check Section 12



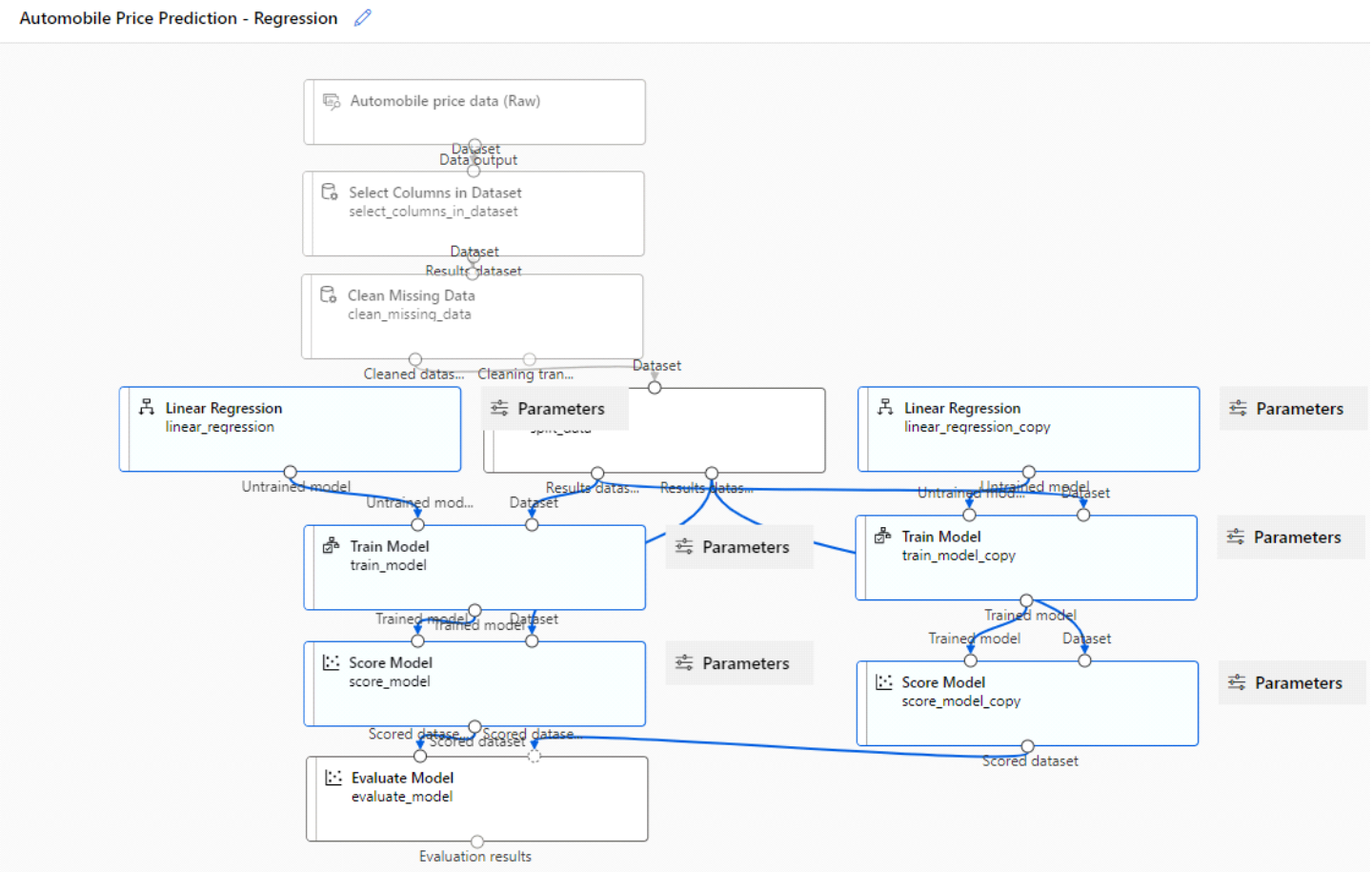
➢ **Evaluating the Results**



Evaluation Results (Outputs + logs - Evaluate Model)



- **Training ML Model with Linear Regression (Ordinary Least Square)**
  - **Copy 3 components from the original pipeline from Linear Regression (Online Gradient Descent) and change to Linear Regression (Ordinary Least Square)**
  - **Connect to the dataset and model evaluation components**



- ✓ **model comparison**

➢ **Training ML Model with Boosted Decision Tree and Decision Forest Regression**

➢ **Creating and Deploying Real-Time Inference Pipeline (with Boosted Decision Tree)**

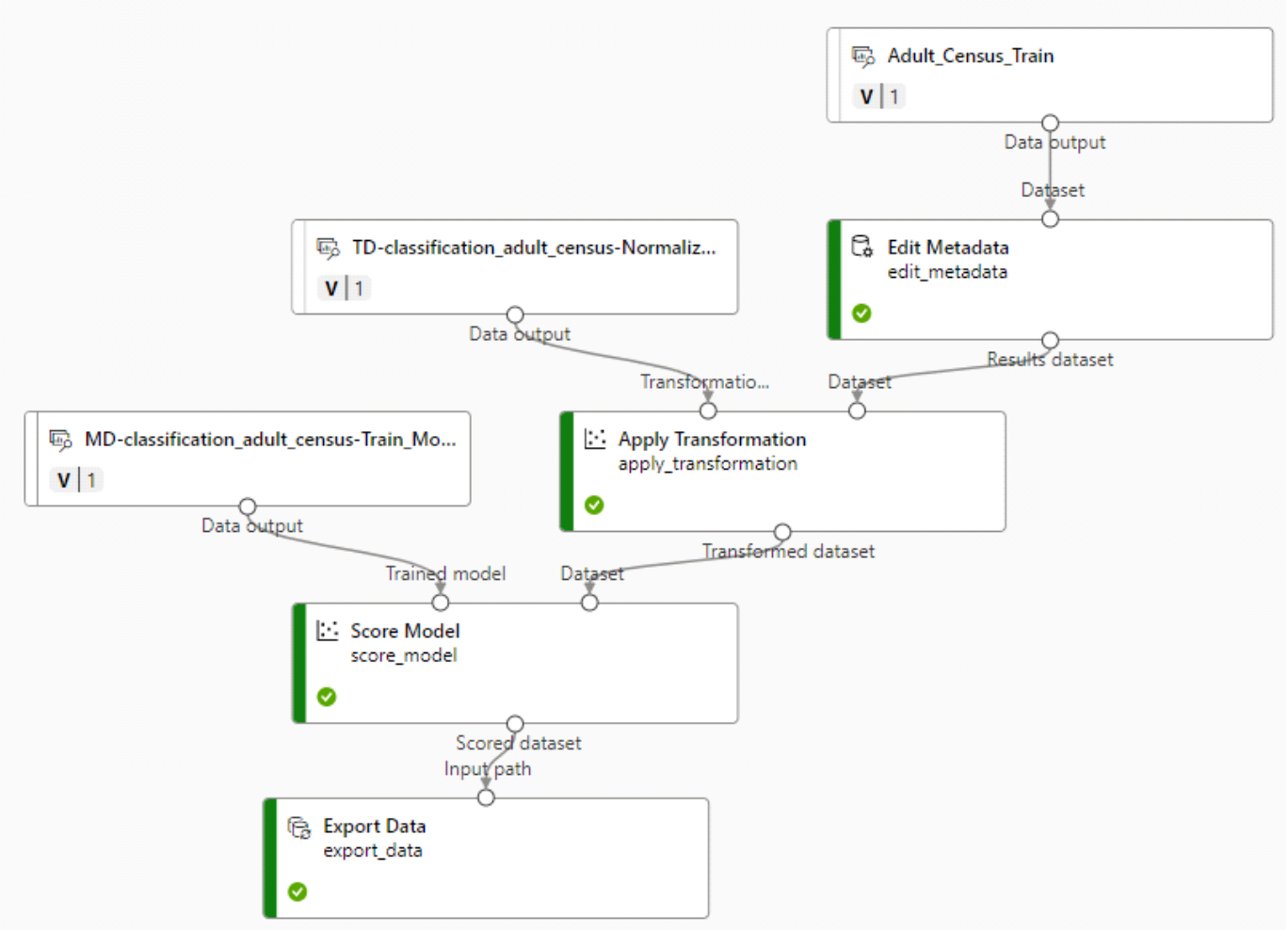
- **Create Real-Time Inference Pipeline**
  - **Delete "Evaluate Model" component**
  - **Create Inference/Kubernetes cluster**
  - **Deploy**

- **Creating and Deploying Batch Inference Pipeline**

Section 7: Project 2: Classification Using AzureML Designer

Wednesday, 26 April, 2023 2:19 PM

- **Creating and Deploying Batch Inference Pipeline**
  1. Set dataset as pipeline parameter
  2. Remove Evaluate Model component
  3. Add Export Data component



4. Publish batch inference pipeline and submit experiment with new test data

Section 8: AzureML SDK: Setting up Azure ML Workspace

Thursday, 27 April, 2023 10:22 AM

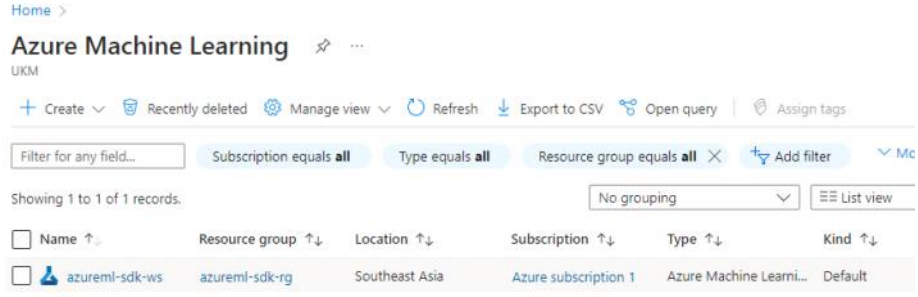
The notebooks in the tutorials: https://drive.google.com/drive/folders/1P_gTElhrWRPWd6k12xgJV8Py420M2Hc3

The Topics:

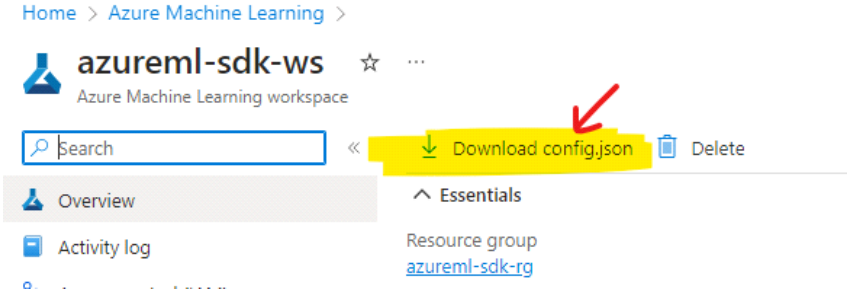
1. **Creating Workspace using AzureML SDK**

```
#Install azure ml SDK package
!pip install azureml-sdk -q

from azureml.core import Workspace

#Creating workspace object
ws = Workspace.create(name='azureml-sdk-ws',
                      subscription_id='5d2abb68-3a0a-4bfe-8994-6915771b5322',
                      resource_group='azureml-sdk-rg',
                      create_resource_group=True,
                      location='southeastasia')
```

(workspace has created)



2. **Creating a Datastore using AzureML SDK**
   a. Download config.json from the workspace created and access the workspace in the notebook

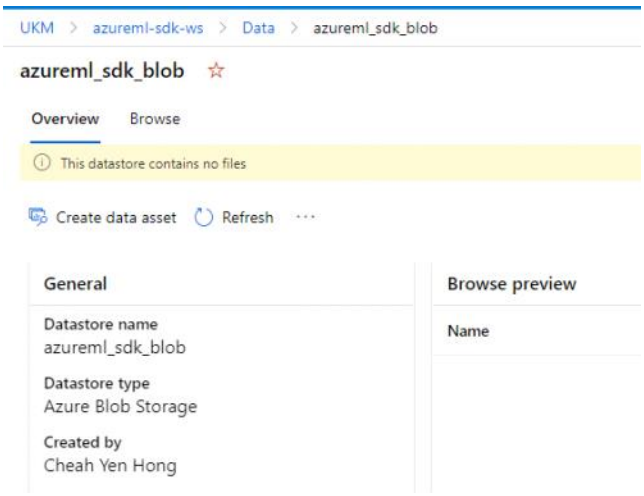



```
from azureml.core import Workspace

# Access the workspace and create the workspace object
ws = Workspace.from_config(path='/content/config.json')
```

   b. Create a storage account in the Azure portal
   c. Create a datastore in Notebook

```
az_dstore = Datastore.register_azure_blob_container(workspace=ws,
                                                    datastore_name='azureml_sdk_blob', # Specify the datastore name
                                                    account_name='azuremlsdkst02',
                                                    container_name='azureml-blob-container',
                                                    account_key='7xMGR1RhljYgYDHsku1ArRD2pmUARir6deAusdiNkSHhAkVKOZXLHON
YR3EYDUepmywIbw2h06EH+AStkSY3ew==')
```

(Check WS, datastore has created)



3. **Creating a Dataset using AzureML SDK**

```
from azureml.core import Datastore, Dataset

# Access the datastore
az_dstore = Datastore.get(workspace=ws,
                          datastore_name='azureml_sdk_blob')

# Creating the dataset
churn_dataset = Dataset.Tabular.from_delimited_files(path=[(az_dstore,'Churn_Modelling.csv')])
```

```
1 # preview first 3 lines
2 churn_dataset.take(3).to_pandas_dataframe()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
0	1	15634602	Hargrave	619	France	Female	42	2	0.00
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86
2	3	15619304	Onio	502	France	Female	42	8	159660.80

```
# Register the dataset
churn_dataset = churn_dataset.register(workspace=ws,
                                       name='Churn Modelling Data SDK')
```

4. **Accessing the Workspace, Datastore and Dataset with AzureML SDK**
   A. **List all the workspaces within a subscription**

```
ws_list = Workspace.list(subscription_id='5d2abb68-3a0a-4bfe-8994-6915771b5322',
                         resource_group='azureml-sdk-rg')
```

   B. **Access the Datastore from workspace**

Accessing the default datastore

```
az_default_dstore = ws.get_default_datastore()
```

```
1 print(az_default_dstore)
```

```
{
  "name": "workspaceblobstore",
  "container_name": "azureml-blobstore-d1ada6ec-7aad-40de-ade6-0b7e09161d5b",
  "account_name": "azuremlsstorage4203b170d",
  "protocol": "https",
  "endpoint": "core.windows.net"
}
```

Accessing all the datastores within a workspace

```
[ ]  1 print(ws.datastores)
```

```
{'azureml_sdk_blob': {
  "name": "azureml_sdk_blob",
  "container_name": "azureml-blob-container",
  "account_name": "azuremlsdkst",
  "protocol": "https",
  "endpoint": "core.windows.net"
}, 'workspaceartifactstore': {
```

```
datastore_list = list(ws.datastores)

print(datastore_list)
```

C. Accessing Datasets

Getting the dataset by it's name

```
az_dataset = Dataset.get_by_name(workspace=ws, name='Churn Modelling Data SDK')
```

```
1 print(az_dataset)
```

```
TabularDataset
{
  "source": [
    "('azureml_sdk_blob', 'Churn_Modelling.csv')"
  ],
  "definition": [
    "GetDatastoreFiles",
    "ParseDelimited",
    "DropColumns",
    "SetColumnTypes"
  ],
  "registration": {
    "id": "53b7eb1b-ef2d-4c90-8091-eba4beb1fbab",
    "name": "Churn Modelling Data SDK",
    "version": 1,
    "workspace": "Workspace.create(name='azureml-sdk-ws', subscription_id='5d2ab
  }
}
```

D. Getting all the datasets within a workspace

```
az_all_datasets = Dataset.get_all(workspace=ws)

print(az_all_datasets)
```
{ 'Churn Modelling Data SDK': DatasetRegistration(id='91595452-031b-4272-b05a-f4a555b9c774', name='Churn Modelling Data SDK', version=1, description='', tags={})}

```
az_all_datasets = list(az_all_datasets)

print(az_all_datasets)
```
['Churn Modelling Data SDK']

E. Getting the list of datasets

```
ws.datasets.keys()
```
KeysView({'Churn Modelling Data SDK': DatasetRegistration(id='91595452-031b-4272-b05a-f4a555b9c774', name='Churn Modelling Data SDK', version=1, description='', tags={})})

```
dataset_list = list(ws.datasets.keys())

print(dataset_list)
```
['Churn Modelling Data SDK']

```
for items in dataset_list:
  print(items)
```

5. AzureML Dataset and Pandas Dataset Conversion

```
# Access the datastore
az_dstore = Datastore.get(workspace=ws, datastore_name = 'azureml_sdk_blob')

# Access the dataset
az_dataset = Dataset.get_by_name(workspace=ws, name='Churn Modelling Data SDK')
```

A. Load Azure ML Dataset as a Dataframe

```
df = az_dataset.to_pandas_dataframe()
```

```
1 df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfPr
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	

```
1 df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfPr
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

```
1 df.shape
```

```
(10000, 14)
```

```
1 df.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

B. Upload and Register the dataframe as an Azure ML Dataset

```
# Upload and Register df_sub
az_dataset_from_df = Dataset.Tabular.register_pandas_dataframe(dataframe=df_sub, target=az_dstore,
                                                               name='Churn Dataset from DF')
```

6. Uploading Local Datasets to Storage Account

```
files_list = ['/content/Dataset 1.csv', '/content/Dataset 2.csv']

# Uploading local files to storage account using the datastore
az_dstore.upload_files(files=files_list, target_path='Churn Data/', overwrite=True)
```

```
# Uploading local folder/directory to storage account using the datastore
az_dstore.upload(src_dir='/content/sample_data/Test Datasets', target_path='Data', overwrite=True)
```

Complete/schedule indicator Loading (allow for ML in prod) Section 9: AzureML SDK: Running Experiments and Training Models

Thursday, 4 May, 2023 4:14 PM

The topics:
Lecture 1 - Running Sample Experiment in AzureML Environment
Lecture 2 - Logging values to Experiment in AzureML Environment
Lecture 3 - Running Script in AzureML Environment
Lecture 4 - Uploading the output file to existing run
Lecture 6 - Running Script for Logistic Regression in AzureML
Lecture 7 - Provisioning Compute Cluster in AzureML SDK
Lecture 8 - Automate Model Training in AzureML SDK

The libraries:

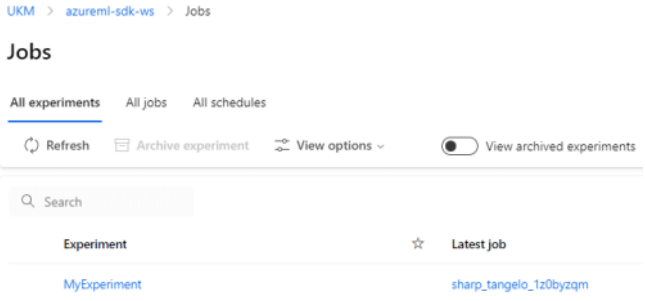
1. Create and access the experiment object
   from azureml.core import Experiment
2. Running an experiment with python script
   from azureml.core import ScriptRunConfig
   (**ScriptRunConfig** is used to define and execute Python scripts outside of a pipeline context, providing more flexibility and control over the execution environment)
3. Creating the custom environment (Importing the classes)
   from azureml.core import Environment
   from azureml.core.environment import CondaDependencies
4. Provision Compute Cluster
   from azureml.core.compute import AmlCompute, ComputeTarget
5. Run configuration steps
   from azureml.core.runconfig import RunConfiguration
6. Defining the pipeline steps
   from azureml.pipeline.steps import PythonScriptStep
   (**PythonScriptStep** is used within an Azure ML pipeline to execute a Python script as a step in the workflow)
   from azureml.pipeline.core import PipelineData
7. Configure and Build the Pipeline
   from azureml.pipeline.core import Pipeline
8. Creating experiment
   from azureml.core import Experiment

- Lecture 1 - **Running Sample Experiment** in AzureML Environment

```
# Import the class
from azureml.core import Experiment

# Create and access the experiment object
experiment = Experiment(workspace=ws, name='MyExperiment')

# Run the experiment
new_run = experiment.start_logging()

# Complete the experiment run
new_run.complete()
```

- Lecture 2 - **Logging values to Experiment** in AzureML Environment
  - Create and run the experiment

```
# Import the class
from azureml.core import Experiment

# Create and access the experiment object
experiment = Experiment(workspace=ws, name='MyExperiment')
```




```
# Run the experiment
new_run = experiment.start_logging()
```

- Log the values in the experiment

```
df = churn_dataset.to_pandas_dataframe()

# getting the shape
shape = df.shape

# Getting the total number of observations
total_observations = len(df)

# logging the values to workspace
new_run.log('Shape', value=shape)
new_run.log('Total Observations', value=total_observations)

# Complete the experiment run
new_run.complete()
```

- Lecture 3 - **Running Script** in AzureML Environment

Running an experiment

a. Creating an experiment

```
# Importing the class
from azureml.core import Experiment

# Create and access the experiment object
experiment = Experiment(workspace=ws, name='Churn_Script')
```

b. Creating an Environment

```
from azureml.core import Environment

myenv = Environment(name='user-managed-environment')
myenv.python.user_managed_dependencies=True
```

c. Running an experiment with python script

```
from azureml.core import ScriptRunConfig

script_config = ScriptRunConfig(source_directory='.',
                script='/content/python_script_1.py',
                environment=myenv)

new_run = experiment.submit(config=script_config)
```

/content/python_script_1.py

1) Installation and Setup

```
[ ] 1 # Importing the classes
    2 from azureml.core import Workspace, Datastore, Dataset, Experiment, Run
```

```
[ ] 1 # Accessing the workspace
    2 ws = Workspace.from_config(path='/content/config.json')

Performing interactive authentication. Please follow the instructions on the terminal.
WARNING:azureml._vendor.azure_cli_core.auth.identity:To sign in, use a web browser to open
Interactive authentication successfully completed.
```

```
[ ] 1 # Accessing the datastore
    2 az_dstore = Datastore.get(workspace=ws, datastore_name='azureml_sdk_blob')
```

```
[ ] 1 # Accessing the default datastore
    2 az_default_dstore = ws.get_default_datastore()
```

```
[ ] 1 # Accessing the dataset
    2 churn_dataset = Dataset.get_by_name(workspace=ws, name='Churn Modelling Data SDK')
```

```
[ ] 1 # Getting the contex of the experiment run
    2 new_run = Run.get_context()
```

Need to refer to the experiment Churn_Script, we don't want to start a new experiment, that's why we need to specify the Run.get_context()
We don't need to create a new experiment in the Python script.

2) Log the values in the experiment

```
[ ]  1 df = churn_dataset.to_pandas_dataframe()
```

```
[ ]  1 # getting the shape
     2 shape = df.shape

(10000, 14)
```

```
[ ]  1 # Getting the total number of observations
     2 total_observations = len(df)

10000
```

```
[ ]  1 # logging the values to workspace
     2 new_run.log('Shape', value=shape)
     3 new_run.log('Total Observations', value=total_observations)
```

3) Complete the experiment run

```
[ ]  1 # Complete the experiment run
     2 new_run.complete()
```

- Lecture 4 - Uploading the output file to existing run in AzureML Environment
  - Running an experiment with python script

```
script_config = ScriptRunConfig(source_directory='.',
                script='/content/python_script_2.py',
                environment=myenv)
```

```
new_run = experiment.submit(config=script_config)
```

```
new_run.wait_for_completion()
```

```
45s  1 new_run = experiment.submit(config=script_config)
     2 new_run.wait_for_completion()

{'runId': 'Churn_Script_1683336573_61e3f9c2',
 'target': 'local',
 'status': 'Finalizing',
 'startTimeUtc': '2023-05-06T01:29:38.028844Z',
 'services': {},
 'properties': {'_azureml.ComputeTargetType': 'local',
  'ContentSnapshotId': 'a473c004-fc1c-4410-aafa-f81d7067d7bc'},
 'inputDatasets': [{'dataset': {'id': '53b7eb1b-ef2d-4c90-8091-eba4beb1fbab'}, 'consumptionDetails': {'type':
'Reference'}}],
 'outputDatasets': [],
 'runDefinition': {'script': 'python_script_2.py',
  'command': '',
  'useAbsolutePath': False,
  'arguments': [],
  'sourceDirectoryDataStore': None,
  'framework': 'Python',
  'communicator': 'None',
  'target': 'local'
```

/content/python_script_2.py

2) Log the values in the experiment

```
[ ]  1 df = churn_dataset.to_pandas_dataframe()
```

```
     1 # getting the shape
     2 shape = df.shape
```

```
(10000, 14)
```

```
[ ]  1 # Getting the total number of observations
     2 total_observations = len(df)
```

```
10000
```

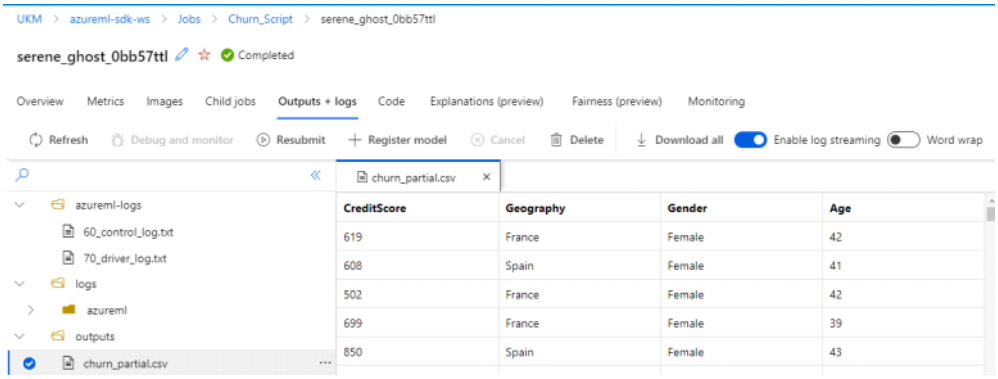
```
[ ]  1 # Selecting four columns from df
     2 df_2 = df[['CreditScore', 'Geography', 'Gender', 'Age']]
```

```
[ ]  1 # save the new file
     2 df_2.to_csv(path_or_buf='./outputs/churn_partial.csv', index=False)
```

```
[ ]  1 # logging the values to workspace
     2 new_run.log('Shape', value=shape)
     3 new_run.log('Total Observations', value=total_observations)
```

3) Complete the experiment run

```
[ ]  1 # Complete the experiment run
     2 new_run.complete()
```



➢ Lecture 6 - Running Script for Logistic Regression in AzureML Environment

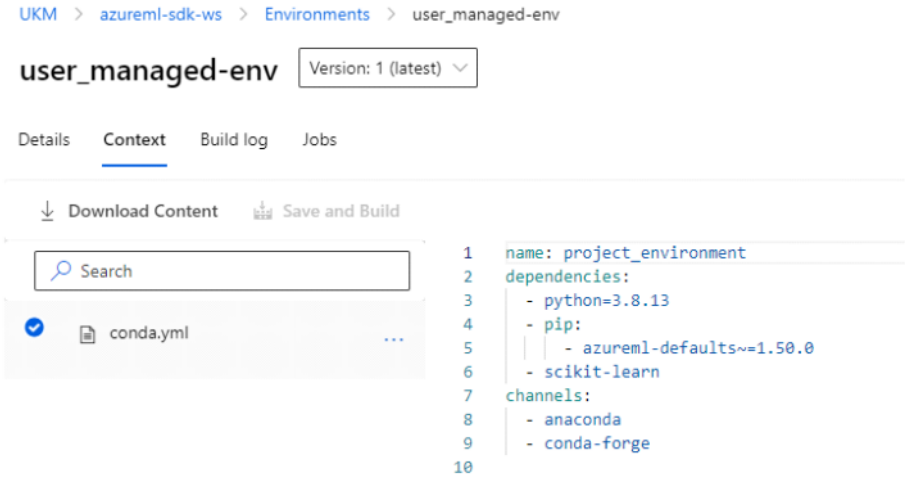
1) Installation and Setup

```
# Install azureml SDK package
! pip install -q azureml-sdk

# Importing the classes
from azureml.core import Workspace, Experiment, ScriptRunConfig

# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

2) Running the experiment

```
# Create and access the experiment object
experiment = Experiment(workspace=ws, name='Logistic_Regression')

# Creating the custom environment (Importing the classes)
from azureml.core import Environment
from azureml.core.environment import CondaDependencies

# Create the environment
myenv = Environment('user-managed-env')
myenv.python.user_managed_dependencies=True

# creting the dependencies object
myenv_dep = CondaDependencies.create(conda_packages=['scikit-learn'])
myenv.python.conda_dependencies = myenv_dep
```

```
# Register the environment
myenv.register(ws)
```




```
# Creating the script run configuration
script_config = ScriptRunConfig(source_directory='.',
                                script='/content/python_script_logistic_regression.py',
                                environment=myenv)

new_run = experiment.submit(config=script_config)
new_run.wait_for_completion()
```

Python Script - Logistic Regression.ipynb

Part 1: Installation and Setup

```
# Importing the classes
from azureml.core import Workspace, Experiment, Run
```

```
# Accessing the workspace
ws = Workspace.from_config(path='/content/config.json')
```

```
# Getting the context of the experiment run
new_run = Run.get_context()
```

***Add Part 1 to the python script to use for the experiment*

Part 2: Data Processing

+ Code + Text

Importing the libraries and dataset

```
import numpy as np
import pandas as pd
```

```
# Loading the dataset
dataset = pd.read_csv('/content/Churn_Modelling.csv')
```

Dealing with the missing data

```
dataset.isnull().values.any()
```

```
False
```

Encoding the categorical variables

Encoding the categorical variables

```
[ ]  1 dataset = dataset.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
```

```
[ ]  1 dataset.select_dtypes(include='object').columns
```

```
Index(['Geography', 'Gender'], dtype='object')
```

```
[ ]  1 dataset = pd.get_dummies(dataset, drop_first=True)
```

Splitting the dataset into train and test set

```
[ ]  1 X = dataset.drop(columns='Exited')
     2 Y = dataset['Exited']
```

```
[ ]  1 from sklearn.model_selection import train_test_split
     2 x_test, x_train, y_test, y_train = train_test_split(X, Y, test_size=0.2, random_state=0)
```

Part 3:Training and evaluating the ML Model

```
[ ]  1 from sklearn.linear_model import LogisticRegression
     2 classifier = LogisticRegression(random_state=0)
     3 classifier.fit(x_train, y_train)
```

```
LogisticRegression(random_state=0)
```

```
[ ]  1 # predicting the results on test set
     2 y_pred = classifier.predict(x_test)
```

```
[ ]  1 # predicting the probabilities
     2 y_prob = classifier.predict_proba(x_test)[:, 1]
```

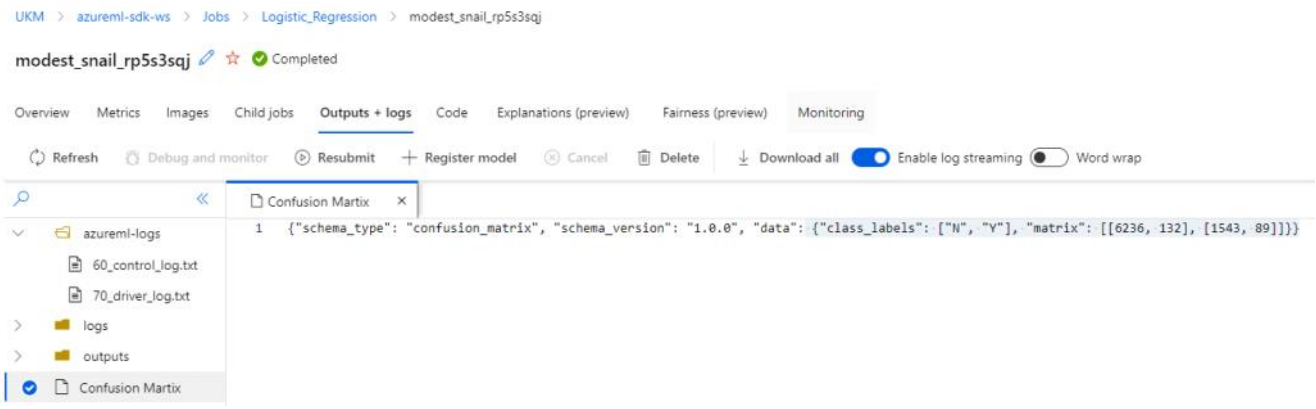
```
[ ]  1 # getting the confusion matrix and the accuracy score
     2 from sklearn.metrics import confusion_matrix, accuracy_score
     3 cm = confusion_matrix(y_test, y_pred)
     4 acc = accuracy_score(y_test, y_pred)
```

```
     1 cm_dict = {
     2       "schema_type": "confusion_matrix",
     3       "schema_version": "1.0.0",
     4       "data": {
     5           "class_labels": ["N", "Y"],
     6           "matrix": cm.tolist()
     7       }
     8   }
```

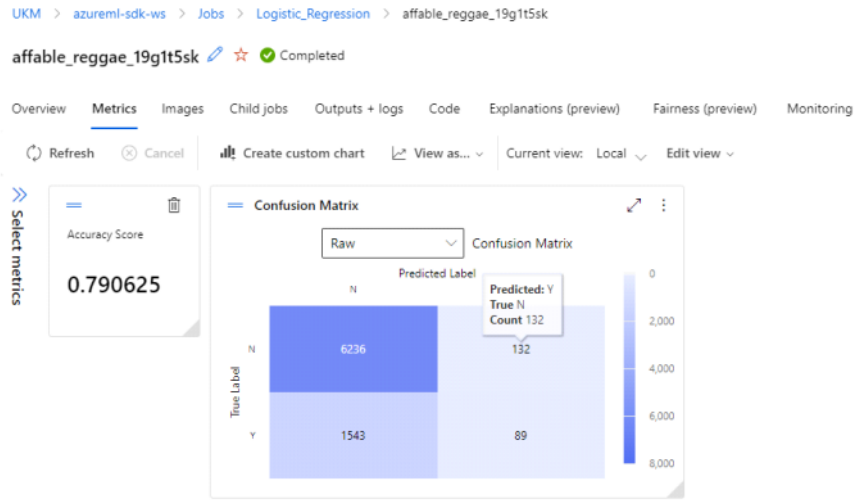
Part 4: Logging the values in experiment run

```
[ ]  1 new_run.log_confusion_matrix('Confusion Martix', cm_dict)
     2 new_run.log('Accuracy Score', acc)
```

***Add Part 4 to the python script to use for the experiment, log the metrics (cm_dict)*




```
new_run.log_confusion_matrix('Confusion Martix', cm_dict)
```



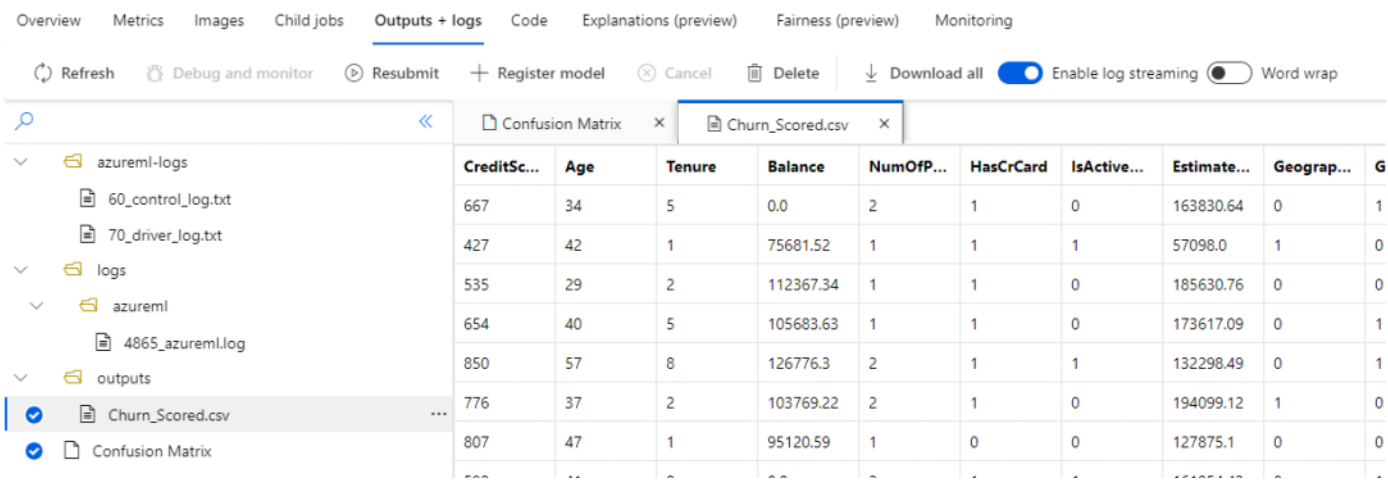
Part 5: Creating the final dataset with predicted values and probabilities

```
[ ]  1 x_test = x_test.reset_index(drop=True)
     2 y_test = y_test.reset_index(drop=True)
     3
     4 y_prob_df = pd.DataFrame(y_prob, columns=['Scored probabilities'])
     5 y_pred_df = pd.DataFrame(y_pred, columns=['Scored Lebels'])
     6
     7 # create the final dataset
     8 scored_dataset = pd.concat([x_test, y_test, y_pred_df, y_prob_df], axis=1)
```

```
[ ]  1 # Upload the dataset
     2 scored_dataset.to_csv('./outputs/Churn_Scored.csv', index=False)
```

```
[ ]  1 # Complete the experiment run
     2 new_run.complete()
```

***Add Part 5 (# Upload the dataset, # Complete the experiment run) to the python script to use for the experiment*

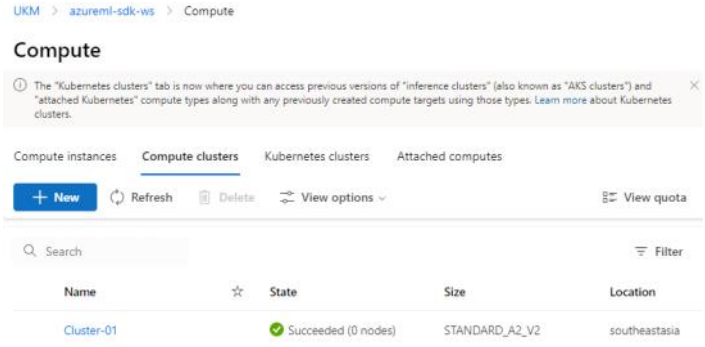


1) Installation and Setup

```
# Install azureml SDK package
! pip install -q azureml-sdk

# Importing the class
from azureml.core import Workspace

# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

2) Provisioning a compute cluster

```
# Specify the name of the compute cluster
cluster_name = 'Cluster-01'

# Import the class
from azureml.core.compute import AmlCompute

# Compute cluster configuration
compute_config = AmlCompute.provisioning_configuration(vm_size='Standard_A2_v2',
                                                       max_nodes=2)

# Create the cluster
cluster = AmlCompute.create(workspace=ws, name=cluster_name, provisioning_configuration=compute_config)
```



➢ Lecture 8 - Automate Model Training in AzureML SDK **(Pipeline Run)**

1) Installation and Setup

```
# Install azureml SDK package
! pip install -q azureml-sdk

# Import the class
from azureml.core import Workspace
```

```
# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

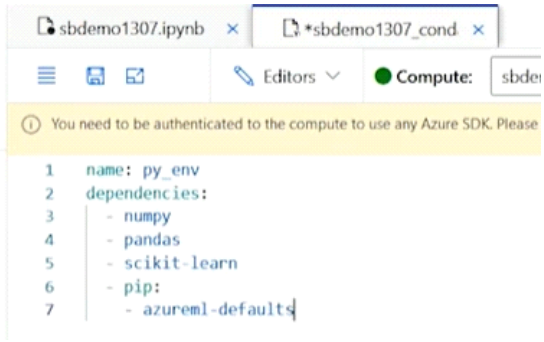
2) Running the Experiment

Creating the custom environment

```
from azureml.core import Environment
from azureml.core.environment import CondaDependencies

myenv = Environment(name="MyEnvironment")
myenv.python.user_managed_dependencies = False
myenv.docker.enabled = True

# Creating the dependencies object
myenv_dep = CondaDependencies.create(conda_packages=['scikit-learn','pandas'],
                                     pip_packages=['azureml-sdk'])
myenv.python.conda_dependencies = myenv_dep
```

OR

```
from azureml.core import Environment

env = Environment.from_conda_specification(
    name='sbdemo1307_environment_static',
    file_path='./sbdemo1307_conda.yml')
print(env)
```




```
# Register the environment to workspace
myenv.register(ws)
```

Provision the compute cluster

```
# Specify the name of the cluster
cluster_name = 'Compute-Cluster'

# Import the classes
from azureml.core.compute import AmlCompute, ComputeTarget

# Configuration for the cluster
compute_config = AmlCompute.provisioning_configuration(vm_size='Standard_DS3_v2',
                                                       max_nodes=2)

# Create the compute cluster
compute_cluster = ComputeTarget.create(workspace=ws, name=cluster_name, provisioning_configuration=compute_config)
compute_cluster.wait_for_completion(show_output=True)
```

Run configuration steps

```
from azureml.core.runconfig import RunConfiguration
run_config = RunConfiguration()

run_config.target = compute_cluster
run_config.environment = myenv
```

Defining the pipeline steps

```
# Importing the classes
from azureml.pipeline.steps import PythonScriptStep
from azureml.pipeline.core import PipelineData
```

(PipelineData - *Data used in pipeline can be produced by one state and consumed in another state by providing pipeline data object as an output of one step and an input of one or more subsequent steps.)*

```
# Access the dataset
input_ds = ws.datasets.get('Churn Modelling Data SDK')
```

(1st step is creating output, and the second step is using that output as an input, this pipeline object is a reference point.)

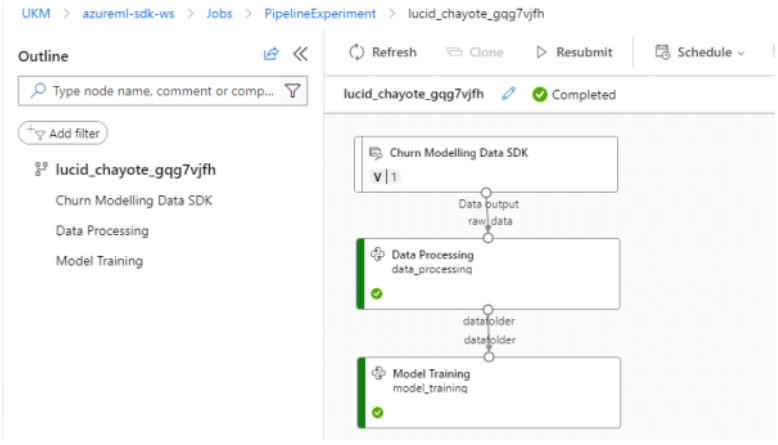
```
# Pipeline data object
dataFolder = PipelineData(name='datafolder', datastore=ws.get_default_datastore())

# Step 1- Data Processing
dataPrep_step = PythonScriptStep(name='Data Processing',
                                 source_directory='.',
                                 script_name='data_processing_script.py',(Data Processing Script.ipynb)
                                 inputs=[input_ds.as_named_input('raw_data')],
                                 outputs=[dataFolder],
                                 runconfig=run_config,
                                 arguments=['--datafolder', dataFolder]) # Location

# Step 2 - Training the Model
train_step = PythonScriptStep(name='Model Training',
                                 source_directory='.',
                                 script_name='model_training_script.py',(Model Training Script.ipynb)
                                 inputs=[dataFolder],
                                 runconfig=run_config,
                                 arguments=['--datafolder', dataFolder])
```

Configure and Build the Pipeline

```
steps = [dataPrep_step, train_step] # List of step names

from azureml.pipeline.core import Pipeline
new_pipeline = Pipeline(workspace=ws, steps=steps)
```

Creating experiment and running the pipeline

```
# Creating the experiment
from azureml.core import Experiment
new_experiment = Experiment(workspace=ws, name='PipelineExperiment')

# Submit the experiment run
new_pipeline_run = new_experiment.submit(new_pipeline)
new_pipeline_run.wait_for_completion(show_output=True)
```



- ***Data Processing Script.ipynb***

```
# Import the class
from azureml.core import Run

# Get the run context
new_run = Run.get_context()

# Access the workspace
ws = new_run.experiment.workspace
```

```
[ ]  1 # Read the input dataset
     2 import pandas as pd
     3 df = new_run.input_datasets['raw_data'].to_pandas_dataframe()
```

```
[ ]  1 # Select relevant columns from the dataset
     2 dataPrep = df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], axis=1)
```

```
[ ]  1 # Create Dummy variables / encoding the categorical variables
     2 dataPrep = pd.get_dummies(dataPrep, drop_first=True)
```

```
[ ]  1 # Normalize the data
     2 from sklearn.preprocessing import MinMaxScaler
     3 scaler = MinMaxScaler()
     4 columns = dataPrep.select_dtypes(include='number').columns
     5 dataPrep[columns] = scaler.fit_transform(dataPrep[columns])
```

```
# Get the arguments from the pipeline job
from argparse import ArgumentParser as AP
parser = AP()
parser.add_argument('--datafolder', type=str) # Adding the argument
args = parser.parse_args()  # Passing the arguments in args

# Create the folder
import os
os.makedirs(args.datafolder, exist_ok=True)

# Create the path
path = os.path.join(args.datafolder, 'churn_prep.csv')

# Save the dataPrep file
dataPrep.to_csv(path, index=False)

# Complete the run
new_run.complete()
```

- ***Model Training Script.ipynb***

```
# Import the class
from azureml.core import Run

# Get the run context
new_run = Run.get_context()

# Access the workspace
ws = new_run.experiment.workspace

# Get the arguments from the pipeline job
from argparse import ArgumentParser as AP
parser = AP()
parser.add_argument('--datafolder', type=str) # Adding the argument
args = parser.parse_args()  # Passing the arguments in args

# read the data from previous step
import os
import pandas as pd

# Create the path
path = os.path.join(args.datafolder, 'churn_prep.csv')
dataPrep = pd.read_csv(path)
```

```
[ ]  1 # Define X and Y
     2 X = dataPrep.drop(['Exited'], axis=1)
     3 Y = dataPrep[['Exited']]
```

```
[ ]  1 # Split the dataset into train and test set
     2 from sklearn.model_selection import train_test_split
     3 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0, stratify=Y)
```

```
[ ]  1 # Build and train the Logistic Regression model
     2 from sklearn.linear_model import LogisticRegression
     3 classifier = LogisticRegression()
     4 classifier.fit(X_train, Y_train)
```

```
[ ]  1 # Predict the output - Scored Label
     2 Y_predict = classifier.predict(X_test)
```

```
[ ]  1 # Scored Probabilities
     2 Y_prob = classifier.predict_proba(X_test)[:, 1]
```

```
[ ]  1 # Confusion matrix and accuracy score
     2 from sklearn.metrics import confusion_matrix
     3 cm    = confusion_matrix(Y_test, Y_predict)
     4 score = classifier.score(X_test, Y_test)
```

```
# Log metrics
# Create the confusion matrix dictionary
cm_dict = {"schema_type": "confusion_matrix",
           "schema_version": "v1",
           "data": {"class_labels": ["N", "Y"],
                    "matrix": cm.tolist()}
           }

new_run.log('TotalObservations', len(dataPrep))
new_run.log_confusion_matrix('ConfusionMatrix', cm_dict)
new_run.log('Score', score)
```

```
     1 # Create the Scored Dataset and upload to outputs folder
     2
     3 X_test = X_test.reset_index(drop=True)
     4 Y_test = Y_test.reset_index(drop=True)
     5
     6 Y_prob_df    = pd.DataFrame(Y_prob, columns=['Scored Probabilities'])
     7 Y_predict_df = pd.DataFrame(Y_predict, columns=['Scored Label'])
     8
     9 scored_dataset = pd.concat([X_test, Y_test, Y_predict_df, Y_prob_df], axis=1)
```

```
# Upload the scored dataset
scored_dataset.to_csv('./outputs/churn_scored.csv', index=False)

# Complete the run
new_run.complete()
```

Notes

Wednesday, 17 May, 2023 4:13 PM

Lecture 8

1. Installation and Setup
    i. Pip install azureml-sdk
    ii. Access the workspace from config file
2. Running the Experiment
    - Creating Custom Environment
        i. Creating Environment
        ii. Creating dependencies object (conda_packages, pip_packages)
        iii. Register the Environment to workspace
    - Provision the compute cluster
        i. Specify the name of the cluster
        ii. Import class AmlCompute
        iii. Provision the configuration using AmlCompute
        iv. Create the compute cluster
    - Run the configuration step

```
from azureml.core.runconfig import RunConfiguration
run_config = RunConfiguration()

run_config.target = compute_cluster
run_config.environment = myenv
```

    - Defining the pipeline steps
        i. Import classes (PythonScriptStep, PipelineData)
        ii. Access the dataset
        iii. Pipeline data object
        iv. Define pipeline (e.g. Step 1-Data Processing, Step 2-Training the Model)

```
# Step 1- Data Processing
dataPrep_step = PythonScriptStep(name='Data Processing',
                                 source_directory='.',
                                 script_name='data_processing_script.py',
                                 inputs=[input_ds.as_named_input('raw_data')],
                                 outputs=[dataFolder],
                                 runconfig=run_config,
                                 arguments=['--datafolder', dataFolder])
```

        - Step 1-Data Processing Python Script
            1. Import class (from azureml.core import Run)
            2. Get the run context
            3. Access the workspace
            4. Read the input dataset (df = new_run.input_datasets['raw_data'].to_pandas_dataframe())
            5. The normal data processing steps
            6. Get the args from the pipeline job (pipeline data object --datafolder)
            7. Create the datafolder
            8. Create the path (os.path.join(args.datafolder, 'churn_prep.csv')
            9. Save the processed data as csv to path
            10. Complete the run

```
# Step 2 - Training the Model
train_step = PythonScriptStep(name='Model Training',
                              source_directory='.',
                              script_name='model_training_script.py',
                              inputs=[dataFolder],
                              runconfig=run_config,
                              arguments=['--datafolder', dataFolder])
```

        - Step 2-Training the Model
            1. Import class (from azureml.core import Run)
            2. Get the run context
            3. Access the workspace

4. Get the args from the pipeline job (pipeline data object --datafolder)
5. Read the data from previous step
6. Data processing, define X and y, train test split, build and train the model, predict the output(label), scored probabilities, get the confusion matrix and acc score
7. Log metrics
    i) Create the CM dictionary
    ii) Log
8. Create the scored dataset and upload it to the /outputs folder
9. Complete the run

- Configure and Build the Pipeline

```
steps = [dataPrep_step, train_step] # List of step names

from azureml.pipeline.core import Pipeline
new_pipeline = Pipeline(workspace=ws, steps=steps)
```

- Create the experiment and run the pipeline
    i. Creating the experiment
    ii. Submit the experiment run

```
# Submit the experiment run
new_pipeline_run = new_experiment.submit(new_pipeline)
new_pipeline_run.wait_for_completion(show_output=True)
```

a. Compute Cluster:
Before creating an experiment, you typically set up a compute cluster in Azure Machine Learning. A compute cluster is a group of virtual machines (VMs) or other compute resources that you can use to run your machine learning workloads. It provides the necessary computational resources to execute your code and train models. By setting up a compute cluster, you ensure that you have the required resources available to run your experiments efficiently.

b. Environment:
Once the compute cluster is in place, you define an environment. An environment in Azure Machine Learning specifies the execution environment for running your machine learning code. It includes the necessary dependencies, packages, and settings required for your code to run successfully. You can define an environment using conda or Docker, specifying the Python packages, libraries, and system dependencies needed for your code execution.

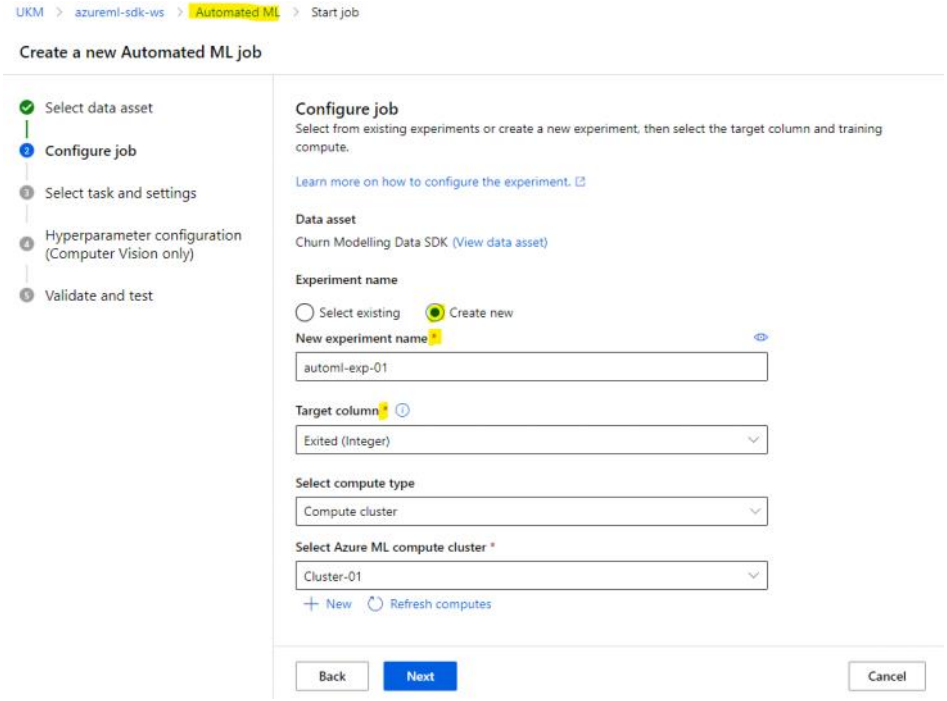
c. Pipeline Steps:
Next, you configure the steps of your machine learning pipeline. Each step represents a specific task or operation, such as data preprocessing, model training, evaluation, or deployment. Each step can reference the compute target and environment you set up earlier. You define the sequence of steps and their dependencies within the pipeline.

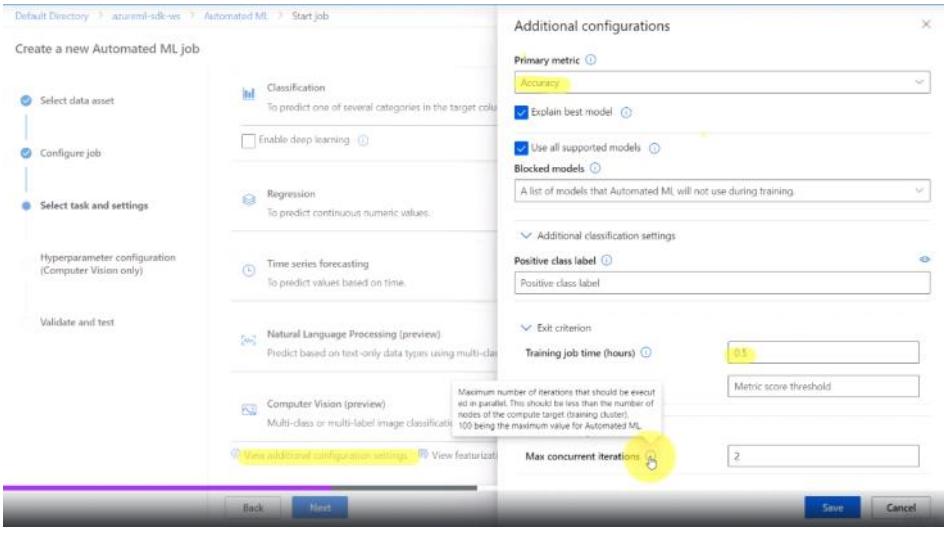
d. Experiment:
Finally, you create an experiment and associate it with the pipeline you defined. The experiment serves as a container or logical unit to organize and track the runs of your machine learning pipeline. It allows you to execute the pipeline multiple times with different configurations or parameters. The experiment captures the metadata, outputs, and results of each run, facilitating comparison, analysis, and tracking of your machine learning iterations.

Section 10: Use Automated ML to Create Optimal Models

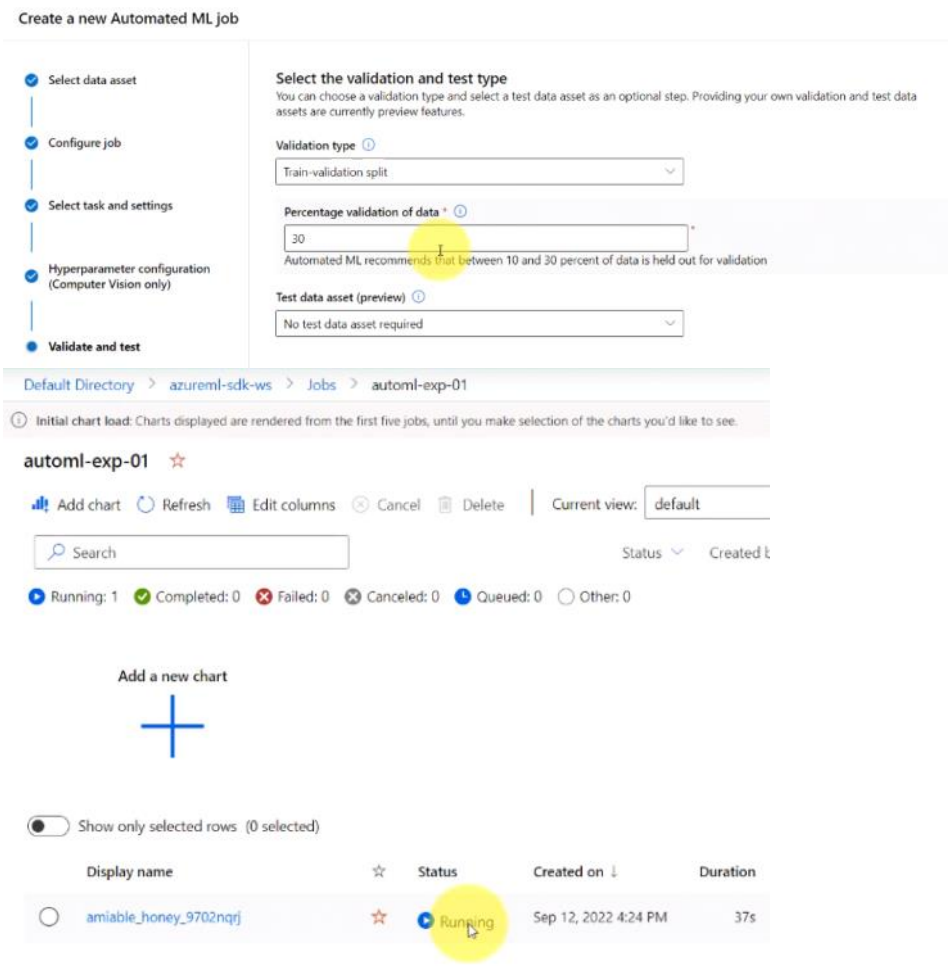
Thursday, 11 May, 2023 1:42 PM

- **Automated ML in Azure ML studio**
  - **Create a new Automated ML job**



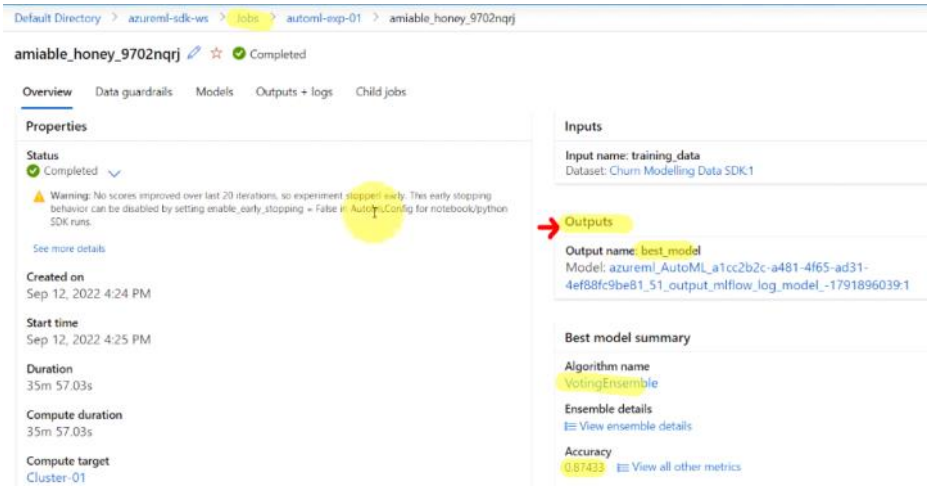


  - **The job/experiment is running**

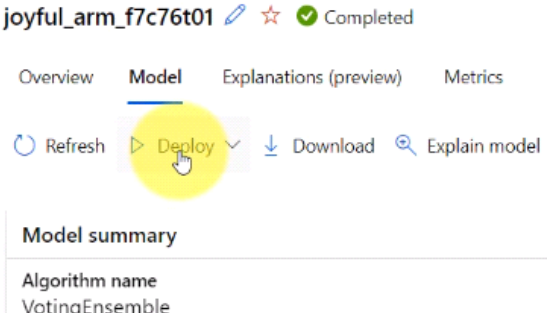


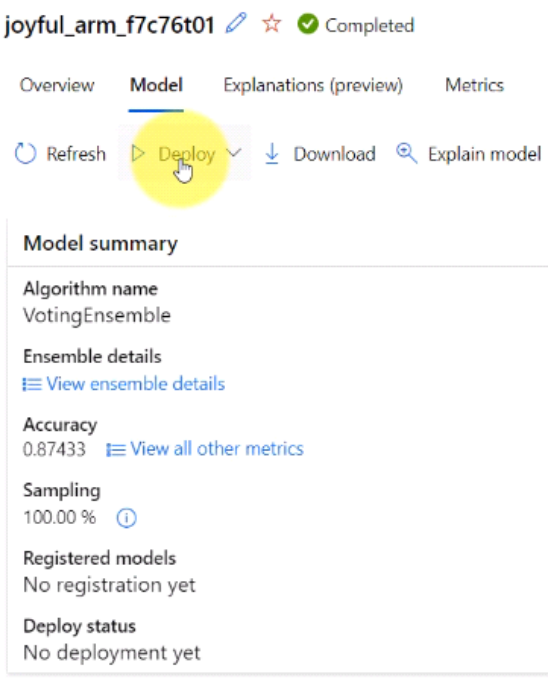
- **The best model is ready**

Overview

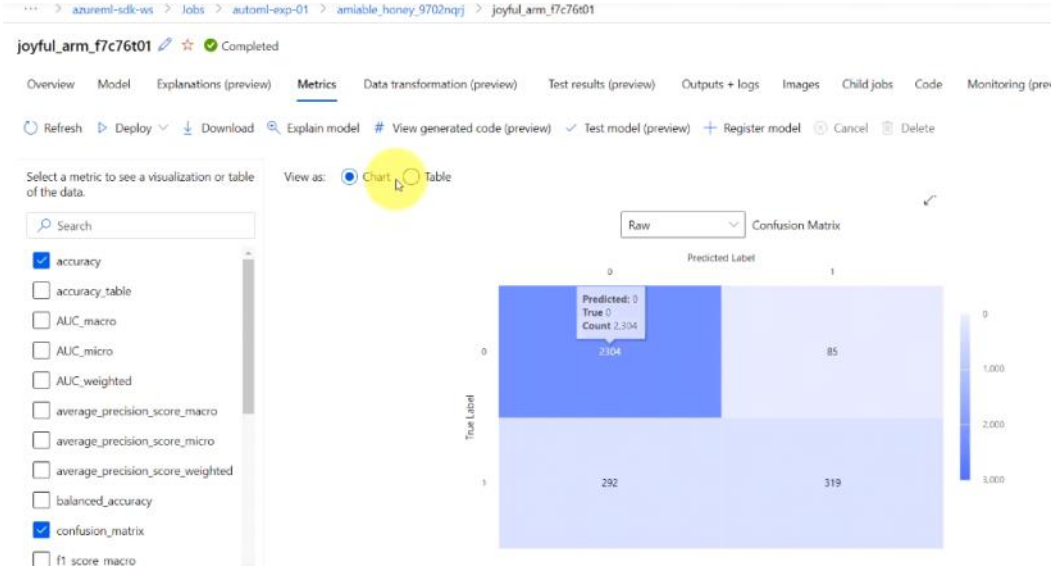


- **The Algo name: VotingEnsemble**

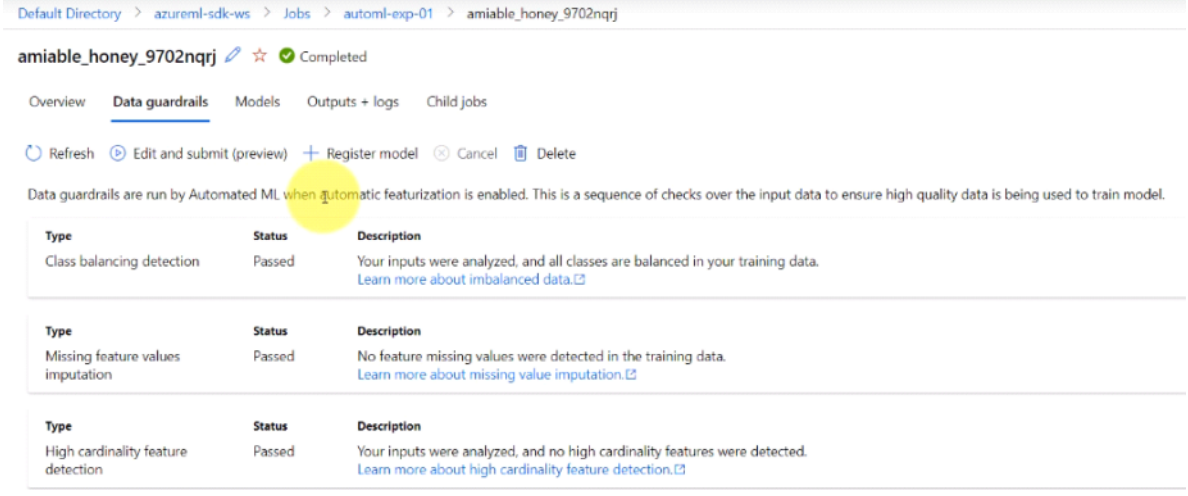




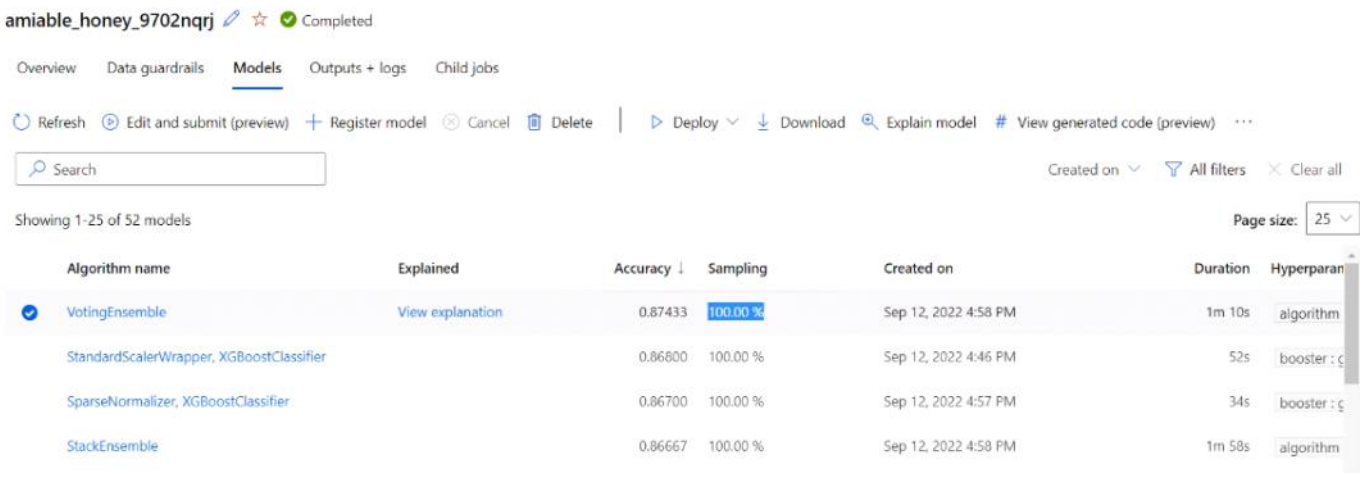
Metrics



Data guardrails



Models



- Lecture 1- Automated ML in Azure Machine Learning SDK

Step 1 - Installation and Setup

```
# Install azureml SDK package
! pip install -q azureml-sdk

# Importing the class
from azureml.core import Workspace

# Accessing the workspace and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

Step 2 - Accessing the input data

```
input_ds = ws.datasets.get('Churn Modelling Data SDK')
```

Step 3 - Creating the compute cluster

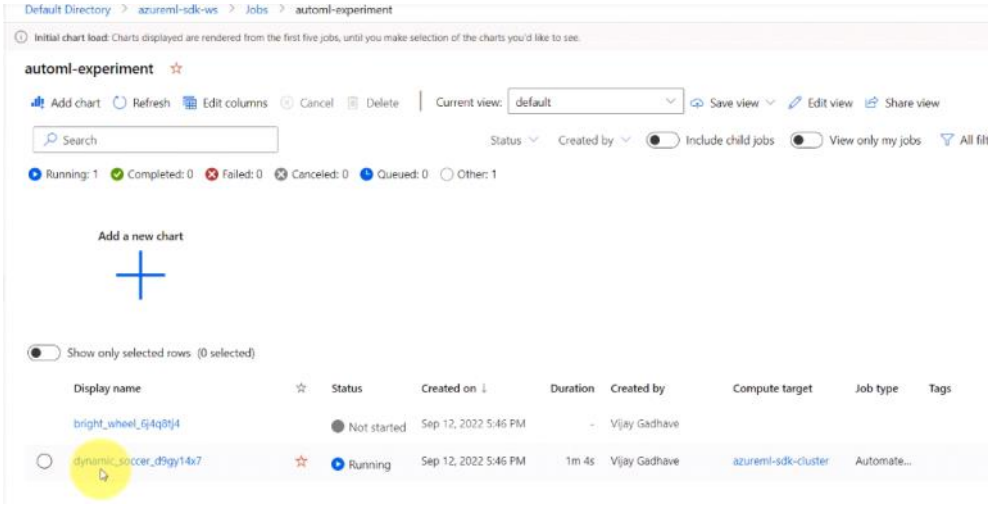
```
cluster_name = 'azureml-sdk-cluster'

# Import the class AmlCompute
from azureml.core.compute import AmlCompute

# Provision the configuration using AmlCompute
if cluster_name not in ws.compute_targets:
  compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D11_V2',
                                                         max_nodes=2)
  cluster = AmlCompute.create(workspace=ws, name=cluster_name, provisioning_configuration=compute_config)
  cluster.wait_for_completion()
else:
  cluster = ws.compute_targets[cluster_name]
```

Step 4 - Configuring the AutoML run

```
# Import the class AutoMLConfig
from azureml.train.automl import AutoMLConfig

# Create an object of the class AutoMLConfig
automl_config = AutoMLConfig(task='classification',
                             compute_target=cluster,
                             training_data=input_ds,
                             validation_size=0.3,
                             label_column_name='Exited',
                             primary_metric='accuracy',
                             iterations=10,
                             max_concurrent_iterations=2,
                             experiment_timeout_hours=0.5,
                             featurization='auto')
# Picture on page "Section 10 Use Automated ML to Create Optimal Models"
```

Step 5 - Creating and submitting the experiment run

```
# Import the class Experiment
from azureml.core.experiment import Experiment

# Create the experiment
new_exp = Experiment(workspace=ws, name='automl-experiment')

# Submit the experiment run
new_run = new_exp.submit(automl_config)
new_run.wait_for_completion(show_output=True)
```




```
Submitting remote run.
```

Experiment	Id	Type	Status	Details Page	Docs Page
automl-experiment	AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347	automl	NotStarted	Link to Azure Machine Learning studio	Link to Documentation

Experiment	Id	Type	Status	Details Page	Docs Page
automl-experiment	AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347	automl	NotStarted	Link to Azure Machine Learning studio	Link to Documentation


```
Current status: FeaturesGeneration. Generating features for the dataset.
Current status: DatasetFeaturizationCompleted. Completed fit featurizers and featurizing the dataset.
Current status: ModelSelection. Beginning model selection.

*****************************************************************************************
DATA GUARDRAILS:

TYPE:         Class balancing detection
STATUS:       PASSED
DESCRIPTION:  Your inputs were analyzed, and all classes are balanced in your training data.
              Learn more about imbalanced data: https://aka.ms/AutomatedMLImbalancedData

*****************************************************************************************

TYPE:         Missing feature values imputation
STATUS:       PASSED
DESCRIPTION:  No feature missing values were detected in the training data.
              Learn more about missing value imputation: https://aka.ms/AutomatedMLFeaturization

*****************************************************************************************


TYPE:         High cardinality feature detection
STATUS:       PASSED
DESCRIPTION:  Your inputs were analyzed, and no high cardinality features were detected.
              Learn more about high cardinality feature handling: https://aka.ms/AutomatedMLFeaturization

*****************************************************************************************

*****************************************************************************************
ITER: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
*****************************************************************************************

 ITER   PIPELINE                                       DURATION            METRIC      BEST
    0   MaxAbsScaler LightGBM                         0:00:14             0.8627    0.8627
    1   MaxAbsScaler XGBoostClassifier                0:00:16             0.8573    0.8627
    2   MaxAbsScaler ExtremeRandomTrees               0:00:15             0.6927    0.8627
    3   SparseNormalizer XGBoostClassifier            0:00:14             0.8633    0.8633
    4   MaxAbsScaler LightGBM                         0:00:14             0.8650    0.8650
    5   MaxAbsScaler LightGBM                         0:00:14             0.7987    0.8650
    6   StandardScalerWrapper XGBoostClassifier       0:00:14             0.8467    0.8650
    7   MaxAbsScaler LogisticRegression               0:00:17             0.8320    0.8650
    8    VotingEnsemble                               0:00:33             0.8673    0.8673
    9    StackEnsemble                                0:00:39             0.8657    0.8673
```

Step 6 - Getting the best model

```
new_run.get_best_child(metric='accuracy')
```

Step 7 - Getting the metrics for all the runs

```
# Get the metrics for all the runs
for run in new_run.get_children():
    print("")
    print("Run ID : ", run.id)
    print(run.get_metrics('accuracy'))
```

Step 6 - Getting the best model

```
[ ]  1 new_run.get_best_child(metric='accuracy')
```

Experiment	Id	Type
automl-experiment	AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347_8	azureml.scriptrun

Step 7 - Getting the metrics for all the runs

```
1 # Get the metrics for all the runs
2 for run in new_run.get_children():
3     print("")
4     print("Run ID : ", run.id)
5     print(run.get_metrics('accuracy'))
```

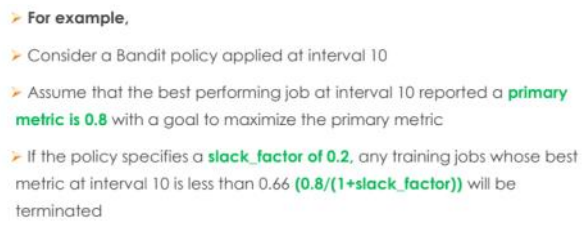
```
Run ID :  AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347_8
{'accuracy': 0.8673333333333333}

Run ID :  AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347_9
{'accuracy': 0.8656666666666667}

Run ID :  AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347_7
{'accuracy': 0.832}

Run ID :  AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347_6
{'accuracy': 0.8466666666666667}

Run ID :  AutoML_62f9970b-54fe-4eb9-afec-60b8c5dcb347_5
{'accuracy': 0.7986666666666666}
```

Section 11: Tune hyperparameters with Azure Machine Learning

Tuesday, 16 May, 2023 8:00 PM

- Specify early termination policy
  - An early termination policy automatically ends the poorly performing jobs, the early termination improves computational efficiency
  - Azure Machine Learning supports the following early termination policies,
    i. Bandit policy
    ii. Median stopping policy
    iii. Truncation selection policy
    iv. No termination policy

Bandit policy
- Based on slack factor/slack amount and evaluation interval
- Ends a job when the primary metric isn't within the specified slack factor/slack amount of the most successful job
- **slack_factor or slack_amount:** the slack allowed with respect to the best performing training job
  1) **slack_factor** specifies the allowable slack as a ratio
  2) **slack_amount** specifies the allowable slack as an absolute amount, instead of a ratio



- Lecture 1 - Configure the Hyperdrive Run

Step 1- Installation and Setup

```
# Install azureml SDK package
! pip install -q azureml-sdk

# Importing the class
from azureml.core import Workspace

# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

Step 2 - Accessing the input data

```
input_ds = ws.datasets.get('Churn Modelling Dataset')
```

Step 3 - Creating the custom environment

```
# Importing the classes
from azureml.core import Environment
from azureml.core.environment import CondaDependencies

# create the environment
myenv = Environment(name='MyEnvironment')
# create the dependencies object
myenv_dep = CondaDependencies.create(conda_packages=['scikit-learn', 'pip', 'pandas'],
                                     pip_packages=['azureml-defaults', 'azureml-sdk'])
myenv.python.conda_dependencies = myenv_dep
# Register the environment to workspace
myenv.register(workspace=ws)
```

Step 4 -Create the compute cluster

```
cluster_name = 'azureml-hyper-cluster'
```

```
# Import the class AmlCompute
from azureml.core.compute import AmlCompute

# Provision the configuration using AmlCompute
if cluster_name not in ws.compute_targets:
  compute_config = AmlCompute.provisioning_configuration(vm_size='Standard_DS3_v2',
                                                          max_nodes=2)
  cluster = AmlCompute.create(workspace=ws, name=cluster_name,
provisioning_configuration=compute_config)
  cluster.wait_for_completion()
else:
  cluster = ws.compute_targets[cluster_name]
```

Step 5 - Script configuration

```
from azureml.core import ScriptRunConfig

script_config = ScriptRunConfig(source_directory='.',
                                script='hyperdrive_training_script.py', (hyperdrive_training_script.py)
                                arguments=['--input-data', input_ds.as_named_input('raw_data')],
                                environment=myenv,
                                compute_target=cluster)
```

Step 6 - Create hyperdrive parameters

```
from azureml.train.hyperdrive import GridParameterSampling, choice

hyper_params = GridParameterSampling(
    {'--n_estimators': choice(10, 20, 30, 50),
     '--min_samples_leaf': choice(1, 2, 3)
     })
```

Step 7 - Configure the Hyperdrive class

```
from azureml.train.hyperdrive import HyperDriveConfig, PrimaryMetricGoal

hyper_config = HyperDriveConfig(run_config=script_config,
                                hyperparameter_sampling=hyper_params,
                                policy=None,
                                primary_metric_name='accuracy',
                                primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                                max_total_runs=20,
                                max_concurrent_runs=2)
```

Step 8 - Create and submit experiment run

```
from azureml.core.experiment import Experiment

# Creating the experiment
new_exp = Experiment(workspace=ws, name='hyperdrive-experiment')

# submit the experiment run
new_run = new_exp.submit(config=hyper_config)
new_run.wait_for_completion()
```

Step 9 - Best run and Best parameters

```
best_run = new_run.get_best_run_by_primary_metric()
print("Best Run ID : ", best_run.id)
print(best_run.get_metrics())
```

```
Best Run ID :  HD_116e48df-682d-4c8a-9420-d37a88069b75_5
{'accuracy': 0.8616666666666667}
```

```
# get best model hyperparameters
best_run.get_tags()
```

```
best_run.get_tags()
```

```
{'_aml_system_hyperparameters': '{"--n_estimators": 20, "--min_samples_leaf": 3}',
 'hyperparameters': '{"--n_estimators": 20, "--min_samples_leaf": 3}',
 '_aml_system_ComputeTargetStatus':
'{"AllocationState":"steady","PreparingNodeCount":0,"RunningNodeCount":1,"CurrentNodeCount":1}'}
```

hyperdrive_training_script.py

```
# Import the class
from azureml.core import Run

# Get the run context
new_run = Run.get_context()

# Access the workspace
ws = new_run.experiment.workspace

# Get the parameters
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--n_estimators', type=int)
parser.add_argument('--min_samples_leaf', type=int)
parser.add_argument('--input-data', type=str)

args = parser.parse_args()
ne = args.n_estimators
msl = args.min_samples_leaf

# read the input dataset
import pandas as pd
df = new_run.input_datasets['raw_data'].to_pandas_dataframe()
```

```
# Select relevant columns from the dataset
dataPrep = df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], axis=1)
```

```
# Create Dummy variables / encoding the categorical variables
dataPrep = pd.get_dummies(dataPrep, drop_first=True)
```

```
# Define X and Y
X = dataPrep.drop(['Exited'], axis=1)
Y = dataPrep[['Exited']]
```

```
# Split the dataset into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0, stratify=Y)
```

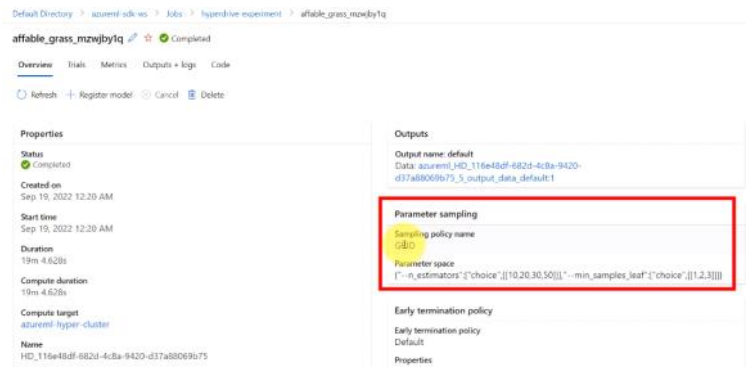
```
# Build and train the Random Forest Classification model
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=ne, min_samples_leaf=msl)
classifier.fit(X_train, Y_train)
```

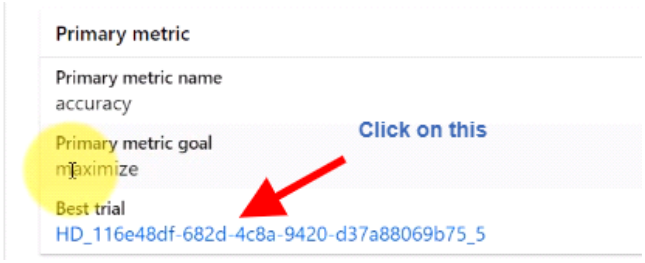
```
# Predict the output - Scored Label
Y_predict = classifier.predict(X_test)
```

```
# Scored Probabilities
Y_prob = classifier.predict_proba(X_test)[:, 1]
```

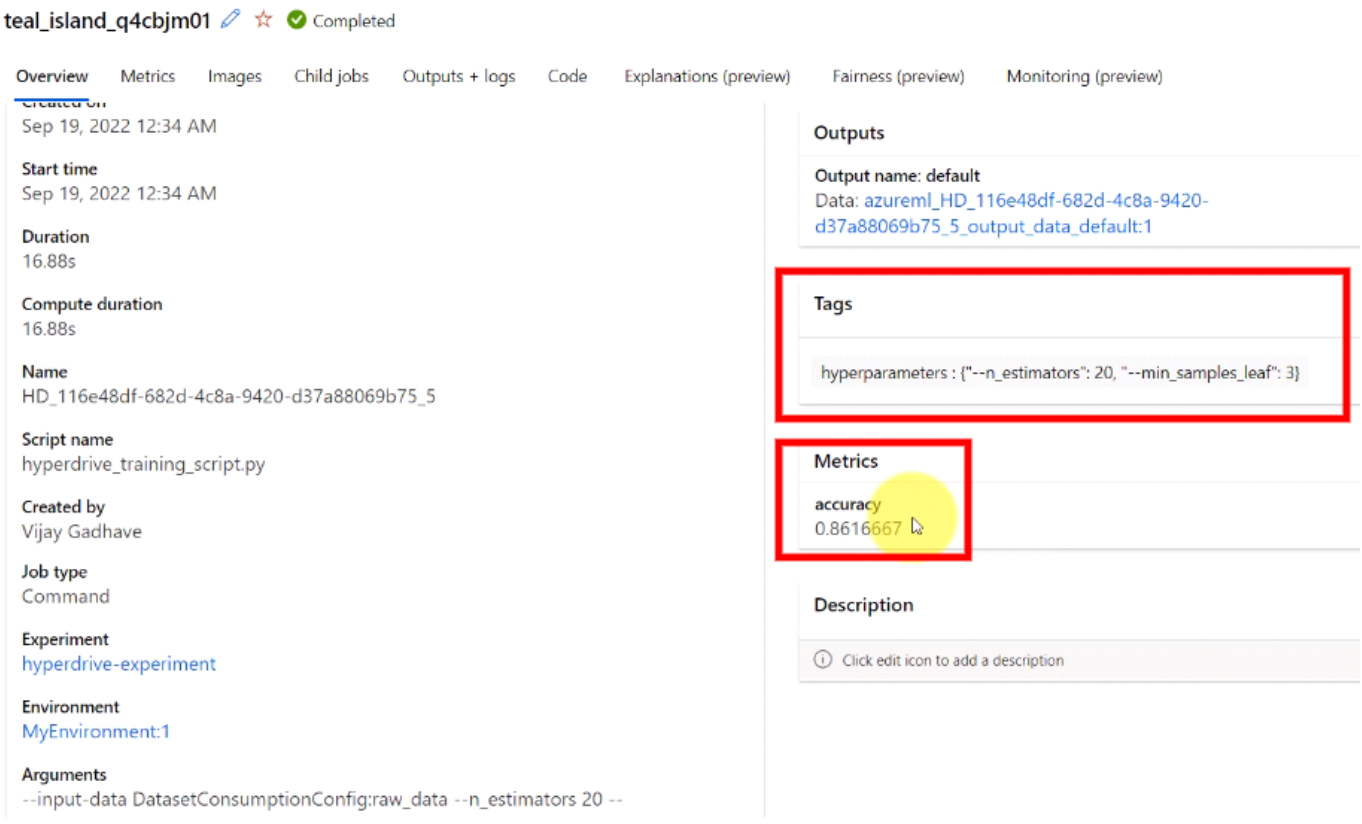
```
# Confusion matrix and accuracy score
from sklearn.metrics import confusion_matrix
cm    = confusion_matrix(Y_test, Y_predict)
score = classifier.score(X_test, Y_test)
```

```
new_run.log('accuracy', score)

new_run.complete()
```





Best model summary:



Notes

Thursday, 18 May, 2023 9:43 AM

1. Installation and Setup
    i. Pip install azureml-sdk
    ii. Access the workspace from config file
2. Accessing the input data
3. Creating the custom environment
    i. Creating Environment
    ii. Creating dependencies object (conda_packages, pip_packages)
    iii. Register the Environment to workspace
4. Create the compute cluster
    i. Specify the name of the cluster
    ii. Import class AmlCompute
    iii. Provision the configuration using AmlCompute
    iv. Create the compute cluster
5. Script Configuration
    i. Import class ScriptRunConfig
    ii. Configuration using ScriptRunConfig

```
script_config = ScriptRunConfig(source_directory='.',
                                script='hyperdrive_training_script.py',
                                arguments=['--input-data', input_ds.as_named_input('raw_data')],
                                environment=myenv,
                                compute_target=cluster)
```

        1. Import class (from azureml.core import Run)
        2. Get the run context
        3. Access the workspace
        4. Get the parameters (with argparse)
        5. Read the input dataset with the run context
        6. Data processing, define X and y, train test split, build and train the model, predict the output(label), scored probabilities, get the confusion matrix and acc score
        7. New_run.log('accuracy', score)
        8. Complete the run
6. Create hyperdrive parameters
    i. Import class GridParameterSampling, choice
    ii. Define the hyperparameters (*hyper_params*) with GridParameterSampling() (grid sampling method)
7. Configure the Hyperdrive class
    i. Import class HyperDriveConfig, PrimaryMetricGoal
    ii. Configuration using HyperDriveConfig

```
hyper_config = HyperDriveConfig(run_config=script_config,
                                hyperparameter_sampling=hyper_params,
                                policy=None,
                                primary_metric_name='accuracy',
                                primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                                max_total_runs=20,
                                max_concurrent_runs=2)
```

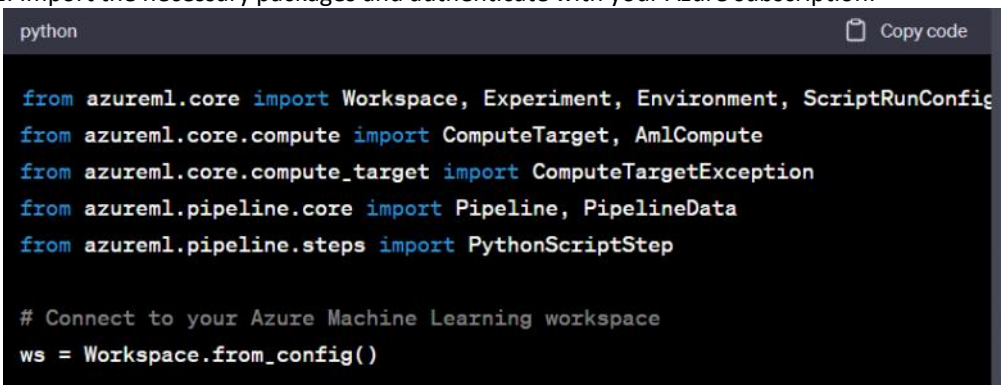
8. Create and submit experiment run

```
# submit the experiment run
new_run = new_exp.submit(config=hyper_config)
new_run.wait_for_completion()
```

Notes2

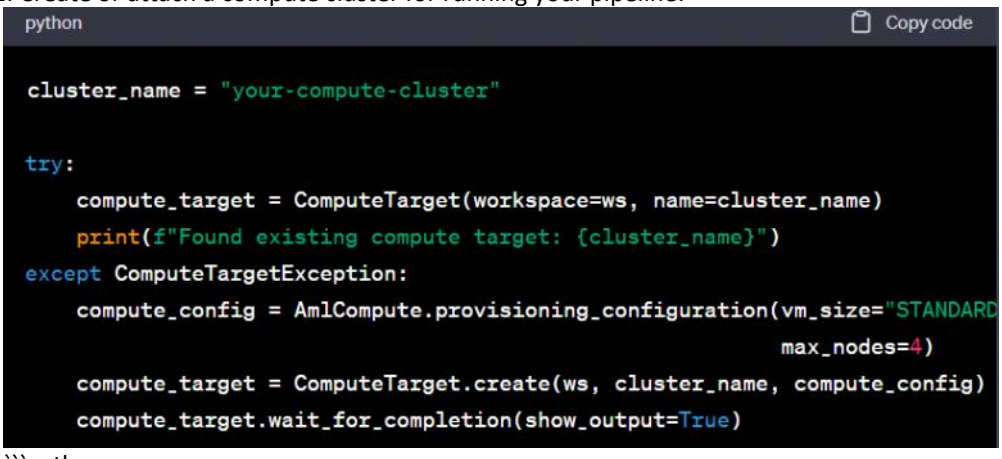
Thursday, 8 June, 2023 4:14 PM

Step 1: Import the necessary packages and authenticate with your Azure subscription:



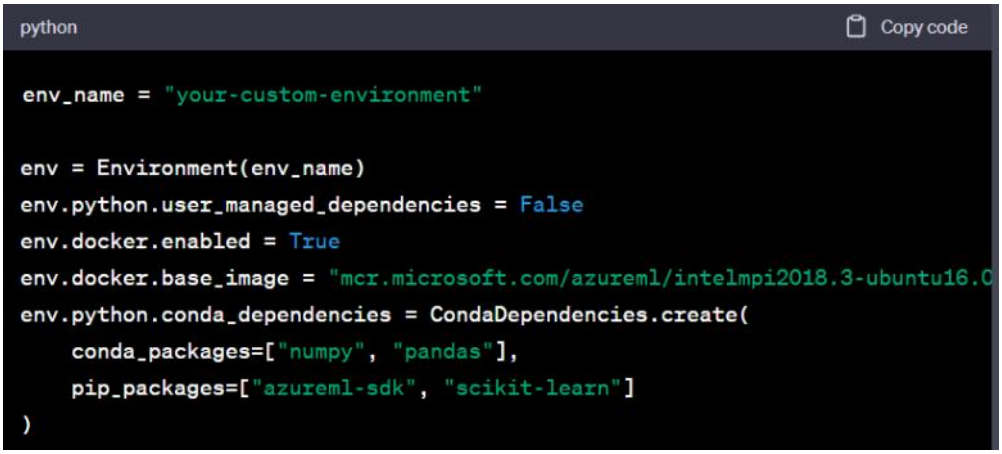

```python
from azureml.core import Workspace, Experiment, Environment, ScriptRunConfig, Dataset
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException
from azureml.pipeline.core import Pipeline, PipelineData
from azureml.pipeline.steps import PythonScriptStep

# Connect to your Azure Machine Learning workspace
ws = Workspace.from_config()
```

Step 2: Create or attach a compute cluster for running your pipeline:



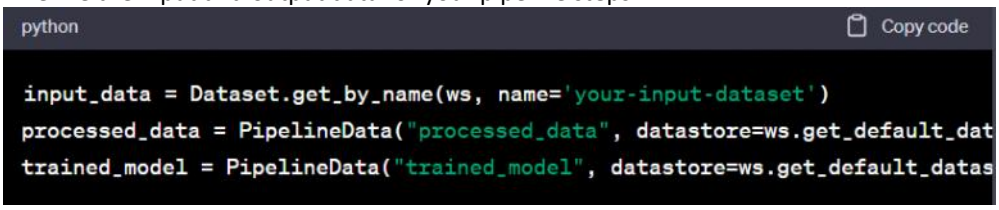

```python
cluster_name = "your-compute-cluster"

try:
    compute_target = ComputeTarget(workspace=ws, name=cluster_name)
    print(f"Found existing compute target: {cluster_name}")
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size="STANDARD_DS3_V2",
                                                           max_nodes=4)
    compute_target = ComputeTarget.create(ws, cluster_name, compute_config)
    compute_target.wait_for_completion(show_output=True)
```

Step 3: Create a custom environment for your pipeline with required packages:



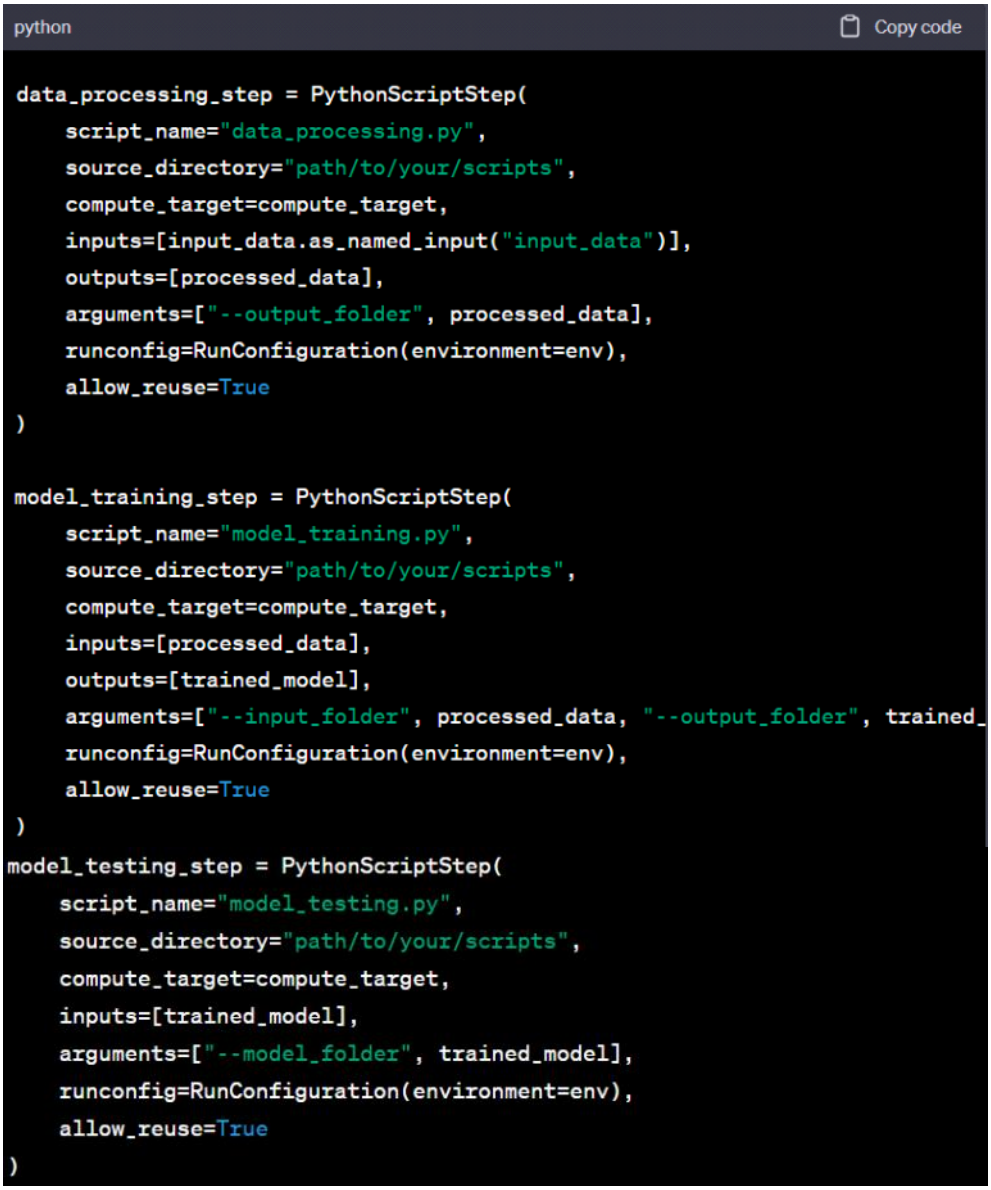

```python
env_name = "your-custom-environment"

env = Environment(env_name)
env.python.user_managed_dependencies = False
env.docker.enabled = True
env.docker.base_image = "mcr.microsoft.com/azureml/intelmpi2018.3-ubuntu16.04"
env.python.conda_dependencies = CondaDependencies.create(
   conda_packages=["numpy", "pandas"],
   pip_packages=["azureml-sdk", "scikit-learn"]
)
```

Step 4: Define the input and output data for your pipeline steps:




```python
input_data = Dataset.get_by_name(ws, name='your-input-dataset')
processed_data = PipelineData("processed_data", datastore=ws.get_default_datastore())
trained_model = PipelineData("trained_model", datastore=ws.get_default_datastore())
```

Step 5: Create the steps for your pipeline:

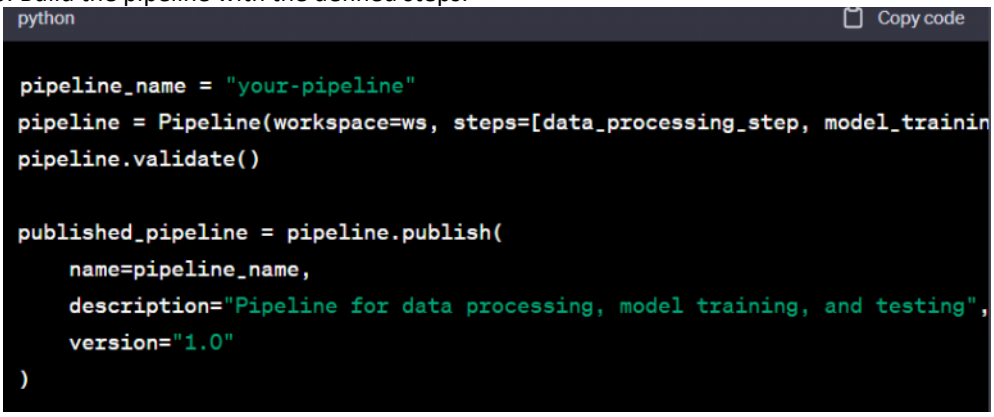


python
data_processing_step = PythonScriptStep(
  script_name="data_processing.py",
  source_directory="path/to/your/scripts",
  compute_target=compute_target,
  inputs=[input_data.as_named_input("input_data")],
  outputs=[processed_data],
  arguments=["--output_folder", processed_data],
  runconfig=RunConfiguration(environment=env),
  allow_reuse=True
)

model_training_step = PythonScriptStep(
  script_name="model_training.py",
  source_directory="path/to/your/scripts",
  compute_target=compute_target,
  inputs=[processed_data],
  outputs=[trained_model],
  arguments=["--input_folder", processed_data, "--output_folder", trained_model],
  runconfig=RunConfiguration(environment=env),
  allow_reuse=True
)

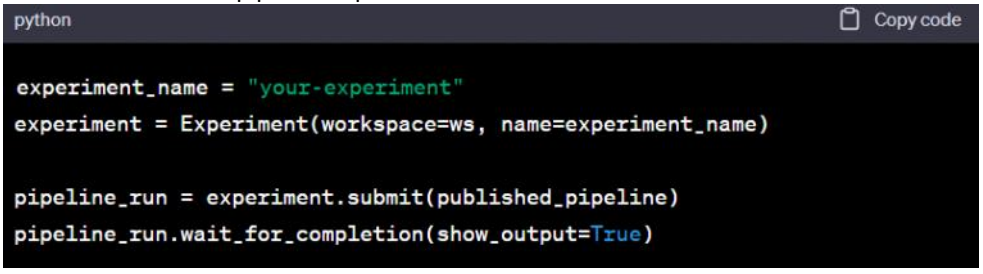
```
model_testing_step = PythonScriptStep(
    script_name="model_testing.py",
    source_directory="path/to/your/scripts",
    compute_target=compute_target,
    inputs=[trained_model],
    arguments=["--model_folder", trained_model],
    runconfig=RunConfiguration(environment=env),
    allow_reuse=True
)
```

Step 6: Build the pipeline with the defined steps:




```python
pipeline_name = "your-pipeline"
pipeline = Pipeline(workspace=ws, steps=[data_processing_step, model_training_step, model_testing_step])
pipeline.validate()

published_pipeline = pipeline.publish(
    name=pipeline_name,
    description="Pipeline for data processing, model training, and testing",
    version="1.0"


)
```

Step 7: Submit and run the pipeline experiment:




```python
experiment_name = "your-experiment"
experiment = Experiment(workspace=ws, name=experiment_name)

pipeline_run = experiment.submit(published_pipeline)
pipeline_run.wait_for_completion(show_output=True)
```

This example provides a basic framework for an end-to-end Azure Machine Learning workflow. You can customize the steps, scripts, and pipeline configuration according to your specific requirements. It covers the creation of a compute cluster, creation of a custom

environment, and building a pipeline with steps for data processing, model training, and testing.

Section 12: Use Model Explainers to Interpret Models

Monday, 22 May, 2023 3:04 PM

The libraries:

1. from interpret_community.tabular_explainer import TabularExplainer

➢ Lecture 1 - Model Explainer on Local Machine

Step 1 - Installation an setup

```
# Install azureml interpret package
! pip install -q azureml-interpret
```

Step 2 - Training the ML model

```
# Import the pandas
import pandas as pd

# Read dataset
df = pd.read_csv('/content/Churn_Modelling.csv')
```

```
df = df.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
```

```
# Dummy variables
df = pd.get_dummies(df, drop_first=True)
```

```
# Define X and Y Variables
X = df.drop(columns='Exited')
Y = df['Exited']
```

```
# Split the X and Y into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0, stratify=Y)
```

```
# Train the ML model
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=0)
trained_ml_model = classifier.fit(X_train, Y_train)
```

```
# Predict the results
y_pred = classifier.predict(X_test)
```

```
# Model evaluation
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, y_pred)
score = classifier.score(X_test, Y_test)
print(cm)
print(score)
```

```
[[2299   90]
 [ 341  270]]
0.8563333333333333
```

Step 3 - Model explainers

Global explanations

A global explanation aims to provide an overall understanding of how the model makes its predictions. It considers the entire model and analyzes its behavior as a whole.

```
from interpret_community.tabular_explainer import TabularExplainer

# Define variables
classes = ['Exited', 'Not Exited']
```

```
features = list(X.columns)

# tabular explainer object
tab_explainer = TabularExplainer(model=trained_ml_model,
                                 initialization_examples=X_train,
                                 features=features,
                                 classes=classes)

# Global explainations
global_explaination = tab_explainer.explain_global(X_train)

# feature importance data
global_feature_imp = global_explaination.get_feature_importance_dict()
```

```
1 global_feature_imp
```

```
{'Age': 0.11569068889397732,
 'NumOfProducts': 0.08005822342620106,
 'IsActiveMember': 0.05316461383325616,
 'Geography_Germany': 0.03189695697579134,
 'Balance': 0.0281066806257185,
 'Gender_Male': 0.027968630058755946,
 'EstimatedSalary': 0.01950655967810963,
 'CreditScore': 0.019100968001856518,
 'Tenure': 0.01330174404392769,
 'Geography_Spain': 0.0063662267453276165,
 'HasCrCard': 0.006194165250576628}
```

Local explanations

Local explanation would help you understand the specific features that influenced the model's decision.

```
# five observations
feature_explain = X_test[0:5]

# Local explanation object
local_explanation = tab_explainer.explain_local(feature_explain)

# extract feature names and importance value
local_features = local_explanation.get_ranked_local_names() # feature names
local_importance = local_explanation.get_ranked_local_values() # corresponding importance values
```

```
1 print(local_features)
```

```
[[['Age', 'Gender_Male', 'Geography_Germany', 'Balance', 'Tenure', 'Geography_Spain',
```

```
1 y_pred[0:5]
```

```
array([0, 0, 0, 0, 0])
```

```
# Print the local explanations
for i in range(0, len(local_features)):
    labels = local_features[i]
    print("\n Feature suppport values for : ", classes[i])

    for j in range(0, len(labels)):
        if y_pred[j] == i:
            print("\n\tObservation number : ", j + 1)
            feature_names = labels[j]

            print("\t\t", "Feature Name".ljust(30), "  Value")
            print("\t\t", "-"*30, "-"*10)

            for k in range(0, len(feature_names)):
                print("\t\t", feature_names[k].ljust(30), round(local_importance[i][j][k], 6))
```

```
Feature suppport values for :  Exited

        Observation number :  1
                Feature Name                    Value
                ------------------------------ ----------
                Age                            0.133111
                Gender_Male                    0.036869
                Geography_Germany              0.032614
                Balance                        0.016073
                Tenure                         0.015483
                Geography_Spain                -0.001166
                HasCrCard                      -0.003277
                EstimatedSalary                -0.00901
                NumOfProducts                  -0.025341
                CreditScore                    -0.03936
                IsActiveMember                 -0.041954

        Observation number :  2
                Feature Name                    Value
                ------------------------------ ----------
                IsActiveMember                 0.147187
                Geography_Spain                0.034841
                Geography_Germany              0.03202
                CreditScore                    0.013241
                Tenure                         0.007167
                NumOfProducts                  -0.007694
                HasCrCard                      -0.011418
                Balance                        -0.023916
                Gender_Male                    -0.025939
                EstimatedSalary                -0.038099
                Age                            -0.243346
```

- Lecture 2 - Model Explainer in AzureML

Step 1 - Installation an setup

```
# Install azureml SDK package
! pip install -q azureml-sdk
# Install azureml interpret package
! pip install -q azureml-interpret

# Importing the class
from azureml.core import Workspace
# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

Step 2 - Accessing the input data

```
input_ds = ws.datasets.get('Churn Modelling Dataset')
```

Step 3 - Creating the custom environment

```
# Importing the classes
from azureml.core import Environment
from azureml.core.environment import CondaDependencies
# Create the environment
myenv = Environment(name="MyEnvironment")
# Create the dependencies object
myenv_dep = CondaDependencies.create(conda_packages=['scikit-learn', 'pip', 'pandas'],
                                     pip_packages=['azureml-defaults', 'azureml-sdk', 'azureml-interpret'])
myenv.python.conda_dependencies = myenv_dep
# Register environment to the workspace
myenv.register(ws)
```

Step 4 - Create the compute Cluster

```
cluster_name = 'azureml-sdk-cluster'
```

```
# Import the class AmlCompute
from azureml.core.compute import AmlCompute
# Provision the configuration using AmlCompute
if cluster_name not in ws.compute_targets:
  compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D11_V2',
                                                          max_nodes=2)
  cluster = AmlCompute.create(workspace=ws, name=cluster_name, provisioning_configuration=compute_config)
  cluster.wait_for_completion()
else:
  cluster = ws.compute_targets[cluster_name]
```

Step 5 - Script configuration for custom environment

```
from azureml.core import ScriptRunConfig
script_config = ScriptRunConfig(source_directory=".",
                                script="training_script_for_the_explainer_job.py",
                                arguments = ['--input-data', input_ds.as_named_input('raw_data')],
                                environment=myenv,
                                compute_target=cluster)
```

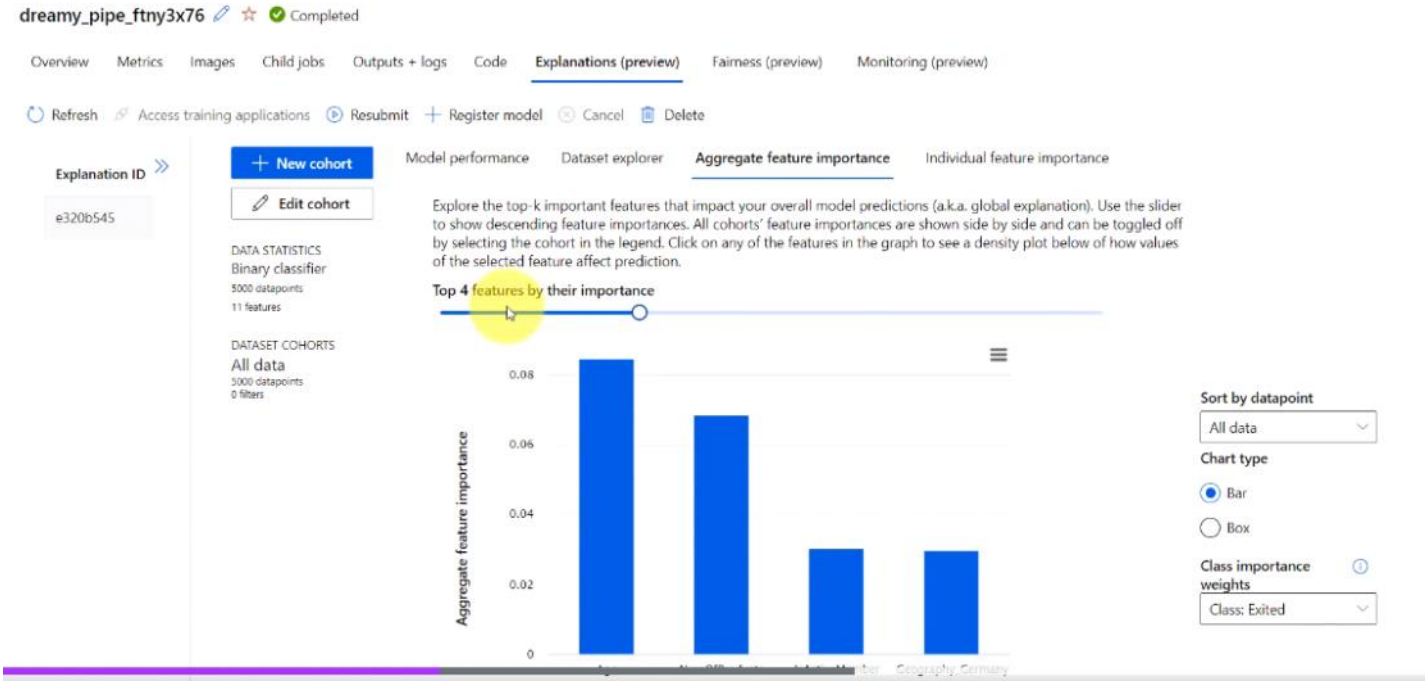
Step 6 - Creating and submitting the Experiment Run

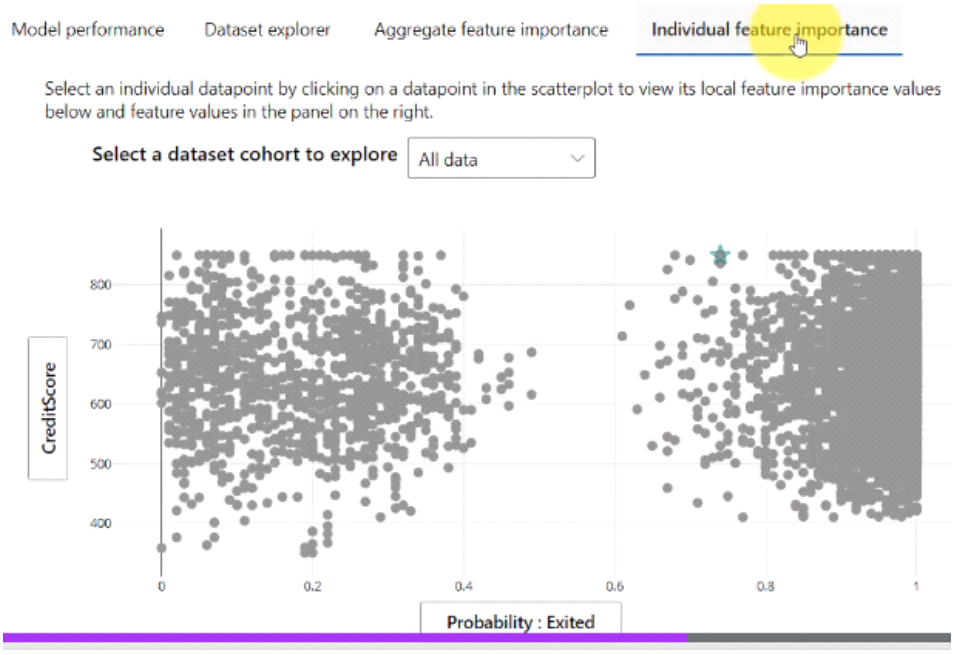
```
# Import the class Experiment
from azureml.core.experiment import Experiment
new_experiment = Experiment(workspace=ws, name='Model_Explainer_Experiment')
new_run = new_experiment.submit(config=script_config)
new_run.wait_for_completion(show_output=True)
```

Training Script for the explainer Job.ipynb

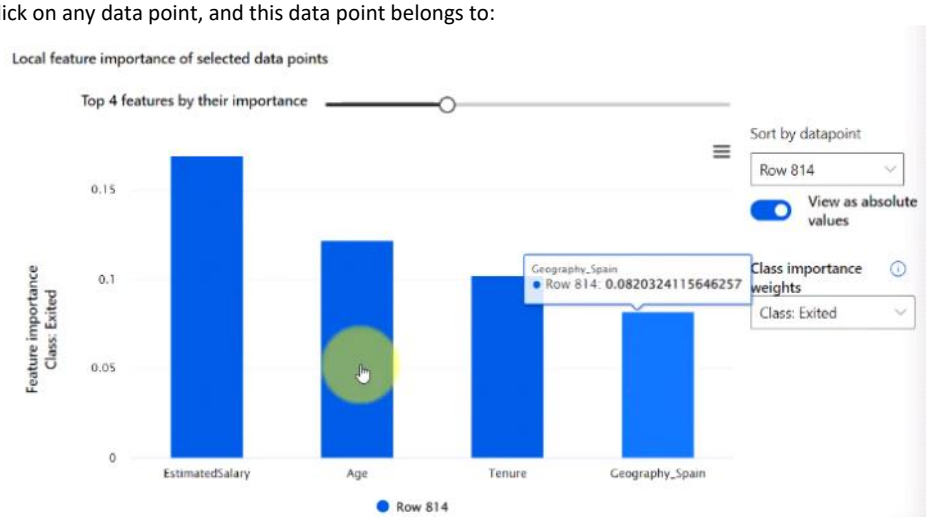
```
from azureml.core import Run
# Get the run context
new_run = Run.get_context()
# Access the workspace
ws = new_run.experiment.workspace

# Get parameters
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--input-data", type=str)
args = parser.parse_args()

import pandas as pd
# Access the dataset
df = new_run.input_datasets['raw_data'].to_pandas_dataframe()
# Drop columns
df = df.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
# Dummy variables
df = pd.get_dummies(df, drop_first=True)
# Define X and Y
X = df.drop(columns='Exited')
Y = df['Exited']
# Split Dataset into train and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 0, stratify=Y)
# Train the ML model
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=0)
trained_ml_model = classifier.fit(X_train, Y_train)
#  Predict the results
y_pred = classifier.predict(X_test)
# Model evaluation
from sklearn.metrics import confusion_matrix
cm    = confusion_matrix(Y_test, y_pred)
score = classifier.score(X_test, Y_test)

# Logging the primary metric
new_run.log("accuracy", score)

# Get explanation
```

```
from interpret_community.tabular_explainer import TabularExplainer
# Define variables
features = list(X.columns)
classes = ['Exited', 'Not Exited']
# tabular explainer object
tab_explainer = TabularExplainer(model=trained_ml_model,
                                 initialization_examples=X_train,
                                 features=features,
                                 classes=classes)
# explainations
explaination = tab_explainer.explain_global(X_train)

# Upload explanations to workspace
from azureml.interpret import ExplanationClient
# Create explanation client
explanation_client = ExplanationClient.from_run(new_run)
# Upload the explanations
explanation_client.upload_model_explanation(explaination, comment="Explanations")
# Complete the run
new_run.complete()
```





For the local importance
Click on any data point, and this data point belongs to:



Section 13: Model Registration and Deployment Using Azureml SDK

Tuesday, 23 May, 2023 3:09 PM

The topics:

- Lecture 1 - Serialization using joblib

```
import pandas as pd
df_2 = pd.read_csv('/content/Churn_Modelling.csv')
columns = df_2.select_dtypes(include='number').columns
```

```
1 df_2[columns].head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	42	2	0.00	1	1	1	101348.88	1
1	608	41	1	83807.86	1	0	1	112542.58	0
2	502	42	8	159660.80	3	1	0	113931.57	1
3	699	39	1	0.00	2	0	0	93826.63	0
4	850	43	2	125510.82	1	1	1	79084.10	0

```
# Load the object
import joblib
obj_file = '/content/scaler.pkl'
sc = joblib.load(obj_file)

# transform the data
df_2[columns] = sc.transform(df_2[columns])
```

```
1 df_2[columns].head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	0.538	0.324324	0.2	0.000000	0.000000	1.0	1.0	0.506735	1.0
1	0.516	0.310811	0.1	0.334031	0.000000	0.0	1.0	0.562709	0.0
2	0.304	0.324324	0.8	0.636357	0.666667	1.0	0.0	0.569654	1.0
3	0.698	0.283784	0.1	0.000000	0.333333	0.0	0.0	0.469120	0.0
4	1.000	0.337838	0.2	0.500246	0.000000	1.0	1.0	0.395400	0.0

- Lecture 2 - Deserialization using joblib

```
# Load the object
import joblib
obj_file = '/content/scaler.pkl'
sc = joblib.load(obj_file)

# transform the data
df_2[columns] = sc.transform(df_2[columns])
```

- Lecture 3 - Handling Dummy Variables

Step 1 - Training Data

```
my_dict = {'Surname':['Clark', 'McDonald', 'Watson', 'Phillipps'],
           'CreditScore': [350, 450, 550, 650],
           'Geography':['France', 'Spain', 'Germany', 'United-States']}

X_train = pd.DataFrame(my_dict)
```

```
1 X_train
```

	Surname	CreditScore	Geography
0	Clark	350	France
1	McDonald	450	Spain
2	Watson	550	Germany
3	Phillipps	650	United-States

```
X_train_dummy = pd.get_dummies(X_train)
```

```
1 X_train_dummy
```

	CreditScore	Surname_Clark	Surname_McDonald	Surname_Phillipps	Surname_Watson	Geography_France	G
0	350	1	0	0	0	1	
1	450	0	1	0	0	0	
2	550	0	0	0	1	0	
3	650	0	0	1	0	0	

```
train_dummy_cols = X_train_dummy.columns
```

```
1 train_dummy_cols
```

```
Index(['CreditScore', 'Surname_Clark', 'Surname_McDonald', 'Surname_Phillipps',
       'Surname_Watson', 'Geography_France', 'Geography_Germany',
       'Geography_Spain', 'Geography_United-States'],
      dtype='object')
```

```
import joblib
obj_file = '/content/scaler.pkl'
joblib.dump(value=train_dummy_cols, filename=obj_file)
```

Step 2 - Production Data

```
ref_cols = joblib.load(obj_file)
```

```
1 ref_cols
```

```
Index(['CreditScore', 'Surname_Clark', 'Surname_McDonald', 'Surname_Phillipps',
       'Surname_Watson', 'Geography_France', 'Geography_Germany',
       'Geography_Spain', 'Geography_United-States'],
      dtype='object')
```

```
my_dict_2 = {'Surname':['Clark'],
             'CreditScore':[450],
             'Geography':['Japan']}

X_prod = pd.DataFrame(my_dict_2)
```

	Surname	CreditScore	Geography
0	Clark	450	Japan

```
X_prod_dummy = pd.get_dummies(X_prod)
```

	CreditScore	Surname_Clark	Geography_Japan
0	450	1	1

```
prod_cols = X_prod_dummy.columns

# Get the missing columns
missing_cols = ref_cols.difference(prod_cols)
```

```
1 missing_cols
```

```
Index(['Geography_France', 'Geography_Germany', 'Geography_Spain',
       'Geography_United-States', 'Surname_McDonald', 'Surname_Phillipps',
       'Surname_Watson'],
      dtype='object')
```

```
# missing columns assign values 0
for cols in missing_cols:
  X_prod_dummy[cols]=0
```

```
1 X_prod_dummy
```

	CreditScore	Surname_Clark	Geography_Japan	Geography_France	Geography_Germany	Geography_Spain
)	450	1	1	0	0	0

```
X_prod_dummy = X_prod_dummy[ref_cols]
```

```
1 X_prod_dummy
```

	CreditScore	Surname_Clark	Surname_McDonald	Surname_Phillipps	Surname_Watson	Geography_France
0	450	1	0	0	0	0

```
# Extract columns present in production data but not in training data
extra_cols = prod_cols.difference(ref_cols)
```

```
1 extra_cols
```

```
Index(['Geography_Japan'], dtype='object')
```

- Lecture 4 - Train ML Model for Webservice Deployment

Step 1 - Installation and Setup

```
[ ]  1 # Install azureml SDK package
     2 ! pip install -q azureml-sdk
```

```
[ ]  1 # Importing the class
     2 from azureml.core import Workspace, Dataset, Experiment
```

```
[ ]  1 # Access the workspace from config file and creating a workspace object
     2 ws = Workspace.from_config(path='/content/config.json')
```

Step 2 - Accessing the Input Dataset

```
     1 input_dataset = Dataset.get_by_name(workspace=ws, name='Churn')
```

Step 3 - Create an Experiment

```
[ ]  1 experiment = Experiment(workspace=ws, name='Webservice-Experiment')
     2
     3 # Run the experiment
     4 new_run = experiment.start_logging()
```

Step 4 - ML Model Training

```
[ ]  1 import pandas as pd
     2
     3 # Load the dataset
     4 df = input_dataset.to_pandas_dataframe()
```

```
[ ]  1 df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	619	France	Female	42	2	0.00	1	1
1	608	Spain	Female	41	1	83807.86	1	0
2	502	France	Female	42	8	159660.80	3	1
3	699	France	Female	39	1	0.00	2	0
4	850	Spain	Female	43	2	125510.82	1	1

```
[ ]  1 # Define X and Y
     2 X = df.iloc[:, :-1]
     3 Y = df.iloc[:, -1:]
```

```
[ ]  1 # Encode the categorical variables
     2 X = pd.get_dummies(X)
```

```
[ ]  1 X.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_
0	619	42	2	0.00	1	1	1	101348.88	
1	608	41	1	83807.86	1	0	1	112542.58	
2	502	42	8	159660.80	3	1	0	113931.57	
3	699	39	1	0.00	2	0	0	93826.63	
4	850	43	2	125510.82	1	1	1	79084.10	

```
[ ]  1 train_dummy_cols = X.columns

[ ]  1 train_dummy_cols

Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Geography_France',
       'Geography_Germany', 'Geography_Spain', 'Gender_Female', 'Gender_Male'],
      dtype='object')

[ ]  1 # Split the dataset
     2 from sklearn.model_selection import train_test_split
     3 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0, stratify=Y)

     1 # Build and Traing ML Model
     2 from sklearn.ensemble import RandomForestClassifier
     3 classifier = RandomForestClassifier(random_state=0)
     4 trained_model = classifier.fit(X_train, Y_train)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was
  after removing the cwd from sys.path.

[ ]  1 # Predict the results
     2 Y_pred = classifier.predict(X_test)
     3
     4 # probability score
     5 Y_prob = classifier.predict_proba(X_test)[:, 1]

[ ]  1 # Confusion matrix and accuracy score
     2 from sklearn.metrics import confusion_matrix, accuracy_score
     3 cm = confusion_matrix(Y_test, Y_pred)
     4 score = accuracy_score(Y_test, Y_pred)

[ ]  1 print(cm)
     2 print(score)

[[2293   96]
 [ 341  270]]
0.8543333333333333
```

Step 5 - Log the primary metric

```
     1 new_run.log("accuracy", score)
```

Step 6 - Saving the transformations and models

```
[ ]  1 import joblib
     2 model_file = './outputs/models.pkl'
     3
     4 joblib.dump(value=[train_dummy_cols, trained_model], filename=model_file)

['./outputs/models.pkl']

[ ]  1 new_run.complete()

[ ]  1 # Getting the run ID
     2 list(experiment.get_runs())

[Run(Experiment: Webservice-Experiment,
 Id: 2263be5b-1103-49a4-ad4f-6a5c5b5bd461,
 Type: None,
 Status: Completed)]
```

- Lecture 5 - Register the Model Using Run ID and pkl File

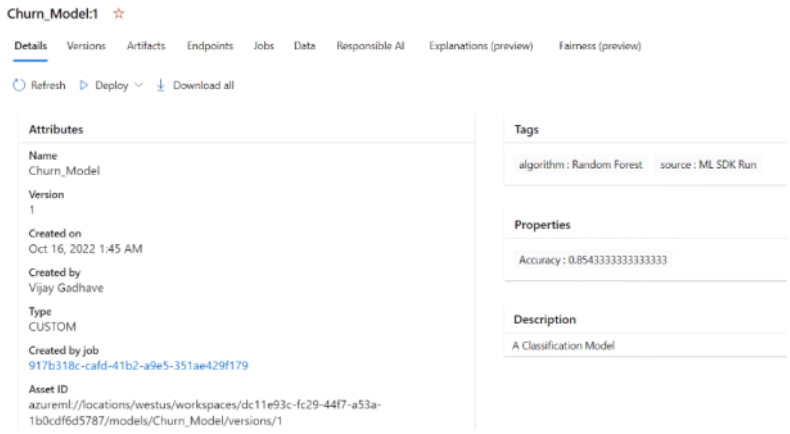
Step 1 - Installation an setup

```
# Install azureml SDK package
```

```
! pip install -q azureml-sdk

# Importing the class
from azureml.core import Workspace, Model

# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

Step 2 - Register the Model

```
# Access the run using run ID
new_run = ws.get_run(run_id='2263be5b-1103-49a4-ad4f-6a5c5b5bd461')

new_run.register_model(model_path='outputs/models.pkl',
                       model_name='Churn_models',
                       tags={'source': 'ML SDK Run', 'algorithm': 'Random Forest'},
                       properties={'Accuracy': new_run.get_metrics()['accuracy']},
                       description='A Classification Model')
```



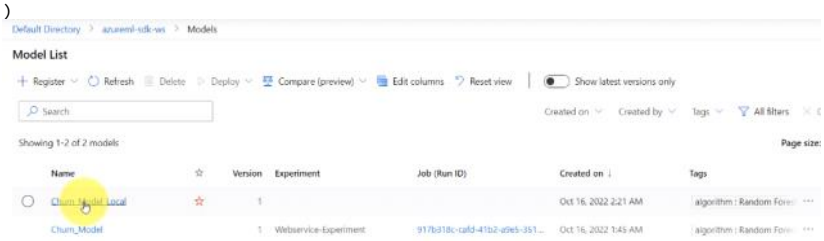
- Lecture 6 - Register the Model Using Local pkl File

Step 1 - Installation an setup

```
# Install azureml SDK package
! pip install -q azureml-sdk

# Importing the class
from azureml.core import Workspace, Model

# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

Step 2 - Register the Model

```
Model.register(workspace=ws,
               model_path='/content/models.pkl',
               model_name='Churn_Model_Local',
               tags={'source': 'ML SDK Run Local', 'algorithm': 'Random Forest'},
               properties={'Accuracy': 0.85},
               description='A Local Classification Model'
)
```



There is no Run ID/Experiment for model register with local pkl file

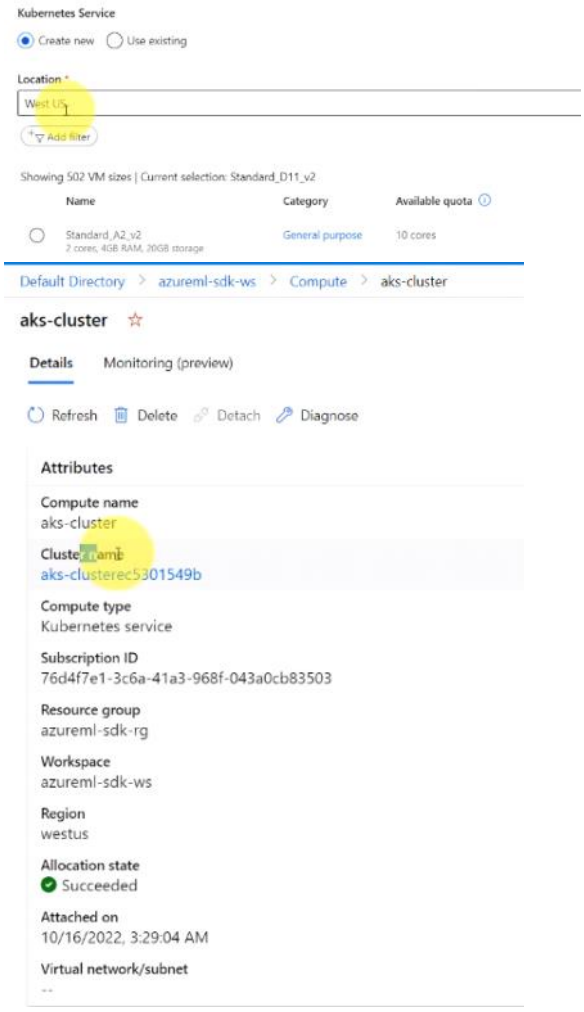
- Lecture 7 - Provision AKS Production Cluster

Step 1 - Installation and Setup

```
# Install azureml SDK package
! pip install -q azureml-sdk

# Importing the class
from azureml.core import Workspace

# Access the workspace from config file and creating a workspace object
ws = Workspace.from_config(path='/content/config.json')
```

Step 2 - Create an Azure Kubernetes Cluster

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

aks_name = 'aks-cluster'
# Create the aks cluster
prov_config = AksCompute.provisioning_configuration(location='westus',
                                                    vm_size='Standard_D11_v2',
                                                    agent_count=1,
                                                    cluster_purpose='DevTest')
# Deployment
aks_target = ComputeTarget.create(workspace=ws,
                                  name=aks_name,
                                  provisioning_configuration=prov_config)
```



- Project 3 Deploy models to Azure Kubernetes Service (AKS)
  - Step 1 - Installation and Setup
  - Step 2 - Train and Serialize ML Model
  - Step 3 - Register the ML Model
  - Step 4 - Register an Environment
  - Step 5 - Create AKS Cluster
  - Step 6 - Inference Configuration
  - Step 7 - Deployment Configuration
  - Step 8 - Create an Endpoint
  - Step 9 - Test the web service
  - Step 10 - Delete the Resources

Step 1 - Installation and Setup

```
# Install azureml sdk package
! pip install -q azureml-sdk

# Import the class workspace
from azureml.core import Workspace

# Access the workspace (that has been created in the Azure portal)
ws = Workspace.from_config(path='/content/config.json')
```

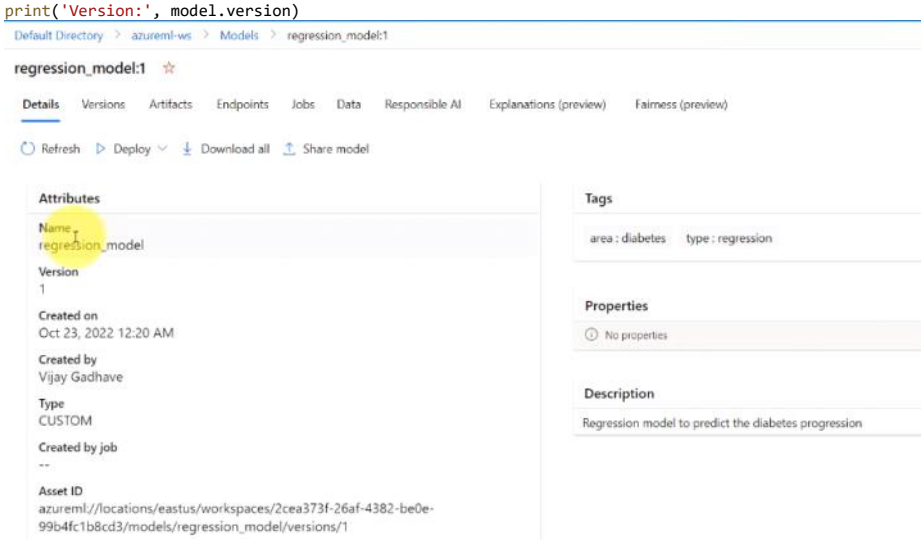
Step 2 - Train and Serialize ML Model

```
import sklearn
from sklearn.datasets import load_diabetes
```

```
from sklearn.linear_model import Ridge

x, y = load_diabetes(return_X_y=True)
model = Ridge().fit(x, y)

import joblib
joblib.dump(model, 'regression_model.pkl')
```

Step 3 - Register the ML Model

```
from azureml.core import Model

# Register the ml model to the workspace
model = Model.register(workspace=ws,
                       model_name='regression_model',
                       model_path='./regression_model.pkl',
                       model_framework=Model.Framework.SCIKITLEARN,
                       model_framework_version='0.19.1',
                       description='Regression model to predict the diabetes progression',
                       tags={'area': 'diabetes', 'type': 'regression'})

print('Name:', model.name)
print('Version:', model.version)
```



Step 4 - Register an Environment

```
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies

environment = Environment(name='my-environment')
environment.python.conda_dependencies = CondaDependencies.create(
    conda_packages=['pip==20.2.4'],
    pip_packages=['azureml-defaults', 'inference-schema[numpy-support]', 'numpy', 'scikit-learn==0.22.1', 'scipy'])
```

Step 5 - Create AKS Cluster

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

cluster_name = 'akscluster'
if cluster_name not in ws.compute_targets:
  print(cluster_name, 'does not exist, Creating new one...')
  print('provisioning config for AKS Cluster...')
  aks_config = AksCompute.provisioning_configuration(location='eastus',
                                                     vm_size='Standard_D11_v2',
                                                     agent_count=1,
                                                     cluster_purpose='DevTest')
  print('Creating the AKS Cluster...')
  production_cluster = ComputeTarget.create(workspace=ws,
                                            name=cluster_name,
                                            provisioning_configuration=aks_config)
  production_cluster.wait_for_completion(show_output=True)
else:
  print(cluster_name, 'Exists, and using the same...')
  production_cluster = ws.compute_targets[cluster_name]
```

Step 6 - Inference Configuration

The inference configuration encapsulates the necessary information to load the model, set up the execution environment, and handle data transformations, ensuring that the deployed model behaves consistently and produces accurate predictions.(software details)

```
from azureml.core.model import InferenceConfig

inference_config = InferenceConfig(entry_script='score_1.py', # also known as scoring script
                                   source_directory='.',
                                   environment=environment)
```

Entry Script:

This Python script defines the code for loading the model, preprocessing input data, making predictions, and post-processing the results. It encapsulates the logic necessary to transform raw input into a format suitable for the model and convert the model's output into the desired format.

Conda Environment:
The inference configuration allows you to specify a Conda environment file that defines the Python packages and dependencies required by the inference script. This ensures that the deployment environment matches the development environment where the model was trained and tested.

Score 1.ipynb

```
# must include 2 functions: init and run
import pickle
import json
import numpy
from sklearn.externals import joblib
from sklearn.linear_model import Ridge
from azureml.core.model import Model

def init():  # init function is called when the service is initialized
    global model
    model_path = Model.get_model_path('regression_model') # specify the model name
    model = joblib.load(model_path) # deserialize the model file back into sklearn model

def run(raw_data): # run is called when new data is submitted to this service
  try:
    data = json.loads(raw_data)['data']
    data = numpy.array(data)
    result = model.predict(data)
    return result.tolist()
  except Exception as e:
    error = str(e)
    return error
```

Typically we use init function to load the model from the model registry and we use the run function to generate the predictions from the input data.

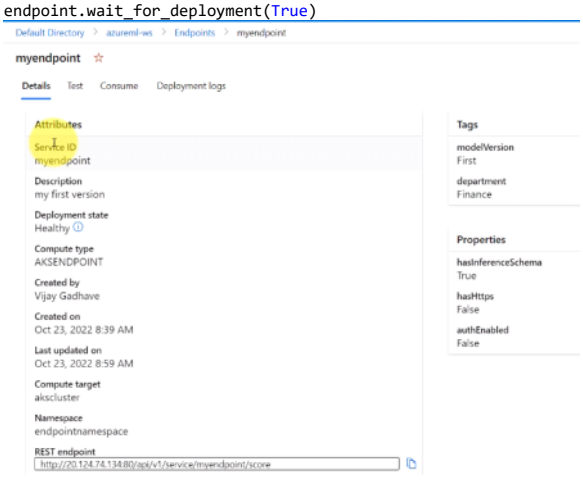
Step 7 - Deployment Configuration

The deployment configuration allows you to define the deployment target and scale the model serving infrastructure according to the expected workload, ensuring efficient resource utilization and meeting performance requirements. (hardware details-such as the compute target, the number of instances to deploy, the type of instance, and the scale settings)

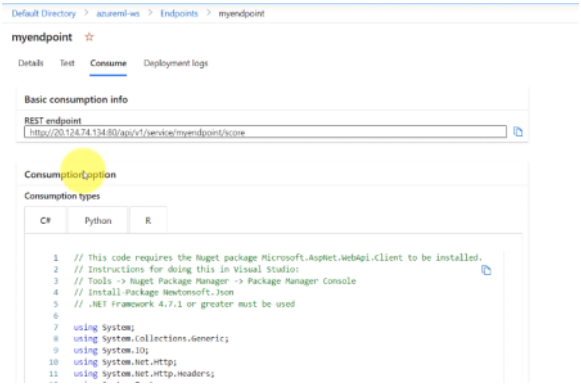
```
namespace_name = 'endpointnamespace'
version_name = 'version1'

from azureml.core.webservice import AksEndpoint

endpoint_deployment_config = AksEndpoint.deploy_configuration(tags={'modelVersion': 'First', 'department': 'Finance'},
                                                              description='my first version',
                                                              namespace=namespace_name,
                                                              version_name=version_name,
                                                              traffic_percentile=10)
```

Step 8 - Create an Endpoint

```
# select the compute
compute = ComputeTarget(workspace=ws, name='akscluster')

# Endpoint name
endpoint_name =endpointnt'

endpoint = Model.deploy(workspace=ws,
                        name=endpoint_name,
                        models=[model],
                        inference_config=inference_config,
                        deployment_config=endpoint_deployment_config,
                        deployment_target=compute)

endpoint.wait_for_deployment(True)
```



We can consume the endpoint model by:



Step 9 - Test the web service

```
import json

test_sample = json.dumps({'data':[
    [1,2,3,4,5,6,7,8,9,10],
    [10,9,8,7,6,5,4,3,2,1]
]})

test_sample_encoded = bytes(test_sample, encoding='utf8') This encoding is necessary to pass the data to the endpoint.
prediction = endpoint.run(input_data=test_sample_encoded)
print(prediction)
```

Step 10 - Delete the Resources

```
endpoint.delete()
```

- Project 4 Deploy models to Azure Kubernetes Service (AKS)
  - Step 1 - Installation and Setup
  - Step 2 - Train and Serialize the ML Model
  - Step 3 - Register the ML Models
  - Step 4 - Register an Environment
  - Step 5 - Inference Configuration
  - Step 6 - Deployment Configuration
  - Step 7 - Deploy Model as Webservice on Azure Container Instance
  - Step 8 - Test the web service

Step 1 - Installation and Setup

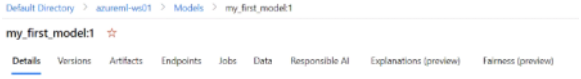
```
# Install azureml sdk package
! pip install -q azureml-sdk

# Import the class workspace
from azureml.core import Workspace

# Access the workspace
ws = Workspace.from_config(path='/content/config.json')
```

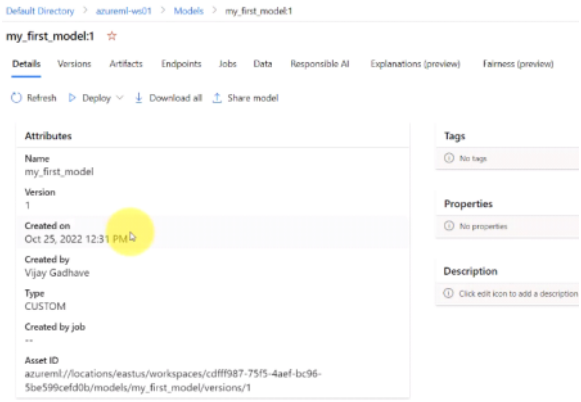
Step 2 - Train and Serialize the ML Model

```
import sklearn
from sklearn.datasets import load_diabetes
from sklearn.linear_model import BayesianRidge, Ridge

x, y = load_diabetes(return_X_y=True)
first_model = Ridge().fit(x, y)
second_model = BayesianRidge().fit(x, y)

import joblib

joblib.dump(first_model, 'first_model.pkl')
joblib.dump(second_model, 'second_model.pkl')
```

Step 3 - Register the ML Models

```
from azureml.core.model import Model

my_model_1 = Model.register(workspace=ws,
                            model_path='first_model.pkl',
                            model_name='my_first_model')

my_model_2 = Model.register(workspace=ws,
                            model_path='second_model.pkl',
                            model_name='my_second_model')
```





It turns version 2 if register 2nd time for the same model

```
1 print('Name:', my_model_1.name)
2 print('Version:', my_model_1.version)
```

```
Name: my_first_model
Version: 2
```

```
[ ]  1 print('Name:', my_model_2.name)
     2 print('Version:', my_model_2.version)
```

```
Name: my_second_model
Version: 2
```

Name	Version	Experiment
my_second_model	2	
my_first_model	2	
my_second_model	1	
my_first_model	1	

Step 4 - Register an Environment

```
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies

environment = Environment(name='my-environment')
environment.python.conda_dependencies = CondaDependencies.create(
    conda_packages=['pip==20.2.4'],
    pip_packages=['azureml-defaults', 'inference-schema[numpy-support]', 'numpy', 'scikit-learn==0.22.1', 'scipy', 'joblib'])
```

Step 5 - Inference Configuration

```
inference_config = InferenceConfig(entry_script='score_2.py',
                                   source_directory='.',
                                   environment=environment)
```

Score 2.ipynb

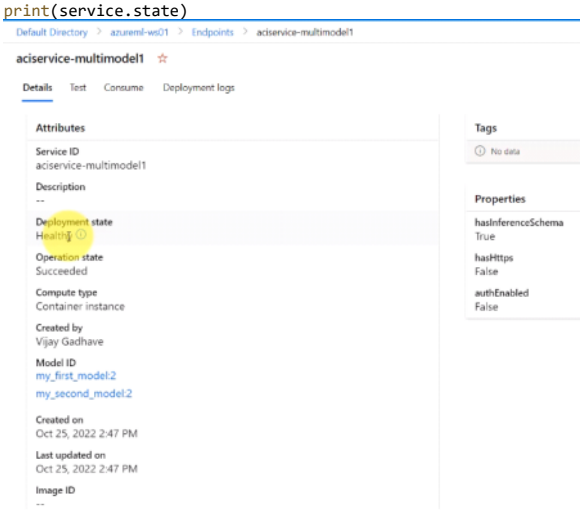
```
import joblib
import json
import numpy as np
from azureml.core.model import Model

def init():
  global model_1, model_2
  # specify the model path
  model_1_path = Model.get_model_path('my_first_model')
  model_2_path = Model.get_model_path('my_second_model')
  # Deserialize the models
  model_1 = joblib.load(model_1_path)
  model_2 = joblib.load(model_2_path)

def run(raw_data):
  try:
    data = json.loads(raw_data)['data']
    data = np.array(data)
    result_1 = model_1.predict(data)
    result_2 = model_2.predict(data)
    return {"prediction1": result_1.tolist(), 'prediction2': result_2.tolist()}

  except Exception as e:
    result = str(e)
    return result
```

Step 6 - Deployment Configuration

```
from azureml.core.webservice import AciWebservice

deployment_config = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1)
```

Step 7 - Deploy Model as Webservice on Azure Container Instance

```
aci_service_name = 'aciservice-multimodel1'

service = Model.deploy(workspace=ws,
                       name=aci_service_name,
                       models=[my_model_1, my_model_2],
                       inference_config=inference_config,
                       deployment_config=deployment_config,
                       overwrite=True)
service.wait_for_deployment(True)
print(service.state)
```



Step 8 - Test the web service

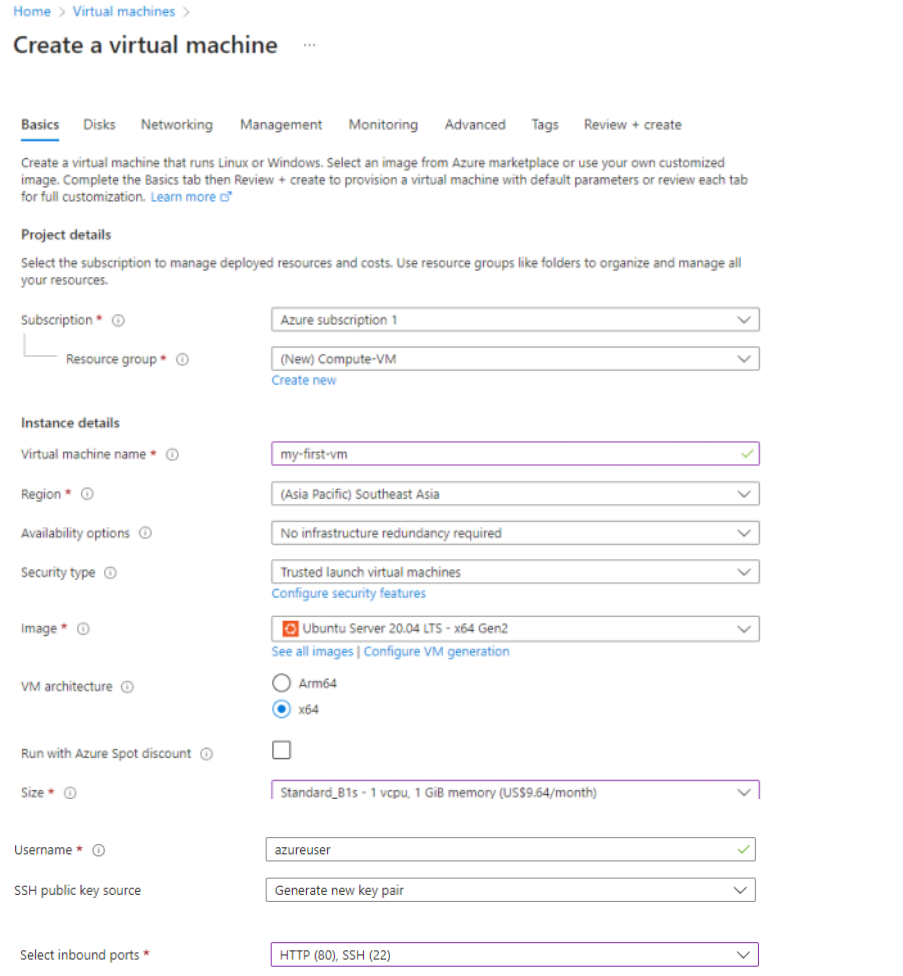
```
import json

test_sample = json.dumps({'data': x[0:2].tolist()})
prediction = service.run(test_sample)
print(prediction)

service.delete()
```

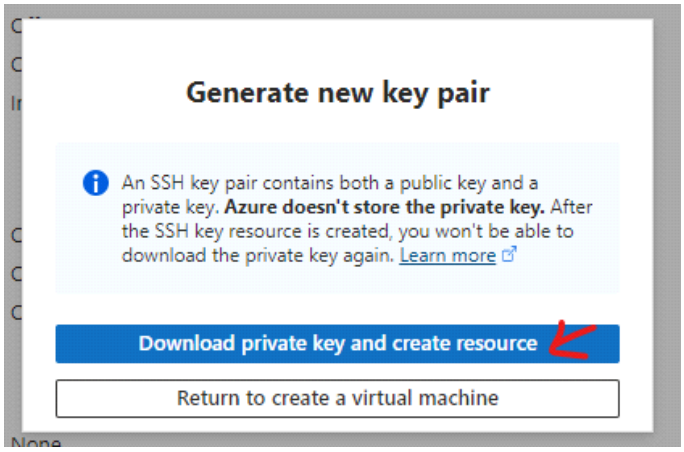
Section 14: Azure Fundamentails: Virtual Machines

Monday, 29 May, 2023 1:11 PM

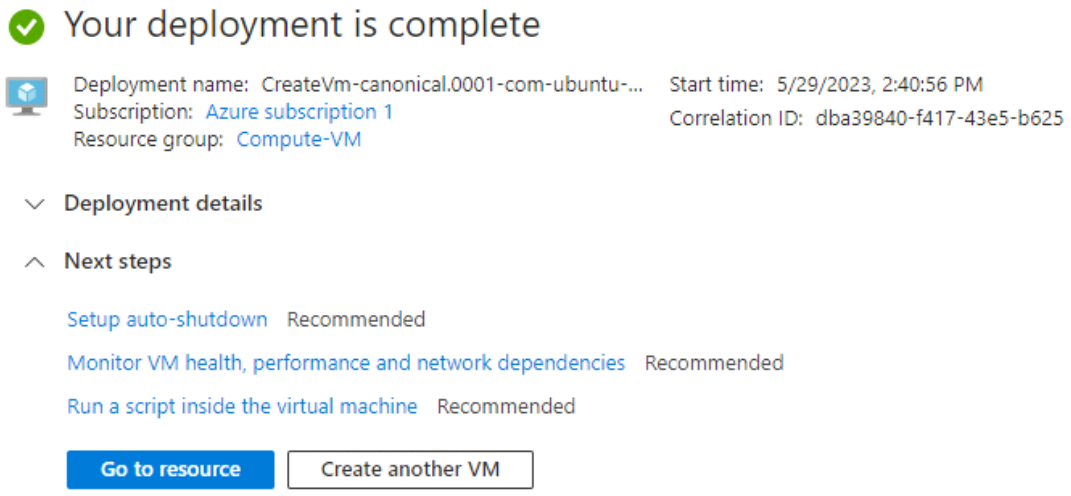
The topics:

➢ Creating Virtual Machine in Azure

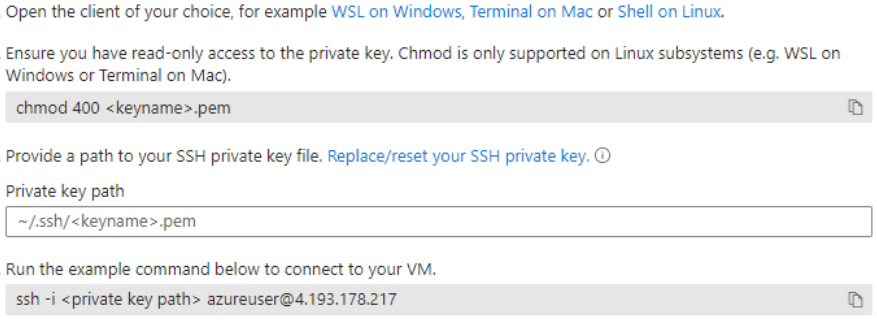




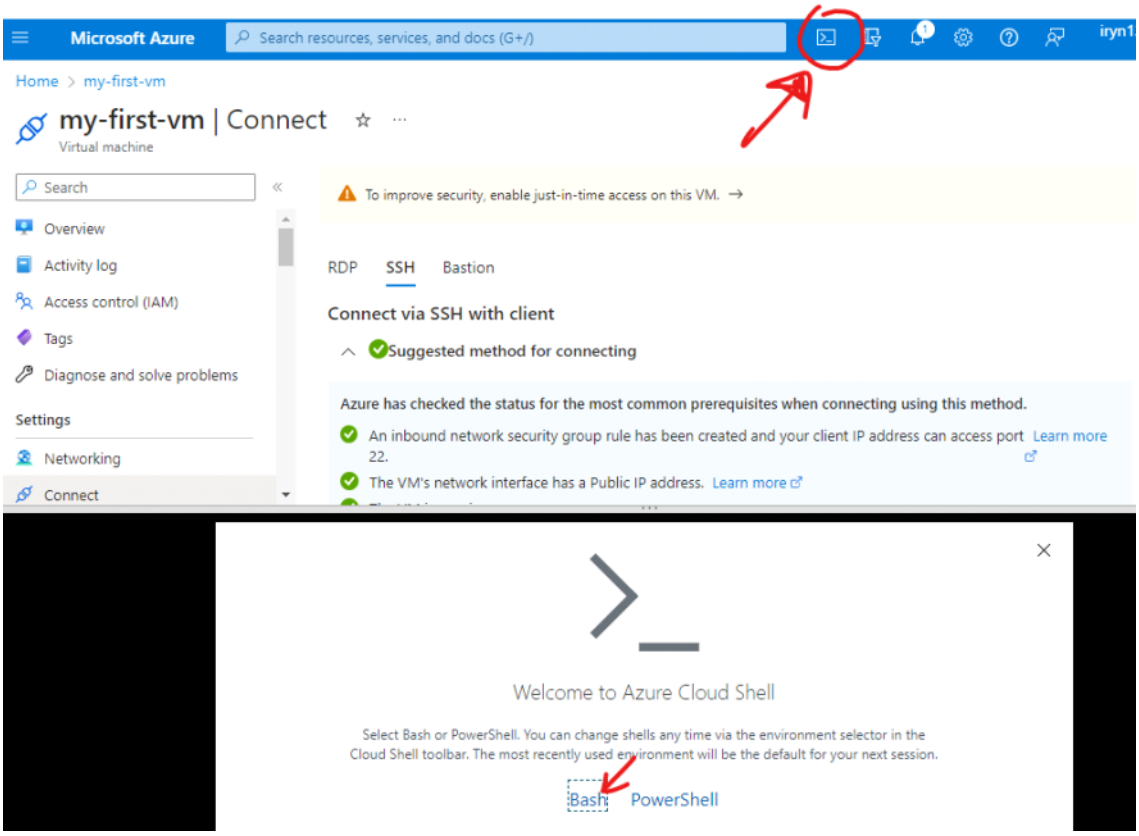
The .pem private key is downloaded:



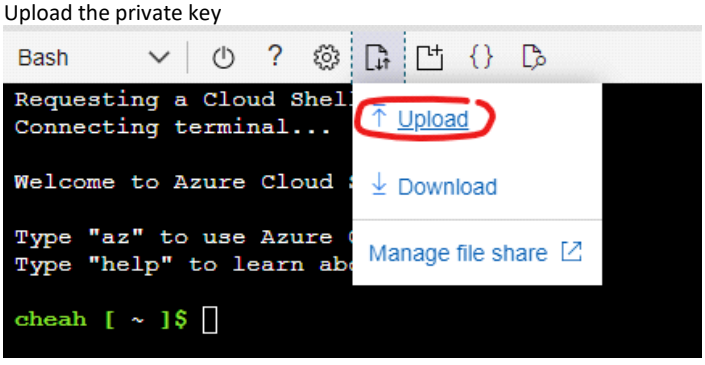
- Connecting to Virtual Machine and Running Commands
  - Connect to the VM



1. Open Cloud Shell, select Bash:



Upload the private key




```
cheah [ ~ ]$ ls
clouddrive  my-first-vm_key.pem
```

2. chmod 400 <keyname>.pem

```
cheah [ ~ ]$ chmod 400 my-first-vm_key.pem
```

3. ssh -i <private key path> azureuser@4.193.178.217

```
cheah [ ~ ]$ ssh -i my-first-vm_key.pem azureuser@4.193.178.217
The authenticity of host '4.193.178.217 (4.193.178.217)' can't be established.
ED25519 key fingerprint is SHA256:+e6m0HFehkpZrSRZfgnEPRLfOyXL+LfDIz3eMCwkkpA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '4.193.178.217' (ED25519) to the list of known hosts.
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1038-azure x86_64)
```

We successfully connected to the VM

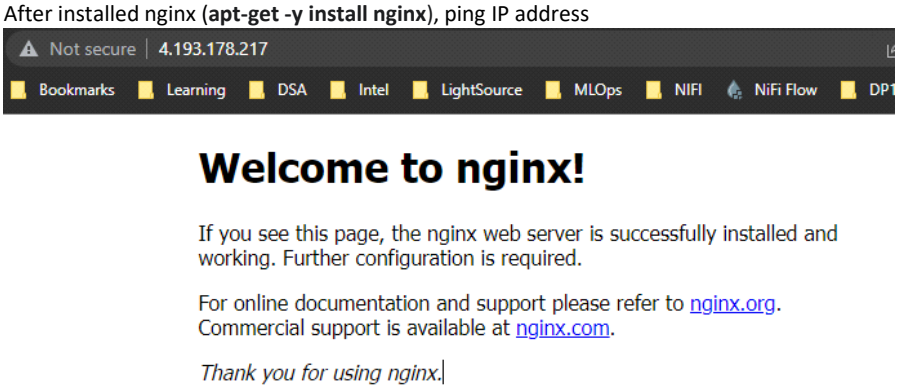
```
azureuser@my-first-vm:~$ whoami
azureuser
azureuser@my-first-vm:~$ hostname -i
10.0.0.4
azureuser@my-first-vm:~$ hostname
my-first-vm
```

- Commands executed in Tutorial

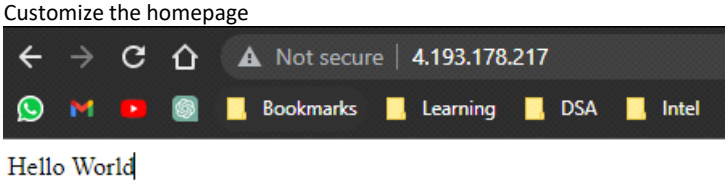
  Below are the commands executed in next tutorial
  - **sudo su -** To become a super user (to install the software)
  - **apt-get -y update -** Update the package index, pull latest changes from the repositories
  - **apt-get -y install nginx -** Install and start nginx web server
  - **echo "Hello World" > /var/www/html/index.html -** Write to index.html
  - **echo "Hello world from $(hostname)" > /var/www/html/index.html -** Write to index.html
- Installing nginx on Azure VM
  - ***nginx***

    Web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache

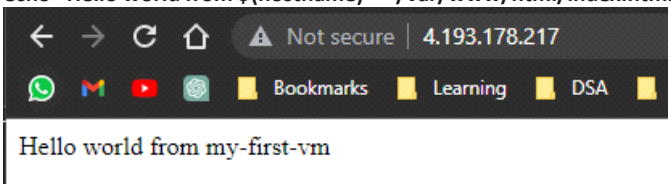
  - After installed nginx (**apt-get -y install nginx**), ping IP address



  - Customize the homepage



  - **echo "Hello world from $(hostname)" > /var/www/html/index.html**

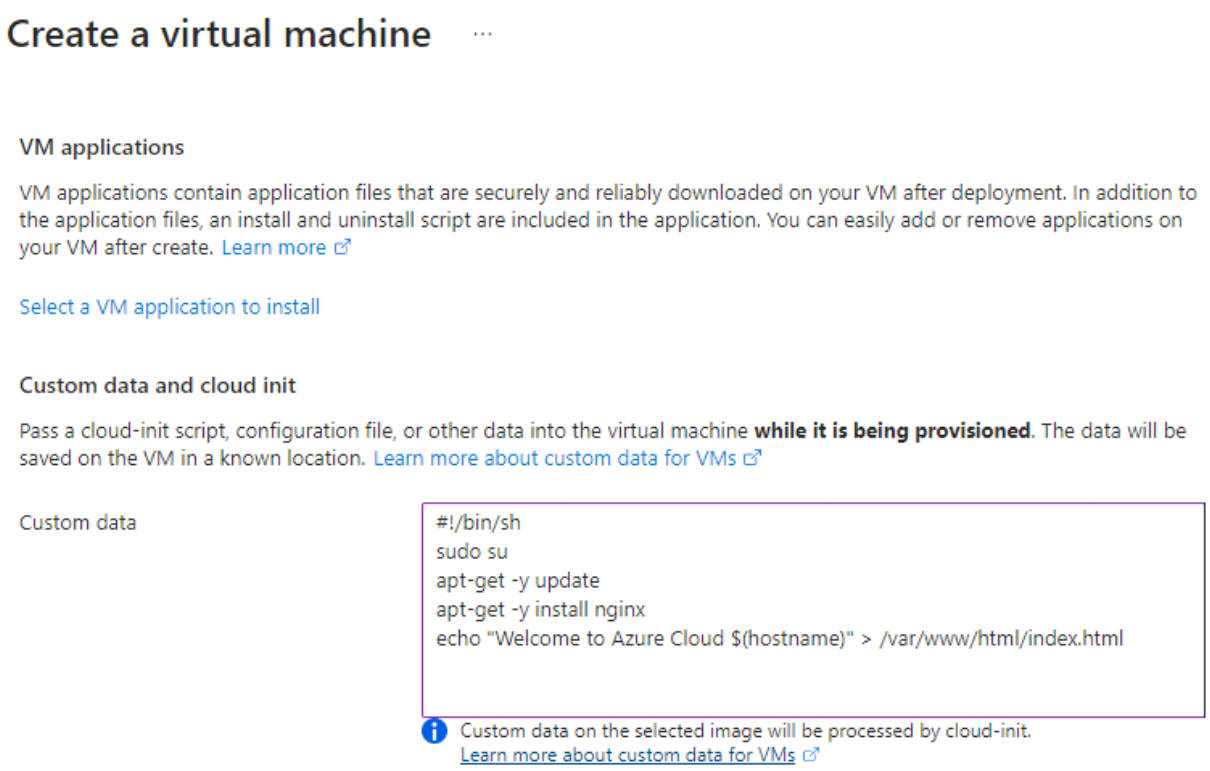


- Commands executed in Tutorial

  Below are the commands executed in next tutorial

  ```
  #!/bin/sh
  sudo su
  apt-get -y update
  apt-get -y install nginx
  echo "Welcome to Azure Cloud $(hostname)" > /var/www/html/index.html
  ```

- Simplification of Software Installation on Azure Virtual Machine
  - Create VM -> Advanced



- Availability Sets and Zones
  - ***Availability***
    - *A logical grouping of VMs that allows Azure to understand how your application is built to provide for redundancy and availability*
    - *The percentage of time an application provides for the expected operations*

Percentage availability in a month (Total runtime of the application in terms of the percentage):

Availability

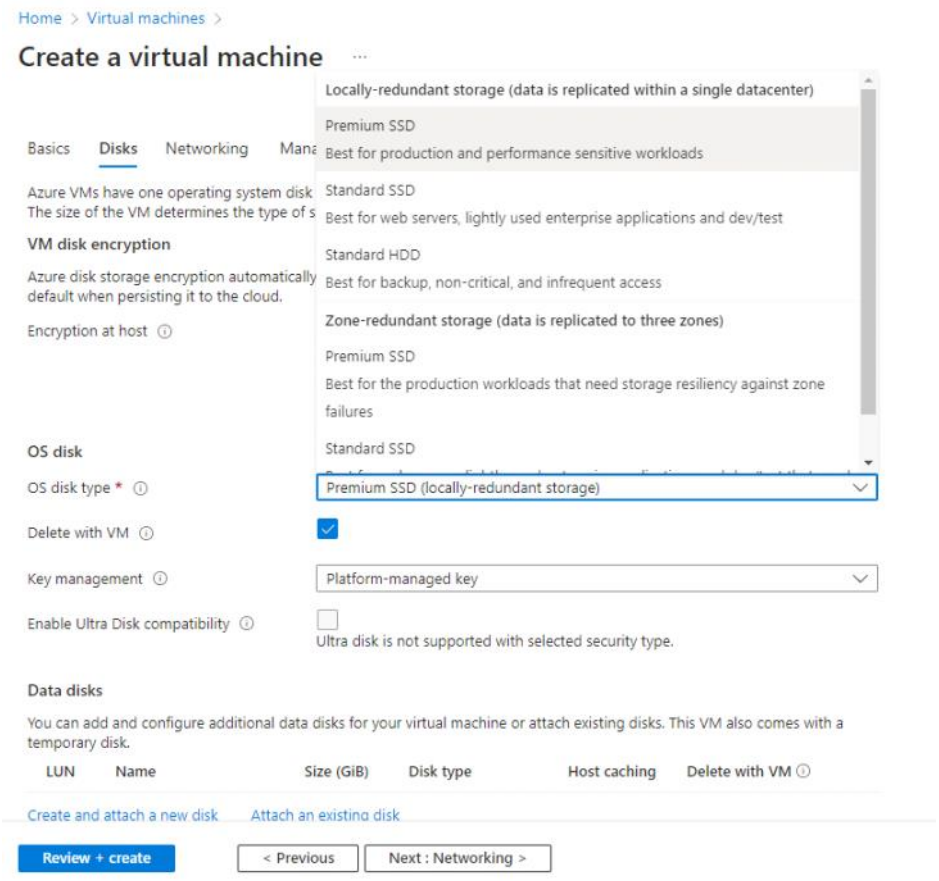
Availability	Downtime (in a month)	Comment
99.95%	22 minutes	
99.99%	4.5 minutes	Online apps aim for this
99.999%	26 seconds	Very tough to achieve

Increasing Availability for Azure VMs

Single instance VMs:

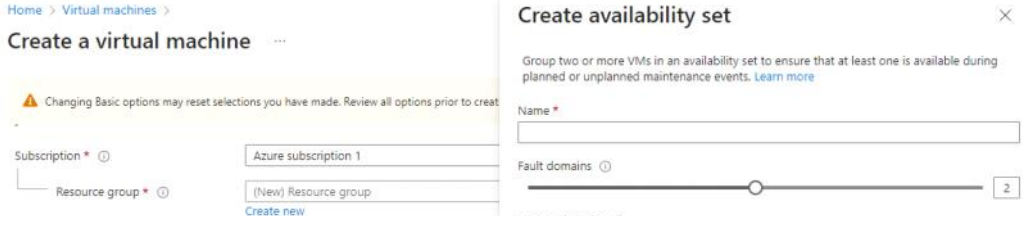
- Premium SSD or Ultra Disk: 99.9%
- Standard SSD Managed Disks: 99.5%
- Standard HDD Managed Disks: 95%

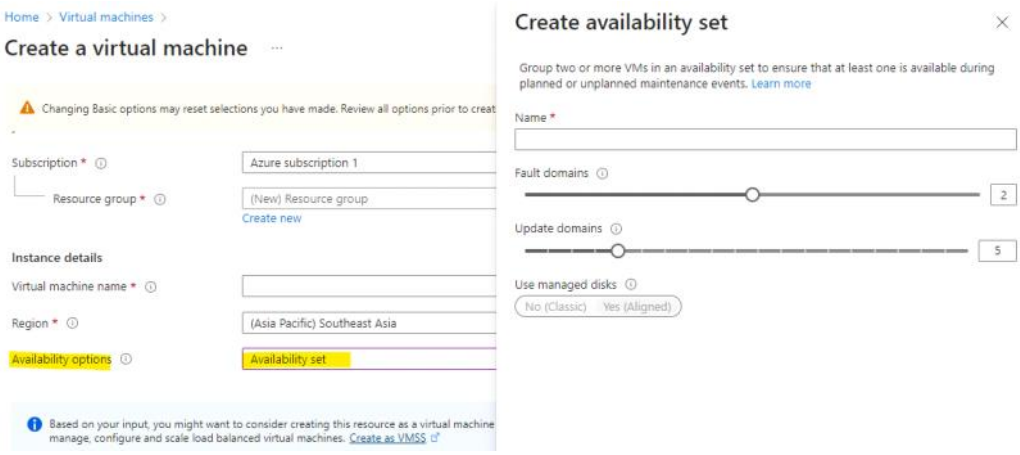
Create VM -> Disks



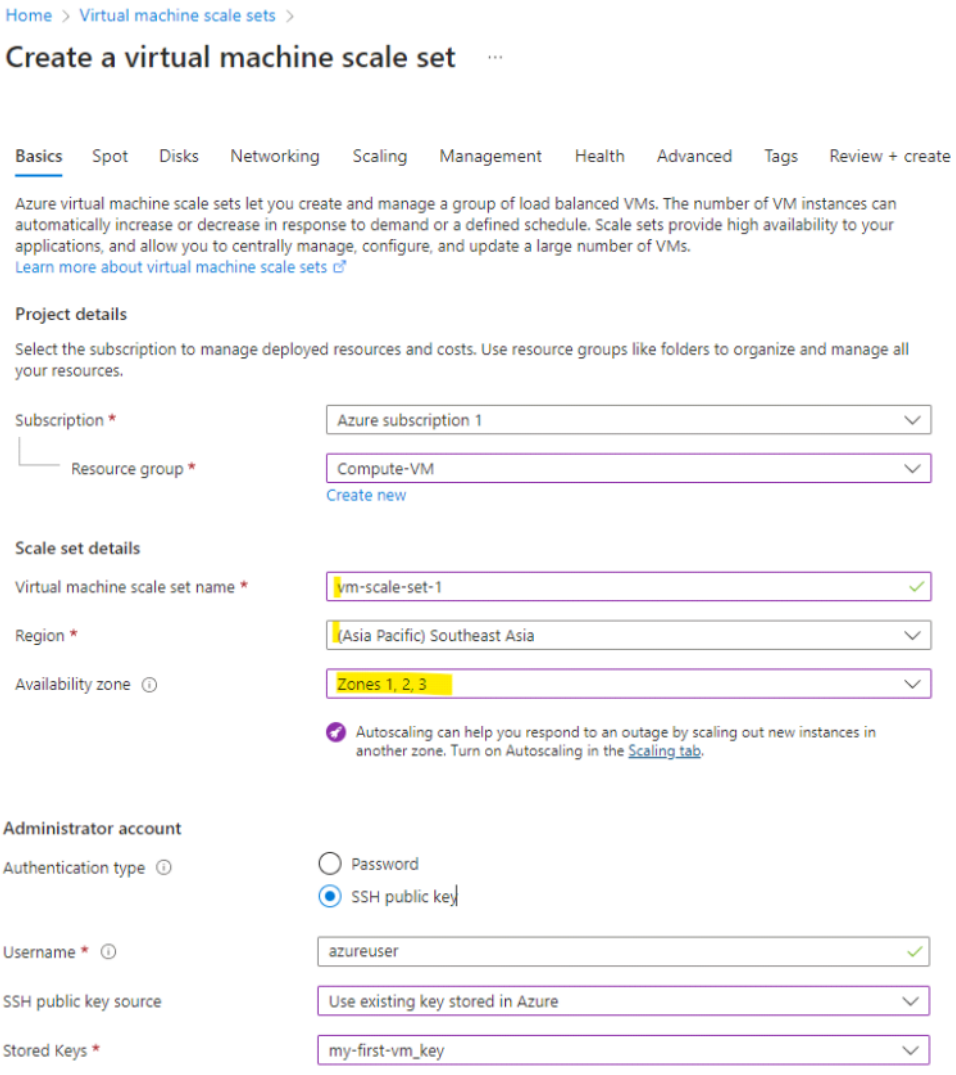
- Availability Sets
  - **Availability set is a logical grouping of VMs**
  - Availability Sets takes the virtual machine and configures multiple copies of it
  - Each copy is isolated within a separate physical server, compute rack, storage units and network switches within a single datacentre within an Azure Region
  - Availability Sets IMP Concepts
    - **Fault domains:** Group of VMs sharing a common power source and network switch, A fault domain represents the physical separation of resources within an Azure data center to protect against hardware failures or infrastructure issues.
      (Problem with having only 1 fault domain:having only 1 fault domain means that if that domain experiences a hardware failure or issue, all your instances within the availability set could be affected simultaneously.)
    - **Update domains:** Group of VMs that are rebooted (updated) at the same time, An update domain is used to manage planned maintenance events, such as host OS updates or hardware maintenance.
      (Problem with having only 1 update domain:having only 1 update domain means that all instances in the availability set would potentially be taken offline simultaneously during maintenance operations)
  - **Further increase the availability for your Azure VM:** Create Two ormore instances in same Availability Set
  - The availability increases from 99.9% to 99.95%
  - **Summary:** Create multiple instances in multiple AZs if you want high availability

    Availability Sets Summary
    - Single instance VMs:
      Premium SSD or Ultra Disk: 99.9%
      Standard SSD Managed Disks: 99.5%
      Standard HDD Managed Disks: 95%
    - Create Two or more instances in same Availability Set: 99.95%
    - **Two or more instances** in **two or more Availability Zones** in the same Azure region: 99.99%
    - **Summary: Create multiple instances in multiple AZs if you want high availability**

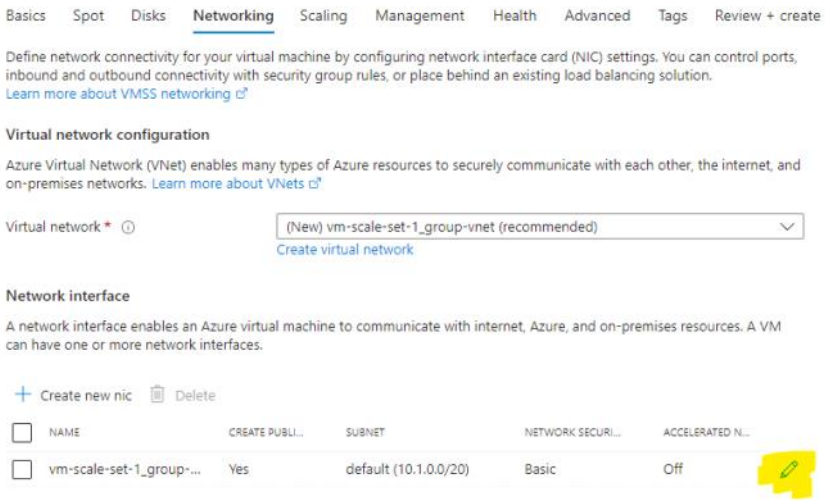




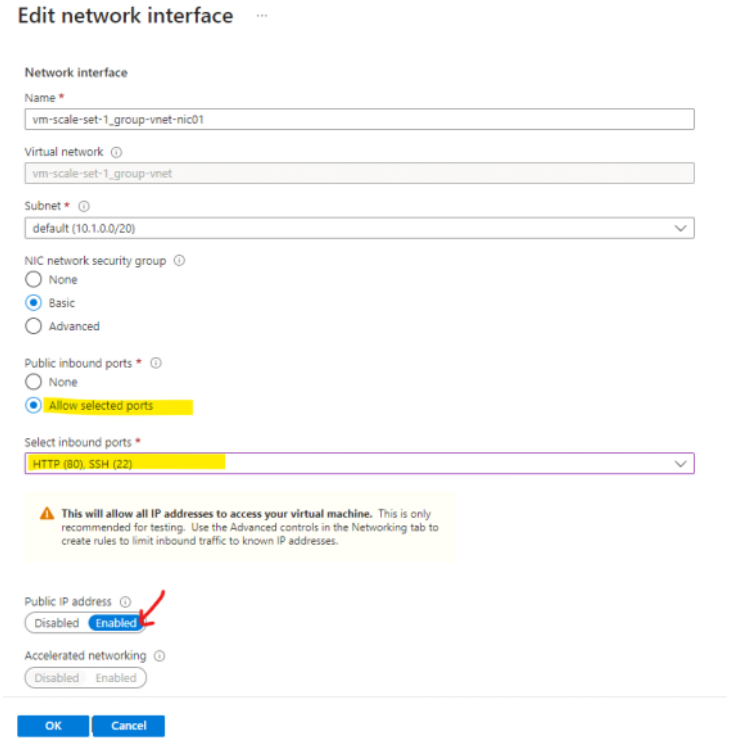
- Virtual Machine Scale Sets



-> Networking -> Edit

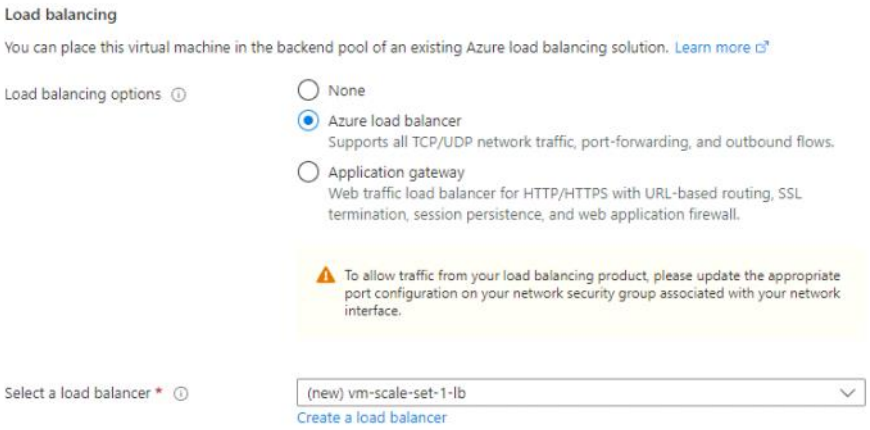


- Change private IP address to public IP address
- Select inbound ports to HTTP and SSH

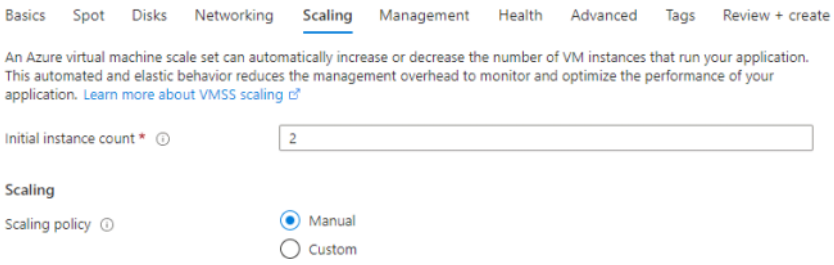


Create a load balancer

*a **Load Balancer** is a network-based service that distributes incoming network traffic across multiple virtual machines (VMs) or services to achieve higher availability, scalability, and performance. It acts as a traffic manager, distributing requests evenly across the backend resources, such as VMs, virtual machine scale sets, or availability sets.*

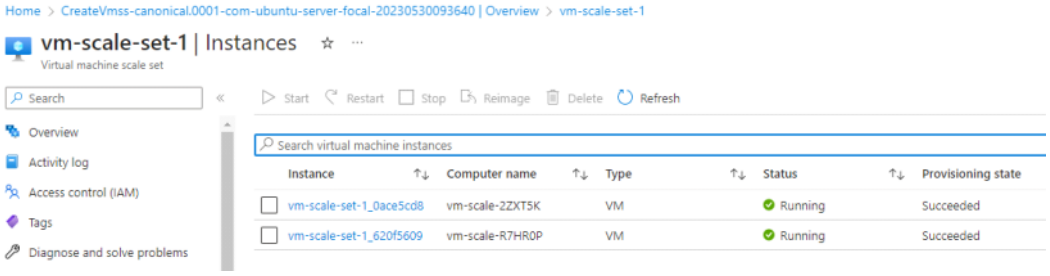


-> Scaling

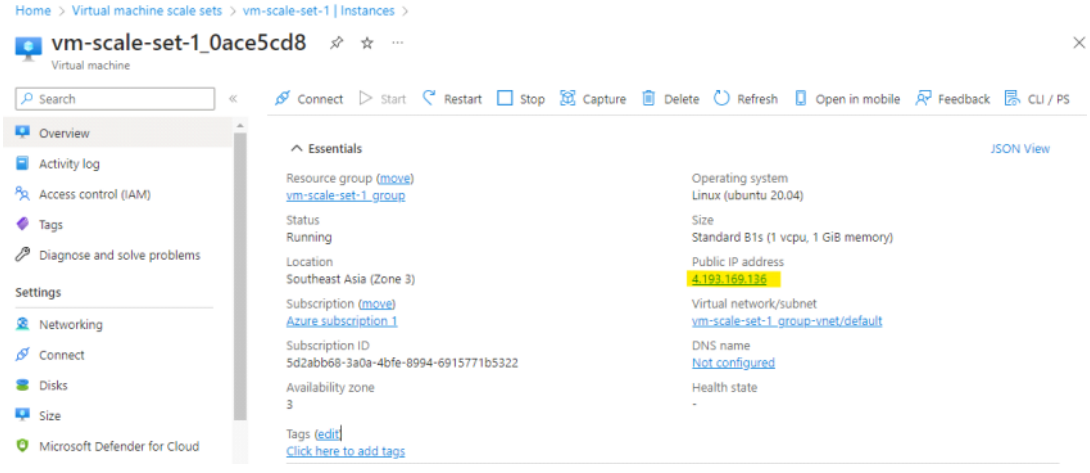


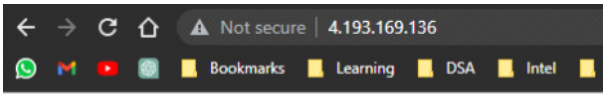
- Scaling and Load Balancing with VM Scale Sets

2 instances (default) have been created



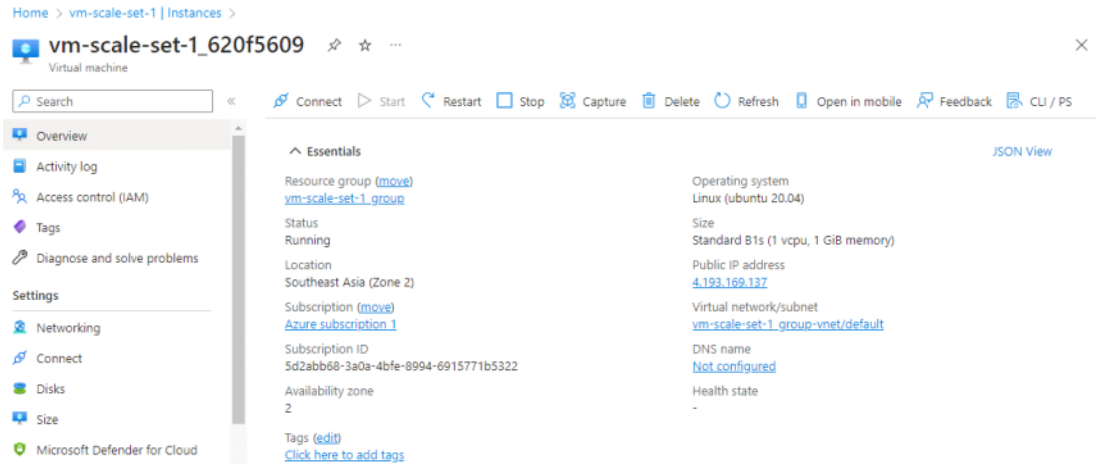
The 1st instance:

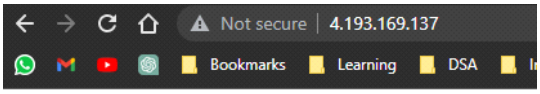




Welcome to Azure Cloud vm-scale-2ZXT5K

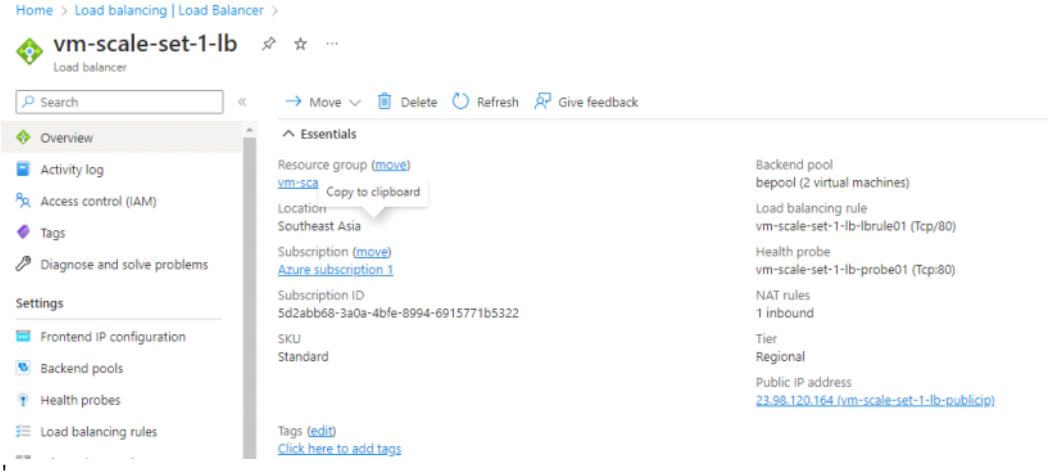
The 2nd instance





Welcome to Azure Cloud vm-scale-R7HR0P

Load Balancer has been created



Curl <IP address>

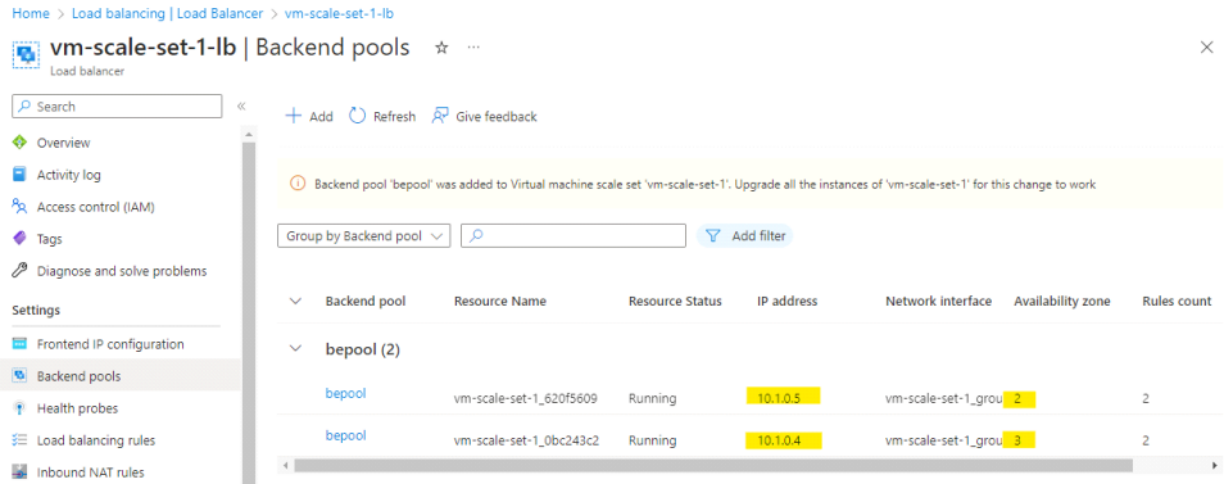
```
cheah [ ~ ]$ curl 23.98.120.164
Welcome to Azure Cloud vm-scale-R7HR0P
cheah [ ~ ]$ curl 23.98.120.164
Welcome to Azure Cloud vm-scale-2ZXT5K
```

Load balancing is happening. There is a load balancer that distributes incoming network traffic across multiple virtual machines (VMs) or services

This means if there is only 1 instance, we will only receive a response from 1 VM instance.

To create new instances, go to Virtual Machine scale sets -> Scaling, increase the initial instance count

>Load Balancer<

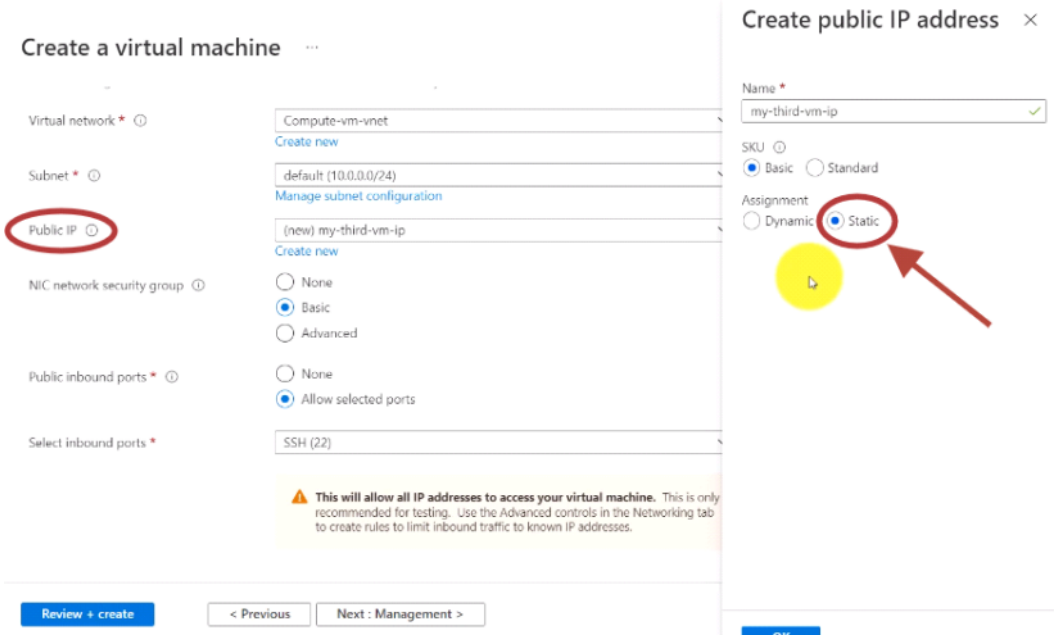


Summary

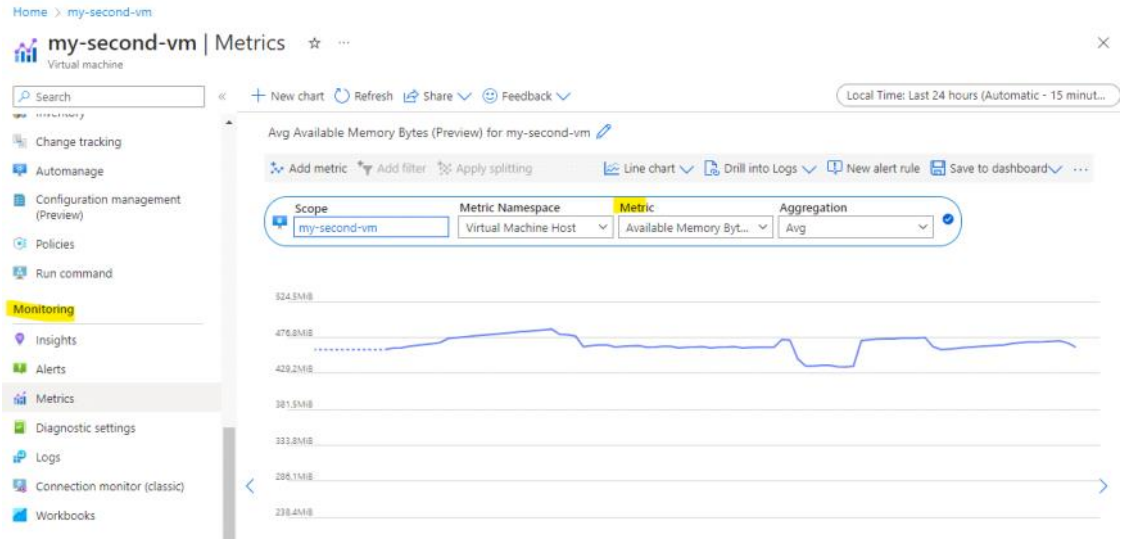
- VM Scale sets – connected with each instance
- Load balancer - round robinin
- Delete one instance from scale set and try round robin under load balancers
- Increase manual scale under scale set and round robin under load balancers
- Load balancer - Check Frontend ip configuration and backend pools

➢ Static IP, Monitoring, Dedicated Host and Reducing the Cost

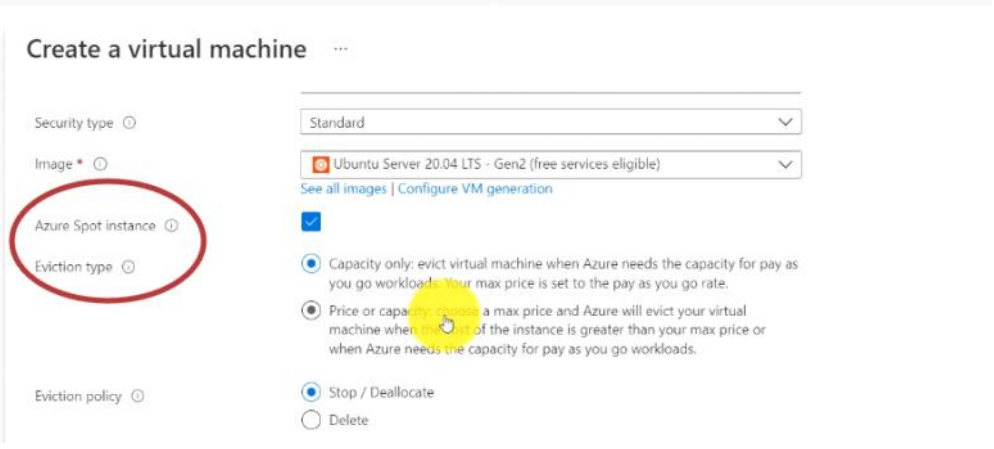
Static IP:



Monitoring:

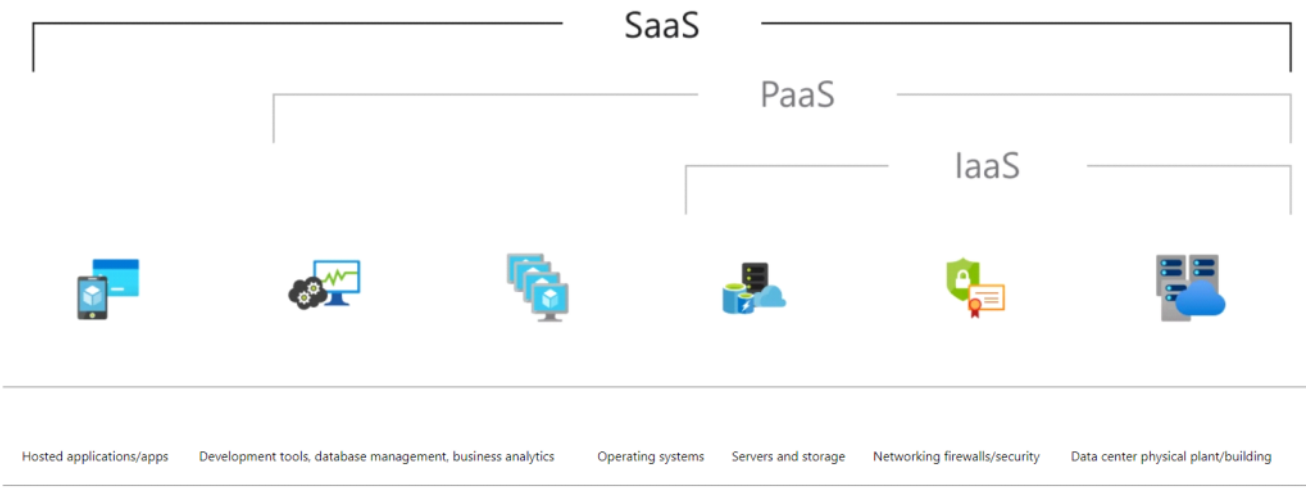


Spot Instance:

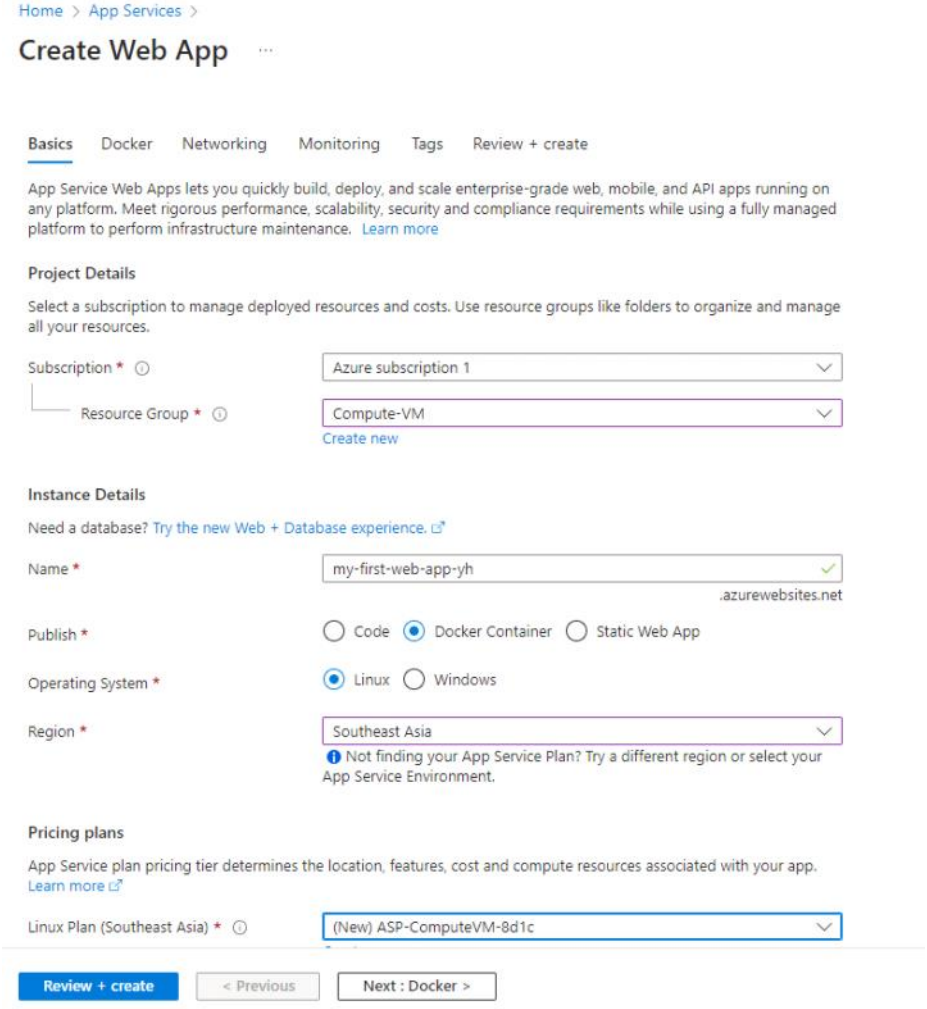


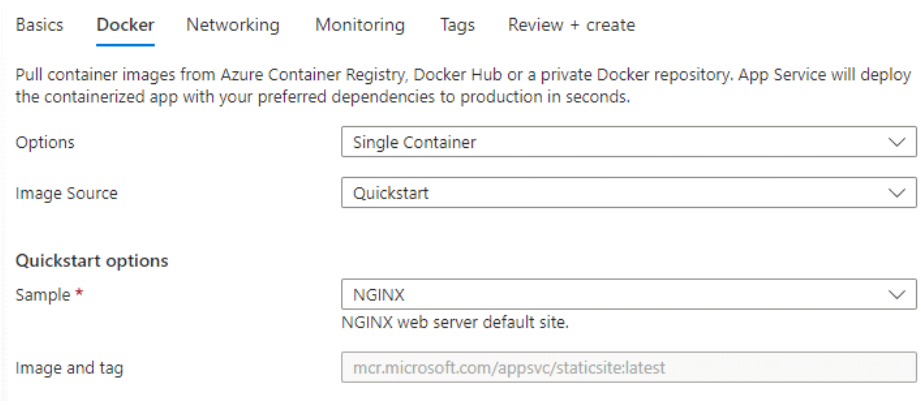
Section 15: Azure Fundamentals: Managed Compute Services

Monday, 5 June, 2023 9:32 AM

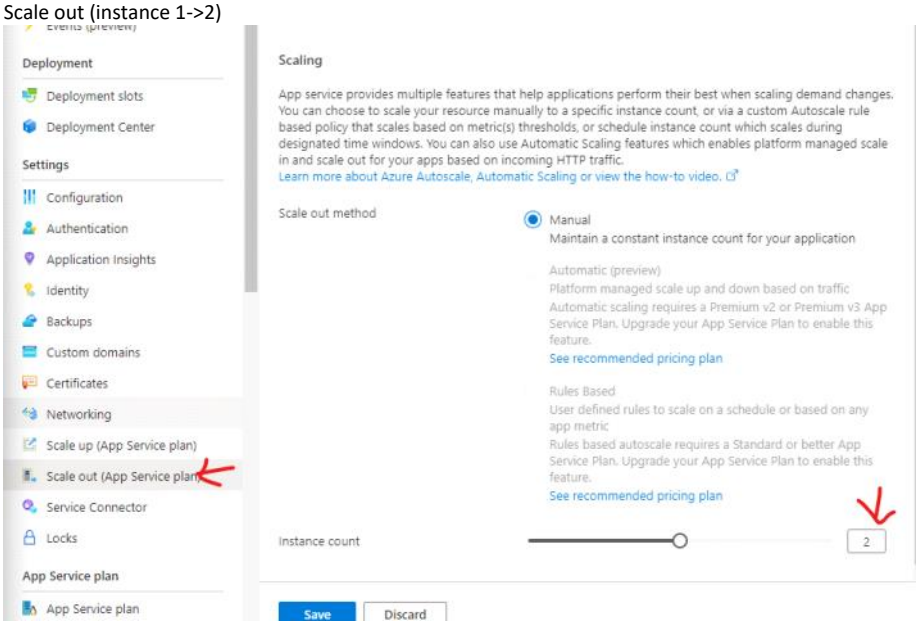


- Creating First Web App using Azure App Service

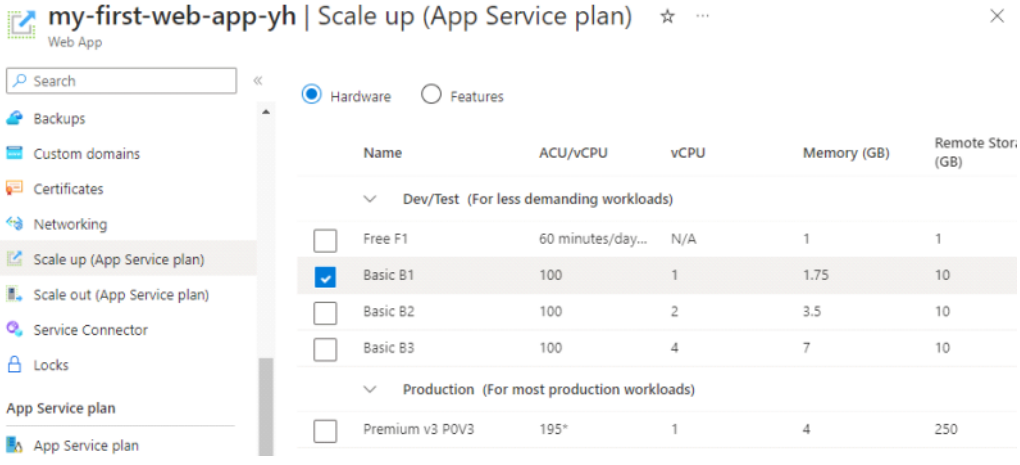




- Scale out (instance 1->2)

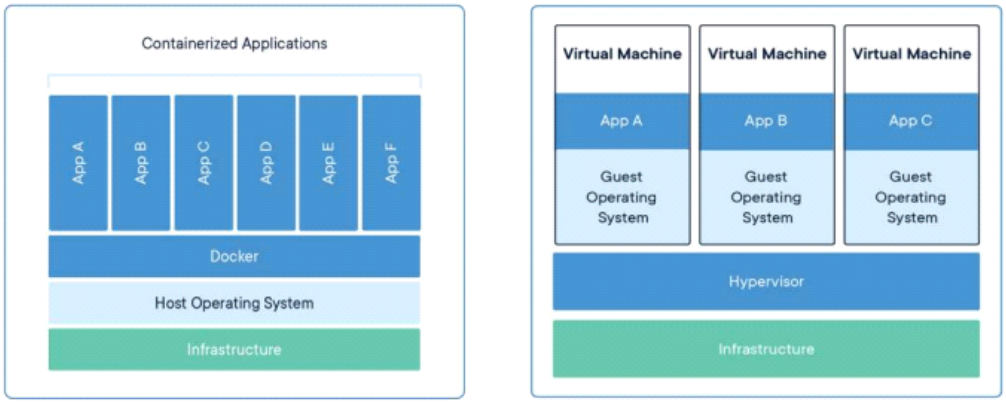


- Scale up (scale up if necessary)

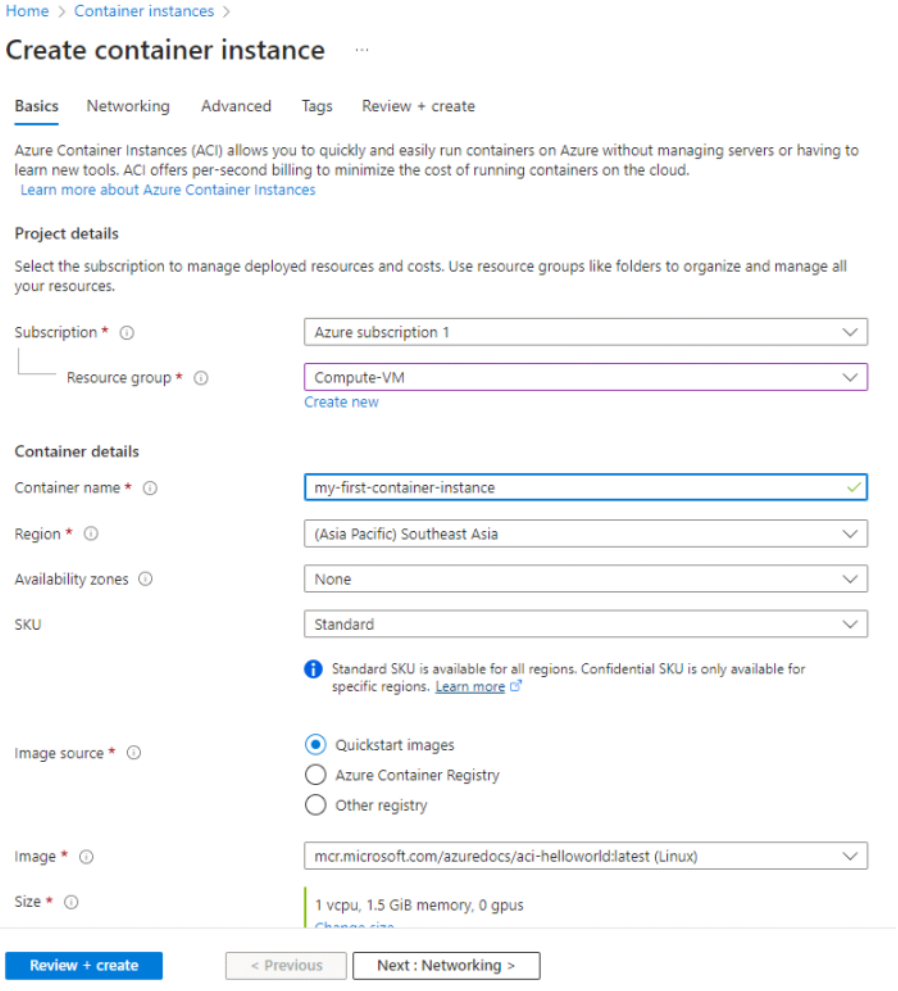


➢ Introduction to Containers

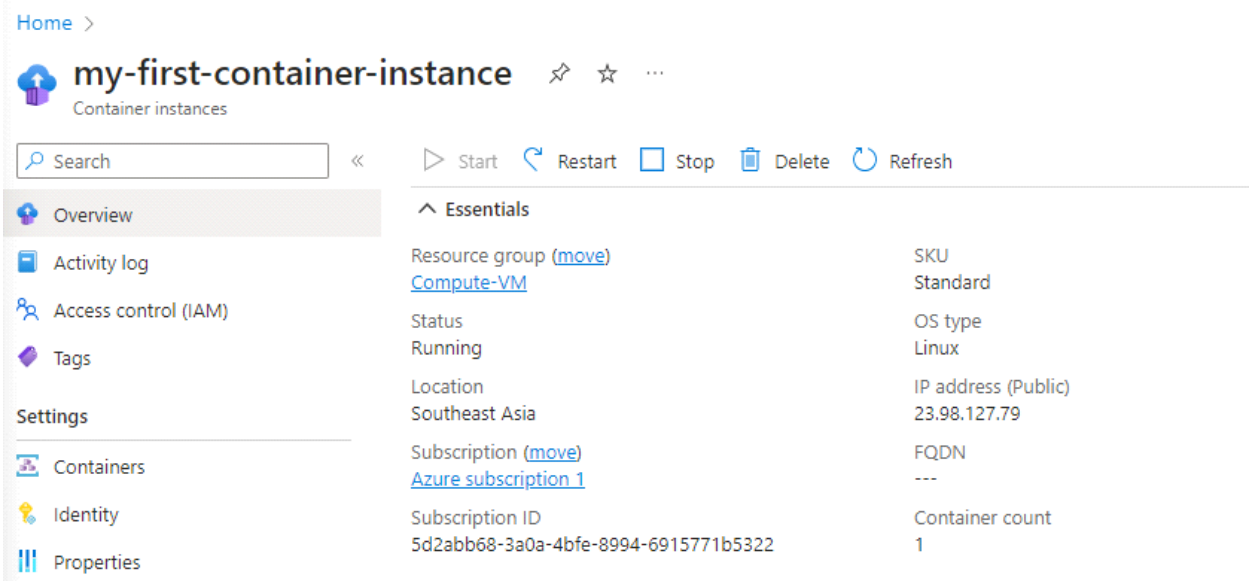
Containers vs Virtual Machines



- Introduction to Azure Container Instances

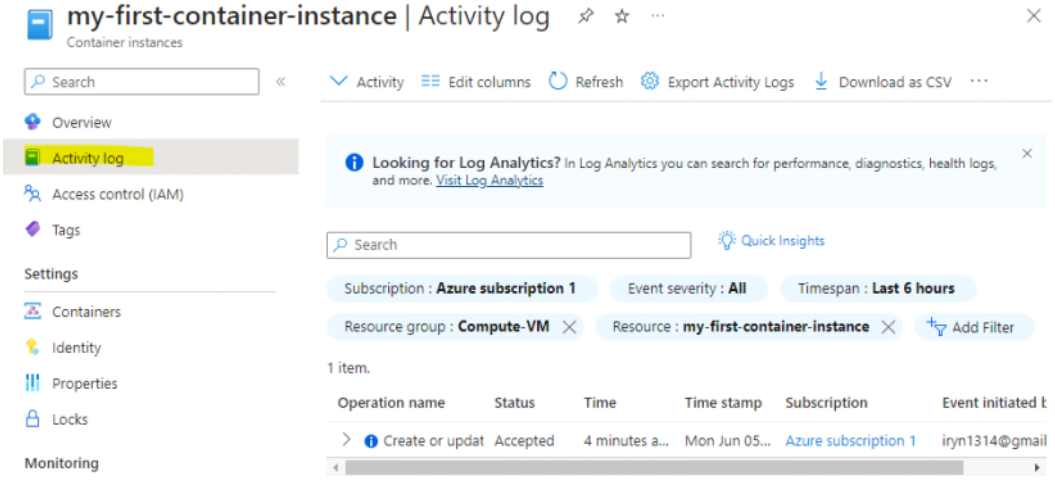


  - Overview of the resources

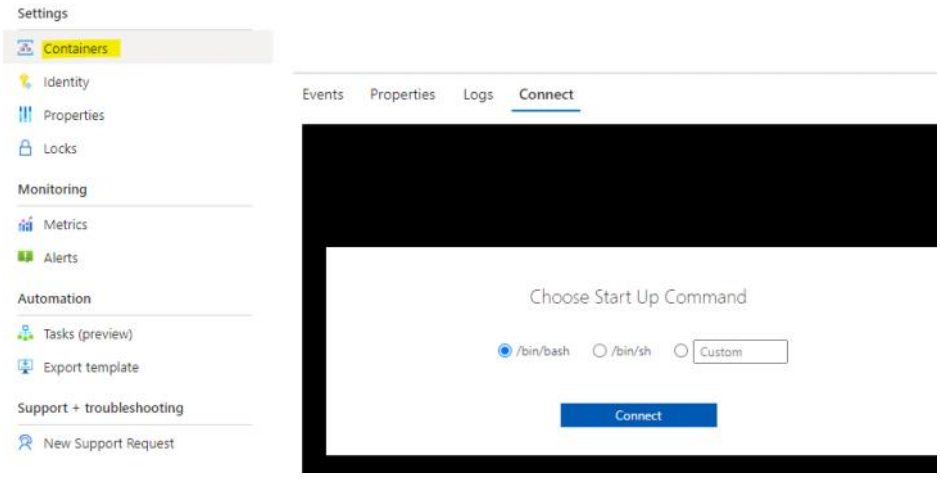


- Copy IP address and run container with IP address

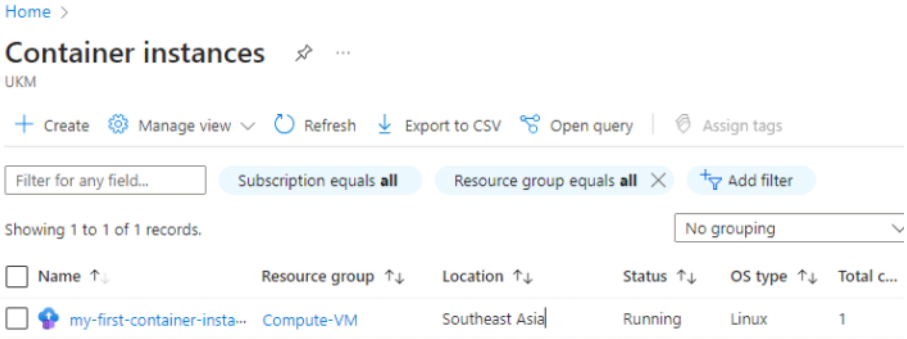
- Activity log



- Containers Setting



- The list of container instances



➢ Container Orchestration - AKS and Service Fabric

Container Orchestration

- You have build container images for your microservices and you are able to run these containers with microservices
- You build containers for Microservice A, B, C, D and E. now you want to manage the deployment of these containers, you have to ensure the auto-scaling of the number of container instances that are running based on the load
- **Kubernetes:** It is one of the most popular open-source container orchestrator solutions

The container orchestrators features:
Auto Scaling, Service Discovery, Self healing, Zero Downtime Deployments

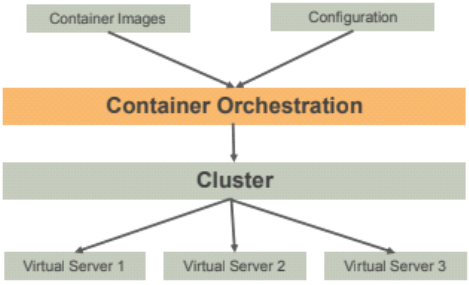
➢ **Container orchestration services in Azure:** AKS and Service Fabric

1) **AKS:** Azure Kubernetes Service - Managed Kubernetes Service

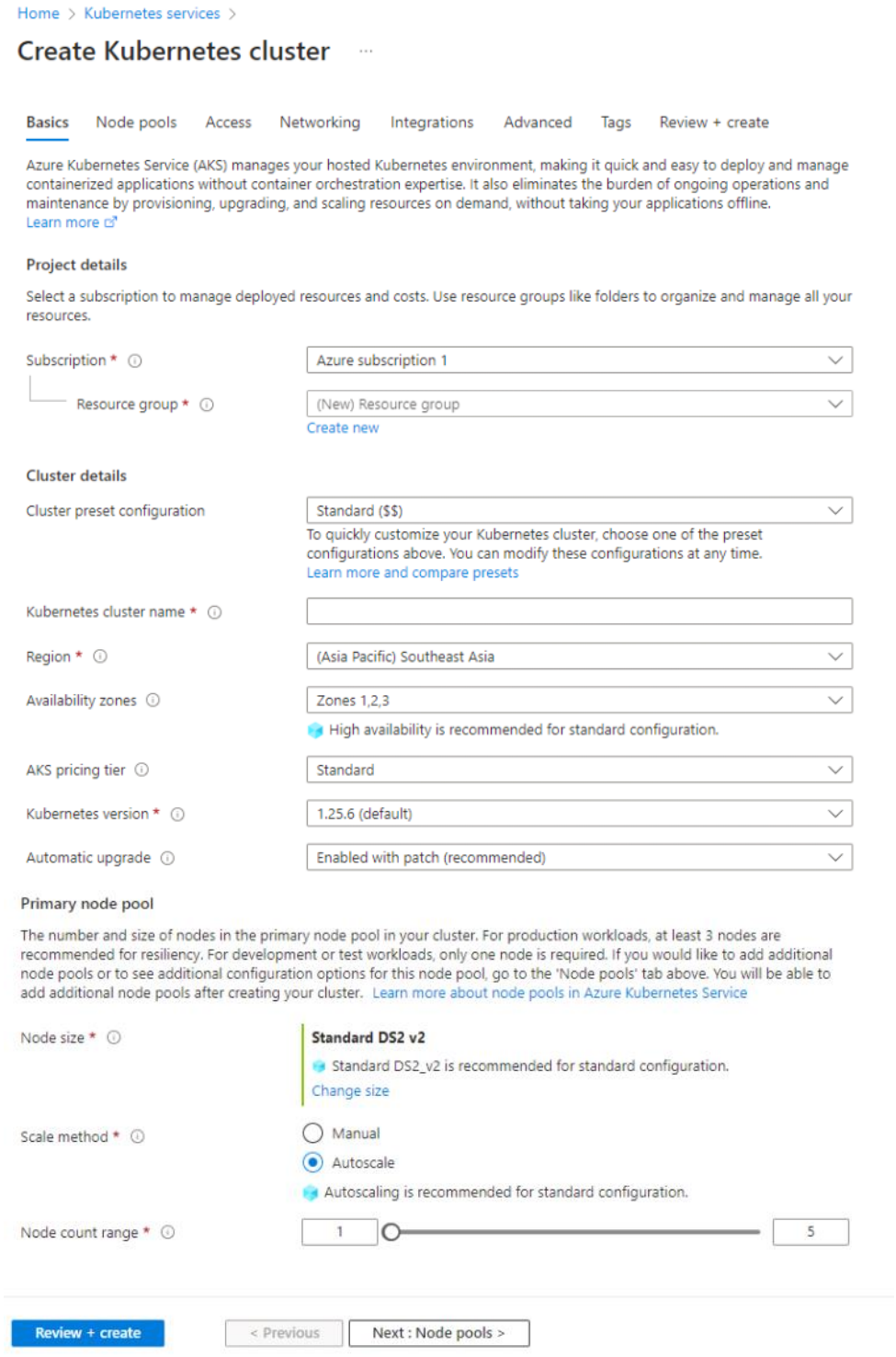
2) **Service Fabric:** Microsoft's Container Orchestrator

Using a Container Orchestrator:

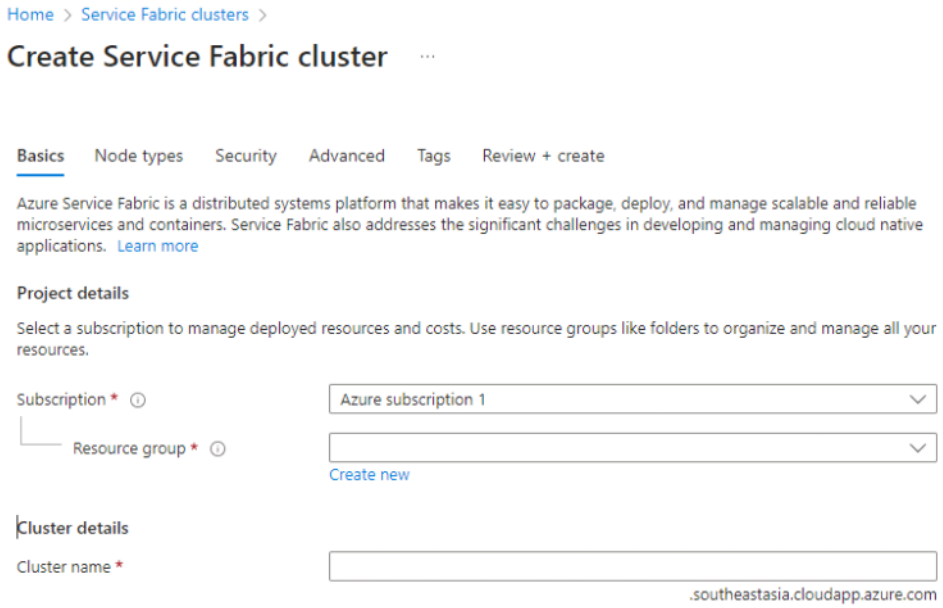
a. Create a cluster with the number of nodes (virtual servers)
b. Deploy and Orchestrate the Microservice



Create Kubernetes cluster



Create Service Fabric cluster

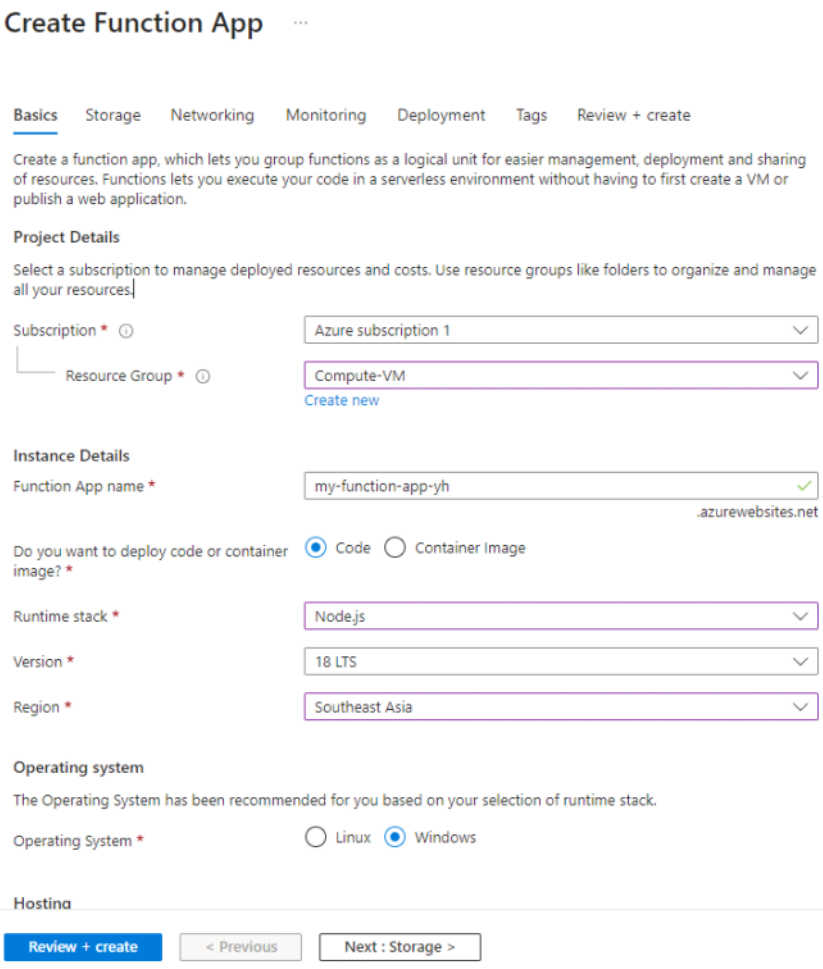


- Azure Serverless Service - Azure Functions

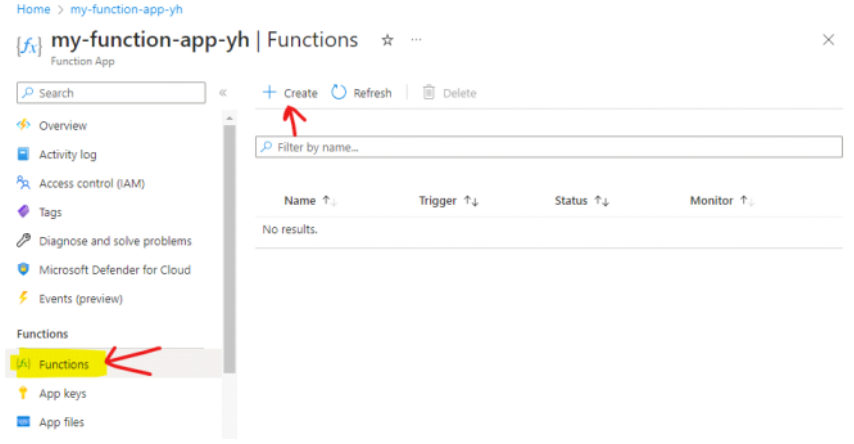
Serverless

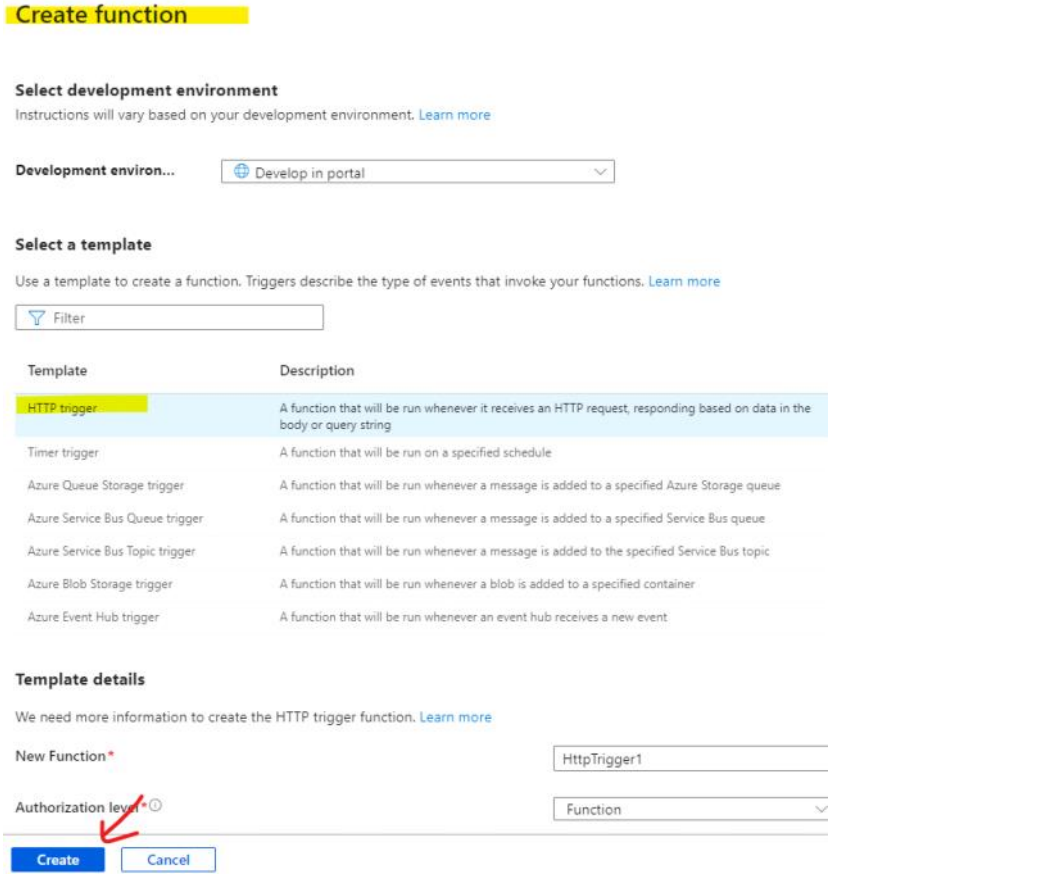
- Serverless computing **enables developers to build applications faster by eliminating the need to manage infrastructure**
- Cloud service provider automatically provisions, scales and manages the infrastructure required to run the code
- **Serverless does not mean "No Servers"**: The tasks associated with infrastructure provisioning and management are invisible to the developer
- This approach enables developers to increase their focus on business logic and deliver more value to core of business
- **Pay for use**: no requests – no costs (Pay for requests, not servers)
- Increases productivity, bring products to market faster, optimise resources and you can focus on innovation

Create Function App

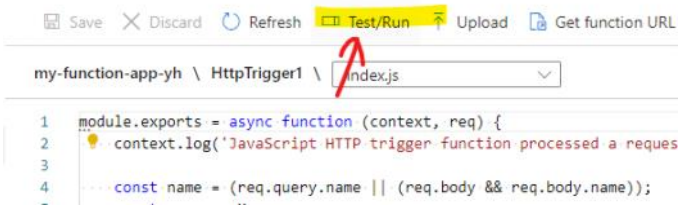


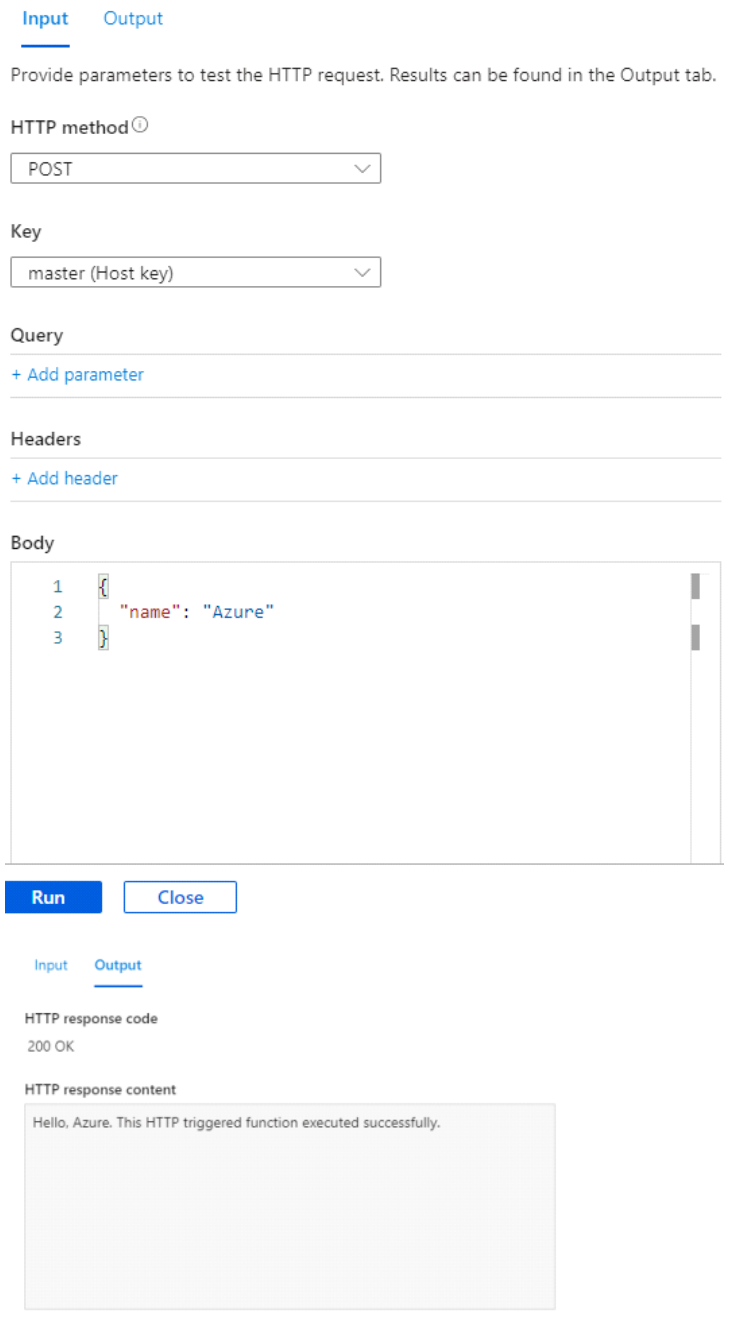
- To write my own functions



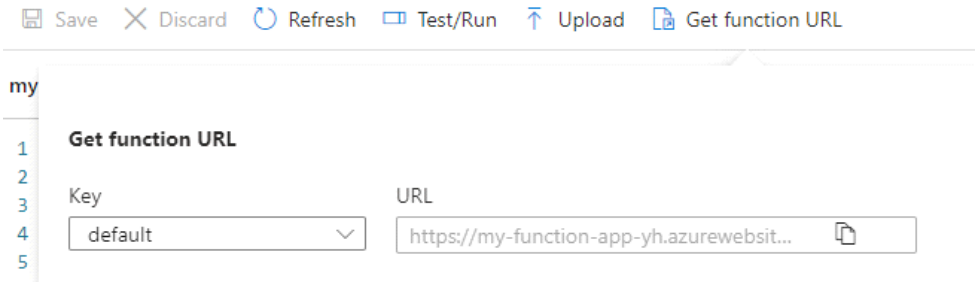


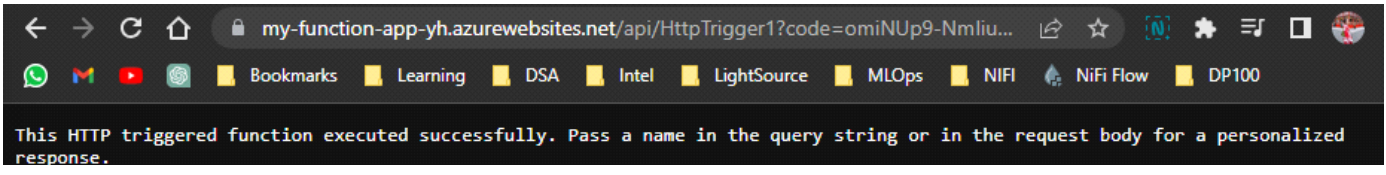
Click on Test/Run





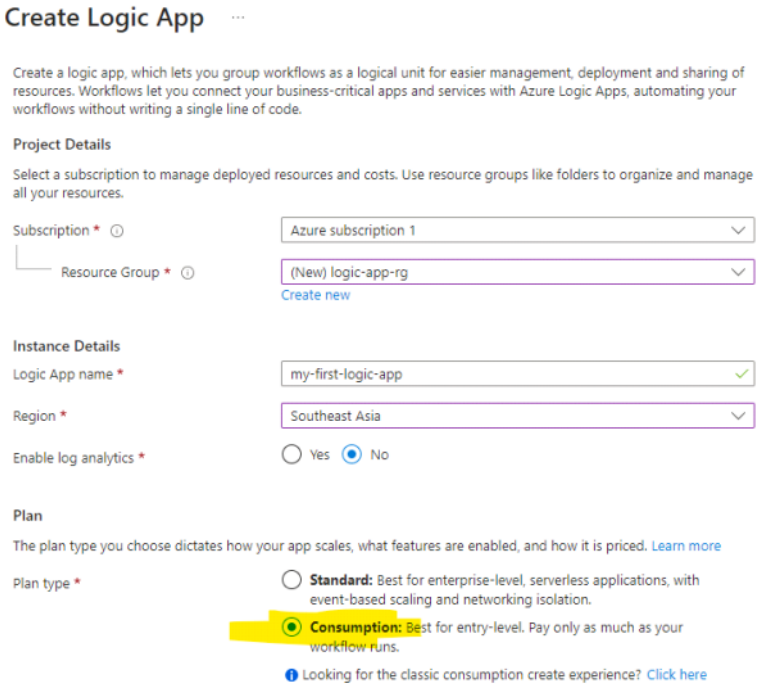
Or Test/Run by **Get Function URL**

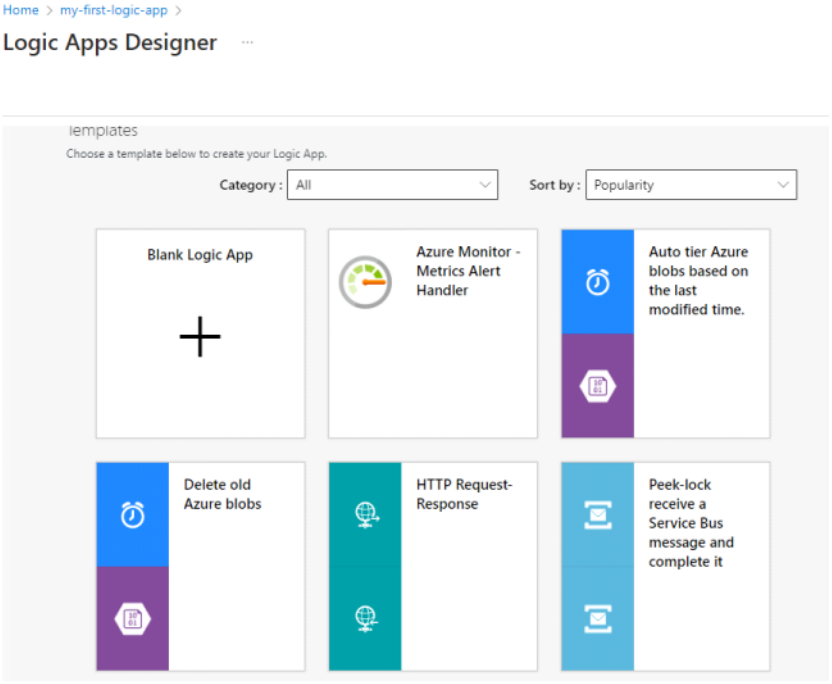




- Logic Apps

Create Logic App





Section 16: Azure Fundamentals: Storage

Monday, 5 June, 2023 8:19 PM

Azure Storage

The Azure Storage platform includes the following data services:

- Azure Blobs: A massively scalable object store for text and binary data. Also includes support for big data analytics through Data Lake Storage Gen2
- Azure Files: Managed file shares for cloud or on-premises deployments
- Azure Queues: A messaging store for reliable messaging between application components
- Azure Tables: A NoSQL store for schemaless storage of structured data
- Azure Disks: Block-level storage volumes for Azure VMs

Azure Storage link: https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction

Azure storage redundancy:
Locally redundant storage (LRS), Zone-redundant storage (ZRS), Geo-redundant storage (GRS), Geo-zone-redundant storage (GZRS)

Managed and Unmanaged Block Storage

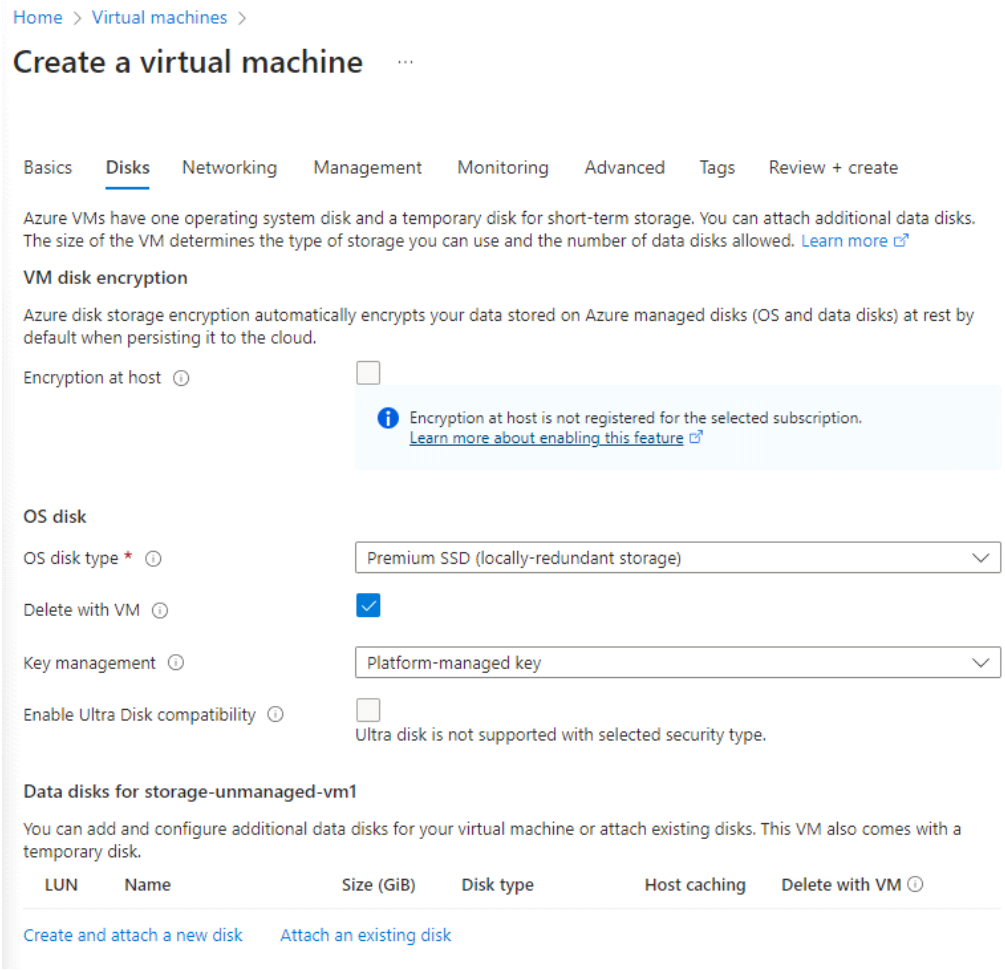
Block Storage

- Block Storage: Hard-disks attached to your computers
- Typically one block storage device can be connected to one virtual server
- You can connect multiple different block storage devices to one virtual server

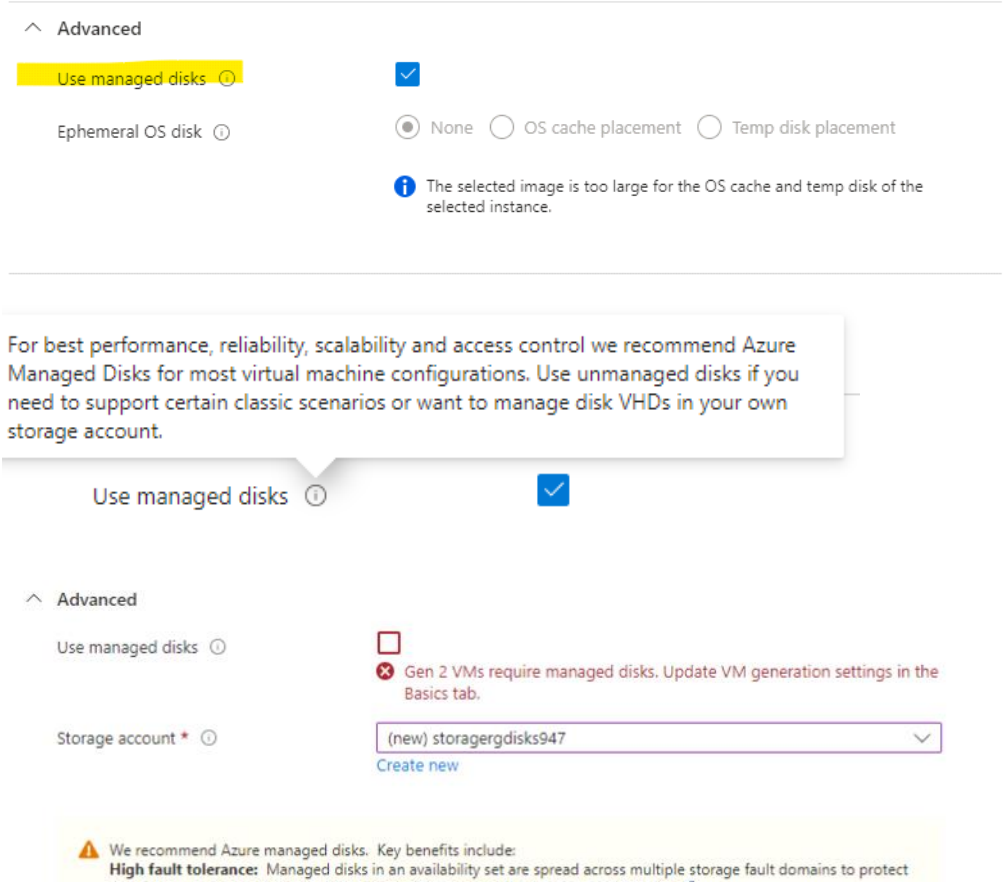
Azure Disk Storage

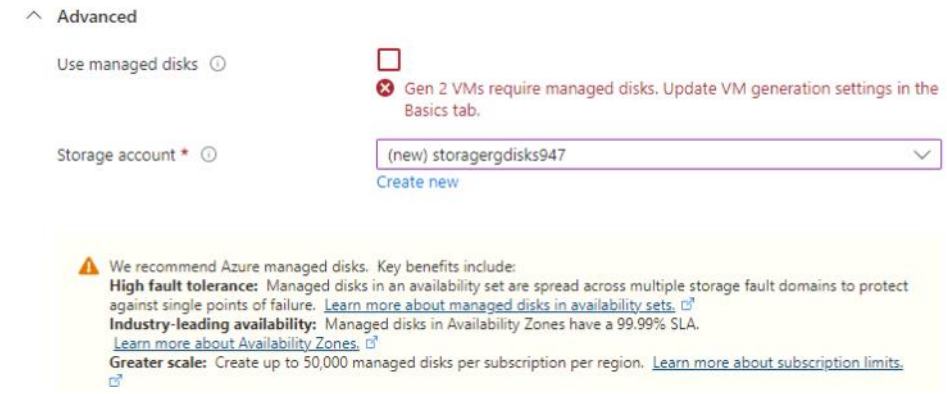
- To store block storage device in azure: Azure disk storage (Disks for Azure VMs)
- Different types of disks:
  i. Standard HDD: Recommended for Backup, non-critical, infrequent access
  ii. Standard SSD: Recommended for web servers, lightly used enterprise applications and dev/test environments
  iii. Premium SSD disks: Recommended for production and performance sensitive workloads
  iv. Ultra disks (SSD): Recommended for IO-intensive workloads such as SAP HANA, top tier databases (for example, SQL, Oracle), and other transaction-heavy workloads
- Premium and Ultra SSD provide very high availability

Managed Disks are easy to use
Unmanaged Disks are old and tricky (Avoid them if you can)

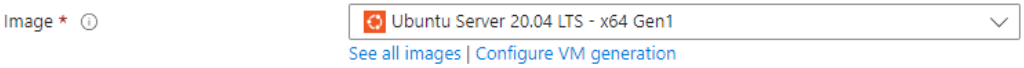


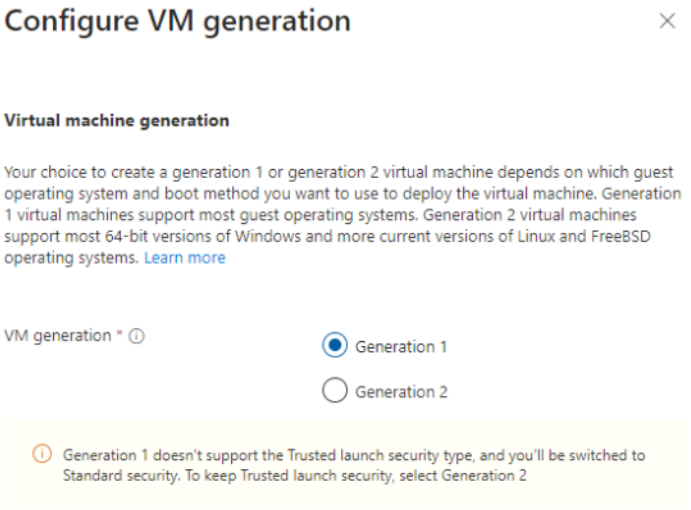
Managed/Un-managed disks:



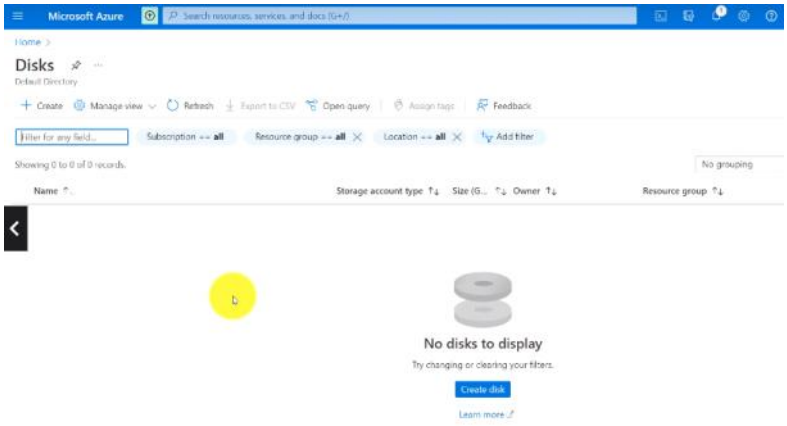


Gen 2 VM require managed disks, so switch to Gen 1 VM for Un-managed disks

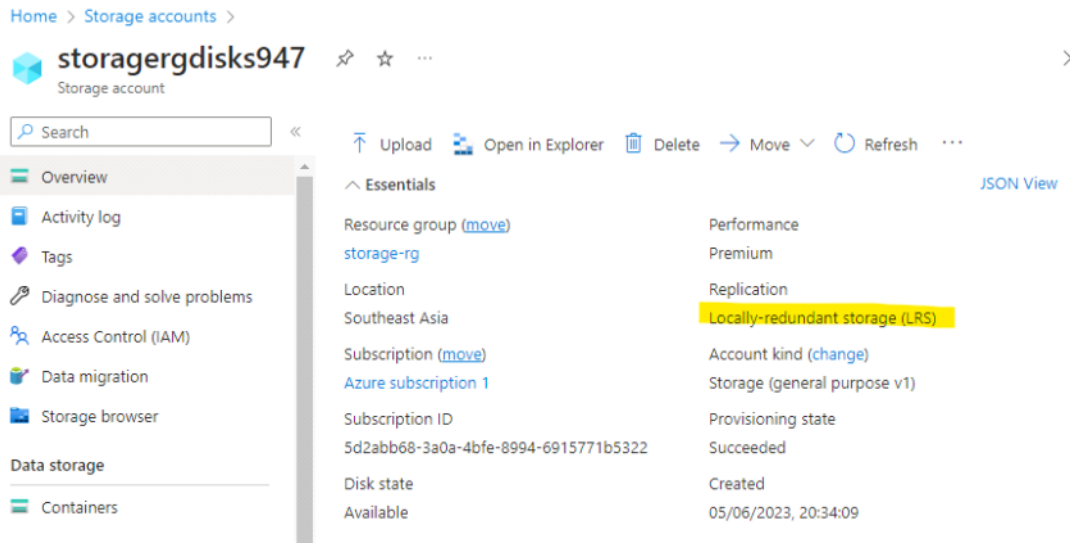




After deployment successful,
We are not able to see any disks available in ***Disks*** directory if we have created an un-managed disk



Its managed by **Azure Storage accounts** *to store the disk data*

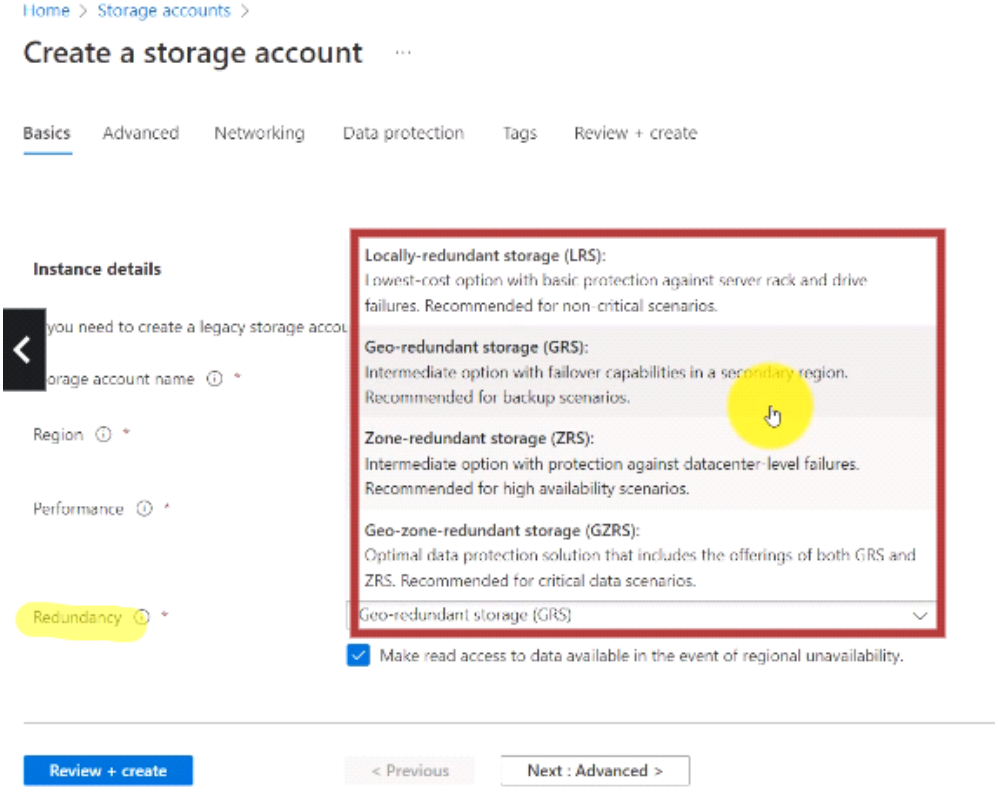


Azure Files

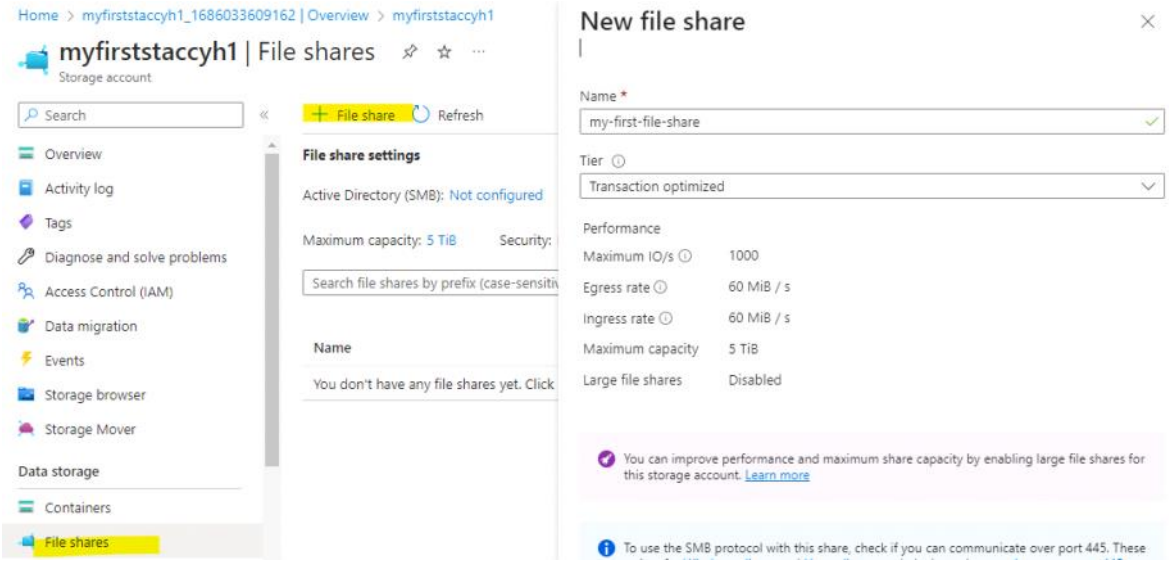
https://learn.microsoft.com/en-us/azure/storage/files/storage-files-introduction

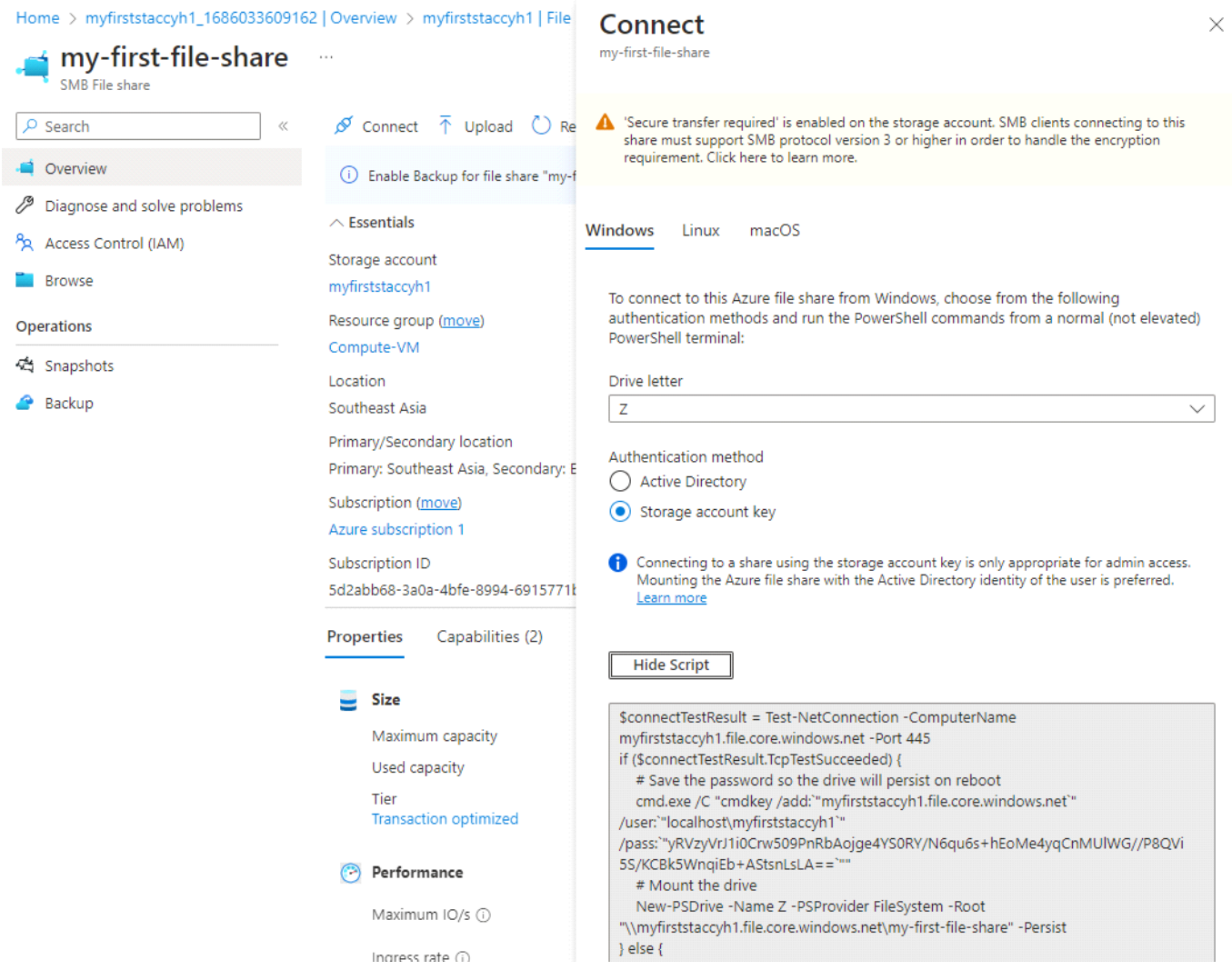
In Azure, a file share is a storage resource that allows you to store and share files in the cloud. It provides a centralized location for multiple users or applications to access and collaborate on files, similar to a network file share on a local server.

Create Storage Account



Create File Share





Azure Blob Storage and Tiers

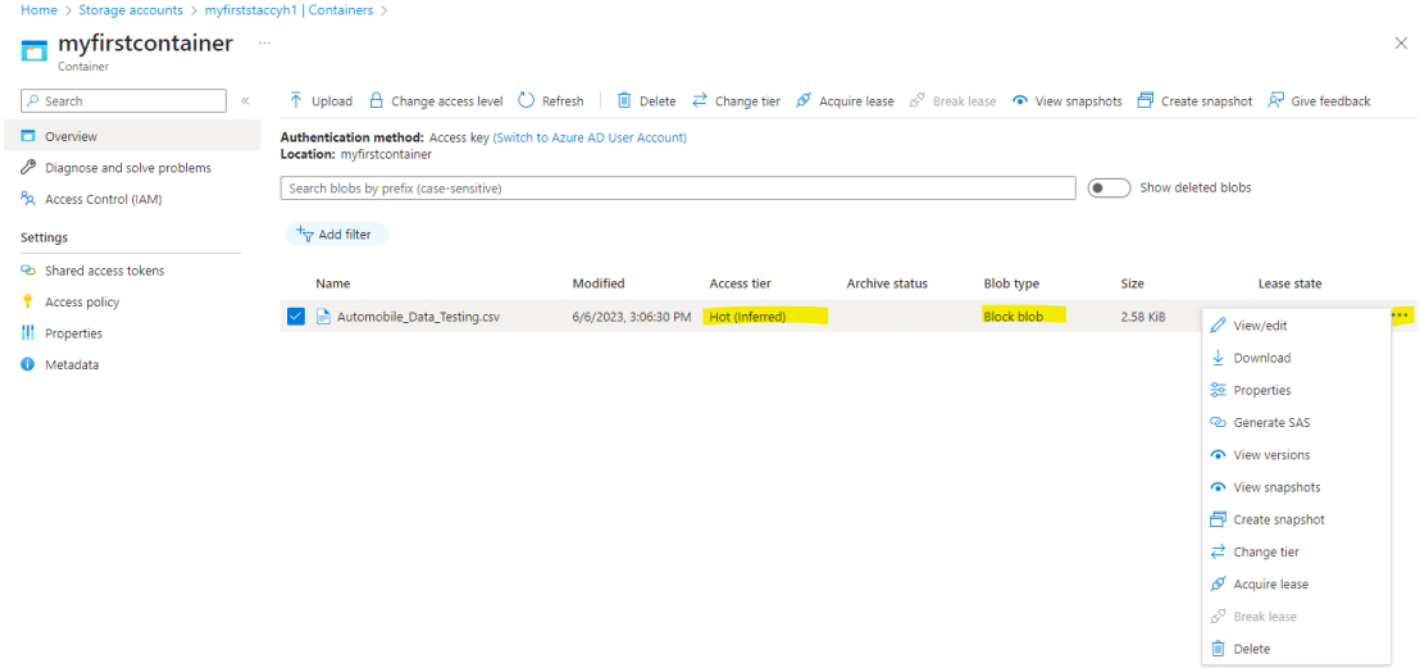
Azure Storage supports three types of blobs:

a. Block blobs: Store text and binary data (videos, archives, etc)
b. Append blobs: Made up of blocks like block blobs, optimized for append operations. Ideal for scenarios such as logging data from virtual machines
c. Page blobs: Store random access files up to 8 TiB in size. Store virtual hard drive (VHD) files and serve as disks for Azure virtual machines

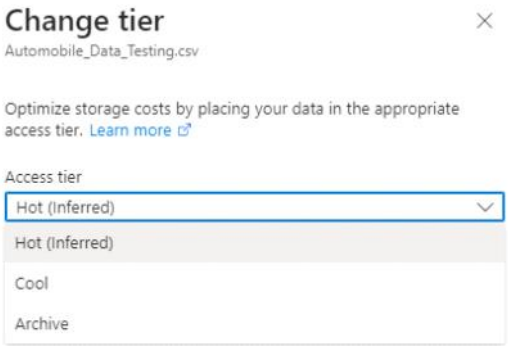
Azure Blob Storage Access Tiers

a. Hot tier: Data accessed or modified frequently. Highest storage costs, but lowest access costs
b. Cool tier: Infrequently accessed data, stored for min for 30 days. Lower storage costs and higher access costs
c. Archive tier: Rarely accessed data, stored for min180 days, access latency in hrs, Lowest storage cost but Highest access cost
d. To access: Rehydrate (Change access tier to hot or cool) or Copy to another blob with access tier hot or cool

Create Storage Account -> Create new container -> Upload files

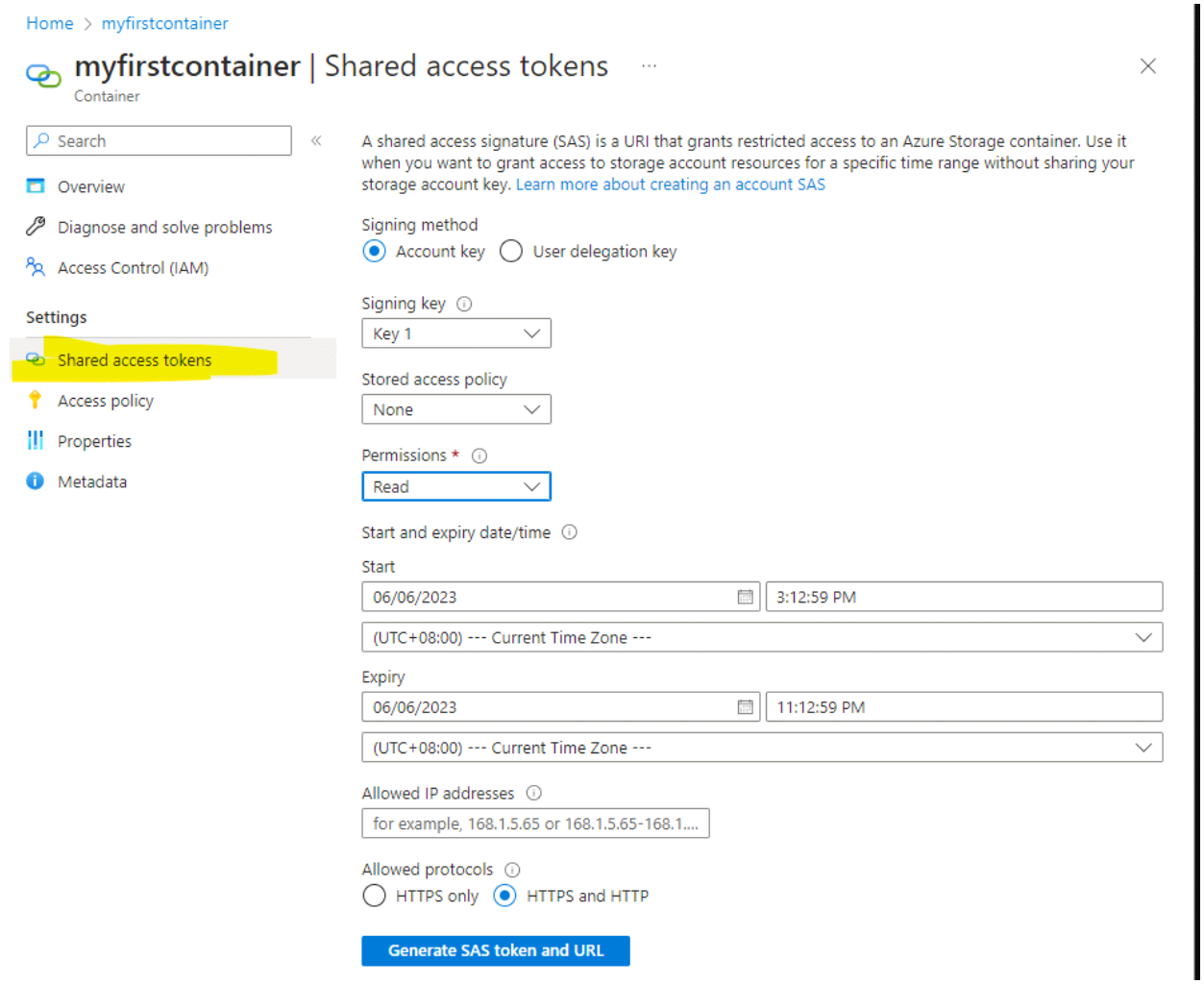


Change tier

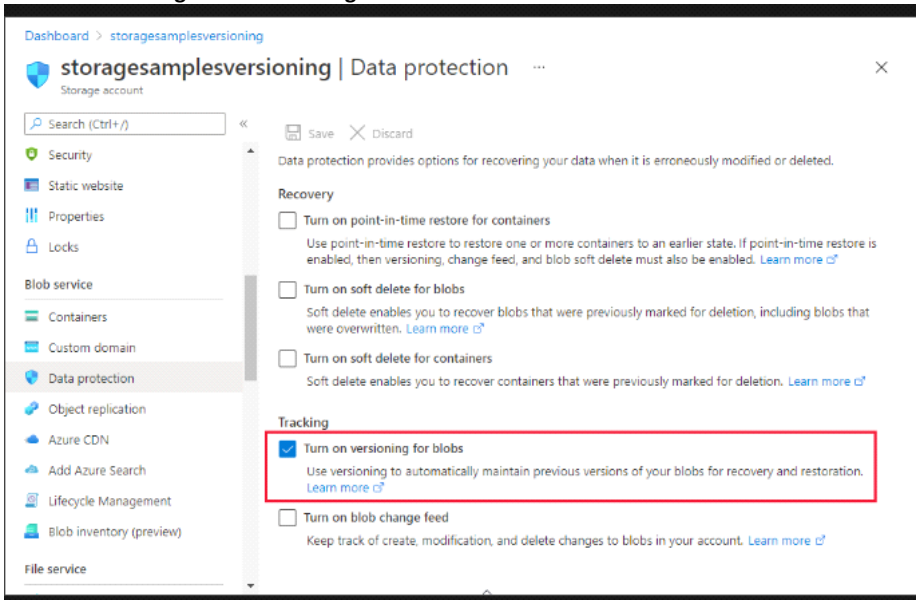


Shared access tokens

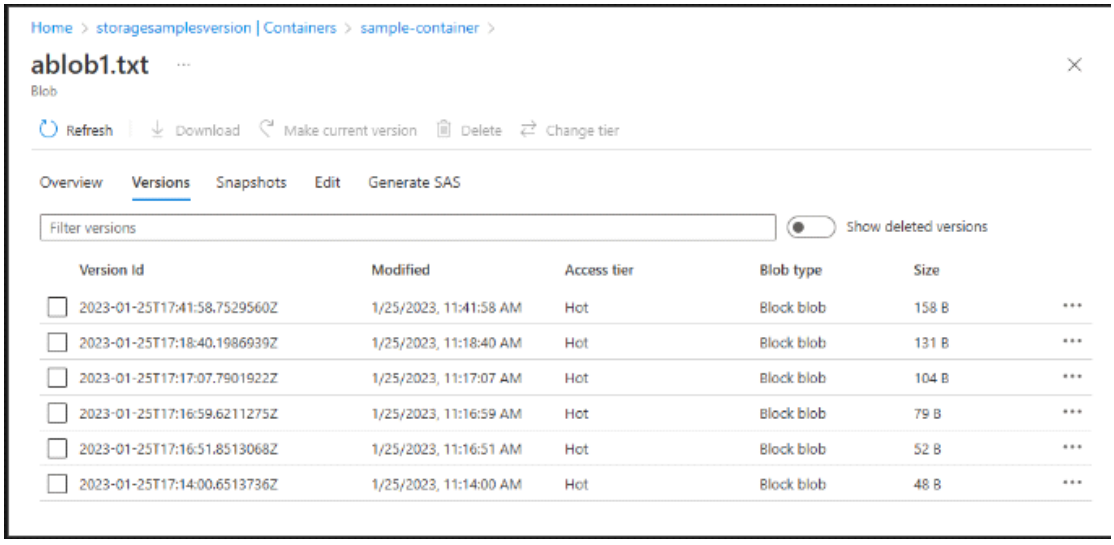
Shared Access Tokens in Azure act as temporary access keys that grant controlled access to specific resources, like containers or files, without sharing the main storage account access key, ensuring fine-grained security and limited permissions.



Enable and manage blob versioning



i. Navigate to your storage account in the portal, then navigate to the container that contains your blob.
ii. Select the blob for which you want to list versions.
iii. Select the Versions tab to display the blob's versions.


Home > storagesamplesversion | Containers > sample-container >
ablob1.txt
Blob
Refresh
Download
Make current version
Delete
Change tier
Overview
Versions
Snapshots
Edit
Generate SAS
Filter versions
Show deleted versions
Version Id
Modified
Access tier
Blob type
Size
2023-01-25T17:41:58.7529560Z
1/25/2023, 11:41:58 AM
Hot
Block blob
158 B
2023-01-25T17:18:40.1986939Z
1/25/2023, 11:18:40 AM
Hot
Block blob
131 B
2023-01-25T17:17:07.7901922Z
1/25/2023, 11:17:07 AM
Hot
Block blob
104 B
2023-01-25T17:16:59.6211275Z
1/25/2023, 11:16:59 AM
Hot
Block blob
79 B
2023-01-25T17:16:51.8513068Z
1/25/2023, 11:16:51 AM
Hot
Block blob
52 B
2023-01-25T17:14:00.6513736Z
1/25/2023, 11:14:00 AM
Hot
Block blob
48 B

Section 17: Azure Fundamentals: Databases

Wednesday, 7 June, 2023 9:50 AM

- **RTO and RPO**
  - The most important parameters of a disaster recovery or data protection plan
  - *Suppose a financial transaction is lost, or a trade transaction is lost or a banking website is down for an hour. The businesses are fine with some downtime but they can't offord data loss*
  - Availability and Durability are the technical measures, **How to measure how quickly we can recover from the failure?**
  - **RTO (Recovery Time Objective)**: Maximum acceptable downtime
    - The duration of time in which a business must restore normal system recovery to avoid consequences that can not only break business continuity but also cause financial loss
  - **RPO (Recovery Point Objective)**: Maximum acceptable period of data loss
    - The time that a disruption occurs before the maximum acceptable data loss that can be tolerated is reached
  - Achieving minimum RTO and RPO is expensive

- **Database Categories**

  Centralized, Distributed, Relational, NoSQL, Cloud, Object-oriented, Hierarchical, Network, Operational, Enterprise, etc

  How to Select a Database ?
  - Fixed schema or schemaless
    - a schema-based database has a predefined structure and enforces strict rules for data organization, while a schemaless database allows for more flexible and dynamic data structures
  - Level of transaction properties (atomicity and consistency)
    - Atomicity:
      Atomicity guarantees that the database remains consistent despite failures or interruptions during transaction processing. In other words, if any part of the transaction fails, the entire transaction is rolled back, and the database remains in its original state.
    - Consistency:
      Consistency ensures that a transaction brings the database from one consistent state to another consistent state. It defines a set of integrity rules or constraints that must be satisfied at all times.
  - Latency (seconds, milliseconds or microseconds)
  - Expected transactions (100, 1000 or a million transactions per second)
  - Amount of data to stored (MBs or GBs or TBs or PBs)

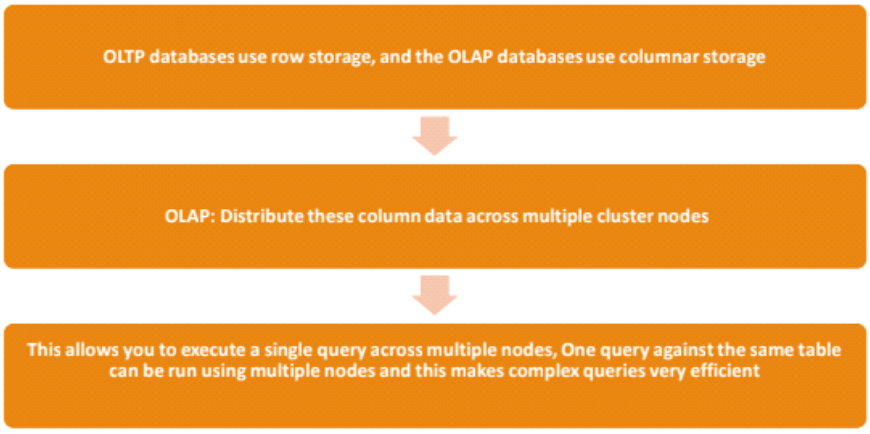
- **Relational Database**
  - a collection of items with pre-established relationships among them. All the items are organised in tables, where columns represent items attributes. Every row of a table represents a single data item.
    Examples of SQL database engine are: Mysql, PostgreSQL, Microsoft SQL Server, SQLlite
  - Up-front schema definition. No adaptation to changing requirements: dynamic changes to an item affect all the other items in the same table
  - Use of Structured Query Language (SQL)
    - Examples of SQL database engine are: Mysql, PostgreSQL, Microsoft SQL Server, SQLlite
  - **OLTP (Online Transaction Processing)**
    - Use cases: Most traditional applications (ERP, CRM, ecommerce, banking)
    - Recommended Azure Managed Services:
      1) Azure SQL Database: Managed Microsoft SQL Server
      2) Azure Database for MySQL: Managed MySQL
      3) Azure Database for PostgreSQL: Managed PostgreSQL

- **OLTP vs OLAP**
  - **OLTP (Online Transaction Processing):**
    - Focuses on day-to-day transactional operations, like processing orders and updating data in real-time.
    - Designed for fast and reliable handling of individual transactions.
    - Typically implemented using relational databases.
    - Emphasizes high concurrency, transaction management, and fast response times.
    - Optimized for capturing and processing real-time transactions.
  - **OLAP (Online Analytical Processing):**
    - Focuses on analyzing and gaining insights from the data generated by transactions.
    - Designed for complex queries, aggregations, and data analysis.

- Can be implemented using relational or non-relational databases.
- Enables slicing, dicing, and drilling down into data for business intelligence.
- Supports strategic decision-making and trend analysis.

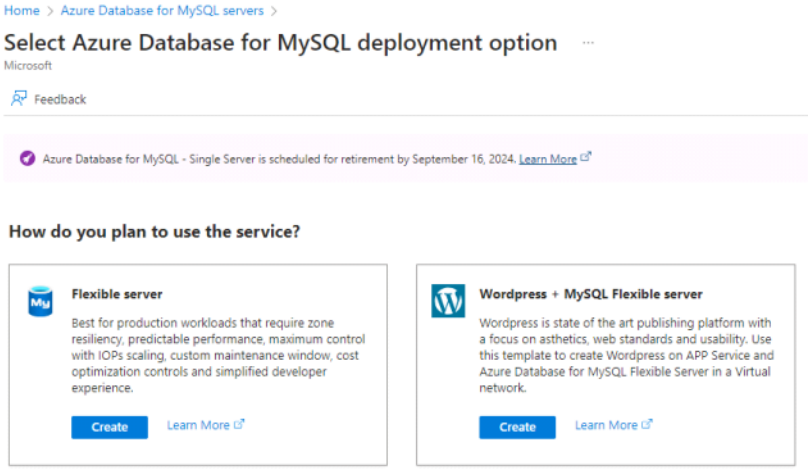
The use of the term "online" emphasizes the importance of immediate access and response in transaction processing and data analysis.

OLAP vs OLTP



	OLAP	OLTP
Characteristics	Handles a large number of small transactions	Handles large volumes of data with complex queries
Query types	Simple standardized queries	Complex queries
Response time	Milliseconds	Seconds, minutes, or hours depending on the amount of data to process
Design	Industry-specific, such as retail, manufacturing, or banking	Subject-specific, such as sales, inventory, or marketing
Data updates	Short, fast updates initiated by user	Data periodically refreshed with scheduled, long-running batch jobs
Productivity	Increases productivity of end users	Increases productivity of business managers, data analysts, and executives
User examples	Customer-facing personnel, clerks, online shoppers	Knowledge workers such as data analysts, business analysts, and executives

➢ Create Azure Database for MySQL servers



Home > Azure Database for MySQL servers > Select Azure Database for MySQL deployment option >

Flexible server

Microsoft

Server names, networking connectivity method, zone redundant HA and backup redundancy cannot be changed after server is created. Review these options carefully before provisioning.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Azure subscription 1

Resource group * : (New) database-rg

Create new

Server details

Enter required settings for this server, including picking a location and configuring the compute and storage resources.

Server name * : mysqlserver-yh

Region * : East US 2

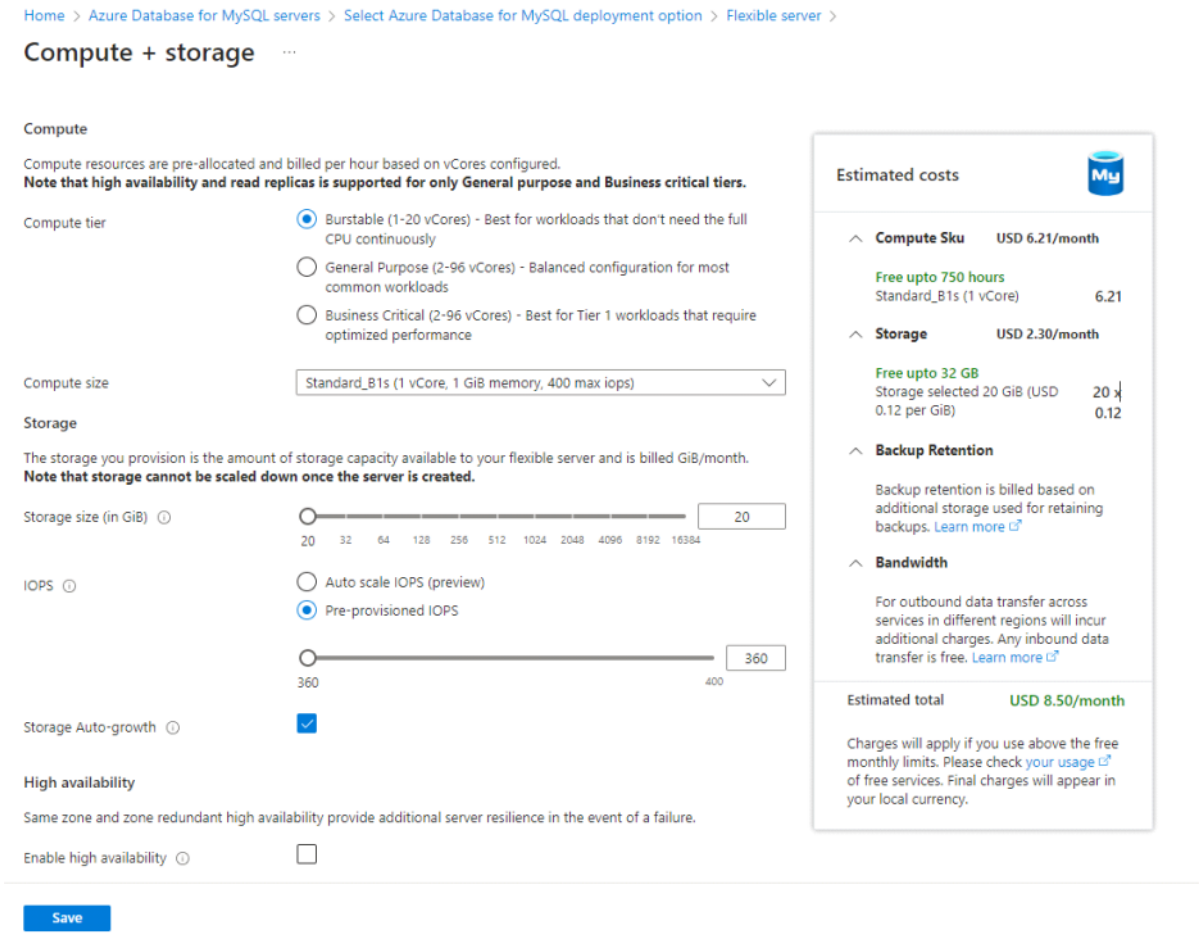
MySQL version * : 5.7

Workload type :

- For small or medium size databases
- Tier 1 Business Critical Workloads
- For development or hobby projects

Compute + storage : **Burstable, B1s**
1 vCores, 1 GiB RAM, 20 GiB storage, 360 IOPS
Geo-redundancy : Disabled
Configure server

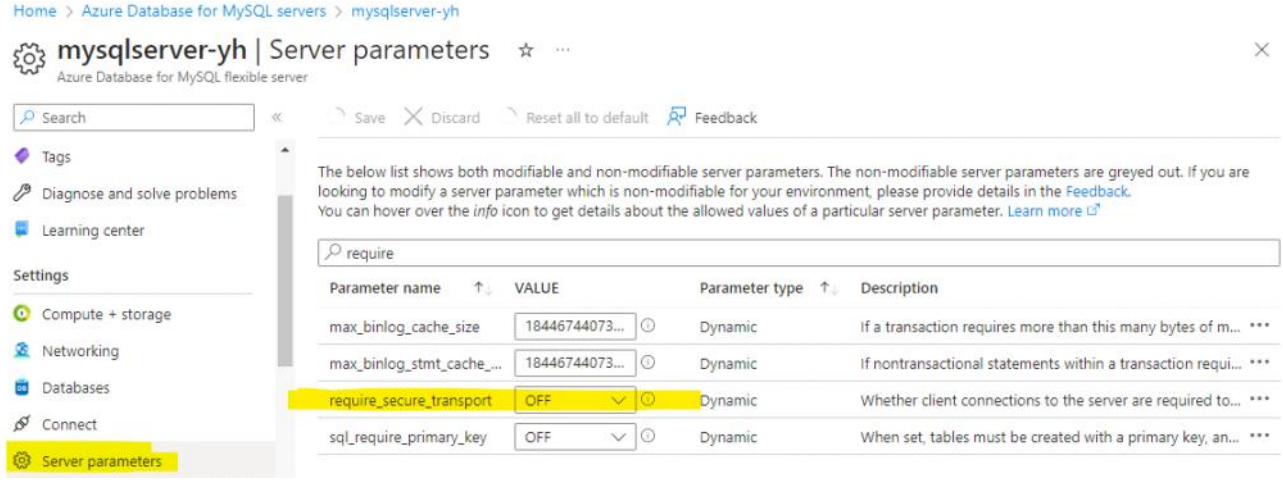
Availability zone : No preference



- Exploring MySQL Server in Azure
  - **Connection details:**
    hostname=mysqlserver-yh.mysql.database.azure.com
    username=mysqlserveryh

    To connect to the server: (in terminal)
    mysql --host=mysqlserver-yh.mysql.database.azure.com --user=mysqlserveryh -p

    To disable SSL



```
cheah [ ~ ]$ mysql --host=mysqlserver-yh.mysql.database.azure.com --user=mysqlserveryh -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20
Server version: 5.7.42-log MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

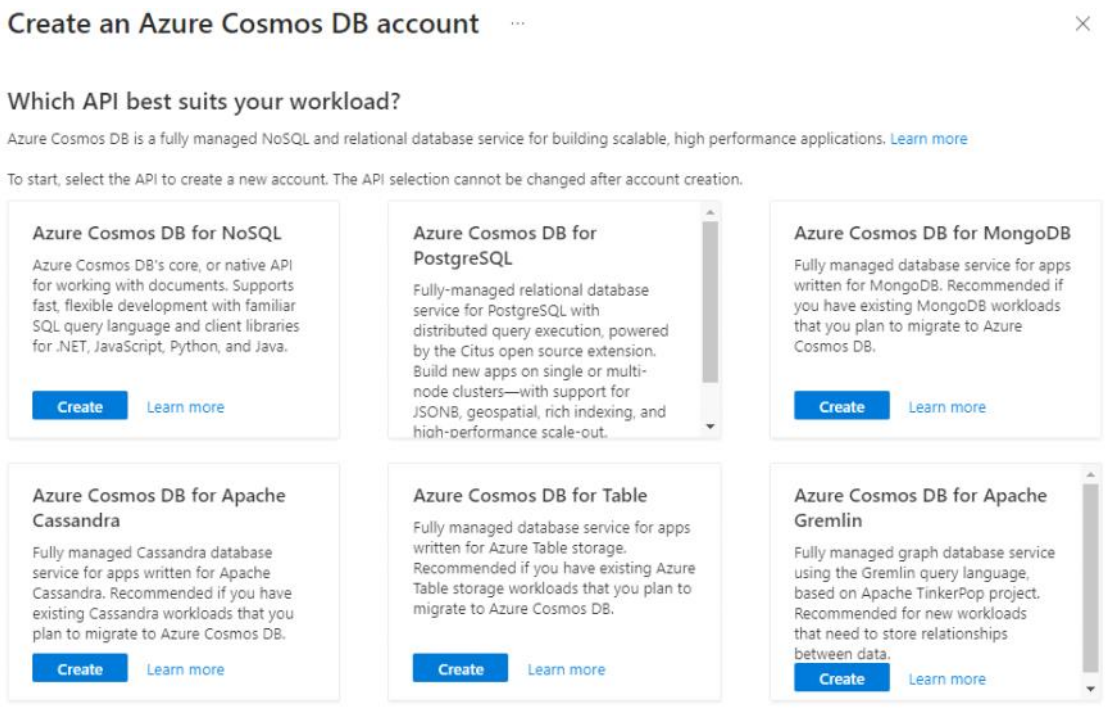
To create database:

```
create database mydatabase;
use mydatabase;
create table user (id integer, username varchar(30));
describe user;
insert into user values (1, 'Vinny');
insert into user values (2, 'Tom');
select * from user;
```

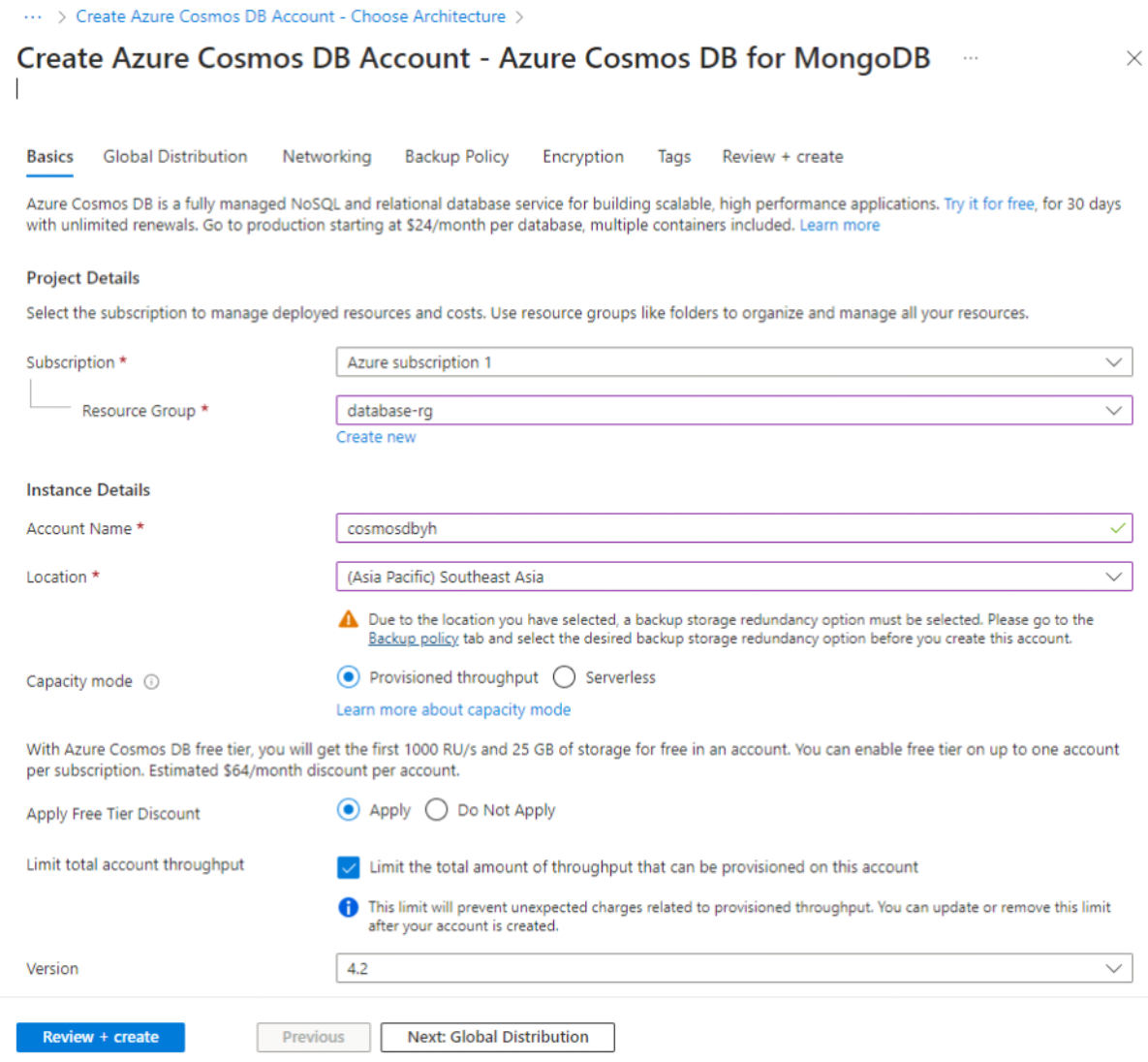
```
mysql>  create database mydatabase;
Query OK, 1 row affected (0.30 sec)

mysql> use mydatabase
Database changed
mysql> create table user (id integer, username varchar(30));
Query OK, 0 rows affected (0.44 sec)

mysql> describe user;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | YES  |     | NULL    |       |
| username | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.28 sec)
```

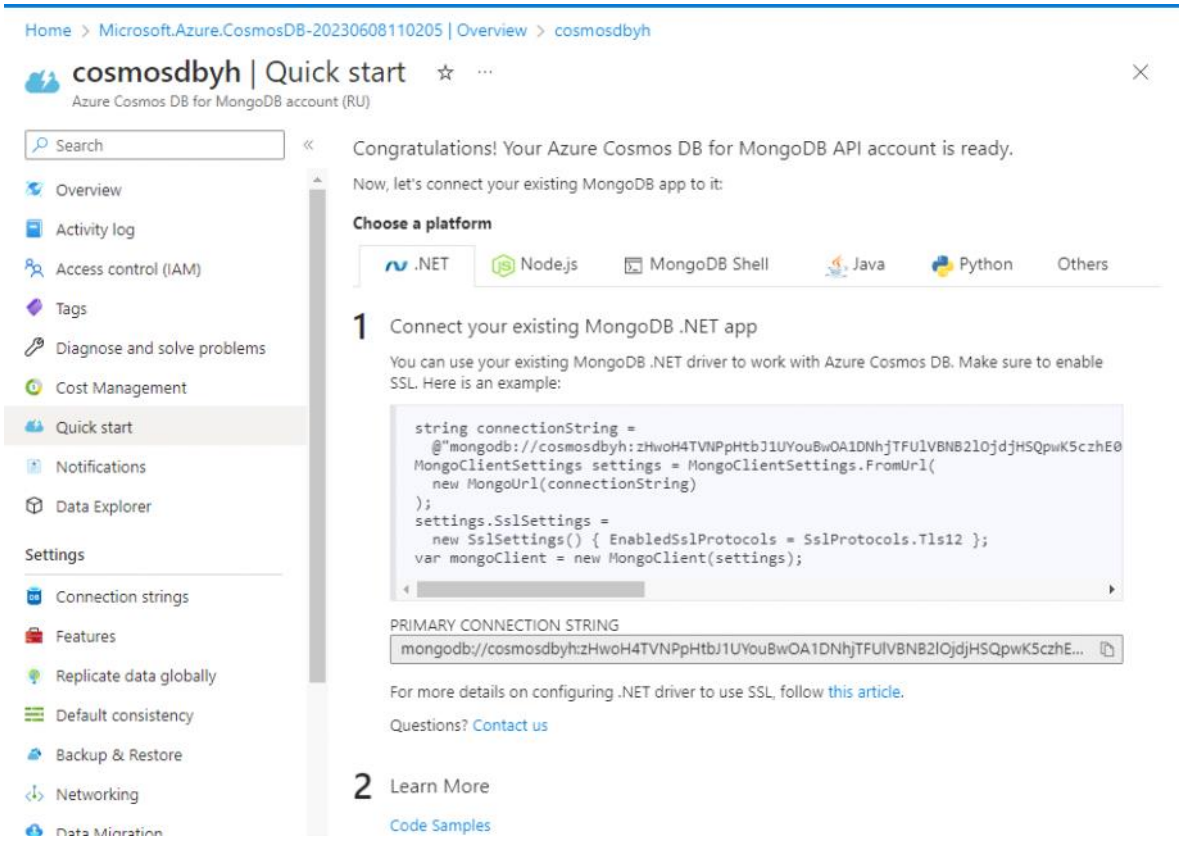
```
mysql> insert into user values (1, 'Vinny');
Query OK, 1 row affected (0.23 sec)

mysql> insert into user values (2, 'Tom');
Query OK, 1 row affected (0.23 sec)

mysql> select * from user;
+------+----------+
| id   | username |
+------+----------+
|    1 | Vinny    |
|    2 | Tom      |
+------+----------+
2 rows in set (0.22 sec)
```

- Azure NoSQL Database: Azure Cosmos DB
  - Create Azure Cosmos DB

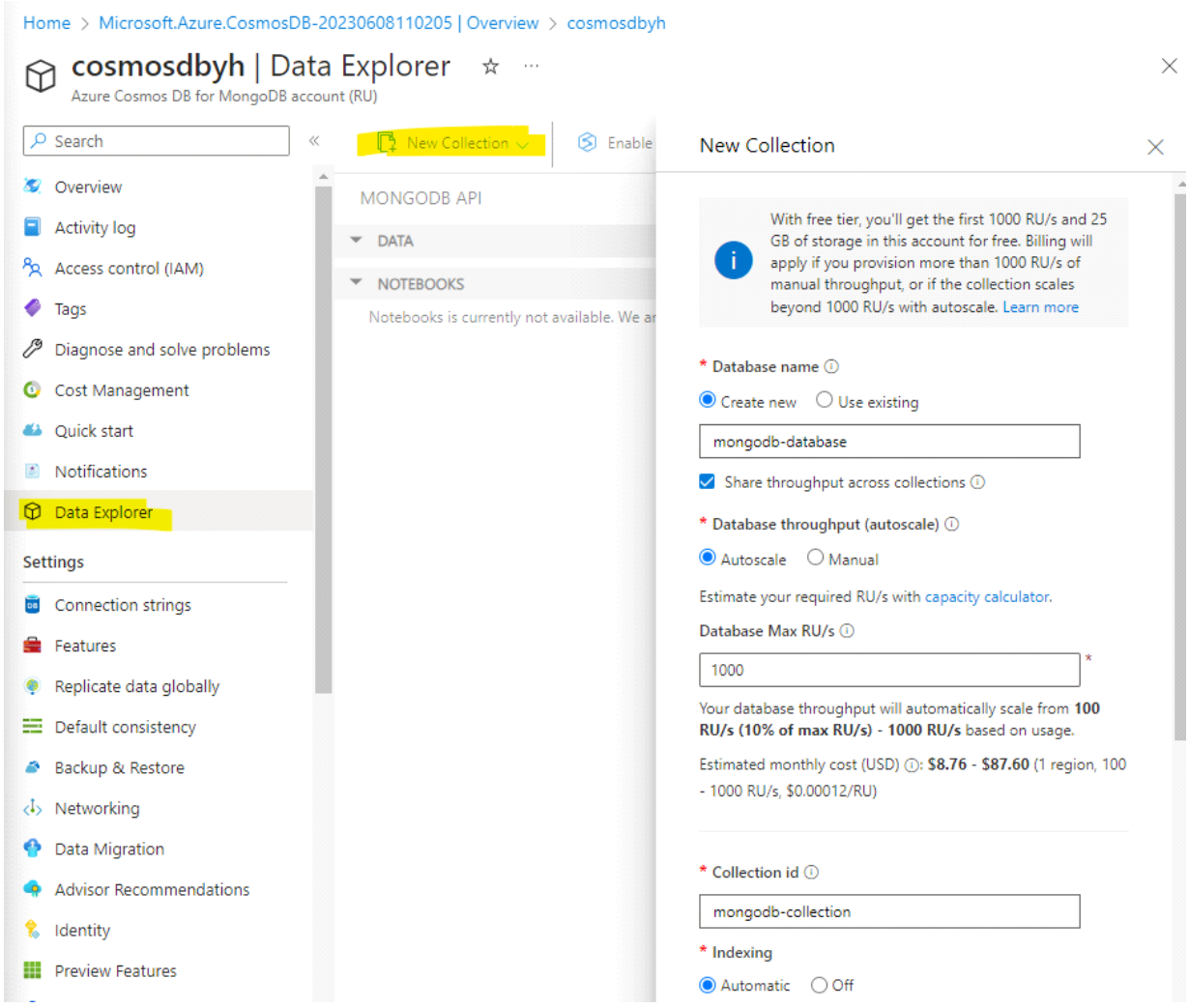


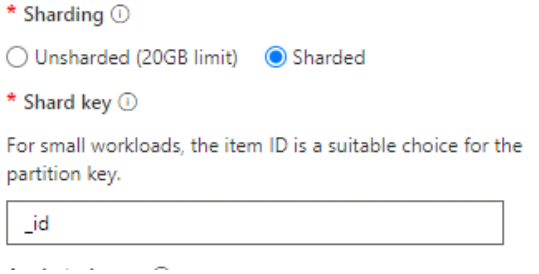
Select <MongoDB>





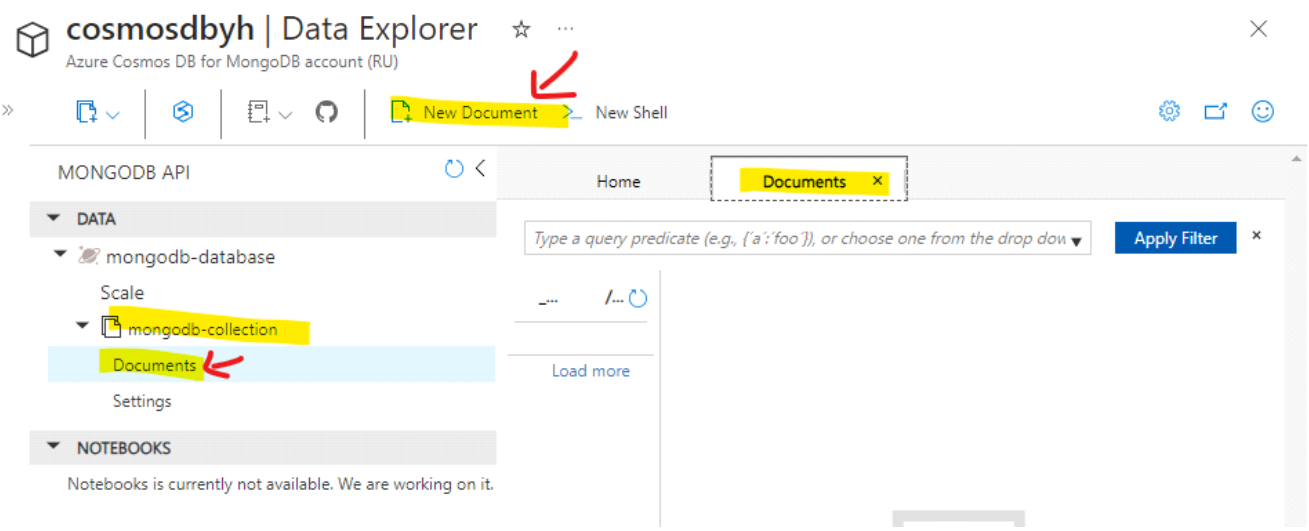
- Exploring Azure NoSQL Database: Azure Cosmos DB



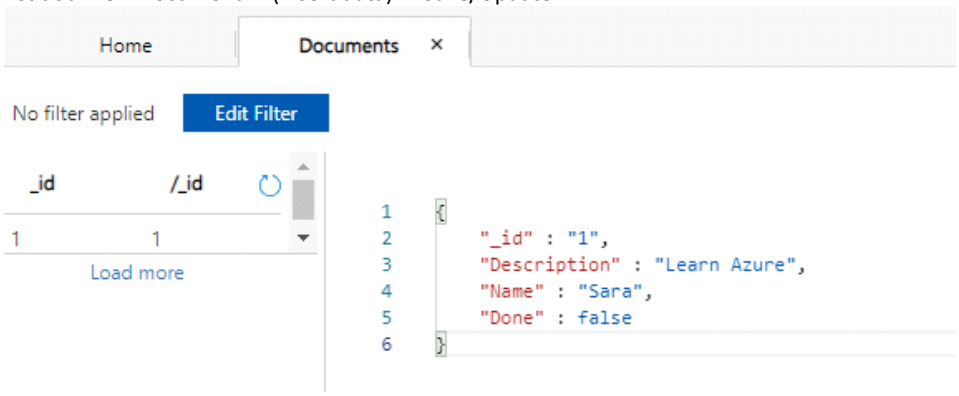


- ***Sharding in NoSQL databases is a technique that involves dividing and distributing data across multiple servers or nodes.***
- ***A shard key, also known as a partition key, is a field or attribute used to determine how data is partitioned and distributed across shards in a sharded database or NoSQL system.***

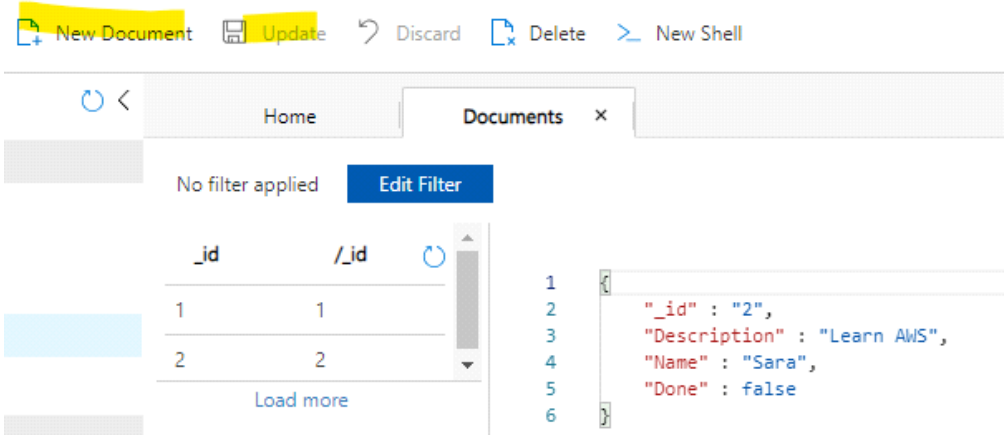
- MongoDB Collection
  - Documents - > New Documents



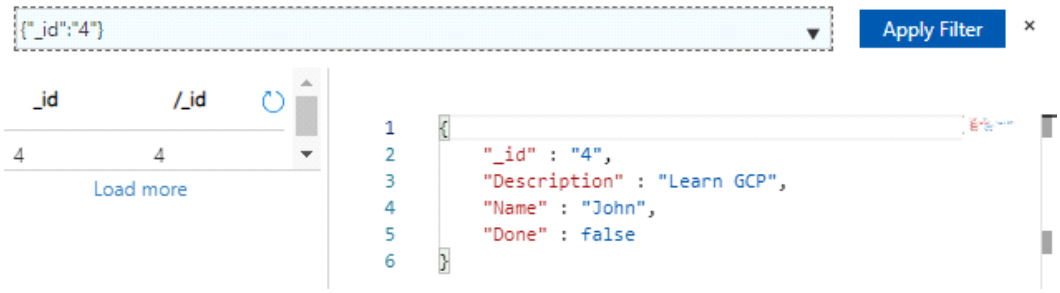
- 1st doc: New Document -> (insert data) -> Save/Update



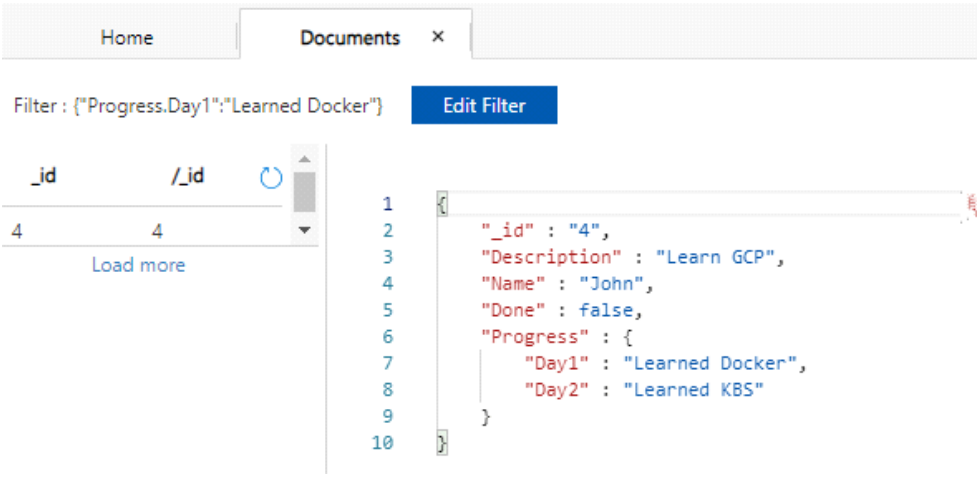
- 2nd doc: New Document -> (insert data) -> Save/Update



- Apply filter to filter the data (e.g. {"_id":"4"})



Or

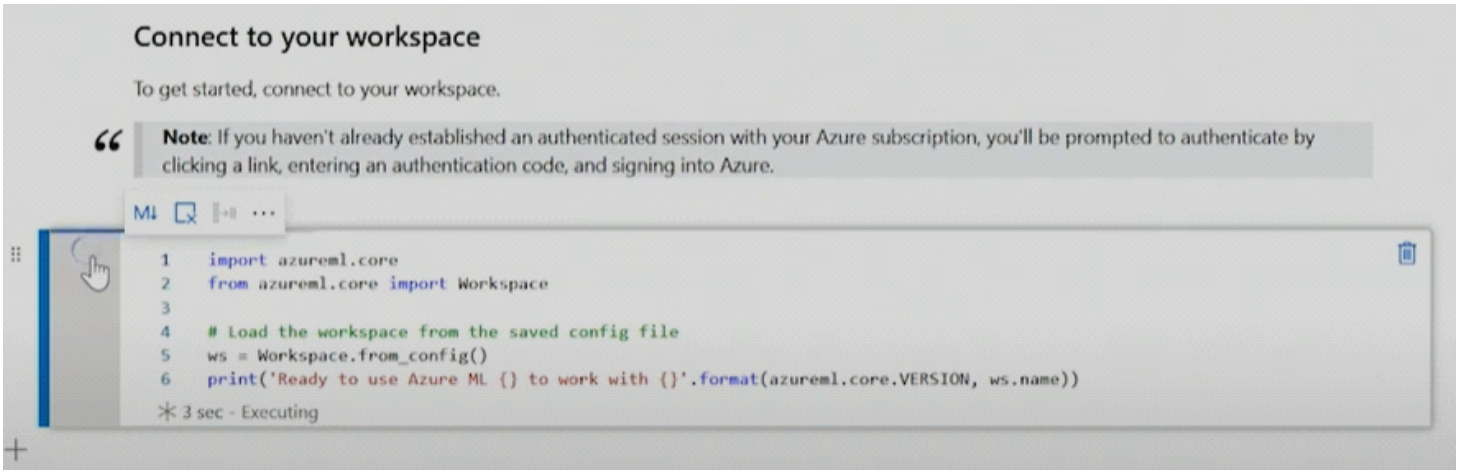

Home
Documents
Filter : {"Progress.Day1":"Learned Docker"}
Edit Filter
_id
/_id
4
4
Load more
{
"_id" : "4",
"Description" : "Learn GCP",
"Name" : "John",
"Done" : false,
"Progress" : {
"Day1" : "Learned Docker",
"Day2" : "Learned KBS"
}
}

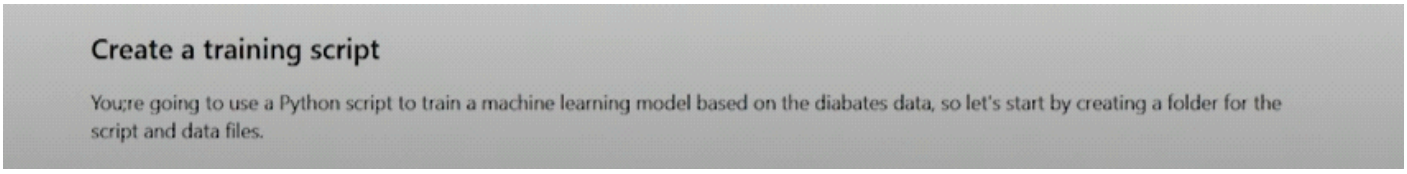
How to pass Exam DP-100 Microsoft Azure Data Scientist in 19 hours

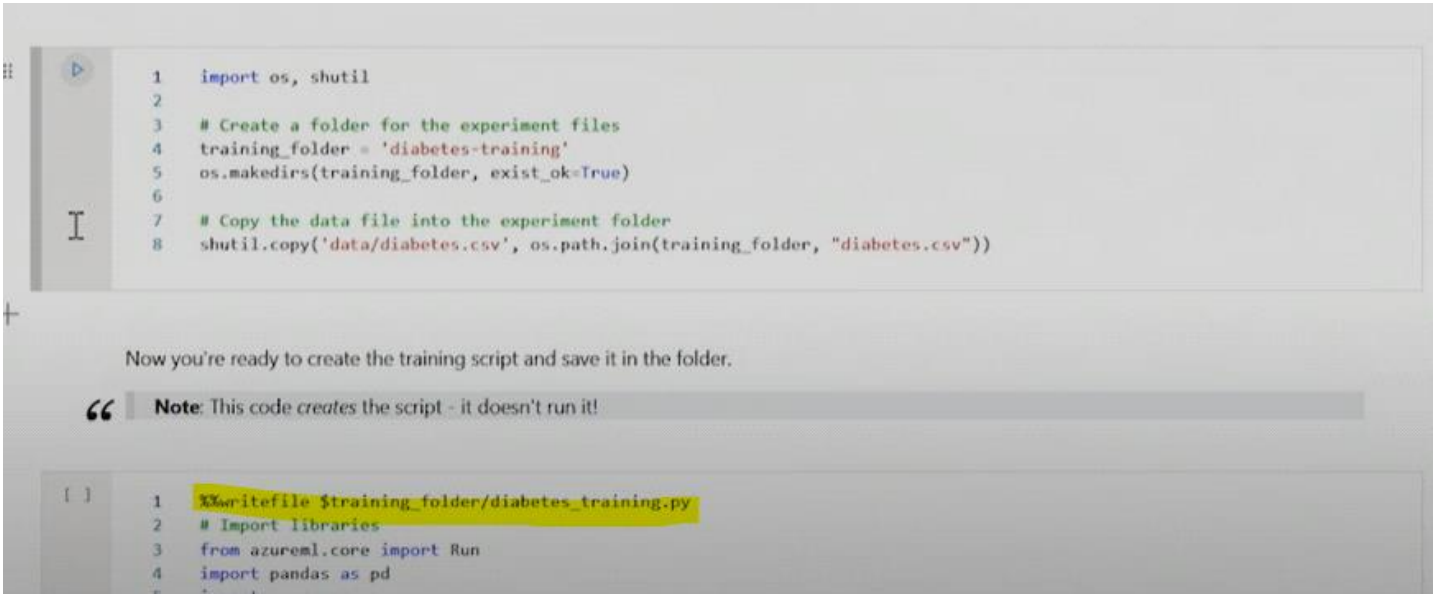
Friday, 22 September, 2023 2:39 PM

Part 07 Azure ML Workspaces

Friday, 22 September, 2023 2:39 PM







```
# Get the experiment run context
run = Run.get_context()

# 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','Diabe

# 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)
run.log('Regularization Rate',  np.float(reg))
model = LogisticRegression(C=1/reg, solver="liblinear").fit(X_train, y_train)
```

```
# Train a logistic regression model
print('Training a logistic regression model with regularization rate of', reg)
run.log('Regularization Rate',  np.float(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)
run.log('Accuracy', np.float(acc))

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

# Save the trained model in the outputs folder
os.makedirs('outputs', exist_ok=True)
joblib.dump(value=model, filename='outputs/diabetes_model.pkl')

run.complete()
```

Run the training script as an experiment

Now you're ready to run the script as an experiment. Note that the default environment does not include the **scikit-learn** package, so you need to explicitly add that to the configuration. The conda environment is built on-demand the first time the experiment is run, and cached for future runs that use the same configuration; so the first run will take a little longer.

```
[ ]
from azureml.core import Experiment, ScriptRunConfig, Environment
from azureml.core.conda_dependencies import CondaDependencies
from azureml.widgets import RunDetails

# Create a Python environment for the experiment
sklearn_env = Environment("sklearn-env")

# Ensure the required packages are installed (we need scikit-learn and Azure ML defaults)
packages = CondaDependencies.create(pip_packages=['scikit-learn','azureml-defaults'])
sklearn_env.python.conda_dependencies = packages

# Create a script config
script_config = ScriptRunConfig(source_directory=training_folder,
                                script='diabetes_training.py',
                                environment=sklearn_env)

# submit the experiment run
experiment_name = 'mslearn-train-diabetes'
experiment = Experiment(workspace=ws, name=experiment_name)
run = experiment.submit(config=script_config)

# Show the running experiment run in the notebook widget
RunDetails(run).show()
```

```
# Block until the experiment run has completed
run.wait_for_completion()
```

```
# Get logged metrics and files
metrics = run.get_metrics()
for key in metrics.keys():
        print(key, metrics.get(key))
print('\n')
for file in run.get_file_names():
    print(file)
```

✓ 1 sec

```
Regularization Rate 0.01
Accuracy 0.774
AUC 0.8483203144435048


azureml-logs/60_control_log.txt
azureml-logs/70_driver_log.txt
logs/azureml/21976_azureml.log
outputs/diabetes_model.pkl
```

Register the trained model

Note that the outputs of the experiment include the trained model file (**diabetes_model.pkl**). You can register this model in your Azure Machine Learning workspace, making it possible to track model versions and retrieve them later.

```
from azureml.core import Model

# Register the model
run.register_model(model_path='outputs/diabetes_model.pkl', model_name='diabetes_model',
                   tags={'Training context':'Script'},
                   properties={'AUC': run.get_metrics()['AUC'], 'Accuracy': run.get_metrics()['Accuracy']})

# List registered models
for model in Model.list(ws):
    print(model.name, 'version:', model.version)
    for tag_name in model.tags:
        tag = model.tags[tag_name]
        print ('\t',tag_name, ':', tag)
    for prop_name in model.properties:
        prop = model.properties[prop_name]
        print ('\t',prop_name, ':', prop)
    print('\n')
```

Again, lets start by creating a folder for the parameterized script and the training data.

```
import os, shutil

# Create a folder for the experiment files
training_folder = 'diabetes-training-params'
os.makedirs(training_folder, exist_ok=True)

# Copy the data file into the experiment folder
shutil.copy('data/diabetes.csv', os.path.join(training_folder, "diabetes.csv"))
```

Now let's create a script with an argument for the regularization rate hyperparameter. The argument is read using a Python **argparse.ArgumentParser** object.

```
%%writefile $training_folder/diabetes_training.py
# Import libraries
from azureml.core import Run
import pandas as pd
import numpy as np
import joblib
import os
import argparse
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

# Get the experiment run context
run = Run.get_context()

# Set regularization hyperparameter
parser = argparse.ArgumentParser()
parser.add_argument('--reg_rate', type=float, dest='reg', default=0.01)
args = parser.parse_args()
reg = args.reg

```

Run the script with arguments

You run the script as an experiment like you did previously, reusing the environment you created; but this time you must provide the **--reg_rate** parameter that the script expects as an argument.

```
# Create a script config
script_config = ScriptRunConfig(source_directory=training_folder,
                                script='diabetes_training.py',
                                arguments = ['--reg_rate', 0.1],
                                environment=sklearn_env)

# submit the experiment
experiment_name = 'mslearn-train-diabetes'
experiment = Experiment(workspace=ws, name=experiment_name)
run = experiment.submit(config=script_config)
RunDetails(run).show()
run.wait_for_completion()
```

k21academy

Friday, 29 September, 2023 2:17 PM

https://k21academy.com/microsoft-azure/dp-100/dp100-self-study-guide/

Azure Datastore and Create Azure Datasets : Complete Overview

Friday, 29 September, 2023 2:17 PM

https://k21academy.com/microsoft-azure/dp-100/datastores-and-datasets-in-azure/

Create a datastore

Completed 100 XP

- 7 minutes

In Azure Machine Learning, datastores are abstractions for cloud data sources. They encapsulate the information needed to connect to data sources, and securely store this connection information so that you don't have to code it in your scripts.
When you create a datastore with an existing storage account on Azure, you have the choice between two different authentication methods:

- Credential-based: Use a *service principal*, *shared access signature* (*SAS*) token or *account key* to authenticate access to your storage account.
- Identity-based: Use your *Azure Active Directory identity* or *managed identity*.

Understand types of datastores

Azure Machine Learning supports the creation of datastores for multiple kinds of Azure data source, including:

- Azure Blob Storage
- Azure File Share
- Azure Data Lake (Gen 1)
- Azure Data Lake (Gen 2)

Use the built-in datastores

Every workspace has four built-in datastores (two Azure Storage blob containers, and two Azure Storage file shares), which are used as system storages by Azure Machine Learning. There's also another datastore that gets added to your workspace if you make use of the open datasets provided as samples (for example, by creating a designer pipeline based on a sample dataset).
In most machine learning projects, you'll likely need to work with data sources of your own – either because you need to store larger volumes of data than the built-in datastores support, or because you need to integrate your machine learning solution with data from existing applications.

Create a datastore

Datastores are attached to workspaces and are used to store connection information to storage services. When you create a datastore, you provide a name that can be used to retrieve the connection information.
Datastores will allow you to easily connect to storage services without having to provide all necessary details every time you want to read or write data. It also creates a protective layer if you want users to use the data, but not connect to the underlying storage service directly.

Create a datastore for an Azure Blob Storage container

You can create a datastore through the graphical user interface, the Azure command-line interface (CLI), or the Python software development kit (SDK).
Depending on the storage service you want to connect to, there are different options for Azure Machine Learning to authenticate.
For example, when you want to create a datastore to connect to an Azure Blob Storage container, you may use an account key:

PythonCopy

```
blob_datastore = AzureBlobDatastore(
                     name = "blob_example",
                     description = "Datastore pointing to a blob container",
                     account_name = "mytestblobstore",
                     container_name = "data-container",
                     credentials = AccountKeyCredentials(
                            account_key="XXXxxxXXXxXXXXxxXXX"),
)
ml_client.create_or_update(blob_datastore)
```

Alternatively, you can create a datastore to connect to an Azure Blob Storage container by using a SAS token to authenticate:

PythonCopy

```
blob_datastore = AzureBlobDatastore(
name="blob_sas_example",
description="Datastore pointing to a blob container",
account_name="mytestblobstore",
container_name="data-container",
credentials=SasTokenCredentials(
sas_token="?xx=XXXX-XX-XX&xx=xxxx&xxx=xxx&xx=xxxxxxxxxxx&xx=XXXX-XX-XXXXX:XX:XXX&xx=XXXX-XX-XXXXX:XX:XXX&xxx=xxxxx&xxx=XXxXXXxxxxxXXXXXXXxXxxxXXXXXxxXXXXXxXXXXxXXXxXXxXX"),
)
ml_client.create_or_update(blob_datastore)
```

Create a data asset

Completed100 XP

- 12 minutes

As a data scientist, you want to focus on training machine learning models. Though you need access to data as input for a machine learning model, you don't want to worry about *how* to get access. To simplify getting access to the data you want to work with, you can use data assets.

Understand data assets

In Azure Machine Learning, data assets are references to where the data is stored, how to get access, and any other relevant metadata. You can create data assets to get access to data in datastores, Azure storage services, public URLs, or data stored on your local device.

The benefits of using data assets are:

- You can share and reuse data with other members of the team such that they don't need to remember file locations.
- You can seamlessly access data during model training (on any supported compute type) without worrying about connection strings or data paths.
- You can version the metadata of the data asset.

There are three main types of data assets you can use:

- URI file: Points to a specific file.
- URI folder: Points to a folder.
- MLTable: Points to a folder or file, and includes a schema to read as tabular data.

Note

URI stands for Uniform Resource Identifier and stands for a storage location on your local computer, Azure Blob or Data Lake Storage, publicly available https location, or even an attached datastore.

When to use data assets

Data assets are most useful when executing machine learning tasks as Azure Machine Learning jobs. As a job, you can run a Python script that takes inputs and generates outputs. A data asset can be parsed as both an input or output of an Azure Machine Learning job.

Let's take a look at each of the types of data assets, how to create them, and how to use the data asset in a job.

Create a URI file data asset

A URI file data asset points to a specific file. Azure Machine Learning will only store the

path to the file, which means you can point to any type of file. When you use the data asset, you'll specify how you want to read the data, which will depend on the type of data you're connecting to.

The supported paths you can use when creating a URI file data asset are:

- Local: ./<path>
- Azure Blob Storage: wasbs://<account_name>.blob.core.windows.net/<container_name>/<folder>/<file>
- Azure Data Lake Storage (Gen 2): abfss://<file_system>@<account_name>.dfs.core.windows.net/<folder>/<file>
- Datastore: azureml://datastores/<datastore_name>/paths/<folder>/<file>

Important

When you create a data asset and point to a file or folder stored on your local device, a copy of the file or folder will be uploaded to the default datastore workspaceblobstore. You can find the file or folder in the LocalUpload folder. By uploading a copy, you'll still be able to access the data from the Azure Machine Learning workspace, even when the local device on which the data is stored is unavailable.

To create a URI file data asset, you can use the following code:

PythonCopy

```
fromazure.ai.ml.entities importData
fromazure.ai.ml.constants importAssetTypes
my_path = '<supported-path>'my_data = Data(
    path=my_path,
    type=AssetTypes.URI_FILE,
    description="<description>",
    name="<name>",
    version="<version>")
ml_client.data.create_or_update(my_data)
```

When you parse the URI file data asset as input in an Azure Machine Learning job, you'll first need to read the data before you can work with it.

Imagine you create a Python script you want to run as a job, and you set the value of the input parameter input_data to be the URI file data asset (which points to a CSV file). You can read the data by including the following code in your Python script:

PythonCopy

```
importargparse
importpandas aspd
parser = argparse.ArgumentParser()
parser.add_argument("--input_data", type=str)
args = parser.parse_args()
df = pd.read_csv(args.input_data)
print(df.head(10))
```

If your URI file data asset points to a different type of file, you'll need to use the appropriate Python code to read the data. For example, if instead of CSV files, you're

working with JSON files, you'd use pd.read_json() instead.

Create a URI folder data asset

A URI folder data asset points to a specific folder. It works similar to a URI file data asset and supports the same paths.
To create a URI folder data asset with the Python SDK, you can use the following code:

PythonCopy

```
fromazure.ai.ml.entities importData
fromazure.ai.ml.constants importAssetTypes
my_path = '<supported-path>'my_data = Data(
    path=my_path,
    type=AssetTypes.URI_FOLDER,
    description="<description>",
    name="<name>",
    version='<version>')
ml_client.data.create_or_update(my_data)
```

When you parse the URI folder data asset as input in an Azure Machine Learning job, you'll first need to read the data before you can work with it.
Imagine you create a Python script you want to run as a job, and you set the value of the input parameter input_data to be the URI folder data asset (which points to multiple CSV files). You may want to read all CSV files in the folder and concatenate them, which you can do by including the following code in your Python script:

PythonCopy

```
importargparse
importglob
importpandas aspd
parser = argparse.ArgumentParser()
parser.add_argument("--input_data", type=str)
args = parser.parse_args()
data_path = args.input_data
all_files = glob.glob(data_path + "/*.csv")
df = pd.concat((pd.read_csv(f) forf inall_files), sort=False)
```

Depending on the type of data you're working with, the code you use to read the files may change.

Create a MLTable data asset

A MLTable data asset allows you to point to tabular data. When you create a MLTable data asset, you specify the schema definition to read the data. As the schema is already defined and stored with the data asset, you don't have to specify how to read the data when you use it.
Therefore, you'll want to use a MLTable data asset when the schema of your data is

complex or changes frequently. Instead of changing how to read the data in every script that uses the data, you only have to change it in the data asset itself.
When you define the schema when creating a MLTable data asset, you can also choose to only specify a subset of the data.
For certain features in Azure Machine Learning, like Automated Machine Learning, you'll need to use a MLTable data asset, as Azure Machine Learning needs to know how to read the data.
To define the schema, it's recommended to include a MLTable file in the same folder as the data you want to read. The MLTable file includes the path pointing to the data you want to read, and how to read the data:

ymlCopy

```
type:mltablepaths:  - pattern:./*.txttransformations:  - read_delimited:      delimiter:','      encoding:ascii
header:all_files_same_headers
```

Tip

Learn more on how to create the MLTable file and which transformations you can include.

To create a MLTable data asset with the Python SDK, you can use the following code:

PythonCopy

```
fromazure.ai.ml.entities importData
fromazure.ai.ml.constants importAssetTypes
my_path = '<path-including-mltable-file>'my_data = Data(
    path=my_path,
    type=AssetTypes.MLTABLE,
    description="<description>",
    name="<name>",
    version='<version>')
ml_client.data.create_or_update(my_data)
```

When you parse a MLTable data asset as input to a Python script you want to run as an Azure Machine Learning job, you can include the following code to read the data:

PythonCopy

```
importargparse
importmltable
importpandas
parser = argparse.ArgumentParser()
parser.add_argument("--input_data", type=str)
args = parser.parse_args()
tbl = mltable.load(args.input_data)
df = tbl.to_pandas_dataframe()
print(df.head(10))
```

A common approach is to convert the tabular data to a Pandas data frame. However, you can also convert the data to a Spark data frame if that suits your workload better.

Compute Targets

Friday, 29 September, 2023 2:40 PM

https://learn.microsoft.com/en-us/azure/machine-learning/how-to-set-up-training-targets?view=azureml-api-1

1. Perform hyperparameter tuning with Azure Machine Learning

Sunday, 1 October, 2023 8:20 PM

https://learn.microsoft.com/en-us/training/modules/perform-hyperparameter-tuning-azure-machine-learning-pipelines/4-configure-early-termination

Configure an early termination policy

There are two main parameters when you choose to use an early termination policy:

- evaluation_interval: Specifies at which interval you want the policy to be evaluated. Every time the primary metric is logged for a trial counts as an interval.
- delay_evaluation: Specifies when to start evaluating the policy. This parameter allows for at least a minimum of trials to complete without an early termination policy affecting them.

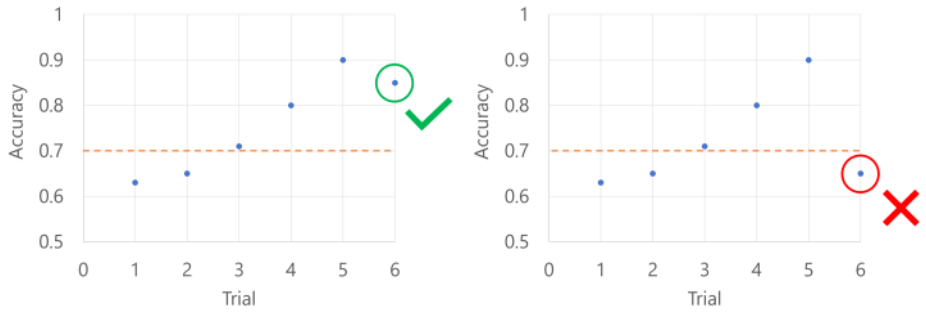
Bandit policy

You can use a bandit policy to stop a trial if the target performance metric underperforms the best trial so far by a specified margin.
For example, the following code applies a bandit policy with a delay of five trials, evaluates the policy at every interval, and allows an absolute slack amount of 0.2.

PythonCopy

```
fromazure.ai.ml.sweep importBanditPolicy
sweep_job.early_termination = BanditPolicy(
    slack_amount = 0.2,
    delay_evaluation = 5,
    evaluation_interval = 1)
```

Imagine the primary metric is the accuracy of the model. When after the first five trials, the best performing model has an accuracy of 0.9, any new model needs to perform better than (0.9-0.2) or 0.7. If the new model's accuracy is higher than 0.7, the sweep job will continue. If the new model has an accuracy score lower than 0.7, the policy will terminate the sweep job.



You can also apply a bandit policy using a slack *factor*, which compares the performance metric as a ratio rather than an absolute value.

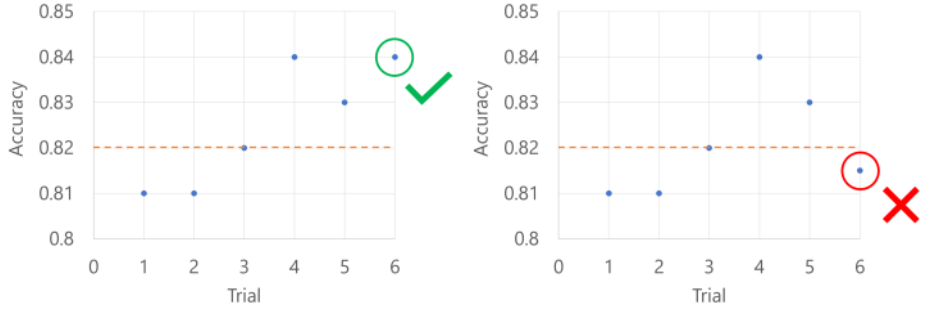
Median stopping policy

A median stopping policy abandons trials where the target performance metric is worse than the median of the running averages for all trials.
For example, the following code applies a median stopping policy with a delay of five trials and evaluates the policy at every interval.

PythonCopy

```
fromazure.ai.ml.sweep importMedianStoppingPolicy
sweep_job.early_termination = MedianStoppingPolicy(
    delay_evaluation = 5,
    evaluation_interval = 1)
```

Imagine the primary metric is the accuracy of the model. When the accuracy is logged for the sixth trial, the metric needs to be higher than the median of the accuracy scores so far. Suppose the median of the accuracy scores so far is 0.82. If the new model's accuracy is higher than 0.82, the sweep job will continue. If the new model has an accuracy score lower than 0.82, the policy will stop the sweep job, and no new models will be trained.



Truncation selection policy

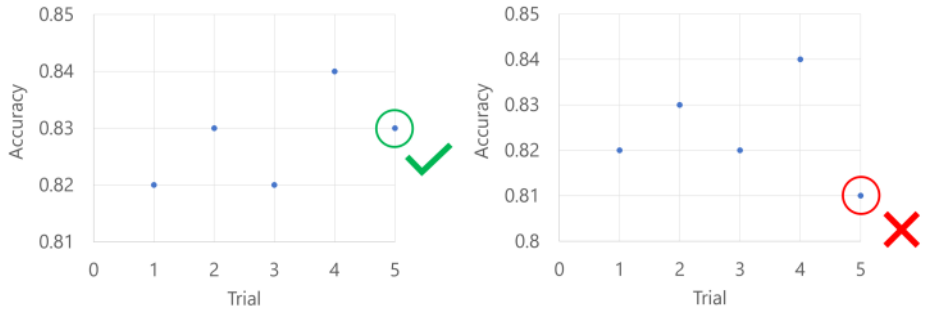
A truncation selection policy cancels the lowest performing *X*% of trials at each evaluation interval based on the *truncation_percentage* value you specify for *X*.

For example, the following code applies a truncation selection policy with a delay of four trials, evaluates the policy at every interval, and uses a truncation percentage of 20%.

PythonCopy

```
fromazure.ai.ml.sweep importTruncationSelectionPolicy
sweep_job.early_termination = TruncationSelectionPolicy(
    evaluation_interval=1,
    truncation_percentage=20,
    delay_evaluation=4)
```

Imagine the primary metric is the accuracy of the model. When the accuracy is logged for the fifth trial, the metric should not be in the worst 20% of the trials so far. In this case, 20% translates to one trial. In other words, if the fifth trial is not the worst performing model so far, the sweep job will continue. If the fifth trial has the lowest accuracy score of all trials so far, the sweep job will stop.



Azure Machine Learning Architecture | Microsoft Docs

Friday, 29 September, 2023 3:11 PM

https://learn.microsoft.com/en-in/azure/machine-learning/concept-azure-machine-learning-v2?view=azureml-api-2&tabs=sdk

Create a workspace (Jupyter Notebook)

https://github.com/Azure/azureml-examples/blob/main/sdk/python/resources/workspace/workspace.ipynb

Compute (Jupyter Notebook)

https://github.com/Azure/azureml-examples/blob/main/sdk/python/resources/compute/compute.ipynb

- Compute instance can be created using the ComputeInstance class.
- A compute cluster can be created using the AmlCompute class.

Datastore (Jupyter Notebook)

https://github.com/Azure/azureml-examples/blob/main/sdk/python/resources/datastores/datastore.ipynb

- The AzureBlobDatastore can be used to create datastores for Azure blob containers.

Model

Attachment:

Create a workspace (Jupyter Notebook):

1.2 Configure where workspace needs to be created.

Before creating a workspace, we need identifier parameters - a subscription and resource group. We will use these parameters to define where the Azure Machine Learning workspace.

In [ ]:
```
# Enter details of your subscription
subscription_id = "<SUBSCRIPTION_ID>"
resource_group = "<RESOURCE_GROUP>"
```

The `MLClient` from `azure.ml` will be used to create the workspace. We use the default default azure authentication for this tutorial. Check the configuration notebook for more details on how to configure credentials and connect to a workspace.

In [ ]:
```
# get a handle to the subscription

from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

ml_client = MLClient(DefaultAzureCredential(), subscription_id, resource_group)
```

1.3 Create a basic workspace

To create a basic workspace, we will define the following attributes

- `name` - Name of the workspace
- `location` - The Azure location for workspace. For e.g. eastus, westus etc.
- `display_name` - Display name of the workspace
- `description` - Description of the workspace
- `hbi_workspace` - Flag to define whether the workspace is a High Impact workspace
- `tags` - (Optional) Tags to help search/filter on workspace easily

Using the `MLClient` created earlier, we will create the workspace. This command will start workspace creation and provide a confirmation.

[ ]:
```
# Creating a unique workspace name with current datetime to avoid conflicts
from azure.ai.ml.entities import Workspace
import datetime

basic_workspace_name = "mlw-basic-prod-" + datetime.datetime.now().strftime(
    "%Y%m%d%H%M"
)

ws_basic = Workspace(
    name=basic_workspace_name,
    location="eastus",
    display_name="Basic workspace-example",
    description="This example shows how to create a basic workspace",
    hbi_workspace=False,
    tags=dict(purpose="demo"),
)

ws_basic = ml_client.workspaces.begin_create(ws_basic).result()
print(ws_basic)
```

3.2 Load workspace from a config file

The config details of a workspace can be saved to a file from the Azure Machine Learning portal. Click on the name of the portal on the top right corner to see the link to save the config file.

This config file can be used to load a workspace using `MLClient`. If no path is mentioned, path is defaulted to current folder. If no file name is mentioned, file name will be defaulted to `config.json`

```
import json
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

# create a demo config file "my_config.json" to be used in this example
# Data to be written
wsconfig = {
    "subscription_id": subscription_id,
    "resource_group": resource_group,
    "workspace_name": ws_basic.name,
}
with open("my_config.json", "w") as outfile:
    json.dump(wsconfig, outfile)

# read the config from the current directory
ws_from_config = MLClient.from_config(
    DefaultAzureCredential(), file_name="my_config.json"
)
print(ws_from_config.workspace_name)
```

Azure ML SDK

Saturday, 30 September, 2023 8:37 AM

Explore Azure Machine Learning workspace resources and assets

https://learn.microsoft.com/en-us/training/modules/explore-azure-machine-learning-workspace-resources-assets/

1. [Deploy a model to a batch endpoint](#)

Monday, 2 October, 2023 9:32 AM

Create the scoring script

The scoring script is a file that reads the new data, loads the model, and performs the scoring.

The scoring script must include two functions:

- `init()`: Called once at the beginning of the process, so use for any costly or common preparation like loading the model.
- `run()`: Called for each mini batch to perform the scoring.

The `run()` method should return a pandas DataFrame or an array/list.

A scoring script may look as follows:

Python Copy

```python
import os
import mlflow
import pandas as pd


def init():
    global model

    # get the path to the registered model file and load it
    model_path = os.path.join(os.environ["AZUREML_MODEL_DIR"], "model")
    model = mlflow.pyfunc.load(model_path)


def run(mini_batch):
    print(f"run method start: {__file__}, run({len(mini_batch)} files)")
    resultList = []

    for file_path in mini_batch:
        data = pd.read_csv(file_path)
        pred = model.predict(data)

        df = pd.DataFrame(pred, columns=["predictions"])
        df["file"] = os.path.basename(file_path)
        resultList.extend(df.values)

    return resultList
```

There are some things to note from the example script:

- AZUREML_MODEL_DIR is an environment variable that you can use to locate the files associated with the model.
- Use global variable to make any assets available that are needed to score the new data, like the loaded model.
- The size of the mini_batch is defined in the deployment configuration. If the files in the mini batch are too large to be processed, you need to split the files into smaller files.
- By default, the predictions will be written to one single file.

Configure and create the deployment

Finally, you can configure and create the deployment with the `BatchDeployment` class.

Python Copy

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

deployment = BatchDeployment(
    name="forecast-mlflow",
    description="A sales forecaster",
    endpoint_name=endpoint.name,
    model=model,
    compute="aml-cluster",
    code_path="./code",
    scoring_script="score.py",
    environment=env,
    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)
```

OR~~~~~~

You're going to use a pipeline to run the batch prediction script, generate predictions from the input data, and save the results as a text file in the output folder. To do this, you can use a **ParallelRunStep**, which enables the batch data to be processed in parallel and the results collated in a single output file named *parallel_run_step.txt*.

In [ ]:

```
from azureml.pipeline.steps import ParallelRunConfig, ParallelRunStep
from azureml.pipeline.core import PipelineData

default_ds = ws.get_default_datastore()

output_dir = PipelineData(name='inferences',
                          datastore=default_ds,
                          output_path_on_compute='diabetes/results')

parallel_run_config = ParallelRunConfig(
    source_directory=experiment_folder,
    entry_script="batch_diabetes.py",
    mini_batch_size="5",
    error_threshold=10,
    output_action="append_row",
    environment=batch_env,
    compute_target=inference_cluster,
    node_count=2)

parallelrun_step = ParallelRunStep(
    name='batch-score-diabetes',
    parallel_run_config=parallel_run_config,
    inputs=[batch_data_set.as_named_input('diabetes_batch')],
    output=output_dir,
    arguments=[],
    allow_reuse=True
)

print('Steps defined')
```

Create and run machine learning pipelines using components with the Azure Machine Learning SDK v2

Monday, 2 October, 2023 2:27 PM

https://learn.microsoft.com/en-us/azure/machine-learning/how-to-create-component-pipeline-python?view=azureml-api-2

Create a pipeline

Completed100 XP

- 6 minutes

In Azure Machine Learning, a pipeline is a workflow of machine learning tasks in which each task is defined as a component.

Components can be arranged sequentially or in parallel, enabling you to build sophisticated flow logic to orchestrate machine learning operations. Each component can be run on a specific compute target, making it possible to combine different types of processing as required to achieve an overall goal.

A pipeline can be executed as a process by running the pipeline as a pipeline job. Each component is executed as a child job as part of the overall pipeline job.

Build a pipeline

An Azure Machine Learning pipeline is defined in a YAML file. The YAML file includes the pipeline job name, inputs, outputs, and settings.

You can create the YAML file, or use the @pipeline() function to create the YAML file.

Tip

Review the reference documentation for the @pipeline() function.

For example, if you want to build a pipeline that first prepares the data, and then trains the model, you can use the following code:

PythonCopy

```
fromazure.ai.ml.dsl importpipeline
@pipeline()defpipeline_function_name(pipeline_job_input):prep_data =
loaded_component_prep(input_data=pipeline_job_input)
    train_model = loaded_component_train(training_data=prep_data.outputs.output_data)
return{
        "pipeline_job_transformed_data": prep_data.outputs.output_data,
        "pipeline_job_trained_model": train_model.outputs.model_output,
    }
```

To pass a registered data asset as the pipeline job input, you can call the function you created with the data asset as input:

PythonCopy

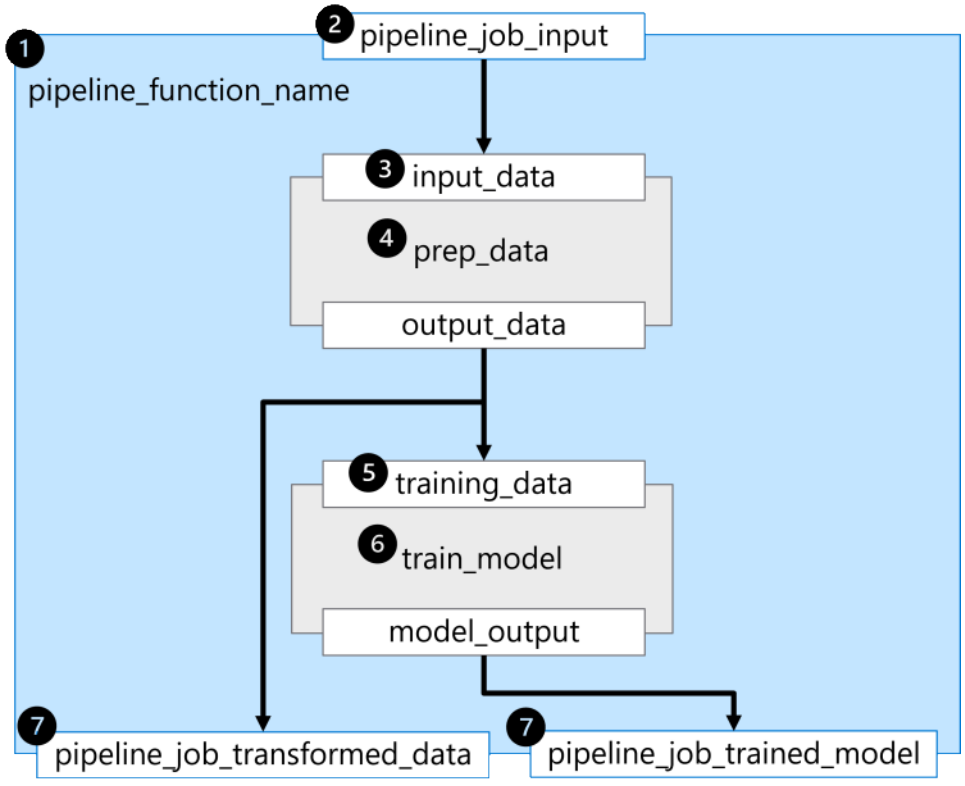
```
fromazure.ai.ml importInput
fromazure.ai.ml.constants importAssetTypes
pipeline_job = pipeline_function_name(
    Input(type=AssetTypes.URI_FILE,
```

```
    path="azureml:data:1"))
```

The @pipeline() function builds a pipeline consisting of two sequential steps, represented by the two loaded components.

To understand the pipeline built in the example, let's explore it step by step:

1. The pipeline is built by defining the function pipeline_function_name.
2. The pipeline function expects pipeline_job_input as the overall pipeline input.
3. The first pipeline step requires a value for the input parameter input_data. The value for the input will be the value of pipeline_job_input.
4. The first pipeline step is defined by the loaded component for prep_data.
5. The value of the output_data of the first pipeline step is used for the expected input training_data of the second pipeline step.
6. The second pipeline step is defined by the loaded component for train_model and results in a trained model referred to by model_output.
7. Pipeline outputs are defined by returning variables from the pipeline function. There are two outputs:
    - pipeline_job_transformed_data with the value of prep_data.outputs.output_data
    - pipeline_job_trained_model with the value of train_model.outputs.model_output



The result of running the @pipeline() function is a YAML file that you can review by printing the pipeline_job object you created when calling the function:

PythonCopy

```
print(pipeline_job)
```

The output will be formatted as a YAML file, which includes the configuration of the pipeline and its components. Some parameters included in the YAML file are shown in the following example.

ymlCopy

```
display_name:pipeline_function_nametype:pipelineinputs:  pipeline_job_input:    type:uri_file
path:azureml:data:1outputs:  pipeline_job_transformed_data:null  pipeline_job_trained_model:nulljobs:
prep_data:    type:command    inputs:    input_data:      path:${{parent.inputs.pipeline_job_input}}
outputs:    output_data:${{parent.outputs.pipeline_job_transformed_data}}  train_model:
type:command    inputs:    input_data:      path:${{parent.outputs.pipeline_job_transformed_data}}
outputs:    output_model:${{parent.outputs.pipeline_job_trained_model}}tags:{}properties:{}settings:{}
```

Sample Questions

Tuesday, 3 October, 2023 8:06 AM

https://www.whizlabs.com/blog/microsoft-azure-dp-100-exam-questions/

https://k21academy.s3.us-west-2.amazonaws.com/Public/DP-100/DP100+sample+question/DP100_Azure_Data_Scientist_Sample_Exam_Guide_ed1.pdf

Service Name	Answer Area
Local web service	Use for low-scale CPU-based workloads that require less than 48 GB of RAM. Doesn't require you to manage a cluster
Azure Machine Learning endpoints	Run inferencing workloads on on-premises, cloud, and edge Kubernetes clusters
Azure Machine Learning Kubernetes	Fully managed computes for real-time (managed online endpoints) and batch scoring (batch endpoints) on serverless compute
Azure Container Instances	Use for limited testing and troubleshooting. Hardware acceleration depends on the use of libraries in the local system

Compute Target	Answer Area
Compute Instances	Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.
Compute Clusters	Deployment targets for predictive services that use your trained models
Inference Clusters	Scalable clusters of virtual machines for on-demand processing of experiment code.
Attached Compute	Development workstations that data scientists can use to work with data and models.