Azure Machine Learning SDK

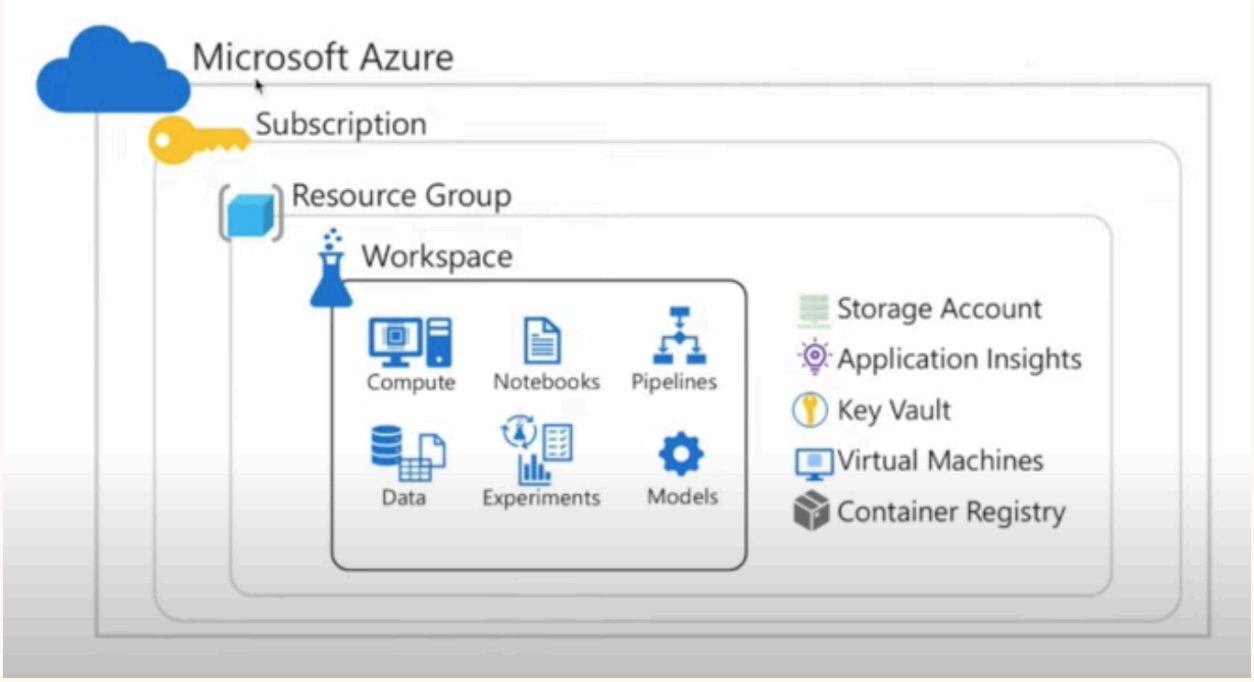
By - Jayabharathi Hari
Deakin University
✉: s224643593@deakin.edu.au

About Azure Machine Learning SDK for Python

Azure Machine Learning provides software development kits(SDK) for Python and which can be used to create, manage and utilize assets in an Azure Machine Learning Workspace by data scientists and AI developers to build and run machine learning projects/workflow((MLops)) on Microsoft's Azure cloud platform. This SDK works in environments like Jupyter Notebooks, Google Colab and Python IDEs.

It provides an intuitive interface for experimentation, model training, deployment, and monitoring within a Python environment. With features like Automated Machine Learning (AutoML) and seamless integration with Azure services, the SDK streamlines the end-to-end machine learning workflow, empowering users to create scalable and efficient machine learning solutions.

1.Workspace:

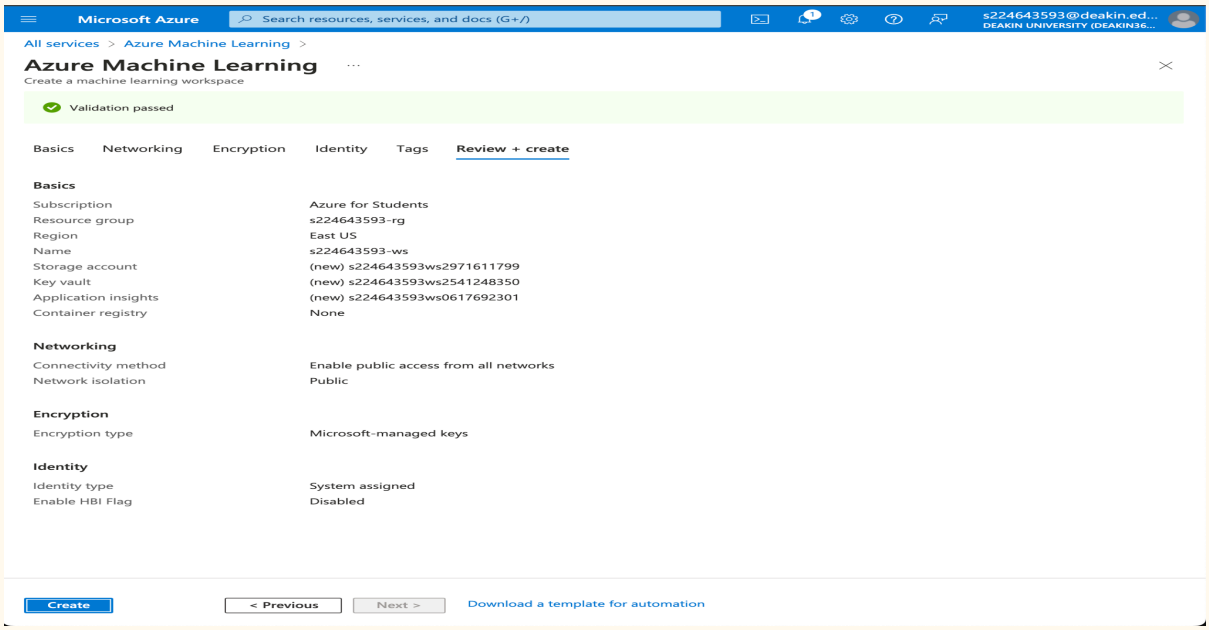


Picture Source: *Microsoft Azure website.*

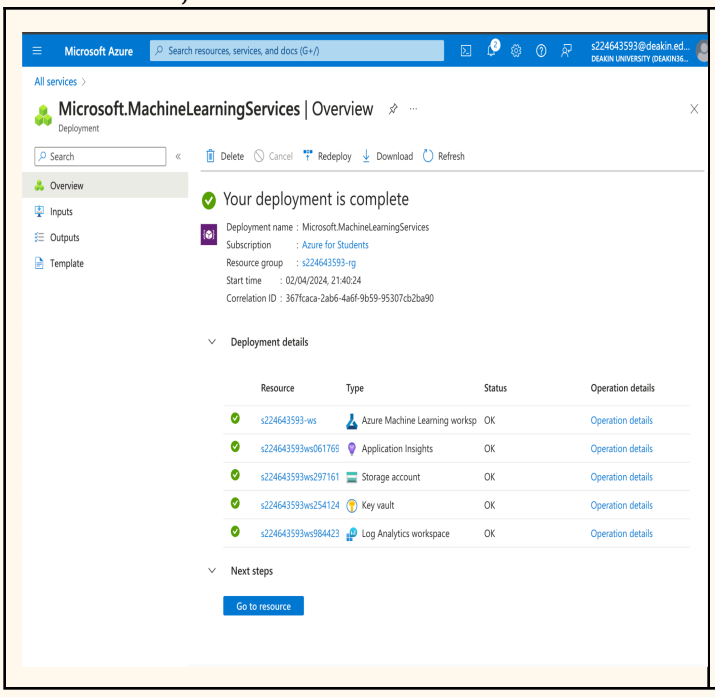
Imagine a cloud storage unit for your machine learning projects. That's a Microsoft Machine Learning workspace. It lives within Azure and holds everything you need: code, data, models. Think of it as a central hub. Data scientists can use this workspace to collaborate, run experiments (train models), and even deploy finished models to use. It all happens securely in the Microsoft cloud.

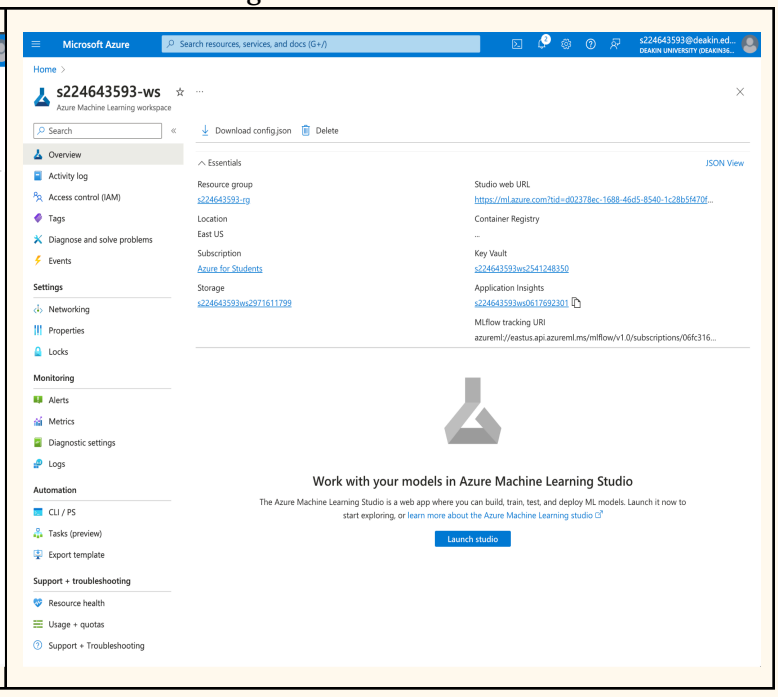
2.Steps to create a workspace for machine learning using the Azure Portal.

1. Sign in to Azure portal and search for "Machine Learning".
2. Click "Create" and choose a unique name for your workspace.
3. Select the subscription and resource group (create a new one or choose existing).
4. Pick a region for your workspace and optionally configure application insights and container registry.
5. Hit "Create" to provision your workspace in Azure.



We would be able to view the below kind of overview in the resource group once workspace got created, then click on “Go to resource” to launch Machine Learning Studio.





3.Lets discuss about Computing Instance & Compute Cluster:

Computing Instance:	**Compute Cluster:**
Imagine a powerful, pre-configured computer in the cloud. This single-user instance is ideal for development and testing tasks. It can run multiple jobs in parallel and even handle some training, but lacks the muscle for large-scale projects.	Think of a team of computers working together. A cluster offers multiple virtual machines (nodes) that can be scaled up or down based on your workload. This allows you to tackle demanding training jobs by distributing the work across the cluster for faster results.
Use a Compute Instance: **Development and Testing:** • Perfect for experimenting with code, exploring data, and testing small-scale models. Its single-user environment keeps things organized. **Quick Iterations:** • Rapidly prototype and iterate on your machine learning models without waiting for clusters to spin up. **Limited Training Needs:** • If your training jobs are relatively small and don't require significant processing power, a compute instance can handle them efficiently. **Cost-Effectiveness:** • For smaller projects or those on a tight budget, a compute instance offers a more cost-friendly option compared to a cluster.	**Use a Compute Cluster:** **Large Datasets and Models:** • When dealing with massive datasets or complex models that require significant processing power, a cluster distributes the workload for faster training. **Scalability:** • Need to ramp up processing power for demanding jobs? A cluster allows you to easily scale the number of virtual machines to meet your needs. **Parallel Processing:** • Tasks that benefit from parallelization, like hyperparameter tuning or running multiple training jobs simultaneously, run much faster on a cluster. **Production Deployments:** • When deploying models for real-world use cases, a cluster provides the reliability and scalability required for handling production workload
Creating a Compute Instance: **Access Azure Machine Learning Studio:** Sign in to Azure and navigate to your Machine	**Creating a Compute Cluster:** **Navigate to Compute:** Follow steps 1 and 2 from creating a compute

Learning workspace.

Go to Compute:

Under the "Manage" section, select "Compute".

Create a New Instance:

Click the "+ New" button and choose "Compute instance".

Configure Settings:

Fill out the details like name, virtual machine size (consider resource requirements for your tasks), and optionally enable a managed identity for access control.

Review and Create:

Review your configuration and hit "Create" to provision the instance.

instance.

Choose Cluster:

Click the "+ New" button and select "Compute cluster".

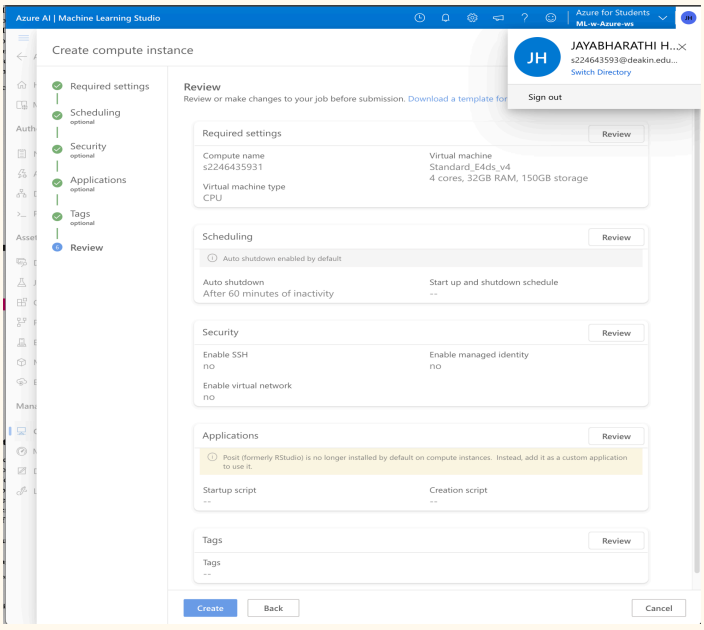
Configure Cluster Details:

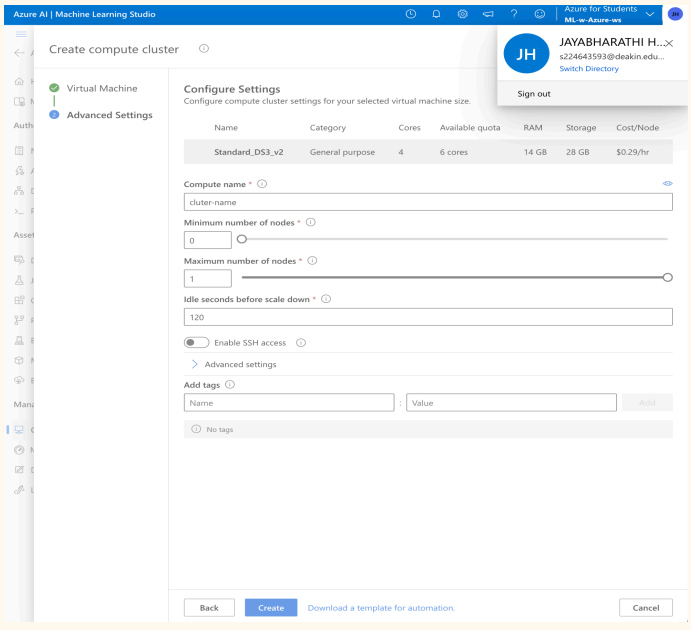
Here you'll define cluster properties like:

- **VM Size:** Choose the virtual machine size for each node in the cluster (based on processing needs).
- **Number of Nodes:** Specify the desired number of virtual machines to work together in the cluster (scales workload).
- **Other Settings:** Configure additional options like initial node count (minimum number of VMs running) and autoscale settings (automatic scaling based on workload).

Review and Create:

After customizing your cluster settings, review the configuration and click "Create" to provision the cluster.





Let's jump into Hands on - Azure Machine learning Python SDK

1. Set Up Azure Machine Learning
2. Train and Register the Model in Azure
3. Deploy the Registered Model
4. Model - Testing/Prediction

1.Set Up Azure Machine Learning

- Installing Azure Machine Learning SDK.

```
!pip install azureml-sdk
```

- Mounting google to gdrive since I'm using google colab.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

- Importing the dependencies.

Importing the dependencies

```
import os
import json
import requests

from azureml.core import Workspace
from azureml.core.model import Model
from azureml.core.environment import Environment
from azureml.core.conda_dependencies import CondaDependencies
from azureml.core.model import InferenceConfig
from azureml.core.webservice import AciWebservice, Webservice
```

- Loading the config file

  Config file has subscription and resource group details. Declared workspace and region info for workspace creation.

```
# laoding the configuration file
config_file_path = "/content/gdrive/MyDrive/MS_DataScience/ML-w-Azure/config.json"

# Read JSON data into a dictionary
with open(config_file_path, 'r') as file:
    data = json.load(file)

subscription_id = data["subscription_id"]
resource_group = data["resource_group"]
workspace_name = data["workspace_name"]
region = data["region"]
```

```
print(resource_group)
print(workspace_name)
print(region)
```

```
ML-w-Azure-rg
ML-w-Azure-ws
East US
```

- Creating the workspace.

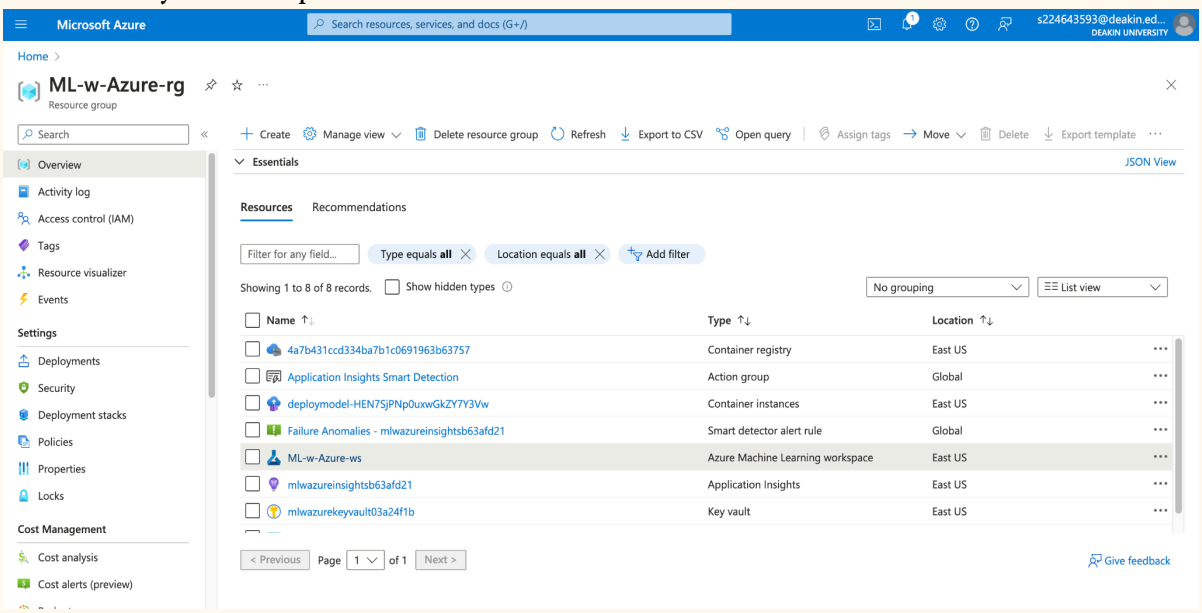
```
# Create a workspace

ws = Workspace.create(name=workspace_name,
                      subscription_id=subscription_id,
                      resource_group=resource_group,
                      location=region)

print(f'Workspace {workspace_name} created')
```

```
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 the page
Interactive authentication successfully completed.
Cleaning up past default Resource Group Deployments on the subscription to avoid limit of 10
Deleting past Resource Group Deployment with name: DeployResourceGroup-47aed8d534
Deploying StorageAccount with name mlwazurestoragefaaa0ad5e.
Deploying KeyVault with name mlwazurekeyvault03a24f1b.
Deploying Workspace with name ML-w-Azure-ws.
Deploying AppInsights with name mlwazureinsightsb63afd21.
Deployed AppInsights with name mlwazureinsightsb63afd21. Took 36.13 seconds.
Deployed Workspace with name ML-w-Azure-ws. Took 35.13 seconds.
Workspace ML-w-Azure-ws created
```

- We can verify in Azure portal that all resources were created.



2.Train and Register the Model in Azure

- Training and Pickling the Model

We need the best model which was trained and the same has to be saved in the pickled format for registering the model in azure. For that I'm using a diabetes dataset.

About the Dataset:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Column Level Details:

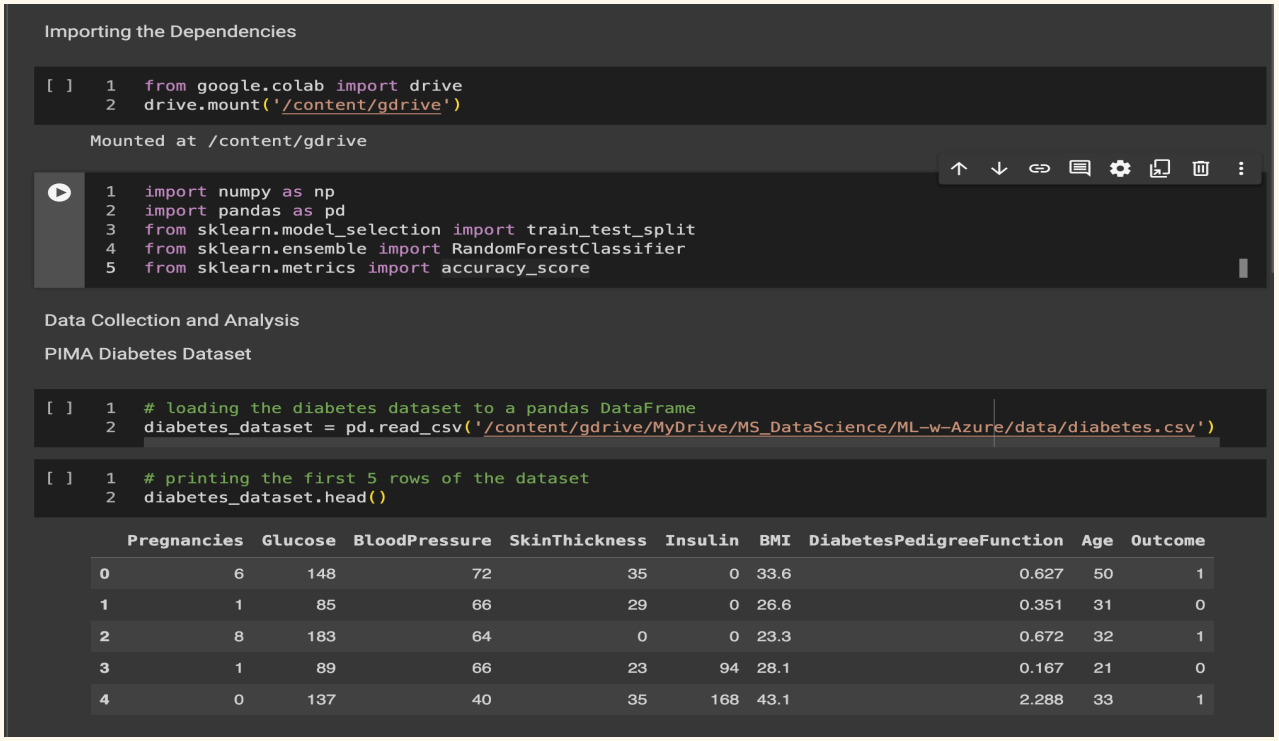
1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)

9. Outcome: Class variable (0 or 1)

Training the Model:

1. Loaded the dataset from gdrive
2. Did EDA and found its an imbalance dataset because class”0”(Non-Diabetic) is 500 in count and class”1”(Diabetic) is 268 in count. There are no duplicate rows.
3. Did Data Pre-Processing - Undersampling to get evenly distributed class to 268 in nos.
4. Did Train Test Split
5. Trained the model with RandomForestClassifier
6. Accuracy Score
7. Saved the Model in Pickle format.

Loaded the dataset from gdrive



Importing the Dependencies

```
[ ]  1  from google.colab import drive
     2  drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.model_selection import train_test_split
4  from sklearn.ensemble import RandomForestClassifier
5  from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

PIMA Diabetes Dataset

```
[ ]  1  # loading the diabetes dataset to a pandas DataFrame
     2  diabetes_dataset = pd.read_csv('/content/gdrive/MyDrive/MS_DataScience/ML-w-Azure/data/diabetes.csv')
```

```
[ ]  1  # printing the first 5 rows of the dataset
     2  diabetes_dataset.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

EDA and Data Pre-Processing:

```
[ ]  1  diabetes_dataset['Outcome'].value_counts()
```

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

0 --> Non-Diabetic

1 --> Diabetic

Undersample to get evenly distributed class

```
 1  df_class_0 = diabetes_dataset[diabetes_dataset["Outcome"]==0]
 2  df_class_1 = diabetes_dataset[diabetes_dataset["Outcome"]==1]
 3
 4  print("After undersampling", len(df_class_0), len(df_class_1))
 5
 6  df_class_0 = df_class_0.sample(len(df_class_1))
 7
 8  print("After undersampling" , len(df_class_0), len(df_class_1))
 9
10  diabetes_dataset = pd.concat([df_class_0, df_class_1])
```

```
After undersampling 500 268
After undersampling 268 268
```

```
[ ]  1  diabetes_dataset['Outcome'].value_counts()
```

```
Outcome
0    268
1    268
Name: count, dtype: int64
```

Train Test Split

Train Test Split

```
[ ]  1  X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=Y, random_state=2)
```

```
[ ]  1  print(X.shape, X_train.shape, X_test.shape)
```

```
(536, 8) (428, 8) (108, 8)
```

Model Training

Training the Model

```
[ ]  1
     2  classifier = RandomForestClassifier(n_estimators=100, random_state=0)
     3
     4  # classifier = svm.SVC(kernel='linear')
```

```
[ ]  1  #training the support vector Machine Classifier
     2  classifier.fit(X_train, Y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

Accuracy Score

Accuracy Score

```
[ ]  1  # accuracy score on the training data
     2  X_train_prediction = classifier.predict(X_train)
     3  training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[ ]  1  print('Accuracy score of the training data : ', training_data_accuracy)
```

```
Accuracy score of the training data :  1.0
```

```
[ ]  1  # accuracy score on the test data
     2  X_test_prediction = classifier.predict(X_test)
     3  test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
[ ]  1  print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data :  0.7592592592592593
```

Picking the Model.

Saving the trained model

```
import pickle

# Specify the path where you want to save the pickle file
SAVE_PATH = '/content/gdrive/MyDrive/MS_DataScience/ML-w-Azure'  # Adjust the path as needed

# Save the object as a pickle file
with open(SAVE_PATH + '/diabetes_model.pickle', 'wb') as outfile:
    pickle.dump(classifier, outfile)
```

```
# filename = 'diabetes_model.pkl'
# pickle.dump(classifier, open(filename, 'wb'))
# loading the saved model
filename = '/content/gdrive/MyDrive/MS_DataScience/ML-w-Azure/diabetes_model.pickle'
loaded_model = pickle.load(open(filename, 'rb'))
```

```
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = loaded_model.predict(input_data_reshaped)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
[1]
The person is diabetic
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifi
  warnings.warn(
```

- Registering the Pickled Model in Azure.

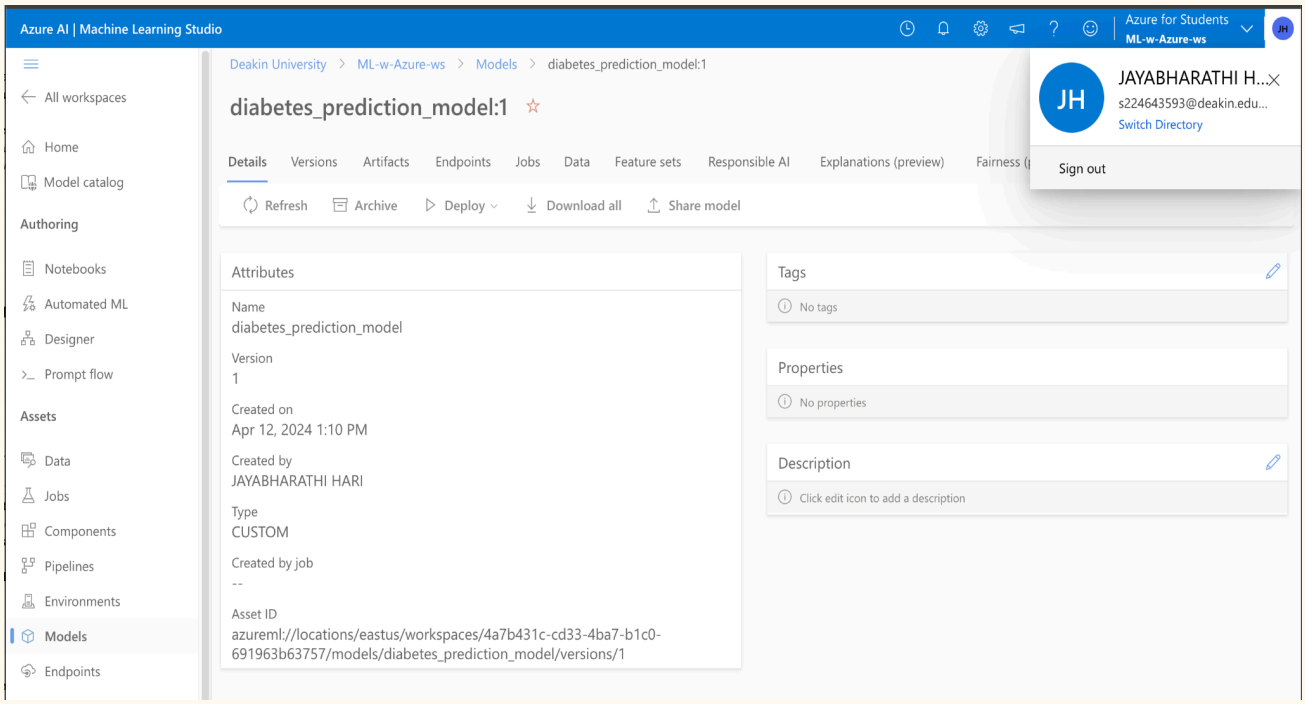
```
# Specify the path to your  model file
import pickle
model_path = '/content/gdrive/MyDrive/MS_DataScience/ML-w-Azure/diabetes_model.pickle'
# Load the pickled model
loaded_model = pickle.load(open(model_path, 'rb'))
```

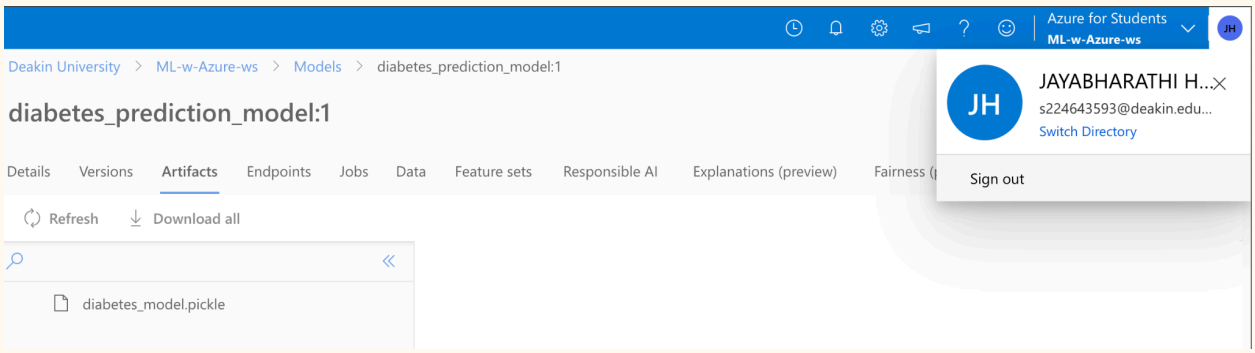
```
model_name='diabetes_prediction_model'
```

```
# Register the model in Azure Machine Learning
registered_model = Model.register(model_path=model_path, model_name=model_name, workspace=ws)
```

```
Registering model diabetes_prediction_model
```

We can see registered model details in the workspace in the name of “diabetes_prediction_model” , also we can see the pickled file





3.Deploy the Registered Model.

1.To deploy the Model , prerequisite files:

- Environment (.yaml) file
- Entry Script (score.py) file.

Environment (.yaml) file:

conda.yaml

```yaml
name: model-env
channels:
  - conda-forge
dependencies:
  - python=3.9
  - numpy=1.23.5
  - pip=23.0.1
  - scikit-learn=1.2.2
  - scipy=1.10.1
  - pip:
    - azureml-defaults==1.53.0
    - inference-schema[numpy-support]==1.5.1
    - joblib==1.2.0
```

Entry Script (score.py) file.

score.py

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

def init():
    global model
    model_path = Model.get_model_path('diabetes_prediction_model')
    model = joblib.load(model_path)

def run(raw_data):
    try:
        data = json.loads(raw_data)['data']
        data = np.array(data).reshape(1, -1)

        # Perform prediction using the loaded scikit-learn model
        result = model.predict(data)

        # You can return the result as a dictionary or in any desired format
        return json.dumps({"result": result.tolist()})
    except Exception as e:
        return json.dumps({"error": str(e)})
```

2. Deploying Model in Azure

We can deploy using python and via Azure Portal - Workspace. I have given below two ways.

- Deployment using python code

```
[ ]  1  cd /content/gdrive/MyDrive/Education/Masters_Data_Science/Azure_ML_SDK/
```

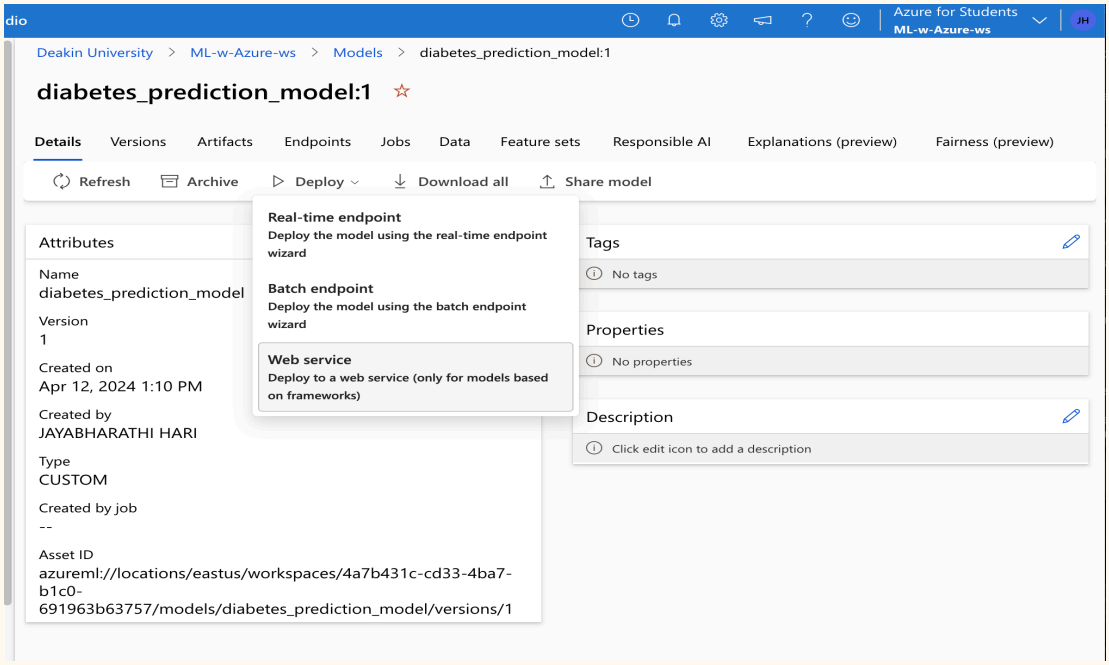
```
[ ]  1  import os
     2
     3  entry_script_path = os.path.join(os.getcwd(), 'score.py')
     4
     5  if not os.path.exists(entry_script_path):
     6      print('Entry script not found at:', entry_script_path)
```

```
[ ]  1
     2  # Load environment from Conda specification file
     3  env = Environment.from_conda_specification(name="azure_ml", file_path="/content/gdrive/MyDrive/MS_DataScience/ML-w-Azure/conda.yaml")
     4
     5  # Define inference configuration
     6  inference_config = InferenceConfig(entry_script="score.py", environment=env)
     7
     8  # Deploy the model as a web service
     9  deployment_config = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1)
    10  service = Model.deploy(workspace=ws,
    11                         name="diabetes-prediction-service", ##Endpoint Name
    12                         models=[model], ##model is variable of registered model
    13                         inference_config=inference_config,
    14                         deployment_config=deployment_config)
    15  service.wait_for_deployment(show_output=True)
```

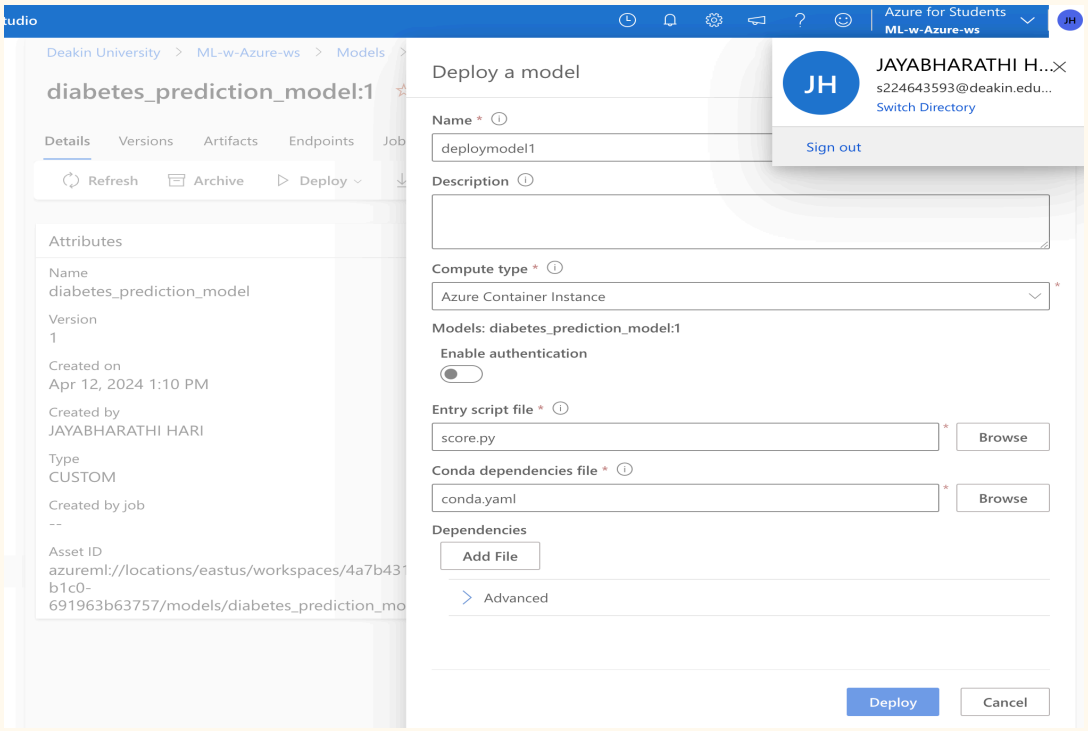
- Deployment via Azure Workspace

We are Deploying model as "Web Service" –

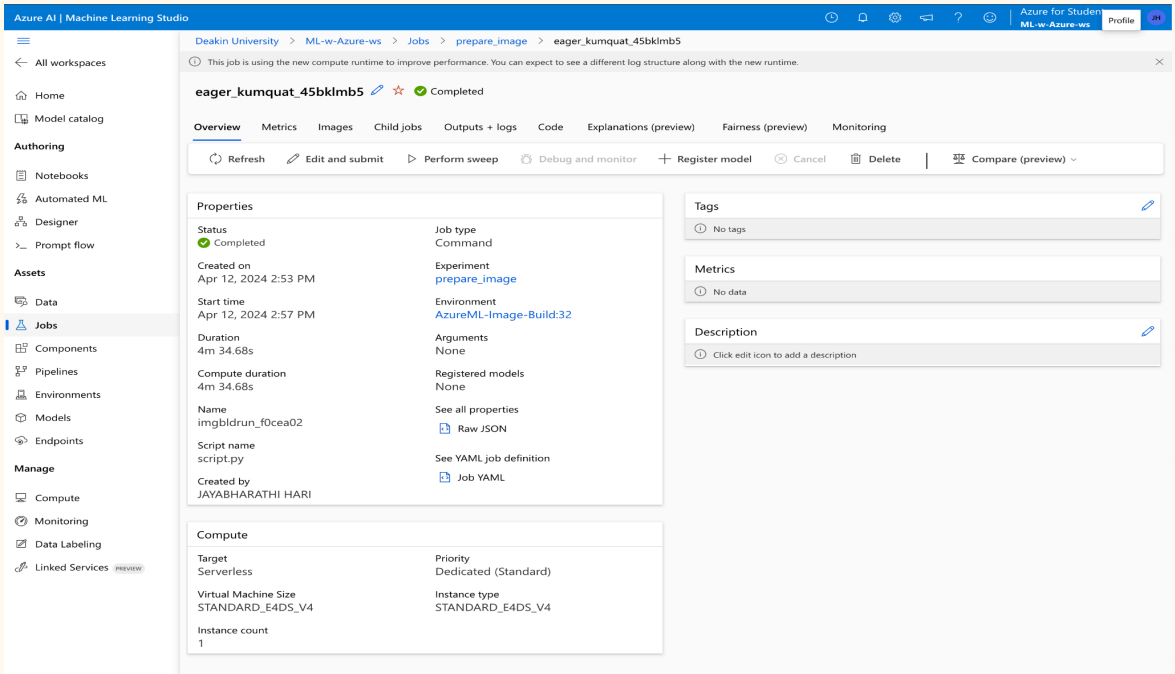
Step1: Model→Select Registered Model which wants to Deploy → Then Select "Deploy" → Select "Web Service" from the dropdown as given below.



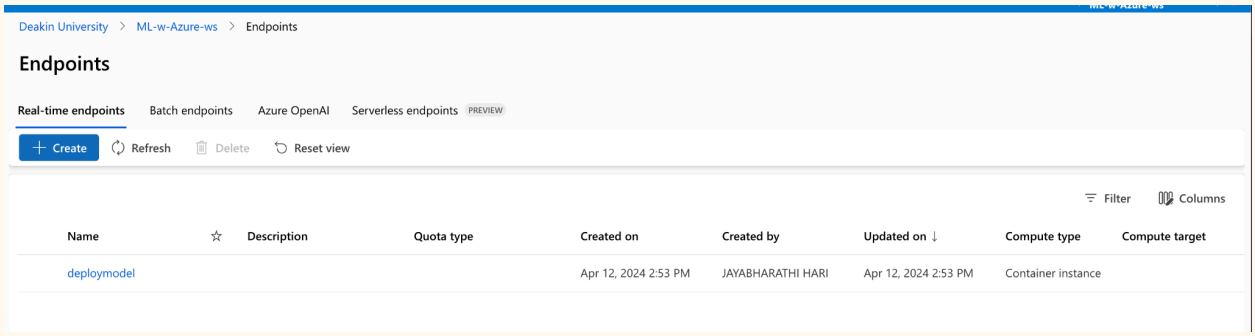
Step2: We need to provide "endpoint name", select compute instance, then upload an entry script and Environment file then deploy the model.



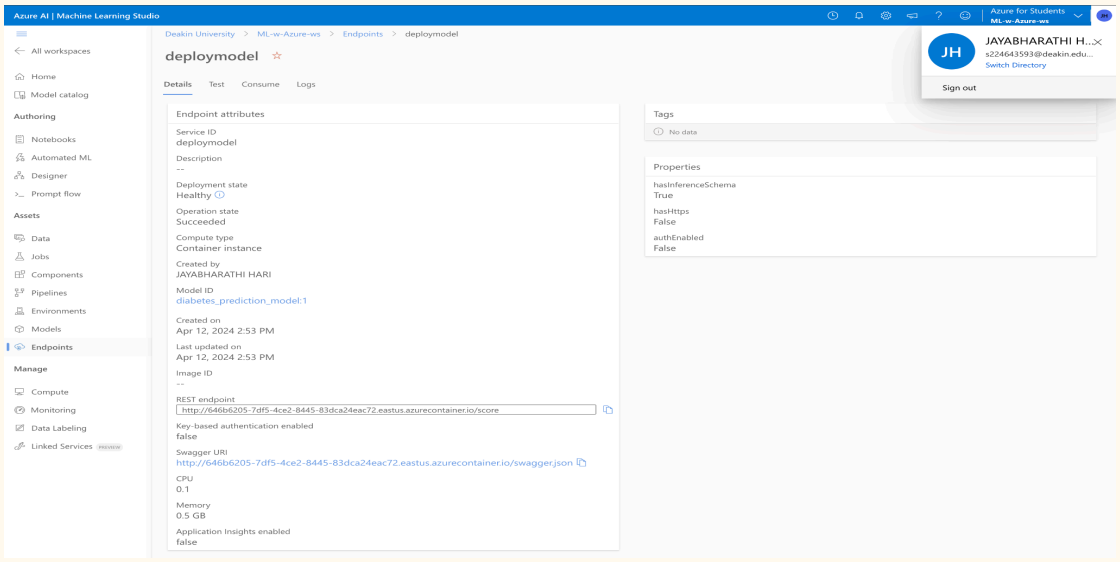
Step3: We can check Jobs as shown below.

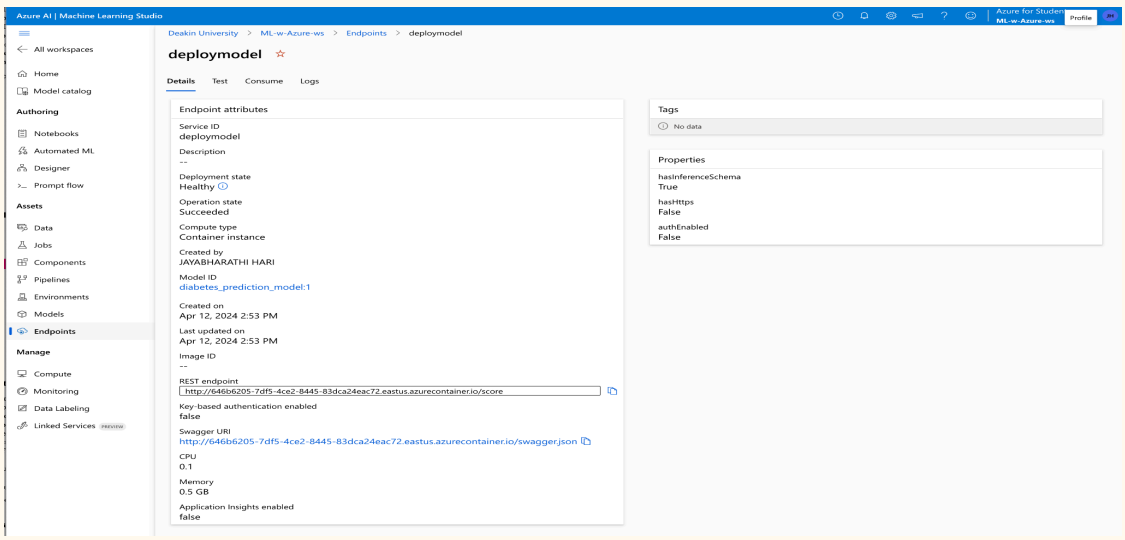


Step4: Once job(Preparing Image) is done, then we can see the Endpoint



Step5: we can use scoring uri for predicting.





3. Model - Testing/Prediction

- Testing with python code and scoring URI
    - This code sends an HTTP POST request to a deployed machine learning model hosted as a web service, passing input data in JSON format. The model processes the data and returns a prediction. If the response status is 200 (OK), it extracts and prints the prediction; otherwise, it prints the error message.

```python
import requests
import json

# Define the input data as a list
input_data = [5,166,72,19,175,25.8,0.587,51]


# Convert the input data to a JSON string
input_data_json = json.dumps({"data": [input_data]})

# Replace with the URL of your deployed service
scoring_uri = "http://646b6205-7df5-4ce2-8445-83dca24eac72.eastus.azurecontainer.io/score"

# Define the content type
headers = {"Content-Type": "application/json"}

# Send an HTTP POST request to the endpoint
response = requests.post(scoring_uri, data=input_data_json, headers=headers)

# Check the response status code
if response.status_code == 200:
    # Parse the JSON response
    result = json.loads(response.json())
    print(result)
    # Extract the prediction (result) from the response
    prediction = result["result"][0]
    print(f"Prediction: {prediction}")
else:
    print(f"Error: {response.text}")
```

```
{'result': [1]}
Prediction: 1
```

- Testing in Workspace - Azure Portal
  - We can predict/test in workspace-Azure Portal “Endpoint→Test→Past the Data”