



Cloud Web App & API Deployment - Iris Data Model

Week 5 Assignment

Name: Nazri

Batch code: LISUM34

Submission date: 04/07/2024

Submitted to: Data Glacier

Table of Contents

Flask Deployment of Iris Data Model	3
Web App Deployment of Iris data Model in Azure Cloud	14
API Deployment of Iris data Model in Azure Cloud	20

Flask Deployment of Iris Data Model

Steps Followed:

1. Pick Iris Toy Data Set

	A	B	C	D	E
1	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
2	5.1	3.5	1.4	0.2	Setosa
3	4.9	3	1.4	0.2	Setosa
4	4.7	3.2	1.3	0.2	Setosa
5	4.6	3.1	1.5	0.2	Setosa
6	5	3.6	1.4	0.2	Setosa
7	5.4	3.9	1.7	0.4	Setosa
8	4.6	3.4	1.4	0.3	Setosa
9	5	3.4	1.5	0.2	Setosa
10	4.4	2.9	1.4	0.2	Setosa
11	4.9	3.1	1.5	0.1	Setosa
12	5.4	3.7	1.5	0.2	Setosa
13	4.8	3.4	1.6	0.2	Setosa
14	4.8	3	1.4	0.1	Setosa
15	4.3	3	1.1	0.1	Setosa
16	5.8	4	1.2	0.2	Setosa
17	5.7	4.4	1.5	0.4	Setosa
18	5.4	3.9	1.3	0.4	Setosa
19	5.1	3.5	1.4	0.3	Setosa
20	5.7	3.8	1.7	0.3	Setosa
21	5.1	3.8	1.5	0.3	Setosa

2. Import necessary libraries:

Prepare the environment by installing necessary libraries like Scikit-learn and importing them. Also ensure the compatibility of Scikit-learn version with the IDE PyCharm

```
[23] !pip install scikit-learn==1.5.1
```

```
Requirement already satisfied: scikit-learn==1.5.1 in /usr/local/lib/python3.10/dist-packages (1.5.1)  
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (1.26.4)  
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (1.13.1)  
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (1.4.2)  
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.5.1) (3.5.0)
```

```
[2] # Import Libraries  
import pandas as pd  
import sklearn  
from sklearn.metrics import accuracy_score  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear_model import LogisticRegression  
from sklearn.model_selection import train_test_split  
import pickle  
import gdown
```

```
print(sklearn.__version__)
```

```
1.5.1
```

3. Downloading the Dataset:

Using 'gdown' download the Iris dataset from Google Drive.

4. Loading the Iris Dataset into a Dataframe:

The dataset is read from the CSV file using pandas from the contents folder.

```
✓ [4] #importing the dataset from drive
4s gdown.download_folder('https://drive.google.com/drive/folders/1Akoln8Xc14yMx01AXQw88YddMFyrfEft?', quiet=True)

['/content/Iris-Dataset/iris.csv']
```

```
✓ [5] #Load the dataset
0s iris_data = pd.read_csv('/content/Iris-Dataset/iris.csv')
```

5. Exploratory Data Analysis:

The head, shape, info, and isnull methods are used to inspect the dataset's structure, datatypes, and check for missing values.

```
✓ 0s iris_data.head()
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
✓ [7] iris_data.shape
0s
```

```
(150, 5)
```

```
✓ 0s iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Sepal_Length    150 non-null   float64
1   Sepal_Width     150 non-null   float64
2   Petal_Length    150 non-null   float64
3   Petal_Width     150 non-null   float64
4   Class           150 non-null   object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```


```
✓ [9] # check fo null vallues
0s iris_data.isnull().sum()
```

```
0
Sepal_Length 0
Sepal_Width  0
Petal_Length 0
Petal_Width  0
Class        0
```

```
dtype: int64
```

Check the summary statistics

```
✓ [13] iris_data.describe()
```



	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
count	149.000000	149.000000	149.000000	149.000000
mean	5.843624	3.059732	3.748993	1.194631
std	0.830851	0.436342	1.767791	0.762622
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.300000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Check for Duplicates

```
✓ [10] # Check for duplicates  
0s num_duplicates = iris_data.duplicated().sum()  
    print(f"Number of duplicate rows: {num_duplicates}")
```

```
➦ Number of duplicate rows: 1
```

```
✓ [11] # Identify duplicate records  
0s duplicates = iris_data[iris_data.duplicated()]  
  
# Print duplicate records  
print("Duplicate records:")  
print(duplicates)
```

```
➦ Duplicate records:  
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Class  
142           5.8           2.7           5.1           1.9  Virginica
```

Note: In the excel sheet, the duplicated data is aligned in the 103 and 144 row.

101	5.7	2.8	4.1	1.3	Versicolor
102	6.3	3.3	6	2.5	Virginica
103	5.8	2.7	5.1	1.9	Virginica
104	7.1	3	5.9	2.1	Virginica
105	6.3	2.9	5.6	1.8	Virginica
142	6.7	3.1	5.6	2.4	Virginica
143	6.9	3.1	5.1	2.3	Virginica
144	5.8	2.7	5.1	1.9	Virginica
145	6.8	3.2	5.9	2.3	Virginica
146	6.7	3.3	5.7	2.5	Virginica

6. Data Preprocessing:

Remove the duplicate rows identified during EDA process.

```

✓ [12] # Remove duplicate rows
0s iris_data = iris_data.drop_duplicates()

# Verify that duplicates are removed
num_duplicates_after = iris_data.duplicated().sum()
print(f"Number of duplicate rows after cleaning: {num_duplicates_after}")

```

➡ Number of duplicate rows after cleaning: 0

7. Splitting the Dataset

Select the features and target variables from the dataset and split the dataset into training and testing sets.

```

✓ [14] # Select independent and dependent variable
0s # Split the data into features and target
X = iris_data[["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]]
y = iris_data["Class"]

```

```

✓ [15] # Split the dataset into train and test
0s X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

8. Model Selection and Training the model:

Train the machine learning model on the pre-processed data with machine learning algorithms such as random forest classifier and logistic regression. Fit the model on the training data.

9. Model Evaluation:

Evaluate the model using the test set. And check its accuracy, precision, recall, and F1-score.

```

✓ [16] # Train the model with Random forest classifier
0s rfc = RandomForestClassifier()

# Fit the model
rfc.fit(X_train, y_train)

```

➡ **RandomForestClassifier** ⓘ ⓘ
RandomForestClassifier()

```

✓ [17] #Evaluate the model
0s y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Random Forest Model Accuracy:', accuracy)

```

➡ Random Forest Model Accuracy: 1.0

```

✓ [18] # Train the model with logistic regression
0s lg= LogisticRegression(max_iter=200)

# Fit the model
lg.fit(X_train, y_train)

```

LogisticRegression

LogisticRegression(max_iter=200)

```

✓ # Evaluate the model
0s y_pred = lg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Logistic Regression Model Accuracy:', accuracy)

```

Logistic Regression Model Accuracy: 1.0

```

✓ # Evaluate Random Forest
0s y_pred_rf = rfc.predict(X_test)
print('Random Forest Classification Report:\n', classification_report(y_test, y_pred_rf))

# Evaluate Logistic Regression
y_pred_lg = lg.predict(X_test)
print('Logistic Regression Classification Report:\n', classification_report(y_test, y_pred_lg))

```

Random Forest Classification Report:

	precision	recall	f1-score	support
Setosa	1.00	1.00	1.00	10
Versicolor	1.00	1.00	1.00	9
Virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Logistic Regression Classification Report:

	precision	recall	f1-score	support
Setosa	1.00	1.00	1.00	10
Versicolor	1.00	1.00	1.00	9
Virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Summary:

When both models achieve an accuracy of 1 on the test data, it might indicate that the models are overfitting, especially if the dataset is small or lacks complexity. Overfitting occurs when a model learns the training data too well, including noise and outliers, leading to poor generalization on unseen data.

10. Perform Cross-Validation on the models

To ensure that the models are truly generalizing well, we should use cross-validation. This involves splitting the dataset into multiple folds and training/evaluating the model on different folds. This process helps in assessing how the model performs across different subsets of the data.

```
✓ 2s [21] from sklearn.model_selection import cross_val_score

# Cross-validation for Random Forest
rf_cv_scores = cross_val_score(rfc, X_train, y_train, cv=5)
print('Random Forest Cross-Validation Scores:', rf_cv_scores)
print('Random Forest Mean CV Score:', rf_cv_scores.mean())

# Cross-validation for Logistic Regression
lg_cv_scores = cross_val_score(lg, X_train, y_train, cv=5)
print('Logistic Regression Cross-Validation Scores:', lg_cv_scores)
print('Logistic Regression Mean CV Score:', lg_cv_scores.mean())
```

⇒ Random Forest Cross-Validation Scores: [1. 0.91666667 0.875 1. 0.95652174]
Random Forest Mean CV Score: 0.9496376811594203
Logistic Regression Cross-Validation Scores: [1. 0.91666667 0.875 1. 0.95652174]
Logistic Regression Mean CV Score: 0.9496376811594203

11. Choosing the Best Model Random Forest:

Choose the model Random Forest Classifier considering the following features

- Handle Non-linearity
- Robustness to Outliers
- Handle large datasets and Complex Patterns

12. Save the trained model using pickle

```
✓ 0s [22] # Choosing the model
# Make pickle file of our model
pickle.dump(rfc, open("model.pkl", "wb"))
```

13. Setting up Flask Application

Create a Flask application (app.py).

- Load the saved model in the Flask app.
- Define routes for prediction, such as /predict.
- Use request to get input from the user and return predictions.

Create a HTML Template (index.html)

- In a templates directory, create index.html for user input

14. Running the Flask App:

- Run the Flask app locally.


```
"C:\Users\nazri_c98ckep\PycharmProject\Flask Deployment-Iris Data\venv\Scripts\python.exe" "C:\Users\nazri_c98ckep\PycharmProject\Flask Deployment-Iris Data\app.py"
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 420-096-619
```

Output

Iris Flower Species Prediction

Iris Flower Species Prediction

Sepal Length
Enter Sepal Length

Sepal Width
Enter Sepal Width

Petal Length
Enter Petal Length

Petal Width
Enter Petal Width

Predict

Iris Flower Species Prediction

Iris Flower Species Prediction

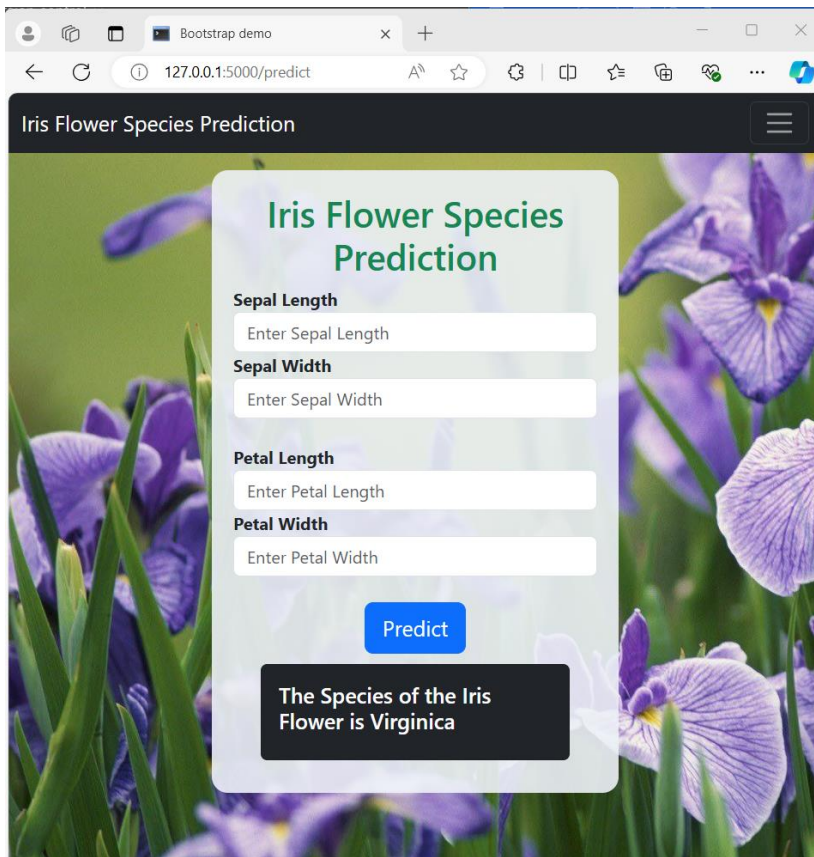
Sepal Length
5.8

Sepal Width
2.7

Petal Length
5.1

Petal Width
1.9

Predict



app.py

```
from flask import Flask, request, render_template
import numpy as np
import pickle
import sklearn
import pandas as pd

# importing model
model = pickle.load(open('model.pkl', 'rb'))

# creating flask app
app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

@app.route("/predict", methods=['POST'])
def predict():
    # Collecting input features from the form
    float_features = [float(x) for x in request.form.values()]
    # print("Received input features:", float_features) # Debugging

    # Defining the feature names as used during the model training
    feature_names = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']

    # Creating a DataFrame with the feature names
```

```

features = pd.DataFrame([float_features], columns=feature_names)
# print("DataFrame created:", features) # Debugging

# Making predictions
prediction = model.predict(features)
# print("Prediction:", prediction) # Debugging

# Rendering the template with the prediction result
return render_template("index.html", prediction_text="The Species of the Iris Flower is
{}".format(prediction[0]))

# Main function to run the Flask app
if __name__ == "__main__":
    app.run(debug=True)

```

index.html

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Bootstrap demo</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"
crossorigin="anonymous">
</head>
<style>
  body {
    background-image: url('{{ url_for('static', filename='img.jpeg') }}');
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
  }
  h1 {
    color: #BE2ED6;
    text-align: center;
  }
  .warning {
    color: red;
    font-weight: bold;
    text-align: center;
  }
  .card {
    margin: 10px auto;
    color: white;
  }
  .container {
    background: rgba(237, 242, 247, 0.9); /* Semi-transparent background */
    font-weight: bold;

```

```

padding: 20px; /* Increased padding for better spacing */
border-radius: 15px;
width: 50%; /* Set width to 50% of the viewport */
max-width: 600px; /* Maximum width */
margin: 0 auto; /* Center the container horizontally */
}
</style>

<body>
  <!--
=====navbar=====
=====-->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container-fluid">
      <a class="navbar-brand" href="/">Iris Flower Species Prediction</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <a class="nav-link active" aria-current="page" href="#">Home</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">Contact</a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">About</a>
          </li>
        </ul>
        <form class="d-flex" role="search">
          <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
          <button class="btn btn-outline-success" type="submit">Search</button>
        </form>
      </div>
    </div>
  </nav>

  <!--
=====
=====-->
  <div class="container my-3 mt-3">
    <h1 class="text-success">Iris Flower Species Prediction<span class="text-success"></span></h1>

    <!-- adding form -->
    <form action="/predict" method="POST">
      <div class="row">
        <div class="col-md-6">
          <label for="Sepal Length">Sepal Length</label>

```

```

        <input type="text" id="Sepal_Length" name="Sepal Length" placeholder="Enter Sepal Length"
class="form-control" required="required">
    </div>
    <div class="col-md-6">
        <label for="Sepal Width">Sepal Width</label>
        <input type="text" id="Sepal_Width" name="Sepal Width" placeholder="Enter Sepal Width"
class="form-control" required="required">
    </div>
</div>

<div class="row mt-4">
    <div class="col-md-6">
        <label for="Petal Length">Petal Length</label>
        <input type="text" id="Petal_Length" name="Petal Length" placeholder="Enter Petal Length"
class="form-control" required="required">
    </div>
    <div class="col-md-6">
        <label for="Petal Width">Petal Width</label>
        <input type="text" id="Petal_Width" name="Petal Width" placeholder="Enter Petal Width"
class="form-control" required="required">
    </div>
</div>

<div class="row mt-4">
    <div class="col-md-12 text-center">
        <button type="submit" class="btn btn-primary btn-lg">Predict</button>
    </div>
</div>
</form>

{% if prediction_text %}
<div class="card bg-dark" style="width: 18rem;">
    <div class="card-body">
        <h5 class="card-title">{{ prediction_text }}</h5>
    </div>
</div>
{% endif %}
</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4ddQ3GYNS7NTKfAdVQSZe"
crossorigin="anonymous"></script>
</body>
</html>

```

Web App Deployment of Iris data Model in Azure Cloud

Web App Deployment: This means the model should also be accessible via a web interface where users can input data directly on a webpage and see the predictions. This could be a simple web page (built with HTML/CSS and perhaps some JavaScript) with a form where users can enter data, click a button, and see the results. This web app will also be hosted on Azure, so anyone with the URL can access it via their web browser.

GitHub Repository of the Source Code

Link: <https://github.com/NazriJasmal/Azure-Deployment.git>

The screenshot shows the GitHub repository page for 'NazriJasmal / Azure-Deployment'. The repository is public and has 11 commits. The file list includes:

File	Commit Message	Time
.github/workflows	Add or update the Azure App Servi...	2 days ago
datasets	Uploaded Code	2 days ago
notebooks	Uploaded Code	2 days ago
static	Uploaded Code	2 days ago
templates	Uploaded Code	2 days ago
README.md	Update README.md	now
app.py	Uploaded Code	2 days ago
model.pkl	Uploaded Code	2 days ago
requirements.txt	Uploaded Code	2 days ago

The README file is selected and shows the title 'Azure Web App Deployment of Iris Data Model'. The right sidebar contains information about the repository, including the number of stars (0), forks (0), and a deployment status (Production 2 days ago). The languages section shows the following distribution:

Language	Percentage
Jupyter Notebook	95.3%
HTML	3.7%
Python	1.0%

Deploy the Flask application to Azure App Service: This involves creating an Azure Web App, configuring the deployment settings, and deploying the application code.

Steps Followed:

- 1) Prepare the Flask Application
 - Make sure the Flask app (app.py) is working locally.
 - Create requirements.txt: To generate a requirements.txt file listing all the project's dependencies, run the command `pip freeze > requirements.txt` in the terminal of IDE Pycharm.
- 2) Log in to the Azure Portal with the Microsoft account
- 3) Create a new Web App
Navigate to **Create a Resource > Web > Web App > Click Create**
- 4) Configure the Web App
 - Subscription: Select the Azure subscription.
 - Resource Group: Create a new resource group or select an existing one.
 - Name: Enter a name for the Web App. This will be part of your web app's URL.
 - Publish: Select Code.
 - Runtime Stack: Choose the version of Python that matches with the local environment, which is Python 3.9. (To get this, Run the command 'python --version' in the terminal of IDE Pycharm.)
 - Region: Select the Azure region where we want to deploy the app.
 - Go to Deployment tab and enable continuous deployment and basic authentication.
 - Connect to GitHub account: Select the repository and branch that contains the source code of the Flask app.
 - Click Review + Create, then Create.

Microsoft... Search resources, services, and docs (G+/) Copilot nazrijasmal@outlook.com DEFAULT DIRECTORY (NAZRIJAS...)

Home > Create a resource >

Create Web App

Basics Database Deployment Networking Monitor + secure Tags Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

Project Details

Select a 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 * Azure_deployment [Create new](#)

Instance Details

Name * IrisSpeciesIdentifierWebApp ✓
-cgc7hda6eufne8f8.uksouth-01.azurewebsites.net

☒ Unique default hostname (preview) on. [More about this update](#)

Publish * ☒ Code ☐ Container ☐ Static Web App

Runtime stack * Python 3.9

Operating System * ☒ Linux ☐ Windows

Region * UK South

Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#)

Linux Plan (UK South) * ASP-Azuredeployment-8025 (B1) [Create new](#)

[Review + create](#) < Previous Next : Database >

Home > Create a resource >

Create Web App

Basics Database **Deployment** Networking Monitor + secure Tags Review + create

Continuous deployment settings

Set up continuous deployment to easily deploy code from your GitHub repository via GitHub Actions. [Learn more](#)

Continuous deployment ☐ Disable ☒ Enable

GitHub settings

Set up GitHub Actions to push content to your app whenever there are code changes made to your repository. Note: Your GitHub account must have write access to the selected repository in order to add a workflow file which manages deployments to your app.

GitHub account NazriJasmal [Change account](#)

Organization * NazriJasmal

Repository * Azure-Deployment

Branch * main

Workflow configuration

Click the button below to preview what the GitHub Actions workflow file will look like before setting up continuous deployment.

[Preview file](#)

Authentication settings

Choose if you would like to allow basic authentication to deploy code to your app. [Learn more](#)

Basic authentication ☐ Disable ☒ Enable

[Review + create](#) [< Previous](#) [Next : Networking >](#)

5) Access the Web App

After creation, go to the Resource page of the new Web App.

6) Deployment will be in progress as we have already enabled continuous deployment.

Enabling continuous deployment in Azure allows your code changes to be automatically updated and deployed to your app or service, without needing manual intervention.

7) If Continuous Deployment is not enabled

- Go to the Resource page of the Web App. Then select Deployment > Deployment Centre, in the left-hand menu.
- Then Select Settings and enter the details of the GitHub Repo and branch that contains the source code of the Flask app. Then Click Save

Microsoft Azure

Search resources, services, and docs (G+/)

Copilot

Home > IrisSpeciesIdentifierApp

IrisSpeciesIdentifierApp | Deployment Center

Web App

Search

Save Discard Browse Manage publish profile Sync Leave Feedback

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Better Together (preview)

Log stream

Deployment

Deployment slots

Deployment Center

Performance

Settings

Environment variables

Configuration

Authentication

Application Insights

Identity

Backups

Custom domains

Certificates

Networking

Settings * Logs FTPS credentials

You're now in the production slot, which is not recommended for setting up CI/CD. Learn more

Deploy and build code from your preferred source and build provider. Learn more

Source *
GitHub

Building with GitHub Actions. Change provider.

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions. Learn more

Signed in as
NazriJasmal Change Account

Organization *
NazriJasmal

Repository *
Azure-Deployment

Branch *
main

Workflow Option *

Microsoft Azure

Search resources, services, and docs (G+/)

Copilot

Home > IrisSpeciesIdentifierApp

IrisSpeciesIdentifierApp | Deployment Center

Web App

Search

Save Discard Browse Manage publish profile Sync Leave Feedback

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Better Together (preview)

Log stream

Deployment

Deployment slots

Deployment Center

Performance

Settings

Environment variables

Configuration

Authentication

Application Insights

Identity

Backups

Custom domains

Certificates

Networking

Workflow Option *

☒ Add a workflow: Add a new workflow file 'main_IrisSpeciesIdentifierApp.yml' in the selected repository and branch.

☐ Use available workflow: Use one of the workflow files available in the selected repository and branch.

Build

Runtime stack
Python

Version
Python 3.9

Authentication settings

Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. Learn more

Authentication type *
☒ User-assigned identity
☐ Basic authentication

Subscription *
Azure subscription 1

Identity *
(Create new)

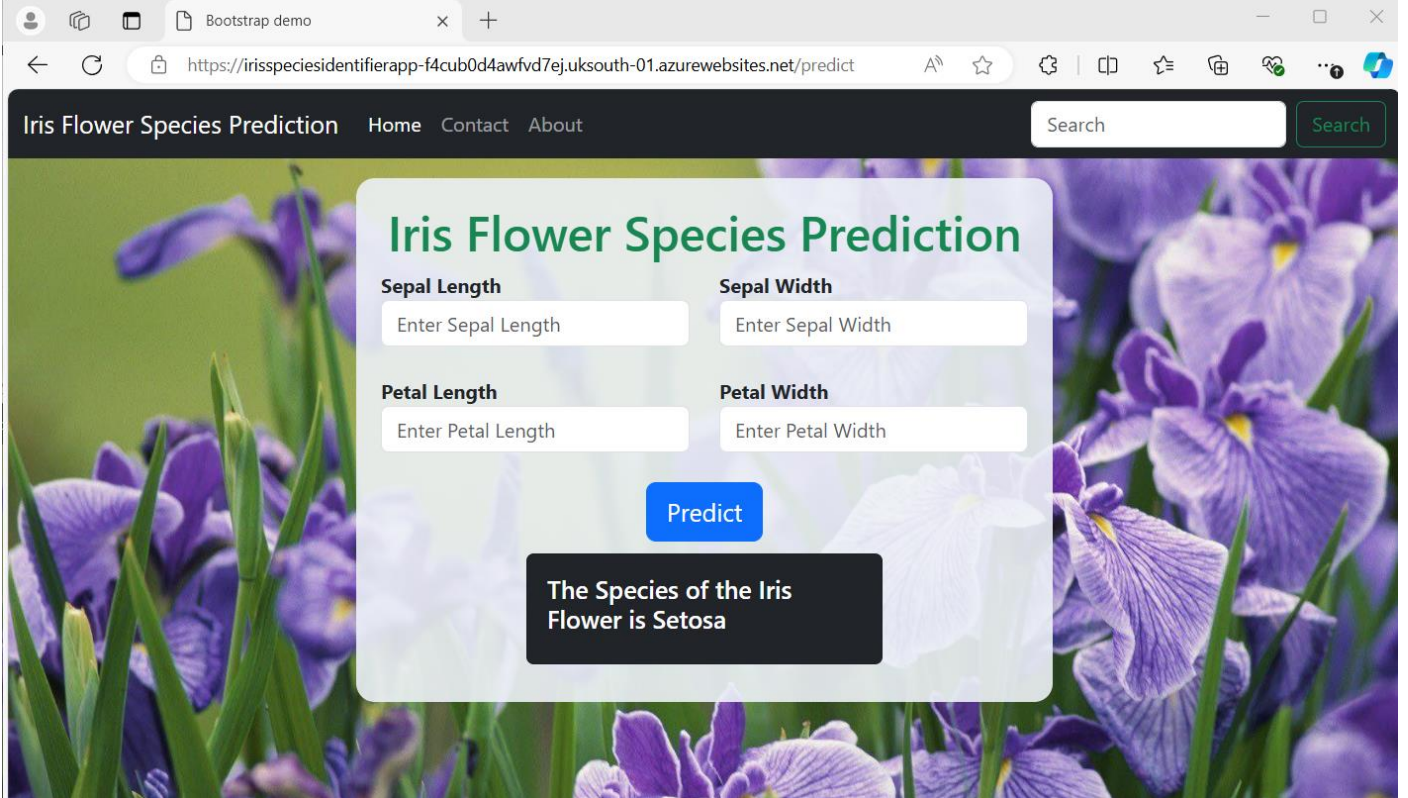
8) Go to the Overview tab and we can see Deployment is successful and the Web App is Created Successfully.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Microsoft Azure logo, an 'Upgrade' button, a search bar, and a 'Copilot' button. The user's profile is 'nazrijasmal@outlook.com'. The main content area is titled 'IrisSpeciesIdentifierApp' and shows the 'Overview' tab. The left sidebar contains a navigation menu with options like Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Better Together (preview), Log stream, Deployment, Deployment slots, Deployment Center, Performance, Settings, Environment variables, Configuration, Authentication, Application Insights, Identity, and Backups. The main content area is divided into several sections: 'Essentials' (Resource group, Status, Location, Subscription, Tags), 'Properties' (Web app, Domains, Hosting), 'Deployment Center' (Deployment logs, Last deployment, Deployment provider), 'Application Insights' (Name), and 'Networking' (Virtual IP address, Outbound IP addresses, Additional Outbound IP addresses). The 'Web app' section shows the name 'IrisSpeciesIdentifierApp', publishing model 'Code', runtime stack 'Python - 3.9', and app service plan 'ASP-Azuredeployment-8025'. The 'Deployment Center' section shows a successful deployment on Thursday 5 September, 07:05:46 PM, using GitHubAction as the deployment provider.

Output

Azure Web App Link: <https://irisspeciesidentifierapp-f4cub0d4awfvd7ej.uksouth-01.azurewebsites.net/>

The screenshot shows a web application titled 'Iris Flower Species Prediction'. The background is a close-up image of purple iris flowers. The application has a dark header with the title 'Iris Flower Species Prediction' and navigation links 'Home', 'Contact', and 'About'. A search bar is located in the top right corner. The main content area features a light blue overlay with the title 'Iris Flower Species Prediction' in green. Below the title are four input fields for 'Sepal Length', 'Sepal Width', 'Petal Length', and 'Petal Width'. The values entered are 5.1, 2.7, 1.4, and 1.8 respectively. A blue 'Predict' button is located at the bottom of the overlay.



API Deployment of Iris data Model in Azure Cloud

API Deployment: This means the model should be accessible through an HTTP endpoint where we can send data and receive predictions. The model is deployed in such a way that it runs in the background, waiting for incoming HTTP requests.

We can use Azure Machine Learning or Azure Web App Services that supports RESTful APIs to expose the model as an API. When we deploy the model as an API, other developers or applications can send data to the API endpoint, and the model will return predictions.

GitHub Repository of the Source Code

Link: <https://github.com/NazriJasmal/AzureDeployment-API.git>

The screenshot shows the GitHub repository page for 'NazriJasmal / AzureDeployment-API'. The repository is public and has 1 branch and 0 tags. The main branch is selected. The repository contains several files and folders, including .github/workflows, datasets, notebooks, README.md, api_app.py, model.pkl, requirements.txt, startup.txt, and test_api.py. The README.md file is open, showing the title '# AzureDeployment-API'. The right sidebar shows the repository's activity, including 0 stars, 1 watching, and 0 forks. The 'Releases' section shows no releases published. The 'Packages' section shows no packages published. The 'Deployments' section shows 3 deployments, with the latest one being 'Production' 5 hours ago. The 'Languages' section shows a bar chart with Jupyter Notebook at 98.4% and Python at 1.6%.

File/Folder	Commit Message	Commit Time
.github/workflows	Add or update the Azure App Service build ...	yesterday
datasets	uploaded the code	yesterday
notebooks	uploaded the code	yesterday
README.md	first commit	yesterday
api_app.py	Update api_app.py	5 hours ago
model.pkl	uploaded the code	yesterday
requirements.txt	uploaded the code	yesterday
startup.txt	uploaded the code	yesterday
test_api.py	uploaded the code	yesterday

Steps Followed:

1. Create an API Application
 - Create dedicated Flask app for the API. This app will only serve the API endpoint and will be deployed separately from the web app.
 - Create a new Python file, `api_app.py`, that contain the API code

api_app.py

```
from flask import Flask, request, jsonify
import numpy as np
import pickle
import pandas as pd

# Import the trained model
model = pickle.load(open('model.pkl', 'rb'))

# Create the Flask app for the API
app = Flask(__name__)

@app.route("/")
def home():
    # Root route for testing
    return "API is up and running!"

@app.route("/api/predict", methods=['POST'])
def api_predict():
    # Expecting a JSON payload
    data = request.get_json(force=True)

    # Extract features from JSON (assuming data['data'] is a list of values)
    features = pd.DataFrame([data['data']], columns=data['columns'])

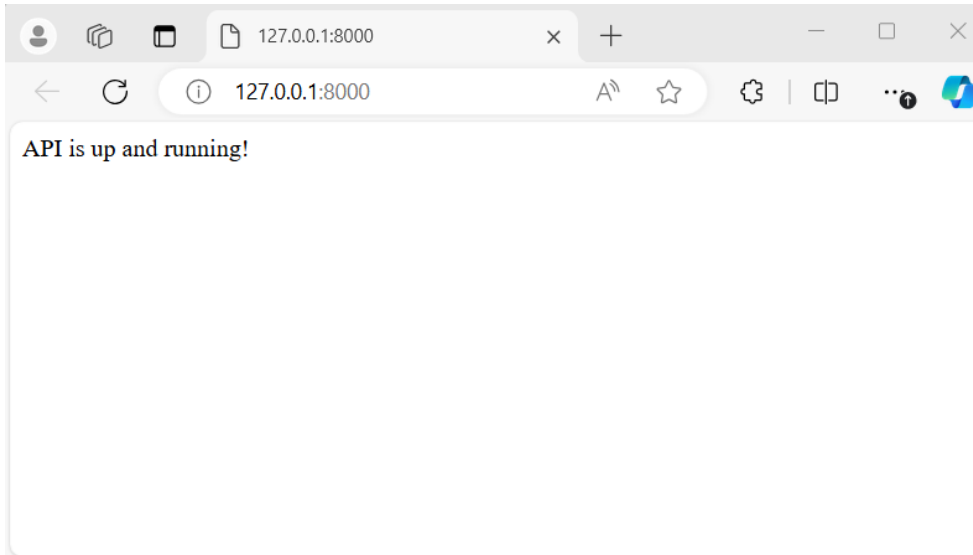
    # Make predictions
    prediction = model.predict(features)

    # Return the prediction in JSON format
    return jsonify({'prediction': prediction[0]})

# Main function to run the API Flask app
if __name__ == "__main__":
    app.run(host='127.0.0.1', port=8000)
```

2. Make sure the Flask app (api_app.py) is working locally.

```
Run api_app x
"C:\Users\nazri_c98ckep\PycharmProject\Flask Deployment-Iris Data\venv\Scripts\python.exe" "C:\Users\nazri_c98ckep\PycharmProject\API Deployment-Iris Data\api_app.py"
* Serving Flask app 'api_app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:8000
Press CTRL+C to quit
```



Testing the API Endpoint

- By Creating a test_api.py: Test Success

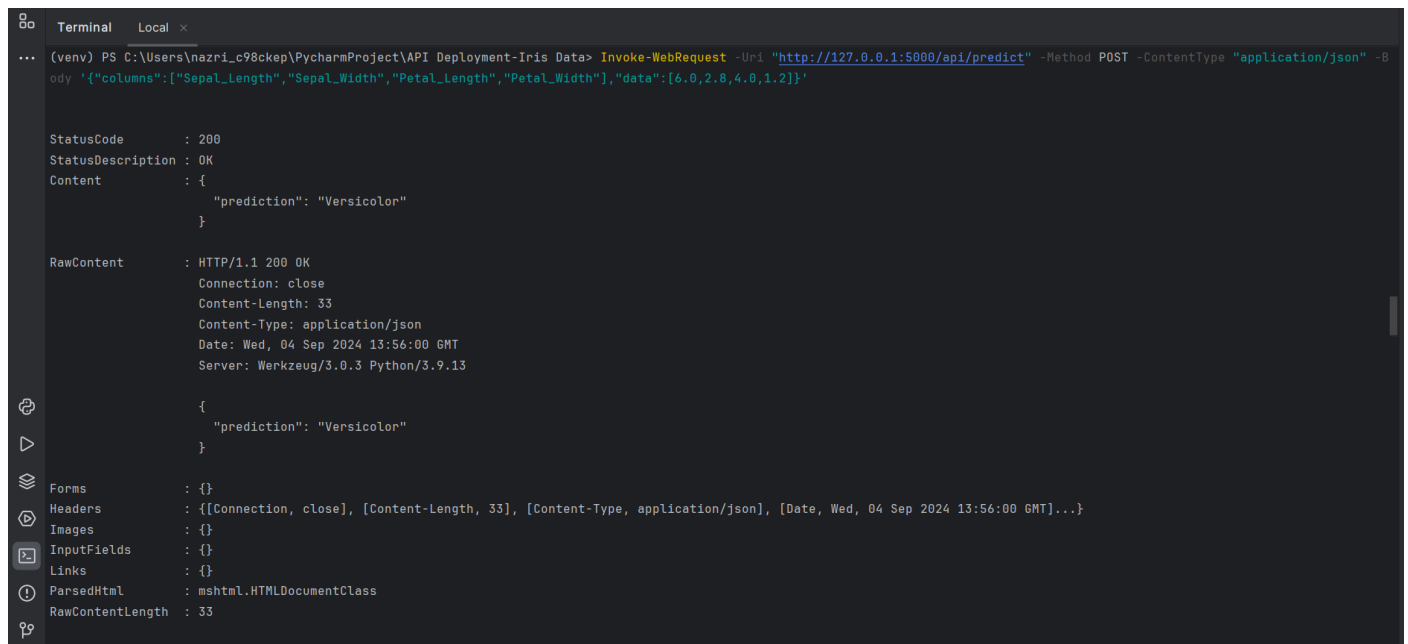
```
test_api.py x
1 import requests
2 import json
3
4 # API URL
5 url = 'http://127.0.0.1:5000/api/predict'
6
7 # Data to be sent
8 data = {
9     "columns": ["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"],
10    "data": [5.1, 3.5, 1.4, 0.2]
11 }
12
13 # Send POST request
14 response = requests.post(url, json=data)
15
16 # Print response
17 print("Status Code:", response.status_code)
18 print("Response JSON:", response.json())
19
```

```
Run test_api x
"C:\Users\nazri_c98ckep\PycharmProject\Flask Deployment-Iris Data\venv\Scripts\python.exe" "C:\Users\nazri_c98ckep\PycharmProject\API Deployment-Iris Data\test_api.py"
Status Code: 200
Response JSON: {'prediction': 'Setosa'}
Process finished with exit code 0
```


- **By Invoking Web Request: Test Success**

Test Case 1:

Invoke-WebRequest -Uri "http://127.0.0.1:5000/api/predict" -Method POST -ContentType "application/json" -Body '{"columns":["Sepal_Length","Sepal_Width","Petal_Length","Petal_Width"],"data":[6.0,2.8,4.0,1.2]}'



```
(venv) PS C:\Users\nazri_c98ckep\PycharmProject\API Deployment-Iris Data> Invoke-WebRequest -Uri "http://127.0.0.1:5000/api/predict" -Method POST -ContentType "application/json" -Body '{"columns":["Sepal_Length","Sepal_Width","Petal_Length","Petal_Width"],"data":[6.0,2.8,4.0,1.2]}'

StatusCode      : 200
StatusDescription : OK
Content         : {
                  "prediction": "Versicolor"
                }

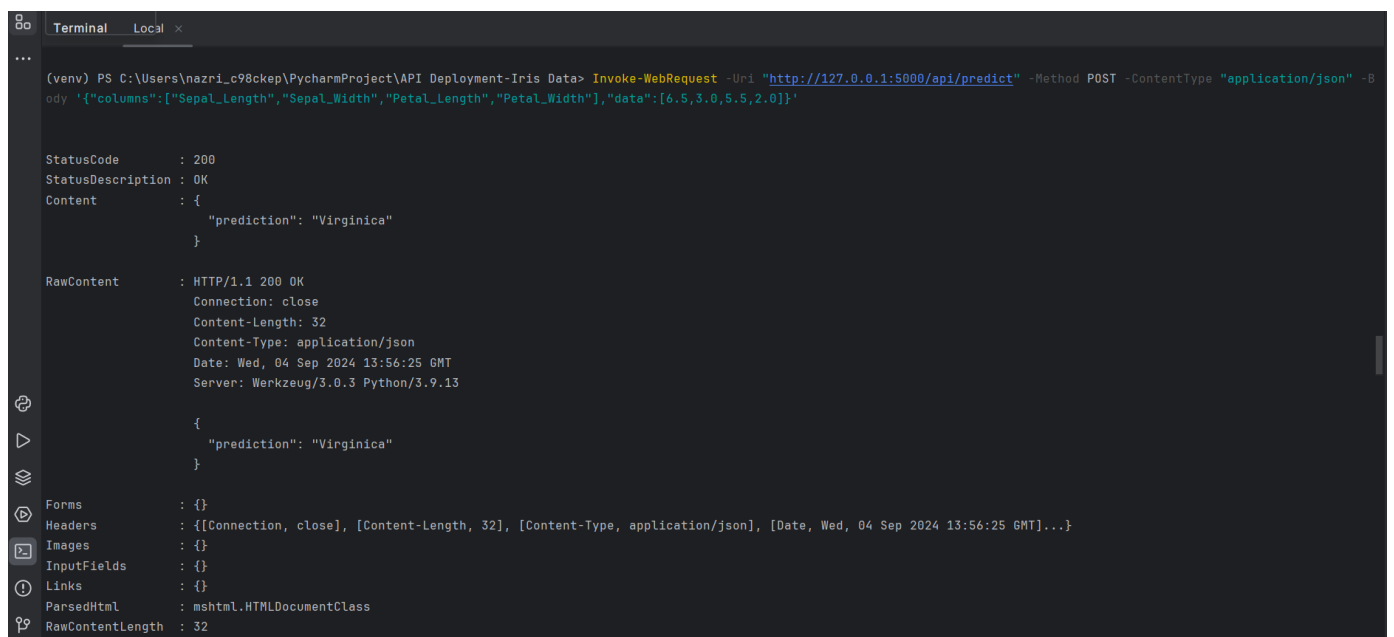
RawContent      : HTTP/1.1 200 OK
                  Connection: close
                  Content-Length: 33
                  Content-Type: application/json
                  Date: Wed, 04 Sep 2024 13:56:00 GMT
                  Server: Werkzeug/3.0.3 Python/3.9.13

{
  "prediction": "Versicolor"
}

Forms          : {}
Headers        : {[Connection, close], [Content-Length, 33], [Content-Type, application/json], [Date, Wed, 04 Sep 2024 13:56:00 GMT]...}
Images         : {}
InputFields    : {}
Links          : {}
ParsedHtml     : mshtml.HTMLDocumentClass
RawContentLength : 33
```

Test Case 2:

Invoke-WebRequest -Uri "http://127.0.0.1:5000/api/predict" -Method POST -ContentType "application/json" -Body '{"columns":["Sepal_Length","Sepal_Width","Petal_Length","Petal_Width"],"data":[6.5,3.0,5.5,2.0]}'



```
(venv) PS C:\Users\nazri_c98ckep\PycharmProject\API Deployment-Iris Data> Invoke-WebRequest -Uri "http://127.0.0.1:5000/api/predict" -Method POST -ContentType "application/json" -Body '{"columns":["Sepal_Length","Sepal_Width","Petal_Length","Petal_Width"],"data":[6.5,3.0,5.5,2.0]}'

StatusCode      : 200
StatusDescription : OK
Content         : {
                  "prediction": "Virginica"
                }

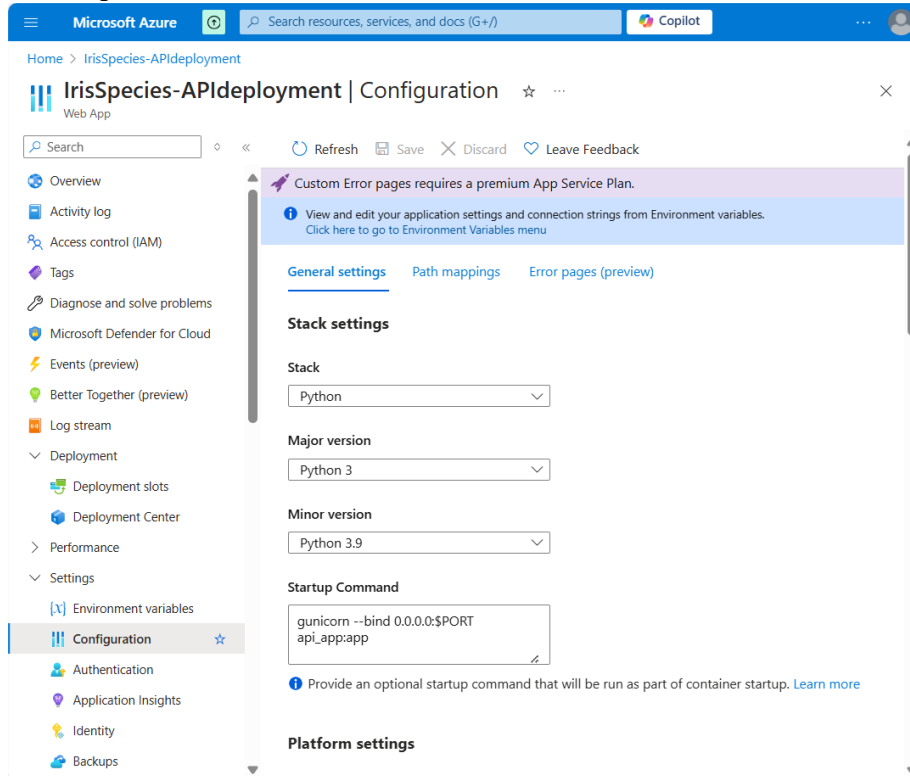
RawContent      : HTTP/1.1 200 OK
                  Connection: close
                  Content-Length: 32
                  Content-Type: application/json
                  Date: Wed, 04 Sep 2024 13:56:25 GMT
                  Server: Werkzeug/3.0.3 Python/3.9.13

{
  "prediction": "Virginica"
}

Forms          : {}
Headers        : {[Connection, close], [Content-Length, 32], [Content-Type, application/json], [Date, Wed, 04 Sep 2024 13:56:25 GMT]...}
Images         : {}
InputFields    : {}
Links          : {}
ParsedHtml     : mshtml.HTMLDocumentClass
RawContentLength : 32
```

3. Prepare the Flask Application

- Create requirements.txt: To generate a requirements.txt file listing all the project's dependencies, run the command `pip freeze > requirements.txt` in the terminal of IDE Pycharm
- Configure Startup Command: Go to Settings > Configuration > General settings > Startup Command. Enter the command to start the Flask app as `gunicorn --bind 0.0.0.0:$PORT api_app:app`. This tells Azure to use gunicorn to serve the Flask app where \$PORT is a placeholder that Azure uses for the port.



- Update app.run()
Update `app.run(host='127.0.0.1', port=8000)` to `app.run(host='0.0.0.0', port=8000)` was made, to ensure that the Flask application is accessible externally when deployed on Azure.
 - `127.0.0.1` binds the app to the local loopback interface, making it accessible only from the local machine (localhost).
 - `0.0.0.0` binds the app to all available network interfaces, allowing it to be accessed externally over the network, which is required when deploying to a cloud platform like Azure.
4. Log in to the Azure Portal with the Microsoft account
 5. Create a new Web App
 6. Navigate to Create a Resource > Web > Web App > Click Create
 7. Configure the Web App
 - Subscription: Select the Azure subscription.
 - Resource Group: Create a new resource group or select an existing one.
 - Name: Enter a name for the Web App. This will be part of your web app's URL.
 - Publish: Select Code.
 - Runtime Stack: Choose the version of Python that matches with the local environment, which is Python 3.9. (To get this, Run the command '`python --version`' in the terminal of IDE Pycharm.)

- **Region:** Select the Azure region where we want to deploy the app.
- Go to Deployment tab and enable continuous deployment and basic authentication.
- **Connect to GitHub account:** Select the repository and branch that contains the source code of the Flask app.
- Click Review + Create, then Create.

This screenshot shows the 'Create Web App' wizard in the Microsoft Azure portal, specifically the 'Basics' tab. The interface includes a top navigation bar with the Azure logo, a search bar, and a Copilot button. The breadcrumb trail indicates the path: Home > Create a resource > Create Web App. The 'Basics' tab is selected, with other tabs like Database, Deployment, Networking, Monitor + secure, Tags, and Review + create visible. A brief description of App Service Web Apps is provided, followed by a 'Project Details' section. In this section, users are prompted to select a subscription (currently 'Azure subscription 1') and a resource group ('API_Deployment', with a 'Create new' link). The 'Instance Details' section includes fields for the app name ('IrisSpecies-APIdeploy'), a unique default hostname toggle (checked), the publish method (set to 'Code'), the runtime stack ('Python 3.9'), the operating system ('Linux'), and the region ('UK South'). A note at the bottom of this section suggests trying a different region if the selected App Service Plan is not found. The 'Pricing plans' section explains that the pricing tier determines location, features, cost, and compute resources. At the bottom, there are three buttons: 'Review + create', '< Previous', and 'Next : Database >'.

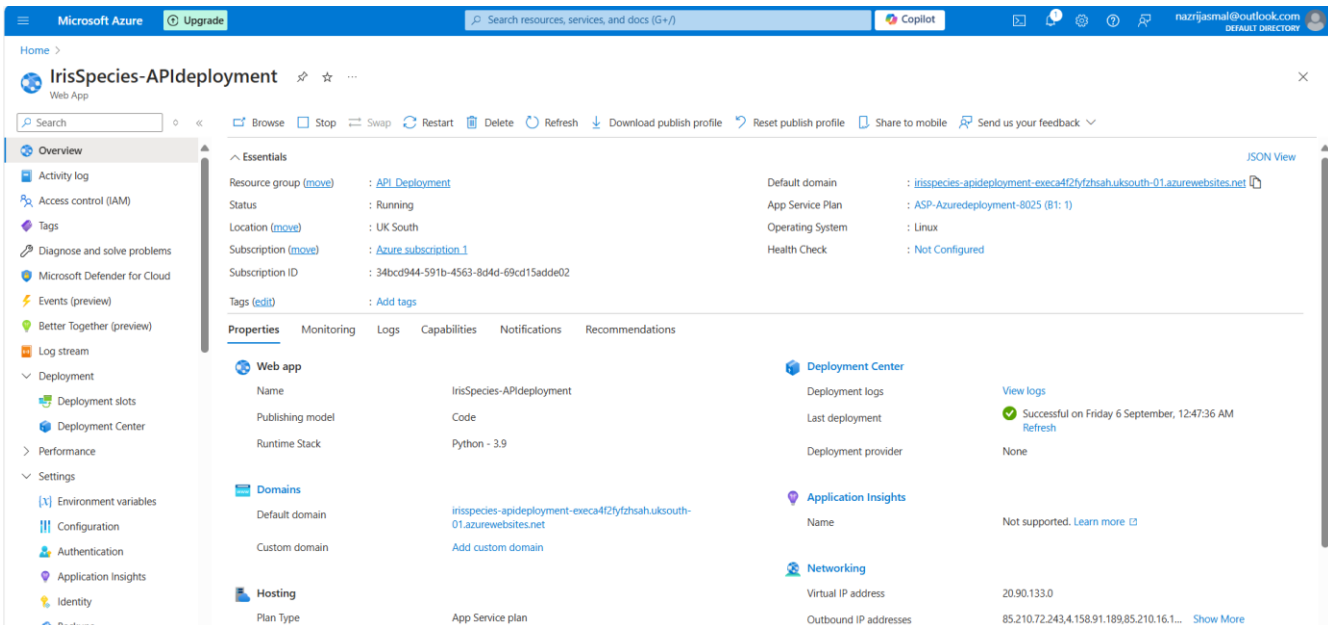
This screenshot shows the 'Create Web App' wizard in the Microsoft Azure portal, specifically the 'Deployment' tab. The breadcrumb trail remains: Home > Create a resource > Create Web App. The 'Deployment' tab is now selected. The 'Continuous deployment settings' section allows users to set up continuous deployment from a GitHub repository via GitHub Actions, with a 'Learn more' link. The 'Continuous deployment' toggle is set to 'Enable'. The 'GitHub settings' section provides instructions on setting up GitHub Actions and includes a 'Change account' button. Below this, there are dropdown menus for 'Organization' (set to 'NazriJasmal'), 'Repository' (set to 'AzureDeployment-API'), and 'Branch' (set to 'main'). The 'Workflow configuration' section includes a 'Preview file' button. The 'Authentication settings' section allows users to choose if they want to allow basic authentication to deploy code, with the 'Basic authentication' toggle set to 'Enable'. At the bottom, there are three buttons: 'Review + create', '< Previous', and 'Next : Networking >'.

8. Access the Web App

After creation, go to the Resource page of the new Web App.

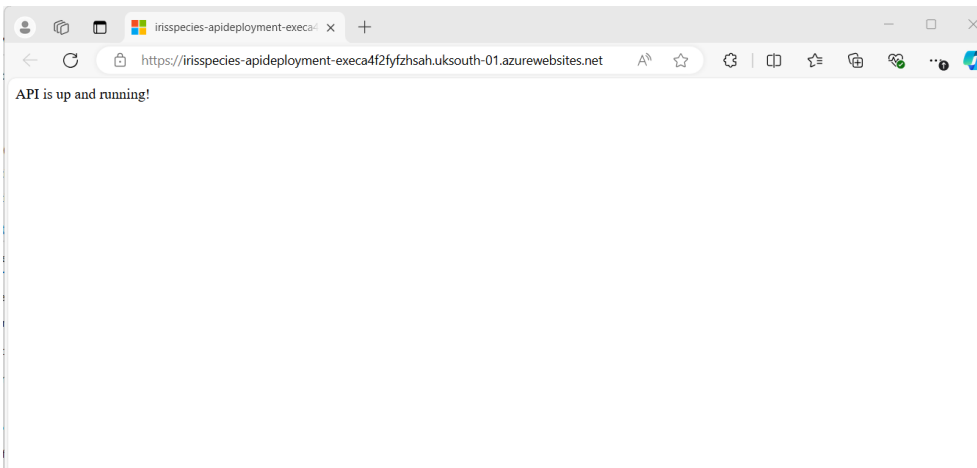
9. Deployment will be in progress as we have already enabled continuous deployment.

10. Go to the Overview tab and we can see API Deployment is Created Successful.



Output

API Domain Link: <https://irisspecies-apideployment-execa4f2fyfzhsah.uksouth-01.azurewebsites.net/api/predict>



Test the API: Test Success

Once deployed, test the API to ensure it's working correctly.

1. Get the API URL:

- After deployment, Azure will provide a URL for the API, which is '<https://irisspecies-apideployment-execa4f2fyfzhsah.uksouth-01.azurewebsites.net>'

2. Send a POST Request:

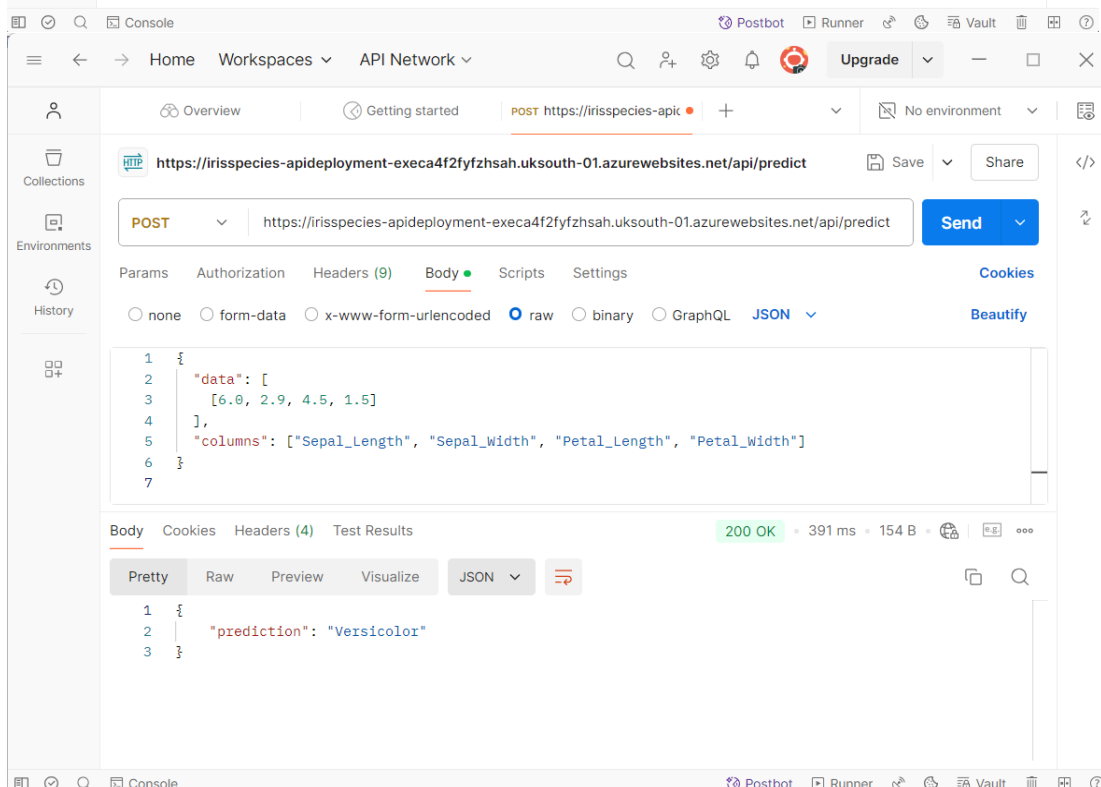
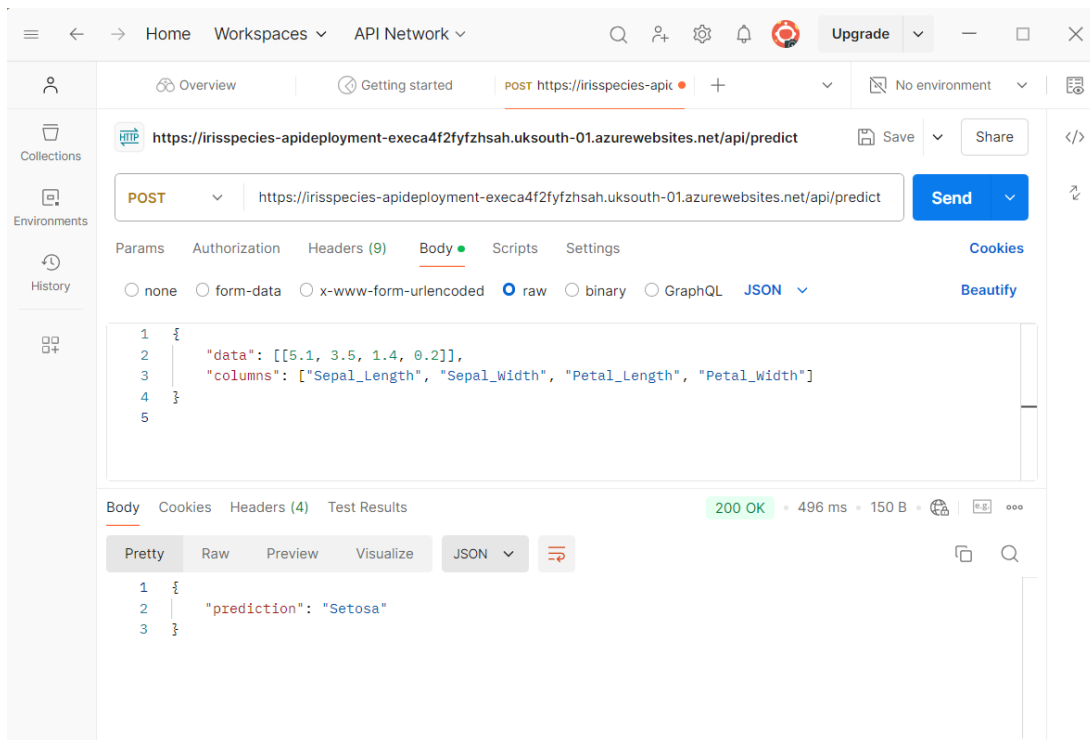
- Use a tool like **Postman** or **curl** to send a POST request to the API with sample data.

Using Postman:

- Open Postman and create a new POST request.
- Set the method to **POST**.
- Set the URL to '<https://irisspecies-apideployment-execa4f2fyfzhsah.uksouth-01.azurewebsites.net/api/predict>'
- Set the Content-Type header to the value application/json.
- In the Body tab, select raw and JSON as the format, then paste in your JSON payload:

```
{
  "data": [[5.1, 3.5, 1.4, 0.2]],
  "columns": ["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]
}
```

- Click **Send** and observe the response.



api_app.py (Used in Azure App Services)

```
from flask import Flask, request, jsonify
import pandas as pd
import pickle
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)

# Import the trained model
model = pickle.load(open('model.pkl', 'rb'))

# Create the Flask app for the API
app = Flask(__name__)

@app.route("/")
def home():
    return "API is up and running!"

@app.route("/api/predict", methods=['POST'])
def api_predict():
    try:
        data = request.get_json(force=True)
        logging.info(f'Received data: {data}')

        # Extract features from JSON
        if 'data' not in data or 'columns' not in data:
            raise ValueError("JSON must contain 'data' and 'columns' keys")

        # Create a DataFrame with the incoming data
        features = pd.DataFrame(data['data'], columns=data['columns'])
        logging.info(f'Features DataFrame: {features}')

        # Check if the columns match what the model was trained with
        expected_columns = ["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]
        if list(features.columns) != expected_columns:
            raise ValueError(f'Expected columns: {expected_columns}, but got: {list(features.columns)}')

        # Make predictions
        prediction = model.predict(features)
        logging.info(f'Prediction: {prediction}')

        # Return the prediction in JSON format
        return jsonify({'prediction': prediction[0]})
    except Exception as e:
        logging.error(f'Error occurred: {str(e)}')
        return jsonify({'error': str(e)}), 500

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8000)
```