Assignment 17: Run your fruit detector on the edge

Question:

It's not just image classifiers that can be run on the edge, anything that can be packaged up into a container can be deployed to an IoT Edge device. Serverless code running as Azure Functions, such as the triggers you've created in earlier lessons can be run in containers, and therefore on IoT Edge.

Pick one of the previous lessons and try to run the Azure Functions app in an IoT Edge container. You can find a guide that shows how to do this using a different Functions app project in the Tutorial: Deploy Azure Functions as IoT Edge modules on Microsoft docs.

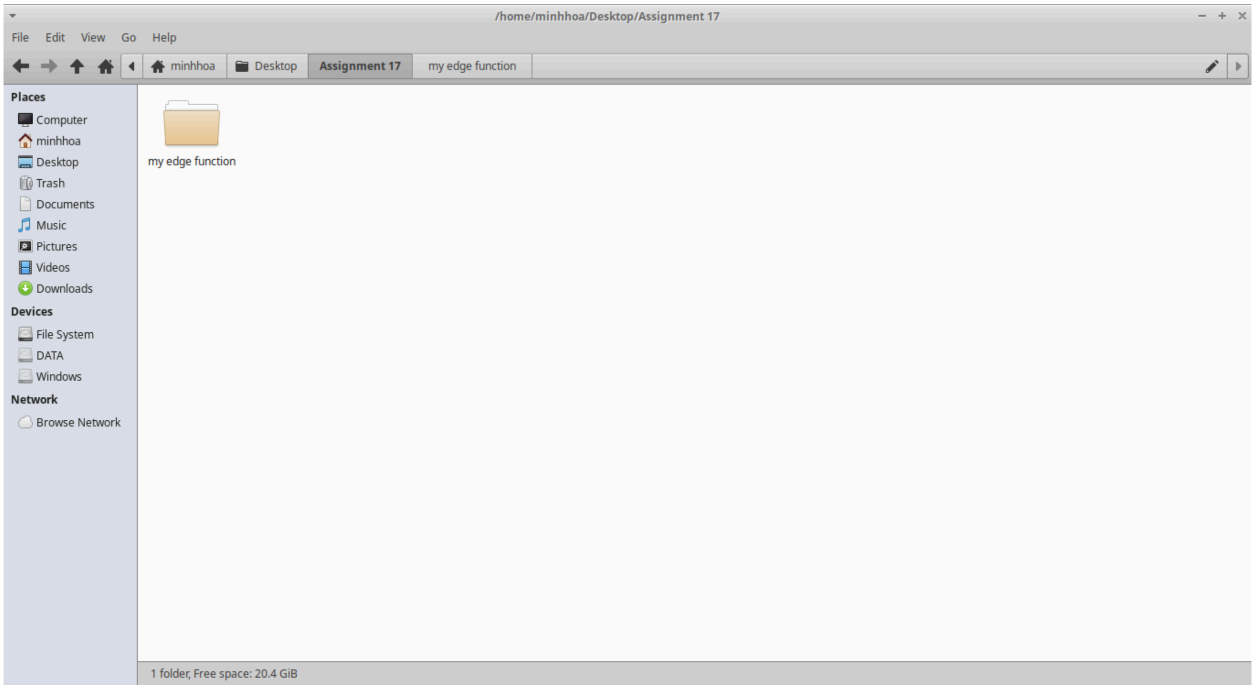
Answer

- In this assignment, I will use the Python Flask app to simulate Azure Function. Using a Docker container and running it on a local PC to simulate IoT Edge. Then I write a script in Python to simulate the IoT devices to send the data to the Python App.

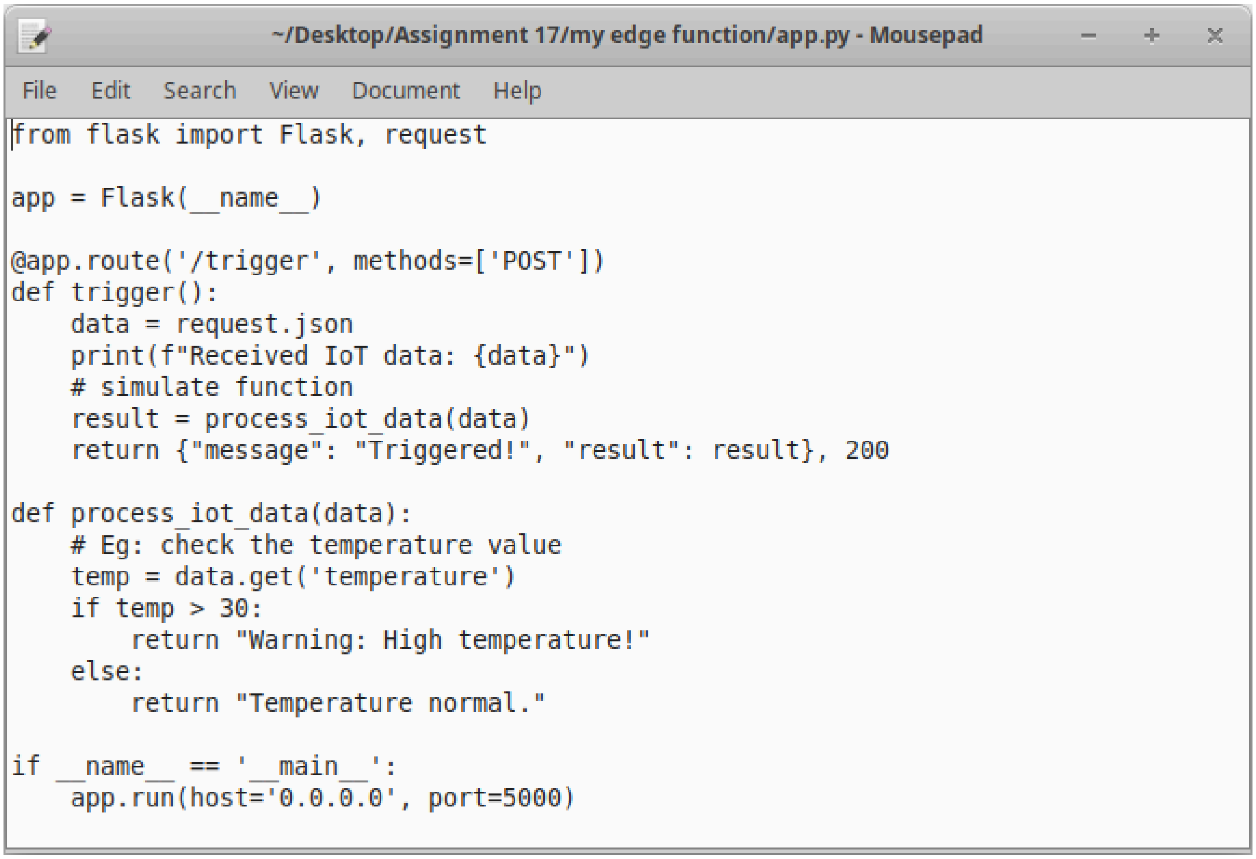
I. Create a simulate system and run it

Step 1: Create Flask app

- First, I create a folder named: “my edge function”



- Then, in this folder, I create a file app.py to simulate Azure Function

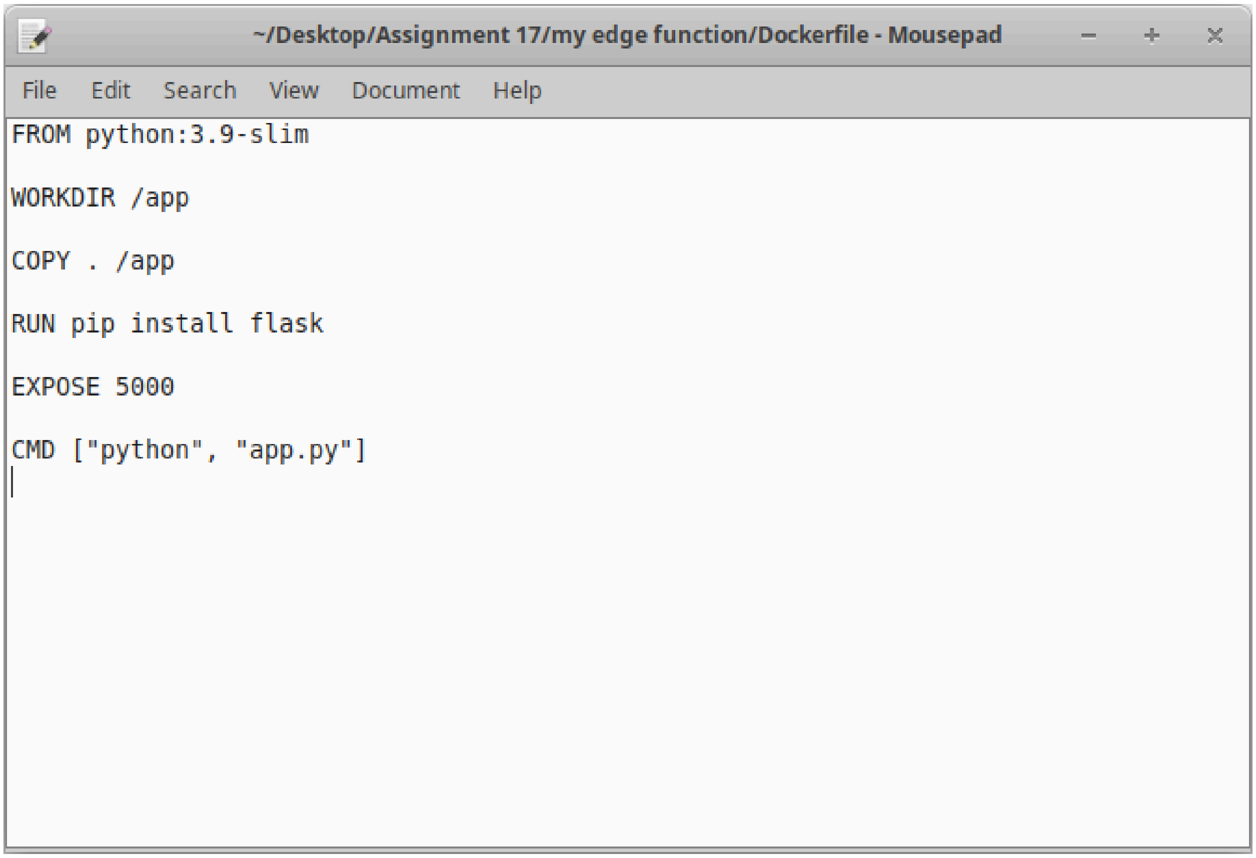


```
from flask import Flask, request

app = Flask(__name__)

@app.route('/trigger', methods=['POST'])
def trigger():
    data = request.json
    print(f"Received IoT data: {data}")
    # simulate function
    result = process_iot_data(data)
    return {"message": "Triggered!", "result": result}, 200

def process_iot_data(data):
    # Eg: check the temperature value
    temp = data.get('temperature')
    if temp > 30:
        return "Warning: High temperature!"
    else:
        return "Temperature normal."

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- This app can check the temperature value. For example, if the temperature value is higher than 30, the App will warn with the message “High temperature”.

Step 2: Create a Dockerfile

- In the same folder with app.py, I create a file named Dockerfile to run.


~/Desktop/Assignment 17/my edge function/Dockerfile - Mousepad
File Edit Search View Document Help

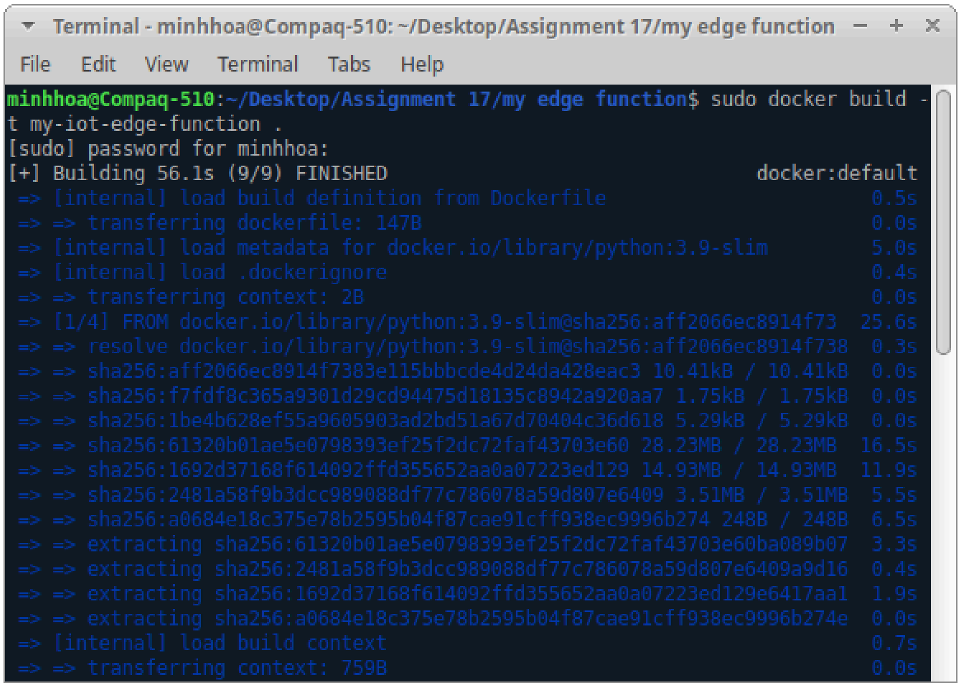
```
FROM python:3.9-slim

WORKDIR /app

COPY . /app

RUN pip install flask

EXPOSE 5000

CMD ["python", "app.py"]
```

- This Dockerfile can connect with the app.py file when it runs.

Step 3: Build Docker image

- I open the terminal in this folder and run the comment:

sudo docker build -t my-iot-edge-function .

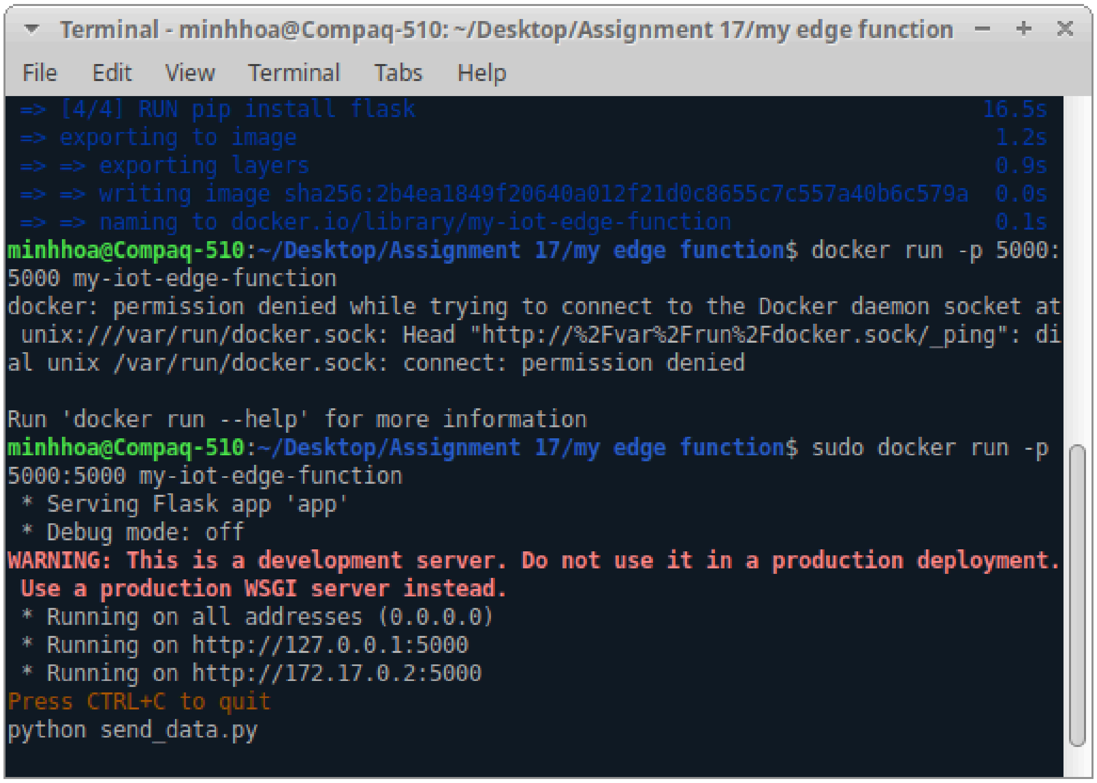

Terminal - minhhoa@Compaq-510: ~/Desktop/Assignment 17/my edge function
File Edit View Terminal Tabs Help

```
minhhoa@Compaq-510:~/Desktop/Assignment 17/my edge function$ sudo docker build -
t my-iot-edge-function .
[sudo] password for minhhoa:
[+] Building 56.1s (9/9) FINISHED                                 docker:default
 => [internal] load build definition from Dockerfile                         0.5s
 => => transferring dockerfile: 147B                                         0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim           5.0s
 => [internal] load .dockerignore                                            0.4s
 => => transferring context: 2B                                              0.0s
 => [1/4] FROM docker.io/library/python:3.9-slim@sha256:aff2066ec8914f73  25.6s
 => => resolve docker.io/library/python:3.9-slim@sha256:aff2066ec8914f738  0.3s
 => => sha256:aff2066ec8914f7383e115bbbcde4d24da428eac3 10.41kB / 10.41kB  0.0s
 => => sha256:f7fdf8c365a9301d29cd94475d18135c8942a920aa7 1.75kB / 1.75kB  0.0s
 => => sha256:1be4b628ef55a9605903ad2bd51a67d70404c36d618 5.29kB / 5.29kB  0.0s
 => => sha256:61320b01ae5e0798393ef25f2dc72faf43703e60 28.23MB / 28.23MB  16.5s
 => => sha256:1692d37168f614092ffd355652aa0a07223ed129 14.93MB / 14.93MB  11.9s
 => => sha256:2481a58f9b3dcc989088df77c786078a59d807e6409 3.51MB / 3.51MB  5.5s
 => => sha256:a0684e18c375e78b2595b04f87cae91cff938ec9996b274 248B / 248B  6.5s
 => => extracting sha256:61320b01ae5e0798393ef25f2dc72faf43703e60ba089b07  3.3s
 => => extracting sha256:2481a58f9b3dcc989088df77c786078a59d807e6409a9d16  0.4s
 => => extracting sha256:1692d37168f614092ffd355652aa0a07223ed129e6417aa1  1.9s
 => => extracting sha256:a0684e18c375e78b2595b04f87cae91cff938ec9996b274e  0.0s
 => [internal] load build context                                            0.7s
 => => transferring context: 759B                                            0.0s
```

Stept 4: Run the container

- After build the Docker image, I use this code to run the container:

sudo docker run -p 5000:5000 my-iot-edge-function

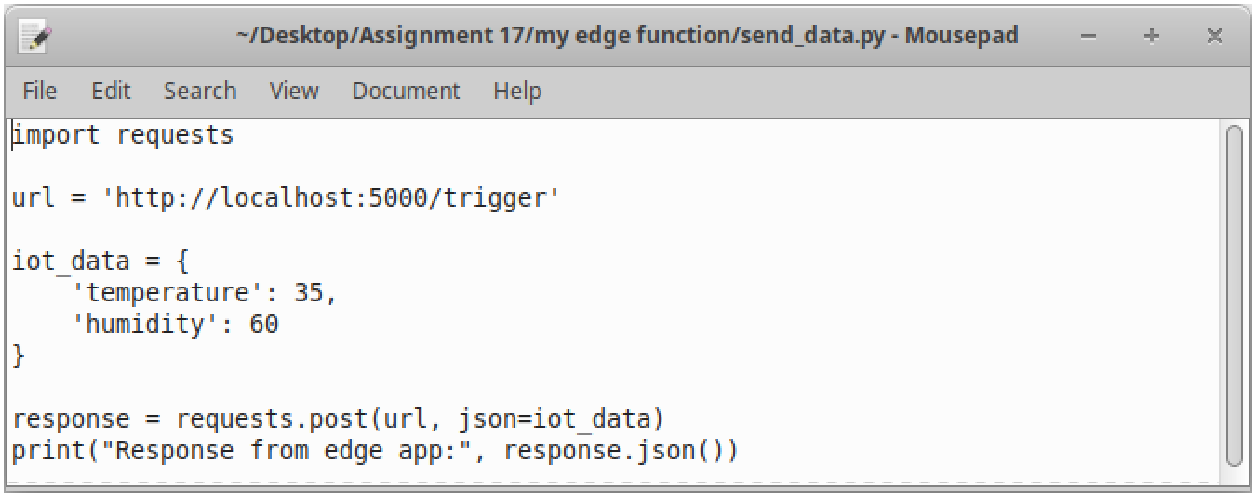

Terminal - minhhoa@Compaq-510: ~/Desktop/Assignment 17/my edge function
File Edit View Terminal Tabs Help

```
 => [4/4] RUN pip install flask                                             16.5s
 => exporting to image                                                       1.2s
 => => exporting layers                                                      0.9s
 => => writing image sha256:2b4ea1849f20640a012f21d0c8655c7c557a40b6c579a  0.0s
 => => naming to docker.io/library/my-iot-edge-function                      0.1s
minhhoa@Compaq-510:~/Desktop/Assignment 17/my edge function$ docker run -p 5000:
5000 my-iot-edge-function
docker: permission denied while trying to connect to the Docker daemon socket at
 unix:///var/run/docker.sock: Head "http://%2Fvar%2Frun%2Fdocker.sock/_ping": di
al unix /var/run/docker.sock: connect: permission denied

Run 'docker run --help' for more information
minhhoa@Compaq-510:~/Desktop/Assignment 17/my edge function$ sudo docker run -p
5000:5000 my-iot-edge-function
 * Serving Flask app '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 all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
python send_data.py
```

- The Container will run the Flask app and wait at 5000 gate.

Step 5: Create the script to simulate IoT device

- In the same folder, I will create a new file send_data.py


~/Desktop/Assignment 17/my edge function/send_data.py - Mousepad
File
Edit
Search
View
Document
Help

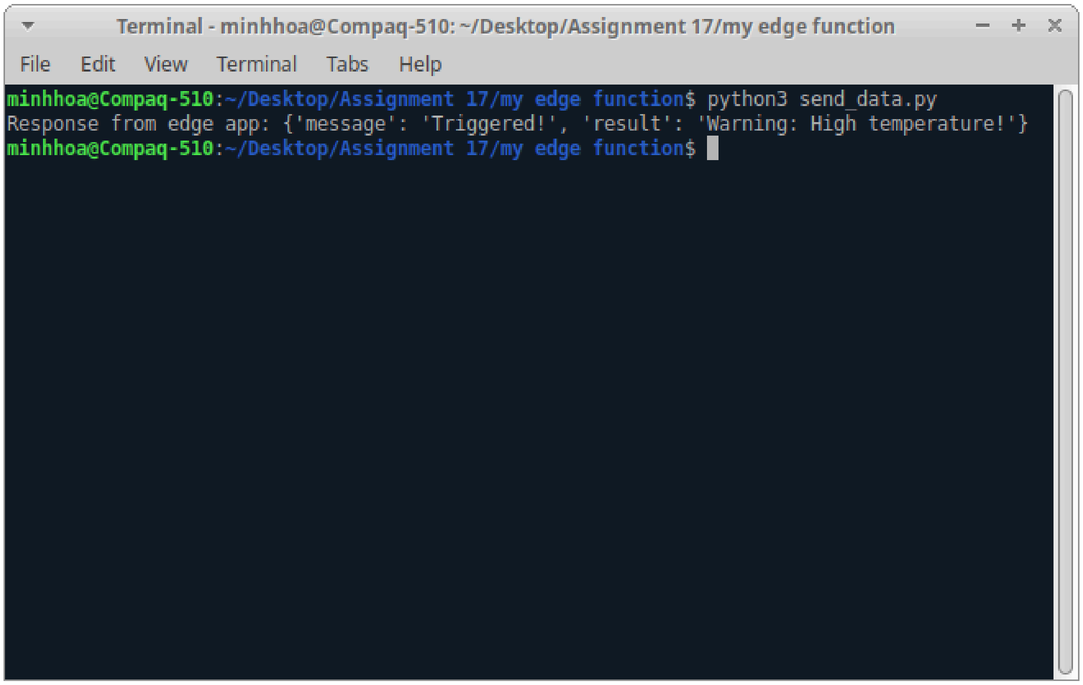
```
import requests

url = 'http://localhost:5000/trigger'

iot_data = {
    'temperature': 35,
    'humidity': 60
}

response = requests.post(url, json=iot_data)
print("Response from edge app:", response.json())
```

- This file can simulate the temperature and humidity.

Step 6: Run all the code

- I open the terminal and using this code to run:

python3 send_data.py


Terminal - minhhoa@Compaq-510: ~/Desktop/Assignment 17/my edge function
File
Edit
View
Terminal
Tabs
Help

```
minhhoa@Compaq-510:~/Desktop/Assignment 17/my edge function$ python3 send_data.py
Response from edge app: {'message': 'Triggered!', 'result': 'Warning: High temperature!'}
minhhoa@Compaq-510:~/Desktop/Assignment 17/my edge function$ 
```

- We can see that the edge app response and the system run successfully.

II. Conclusion

In this exercise, instead of using Azure Functions deployed to a real IoT Edge device, we simulated the edge environment locally using a Python Flask application packaged inside a Docker container. This allowed us to demonstrate how additional services can run on the edge without requiring physical IoT hardware or an Azure subscription.

We successfully built and ran the Flask-based service in a Docker container, exposing it on port 5000. Additionally, we implemented a Python script (send_data.py) to simulate an IoT device sending data to the service. This setup effectively emulated how edge services process incoming IoT data and respond accordingly.

Through this approach, we gained practical experience with:

- Building containerized edge services.
- Deploying and testing services locally.
- Simulating IoT data flow without relying on physical devices.

This exercise provided valuable insights into the flexibility of edge computing and how various types of services, not limited to image classifiers or cloud triggers, can be deployed and tested on the edge using container technologies.