CSC8112 – Internet of Things

Report 1

Name: Aditya Sai Chinnathambi Narayanan

Student ID: 230336355

I. Introduction

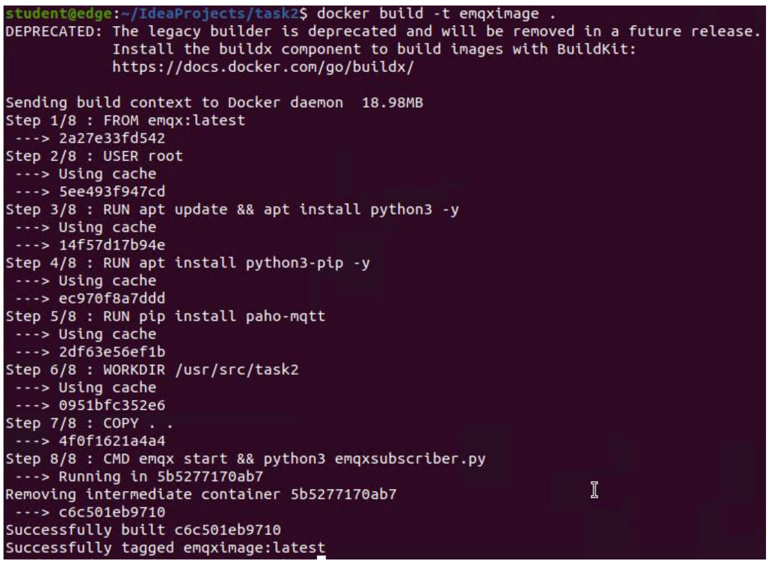
A project to understand how to process Internet of Things (IoT) sensor data in the edge-cloud setting. To develop a machine learning based IoT data processing pipeline (data collection, data preprocessing, prediction and visualisation), using Docker to implement IoT data processing pipeline. This includes integration of Docker containers for data transmission to cloud-based preprocessing using python scripts, EMQX, RabbitMQ and machine learning for forecasting.

II. Task Response: Detailed Breakdown

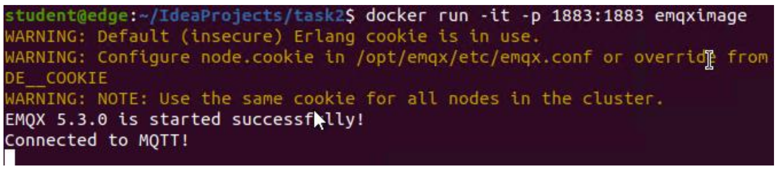
Task 1: Design a data injector component by leveraging Newcastle Urban Observatory IoT Data Streams

1. **Pull and run the Docker image "emqx/emqx" from Docker hub in the virtual machine running on Azure lab (Edge).**

   Successfully pulled and ran the Docker image "emqx/emqx" from Docker hub which is a public repository for all the public Docker images. This was done using the CLI (Command Line Interface) of the virtual machine (Edge) which is running on Azure lab with Linux OS.



```
student@edge:~/IdeaProjects/task2$ docker build -t emqximage .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  18.98MB
Step 1/8 : FROM emqx:latest
 ---> 2a27e33fd542
Step 2/8 : USER root
 ---> Using cache
 ---> 5ee493f947cd
Step 3/8 : RUN apt update && apt install python3 -y
 ---> Using cache
 ---> 14f57d17b94e
Step 4/8 : RUN apt install python3-pip -y
 ---> Using cache
 ---> ec970f8a7ddd
Step 5/8 : RUN pip install paho-mqtt
 ---> Using cache
 ---> 2df63e56ef1b
Step 6/8 : WORKDIR /usr/src/task2
 ---> Using cache
 ---> 0951bfc352e6
Step 7/8 : COPY . .
 ---> 4f0f1621a4a4
Step 8/8 : CMD emqx start && python3 emqxsubscriber.py
 ---> Running in 5b5277170ab7
Removing intermediate container 5b5277170ab7
 ---> c6c501eb9710
Successfully built c6c501eb9710
Successfully tagged emqximage:latest
```



```
student@edge:~/IdeaProjects/task2$ docker run -it -p 1883:1883 emqximage
WARNING: Default (insecure) Erlang cookie is in use.
WARNING: Configure node.cookie in /opt/emqx/etc/emqx.conf or override from
DE__COOKIE
WARNING: NOTE: Use the same cookie for all nodes in the cluster.
EMQX 5.3.0 is started successfully!
Connected to MQTT!
```

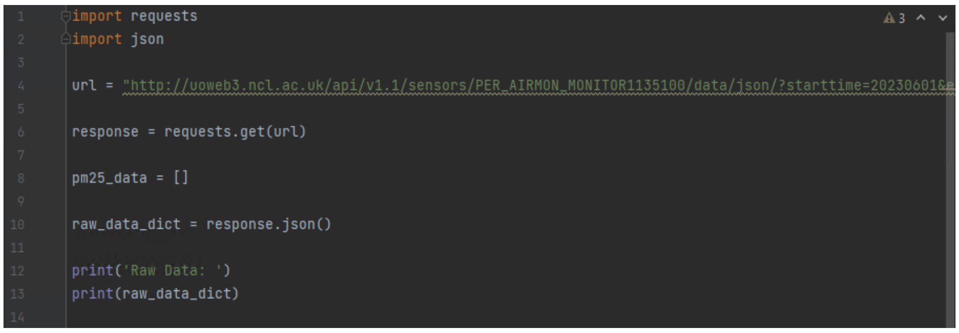
Commands Used:

- Docker build -t emqximage.
- Docker run -it -p 1883:1883 emqximage

- Using the build command, we build a Docker image that consists of both subscriber code in python as well as the emqx image from Docker hub.
- Both the code and the docker image will run as a container.
- Using docker run command we pull the emqx image that is present in Docker hub if it is not available locally, If the image is available locally it is cached from the memory and used to run the image and create a container.
- Used the -it flag to start the image in interactive mode such that the logs can be seen the terminal directly.
- -p flag is used to assign the port 1883 of the service running in the image to the port of the local machine (i.e., Virtual Machine (Edge) in this case).

2. Develop a data injector component with following functions in Azure Lab (Edge) or the Azure lab localhost:

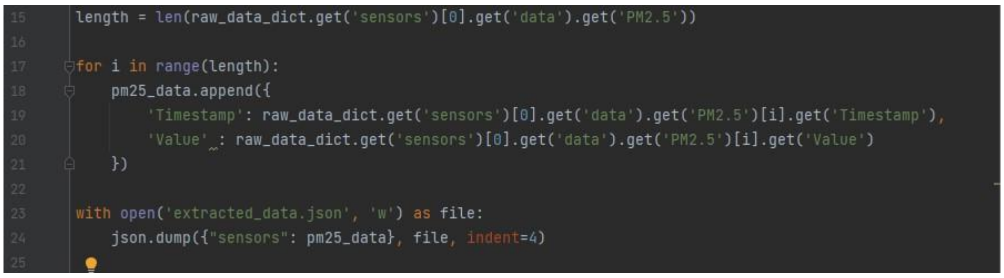
(a). Collect data from Urban Observatory Platform by sending HTTP request to the URL and please print the raw data streams to the console.



```python
import requests
import json

url = "http://uoweb3.ncl.ac.uk/api/v1.1/sensors/PER_AIRMON_MONITOR1135100/data/json/?starttime=20230601&e

response = requests.get(url)

pm25_data = []

raw_data_dict = response.json()

print('Raw Data: ')
print(raw_data_dict)
```

- Using package requests in python3, an HTTP request was sent to Urban Observatory Platform.
- The API of the webpage responds with all the raw data that was requested, and the requested data is stored into a dictionary.

(b). Collect only PM2.5 data and Timestamp and Value metadata from the raw data stream.

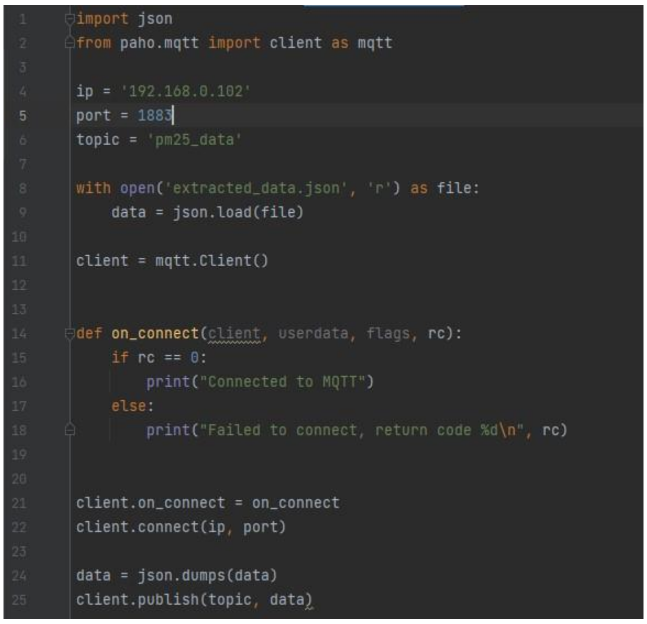


```python
length = len(raw_data_dict.get('sensors')[0].get('data').get('PM2.5'))

for i in range(length):
    pm25_data.append({
        'Timestamp': raw_data_dict.get('sensors')[0].get('data').get('PM2.5')[i].get('Timestamp'),
        'Value' : raw_data_dict.get('sensors')[0].get('data').get('PM2.5')[i].get('Value')
    })

with open('extracted_data.json', 'w') as file:
    json.dump({"sensors": pm25_data}, file, indent=4)
```

- Extracting PM2.5 data from the raw data stream as well as storing only Timestamp and Value metadata of the PM2.5 data.

(c). Send all PM2.5 data to be used by Task 2.2(a) to EMQX service of Azure lab (Edge).

Code Snippet:



```python
import json
from paho.mqtt import client as mqtt

ip = '192.168.0.102'
port = 1883
topic = 'pm25_data'

with open('extracted_data.json', 'r') as file:
    data = json.load(file)

client = mqtt.Client()


def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT")
    else:
        print("Failed to connect, return code %d\n", rc)


client.on_connect = on_connect
client.connect(ip, port)

data = json.dumps(data)
client.publish(topic, data)
```

- Using paho module in python3, we import all the necessary functions to send the data using MQTT protocol.
- Assigning the IP address to the edge virtual machine running on Azure lab (Edge). [Edge's IP: 192.168.0.102]
- Assigning the port to 1883 as that is the common port used by the EMQX service for sending MQTT messages.
- We import the PM2.5 data from a json file stored locally which was created previously when we requested and sorted the data based on our needs.
- Finally, we connect to the specified IP and port assuming the Edge virtual machine is actively listening for incoming connections and send the data to emqx.

Task 2: Data preprocessing operator design

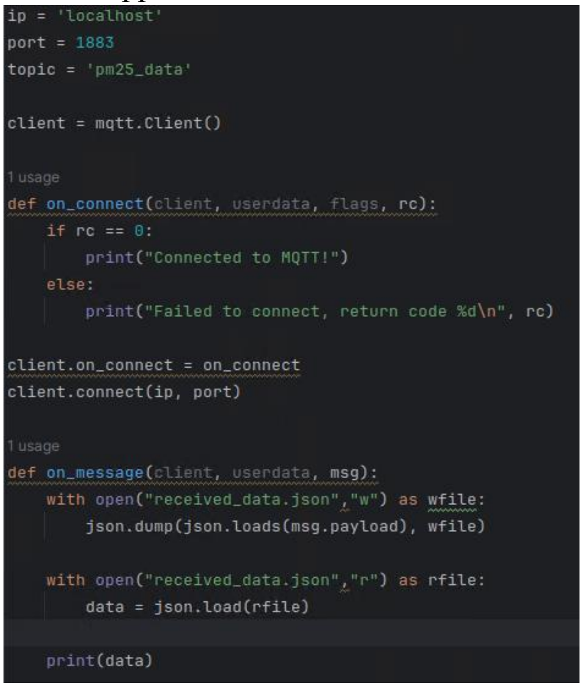
1. **Define a Docker-compose file which contains necessary configurations and instructions for deploying and instantiating the following set of Docker images on Azure lab (Cloud):**
   (a) Download and run RabbitMQ image (rabbitmq:management);

   Defining the necessary configuration to pull and run rabbitmq from Docker hub as shown in the image above.

2. **Design a data preprocessing operator with the following functions in Azure lab (Edge):**

   (a) **Collect all PM2.5 data published by Task 1.2 (c) from EMQX service, and print out the PM2.5 data to the console.**

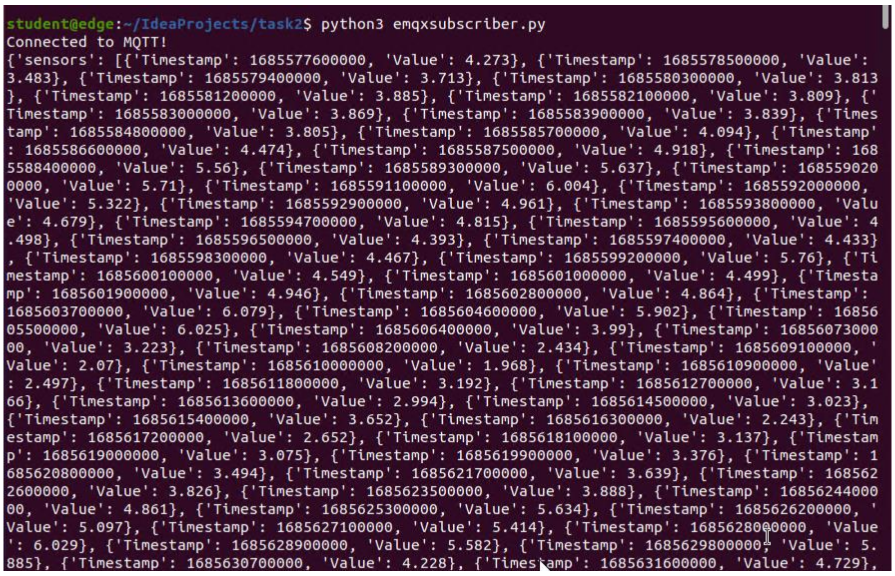
   Code Snippet:



```
ip = 'localhost'
port = 1883
topic = 'pm25_data'

client = mqtt.Client()

1 usage
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT!")
    else:
        print("Failed to connect, return code %d\n", rc)

client.on_connect = on_connect
client.connect(ip, port)

1 usage
def on_message(client, userdata, msg):
    with open("received_data.json","w") as wfile:
        json.dump(json.loads(msg.payload), wfile)

    with open("received_data.json","r") as rfile:
        data = json.load(rfile)

    print(data)
```

- Creating a subscriber which listens continuously for any connection in the specified port (Default port for emqx is 1883).
- Before listening for connections, we must run the emqx service, this is done using Docker.
- We pull and run emqx image from Docker hub as specified in task 1.1.
- Once the image is set up and running in the container, we run the code and listen for any connection for the specified port.
- When the publisher sends the data to the Edge virtual machine, the data is received by the subscriber and further actions takes place.

Output:

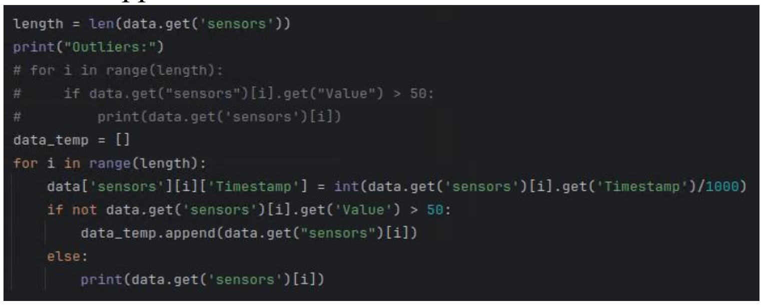


```
student@edge:~/IdeaProjects/task2$ python3 emqxsubscriber.py
Connected to MQTT!
{'sensors': [{'Timestamp': 1685577600000, 'Value': 4.273}, {'Timestamp': 1685578500000, 'Value':
3.483}, {'Timestamp': 1685579400000, 'Value': 3.713}, {'Timestamp': 1685580300000, 'Value': 3.813
}, {'Timestamp': 1685581200000, 'Value': 3.885}, {'Timestamp': 1685582100000, 'Value': 3.809}, {'
Timestamp': 1685583000000, 'Value': 3.869}, {'Timestamp': 1685583900000, 'Value': 3.839}, {'Times
tamp': 1685584800000, 'Value': 3.805}, {'Timestamp': 1685585700000, 'Value': 4.094}, {'Timestamp'
: 1685586600000, 'Value': 4.474}, {'Timestamp': 1685587500000, 'Value': 4.918}, {'Timestamp': 168
5588400000, 'Value': 5.56}, {'Timestamp': 1685589300000, 'Value': 5.637}, {'Timestamp': 168559020
0000, 'Value': 5.71}, {'Timestamp': 1685591100000, 'Value': 6.004}, {'Timestamp': 1685592000000,
'Value': 5.322}, {'Timestamp': 1685592900000, 'Value': 4.961}, {'Timestamp': 1685593800000, 'Valu
e': 4.679}, {'Timestamp': 1685594700000, 'Value': 4.815}, {'Timestamp': 1685595600000, 'Value': 4
.498}, {'Timestamp': 1685596500000, 'Value': 4.393}, {'Timestamp': 1685597400000, 'Value': 4.433}
, {'Timestamp': 1685598300000, 'Value': 4.467}, {'Timestamp': 1685599200000, 'Value': 5.76}, {'Ti
mestamp': 1685600100000, 'Value': 4.549}, {'Timestamp': 1685601000000, 'Value': 4.499}, {'Timesta
mp': 1685601900000, 'Value': 4.946}, {'Timestamp': 1685602800000, 'Value': 4.864}, {'Timestamp':
1685603700000, 'Value': 6.079}, {'Timestamp': 1685604600000, 'Value': 5.902}, {'Timestamp': 16856
05500000, 'Value': 6.025}, {'Timestamp': 1685606400000, 'Value': 3.99}, {'Timestamp': 16856073000
00, 'Value': 3.223}, {'Timestamp': 1685608200000, 'Value': 2.434}, {'Timestamp': 1685609100000, '
Value': 2.07}, {'Timestamp': 1685610000000, 'Value': 1.968}, {'Timestamp': 1685610900000, 'Value'
: 2.497}, {'Timestamp': 1685611800000, 'Value': 3.192}, {'Timestamp': 1685612700000, 'Value': 3.1
66}, {'Timestamp': 1685613600000, 'Value': 2.994}, {'Timestamp': 1685614500000, 'Value': 3.023},
{'Timestamp': 1685615400000, 'Value': 3.652}, {'Timestamp': 1685616300000, 'Value': 2.243}, {'Tim
estamp': 1685617200000, 'Value': 2.652}, {'Timestamp': 1685618100000, 'Value': 3.137}, {'Timestam
p': 1685619000000, 'Value': 3.075}, {'Timestamp': 1685619900000, 'Value': 3.376}, {'Timestamp': 1
685620800000, 'Value': 3.494}, {'Timestamp': 1685621700000, 'Value': 3.639}, {'Timestamp': 168562
2600000, 'Value': 3.826}, {'Timestamp': 1685623500000, 'Value': 3.888}, {'Timestamp': 16856244000
00, 'Value': 4.861}, {'Timestamp': 1685625300000, 'Value': 5.634}, {'Timestamp': 1685626200000, '
Value': 5.097}, {'Timestamp': 1685627100000, 'Value': 5.414}, {'Timestamp': 1685628000000, 'Value
': 6.029}, {'Timestamp': 1685628900000, 'Value': 5.582}, {'Timestamp': 1685629800000, 'Value': 5.
885}, {'Timestamp': 1685630700000, 'Value': 4.228}, {'Timestamp': 1685631600000, 'Value': 4.729},
```

- Hence, when the data is received by the subscriber, it is printed to console.

(b) Filter out outliers (the value greater than 50) and print out outliers to the console.

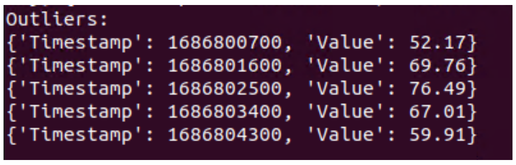
Code Snippet:



```python
length = len(data.get('sensors'))
print("Outliers:")
# for i in range(length):
#     if data.get("sensors")[i].get("Value") > 50:
#         print(data.get('sensors')[i])
data_temp = []
for i in range(length):
    data['sensors'][i]['Timestamp'] = int(data.get('sensors')[i].get('Timestamp')/1000)
    if not data.get('sensors')[i].get('Value') > 50:
        data_temp.append(data.get("sensors")[i])
    else:
        print(data.get('sensors')[i])
```

- We sort out the values based on the outliers hence we take an list and iterate through the dictionary.
- During iteration, we filter out the data which has Value greater than 50 and display it on the console.
- Rest of the data is put into a list and are used for further analysis.

Output:

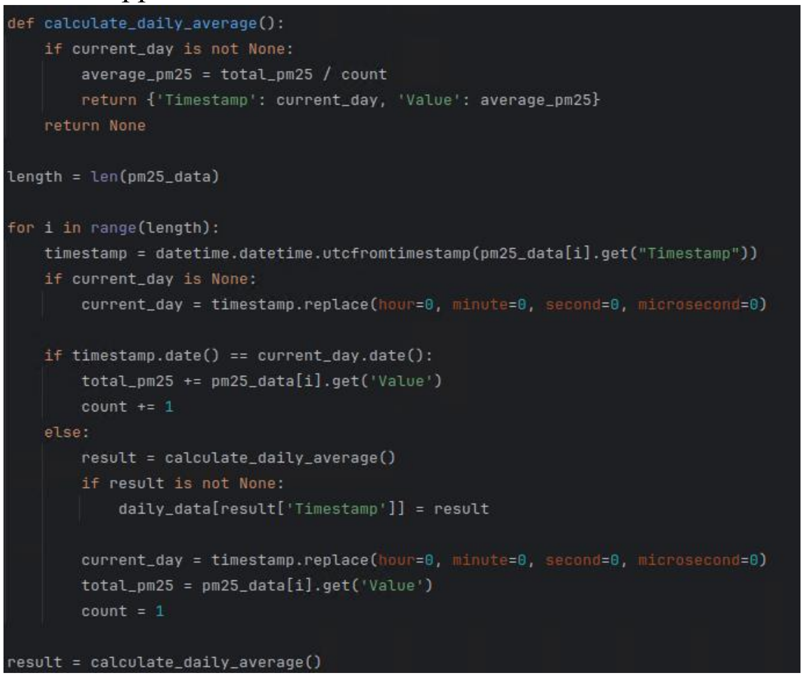


```
Outliers:
{'Timestamp': 1686800700, 'Value': 52.17}
{'Timestamp': 1686801600, 'Value': 69.76}
{'Timestamp': 1686802500, 'Value': 76.49}
{'Timestamp': 1686803400, 'Value': 67.01}
{'Timestamp': 1686804300, 'Value': 59.91}
```

- As mentioned above the outliers are printed out to the console.

(c) Calculate the average value of PM2.5 data on daily basis and print out the result to the console.

Code Snippet:



```python
def calculate_daily_average():
    if current_day is not None:
        average_pm25 = total_pm25 / count
        return {'Timestamp': current_day, 'Value': average_pm25}
    return None

length = len(pm25_data)

for i in range(length):
    timestamp = datetime.datetime.utcfromtimestamp(pm25_data[i].get("Timestamp"))
    if current_day is None:
        current_day = timestamp.replace(hour=0, minute=0, second=0, microsecond=0)

    if timestamp.date() == current_day.date():
        total_pm25 += pm25_data[i].get('Value')
        count += 1
    else:
        result = calculate_daily_average()
        if result is not None:
            daily_data[result['Timestamp']] = result

        current_day = timestamp.replace(hour=0, minute=0, second=0, microsecond=0)
        total_pm25 = pm25_data[i].get('Value')
        count = 1

result = calculate_daily_average()
```

Calculation of average value of PM2.5 data on daily basis is calculated. Firstly, we define a function which calculates the daily average, this function is called when we have iterated through the values which belong to the same day. Once the function is called, its job is to find the average value of that day and create a key, value pair and return both the values as a dictionary with keys: ‘Timestamp’ and ‘Value’. This returned variable with the type ‘dict’ is added into a list which consists of all the averaged data on daily basis.

The for loop present here is the one that iterates through the raw data and converts the timestamp into datetime object and accumulates at the values that belong to the same day, this is done using the if condition inside the loop. If timestamp date and current_date is not equal all the previous accumulated values are used to calculate the average data of that day, to do this the calculate_daily_average function is called which was elaborated previous.

Finally, once all the raw data is finally iterated, we’ll end up with a list of dictionaries which consists of all the averaged data daily.

(d) Transfer all results (averaged PM2.5 data) into RabbitMQ service on Azure lab (Cloud)

Code Snippet:

```python
import pika
import json

ip = '192.168.0.100'
port = 5672
queue = 'finaldata'

with open('final_data.json', 'r') as file:
    data = json.load(file)

data_bytes = data
connection = pika.BlockingConnection(pika.ConnectionParameters(host=ip, port=port))
channel = connection.channel()

channel.queue_declare(queue=queue)
channel.basic_publish(exchange='', routing_key=queue, body=json.dumps(data))

connection.close()
```

RabbitMQ is based on producer and consumer model, which in short uses queues to transfer data from the producer to the respective consumer. Hence, consumer must be in the same queue to that of the producer to receive the data. As seen in the code we use pika module in python to send data to RabbitMQ in cloud. We use the queue named as ‘finaldata’. We proceed to import the final data that is stored locally in the Edge Virtual machine and send it to the consumer that is actively listening in the Cloud virtual machine.

3. Define a dockerfile to migrate your “data preprocessing operator” source code into a Docker image and then define a docker-compose file to run it as a container locally on the Azure lab (Edge).

To run the producer file, we use docker. We define a docker-compose file that automatically builds the Docker image and runs in a container. The docker-compose file is as follows:

```yaml
version: "3"
services:
  python-script:
    build:
      context: .
      dockerfile: Dockerfilerabbit
    command: python producer.py
```

- Since it’s a python file we define the Dockerfile that is named as ‘Dockerfilerabbit’ which consists of all the information on how the docker image must be run.
- It imports python from Docker hub and install all the dependencies that are required for the program to run.
- Finally, we run the file using the command ‘python producer.py’ which is defined in the docker-compose file as mentioned above in the image.

Task 3: Time-series data prediction and visualization

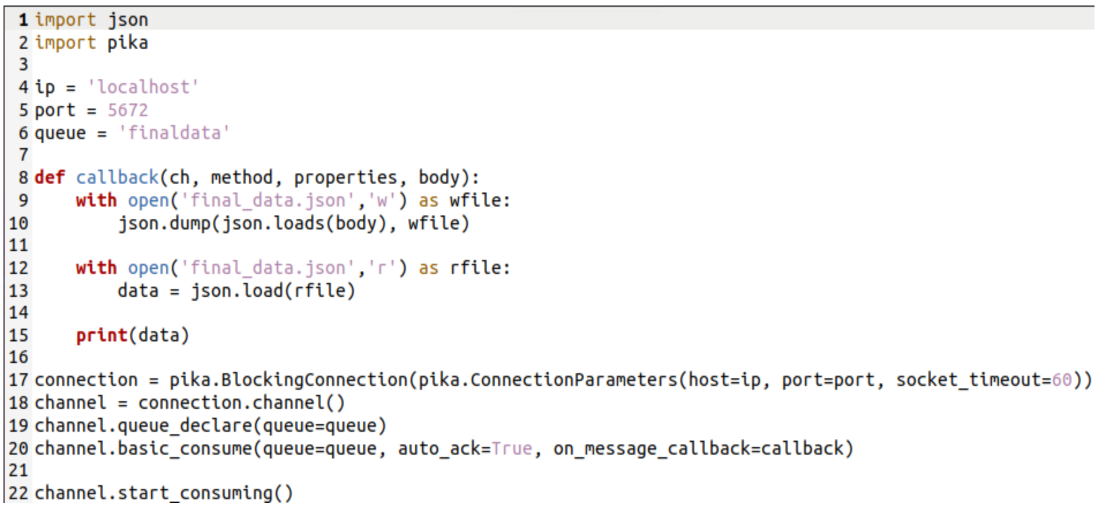
1. **Download a pre-defined Machine Learning (ML) engine code.**

   The pre-defined Machine Learning engine code was successfully downloaded and was named as ml_engine.py to predict the trend of PM2.5 for the next 15 days.

2. **Design a PM2.5 prediction operator with the following functions(code) in Azure Lab (Cloud) or the Azure Lab localhost:**

   (a) Collect all averaged daily PM2.5 data computed from RabbitMQ service and print them to the console.

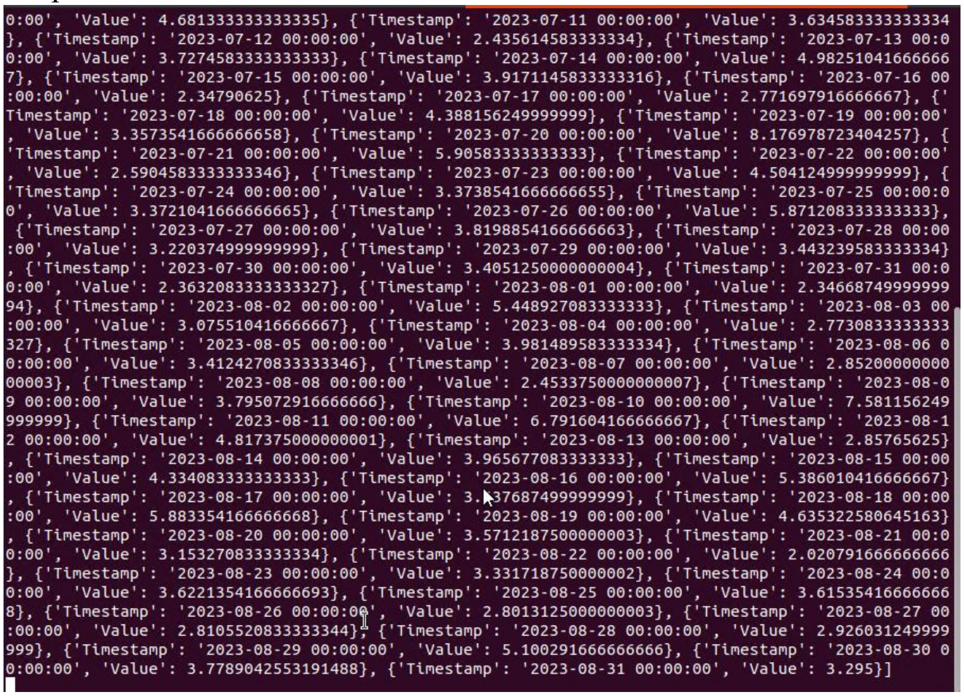
   Code Snippet:



```python
import json
import pika

ip = 'localhost'
port = 5672
queue = 'finaldata'

def callback(ch, method, properties, body):
    with open('final_data.json','w') as wfile:
        json.dump(json.loads(body), wfile)

    with open('final_data.json','r') as rfile:
        data = json.load(rfile)

    print(data)

connection = pika.BlockingConnection(pika.ConnectionParameters(host=ip, port=port, socket_timeout=60))
channel = connection.channel()
channel.queue_declare(queue=queue)
channel.basic_consume(queue=queue, auto_ack=True, on_message_callback=callback)

channel.start_consuming()
```

   Using pika module that comes with python, it is used to receive the data that is being sent from Edge virtual machine, storing the data that was received in a json file named as 'final_data'.

Output:



```
0:00', 'Value': 4.681333333333335}, {'Timestamp': '2023-07-11 00:00:00', 'Value': 3.634583333333334
}, {'Timestamp': '2023-07-12 00:00:00', 'Value': 2.435614583333334}, {'Timestamp': '2023-07-13 00:0
0:00', 'Value': 3.727458333333333}, {'Timestamp': '2023-07-14 00:00:00', 'Value': 4.98251041666666
7}, {'Timestamp': '2023-07-15 00:00:00', 'Value': 3.9171145833333316}, {'Timestamp': '2023-07-16 00
:00:00', 'Value': 2.34790625}, {'Timestamp': '2023-07-17 00:00:00', 'Value': 2.771697916666667}, {'
Timestamp': '2023-07-18 00:00:00', 'Value': 4.388156249999999}, {'Timestamp': '2023-07-19 00:00:00'
, 'Value': 3.3573541666666658}, {'Timestamp': '2023-07-20 00:00:00', 'Value': 8.176978723404257}, {
'Timestamp': '2023-07-21 00:00:00', 'Value': 5.905833333333333}, {'Timestamp': '2023-07-22 00:00:00'
, 'Value': 2.5904583333333346}, {'Timestamp': '2023-07-23 00:00:00', 'Value': 4.504124999999999}, {
'Timestamp': '2023-07-24 00:00:00', 'Value': 3.3738541666666655}, {'Timestamp': '2023-07-25 00:00:0
0', 'Value': 3.3721041666666665}, {'Timestamp': '2023-07-26 00:00:00', 'Value': 5.871208333333333},
 {'Timestamp': '2023-07-27 00:00:00', 'Value': 3.8198854166666663}, {'Timestamp': '2023-07-28 00:00
:00', 'Value': 3.220374999999999}, {'Timestamp': '2023-07-29 00:00:00', 'Value': 3.443239583333334}
, {'Timestamp': '2023-07-30 00:00:00', 'Value': 3.4051250000000004}, {'Timestamp': '2023-07-31 00:0
0:00', 'Value': 2.3632083333333327}, {'Timestamp': '2023-08-01 00:00:00', 'Value': 2.34668749999999
94}, {'Timestamp': '2023-08-02 00:00:00', 'Value': 5.448927083333333}, {'Timestamp': '2023-08-03 00
:00:00', 'Value': 3.075510416666667}, {'Timestamp': '2023-08-04 00:00:00', 'Value': 2.7730833333333
327}, {'Timestamp': '2023-08-05 00:00:00', 'Value': 3.981489583333334}, {'Timestamp': '2023-08-06 0
0:00:00', 'Value': 3.4124270833333346}, {'Timestamp': '2023-08-07 00:00:00', 'Value': 2.85200000000
00003}, {'Timestamp': '2023-08-08 00:00:00', 'Value': 2.4533750000000007}, {'Timestamp': '2023-08-0
9 00:00:00', 'Value': 3.795072916666666}, {'Timestamp': '2023-08-10 00:00:00', 'Value': 7.581156249
999999}, {'Timestamp': '2023-08-11 00:00:00', 'Value': 6.791604166666667}, {'Timestamp': '2023-08-1
2 00:00:00', 'Value': 4.817375000000001}, {'Timestamp': '2023-08-13 00:00:00', 'Value': 2.85765625}
, {'Timestamp': '2023-08-14 00:00:00', 'Value': 3.965677083333333}, {'Timestamp': '2023-08-15 00:00
:00', 'Value': 4.334083333333333}, {'Timestamp': '2023-08-16 00:00:00', 'Value': 5.386010416666667}
, {'Timestamp': '2023-08-17 00:00:00', 'Value': 3.[illegible]37687499999999}, {'Timestamp': '2023-08-18 00:00
:00', 'Value': 5.883354166666668}, {'Timestamp': '2023-08-19 00:00:00', 'Value': 4.635322580645163}
, {'Timestamp': '2023-08-20 00:00:00', 'Value': 3.5712187500000003}, {'Timestamp': '2023-08-21 00:0
0:00', 'Value': 3.153270833333334}, {'Timestamp': '2023-08-22 00:00:00', 'Value': 2.02079166666666
}, {'Timestamp': '2023-08-23 00:00:00', 'Value': 3.331718750000002}, {'Timestamp': '2023-08-24 00:0
0:00', 'Value': 3.6221354166666693}, {'Timestamp': '2023-08-25 00:00:00', 'Value': 3.61535416666666
8}, {'Timestamp': '2023-08-26 00:00:00', 'Value': 2.8013125000000003}, {'Timestamp': '2023-08-27 00
:00:00', 'Value': 2.8105520833333344}, {'Timestamp': '2023-08-28 00:00:00', 'Value': 2.926031249999
999}, {'Timestamp': '2023-08-29 00:00:00', 'Value': 5.100291666666666}, {'Timestamp': '2023-08-30 0
0:00:00', 'Value': 3.7789042553191488}, {'Timestamp': '2023-08-31 00:00:00', 'Value': 3.295}]
```

Once the data is received from Edge virtual machine, we print them out to the console.

(b) Convert timestamp to date time format (year-month-day hour:minute:second).

```python
for value in daily_data.values():
    converted_data.append({
        'Timestamp': value['Timestamp'].strftime('%Y-%m-%d %H:%M:%S'),
        'Value': value['Value']
    })
```

Since, converting the timestamp value to datetime object while calculating the average value on daily basis. String format time function in python is used to convert datetime objects into 'year-month-day hour:minute:second' format. Hence, with the help of the function 'strftime' it was converted from datetime object into the desired format.

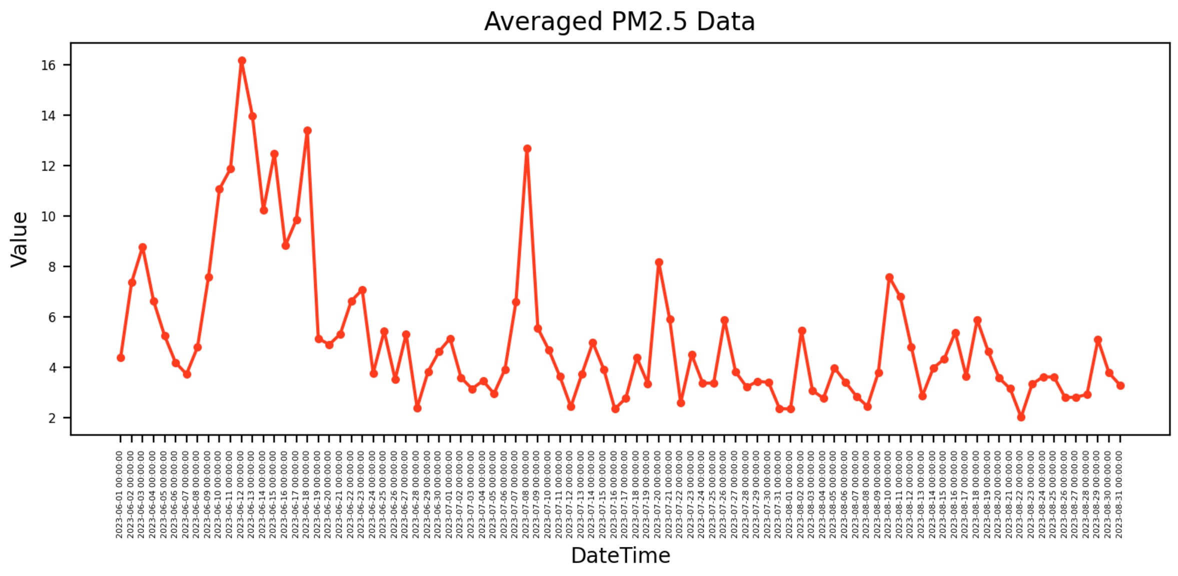
(c) **Use the line chart component of matplotlib to visualize averaged PM2.5 daily data, directly display the figure or save it as a file.**

Code Snippet:

```python
import matplotlib.pyplot as plt
import pandas as pd
import json
with open('final_data.json','r') as file:
    data = json.load(file)

data_df = pd.DataFrame(data)

plt.figure(figsize=(6,3), dpi=200)
plt.plot(data_df["Timestamp"], data_df["Value"], color="#FF3B1D", marker='.', linestyle="-")
plt.xticks(rotation=90, fontsize=4)
plt.yticks(fontsize=6)
plt.title("Averaged PM2.5 Data")
plt.xlabel("DateTime")
plt.ylabel("Value")

plt.tight_layout()
plt.savefig("figure.png")
plt.show()
```

- Matplotlib is used to plot graph based on the processed data.
- The previously stored data is imported and is plot into a two-dimensional table using the pandas library in python.
- Lastly, we adjust the characteristics of the graph and save the graph as a image and is displayed as well.

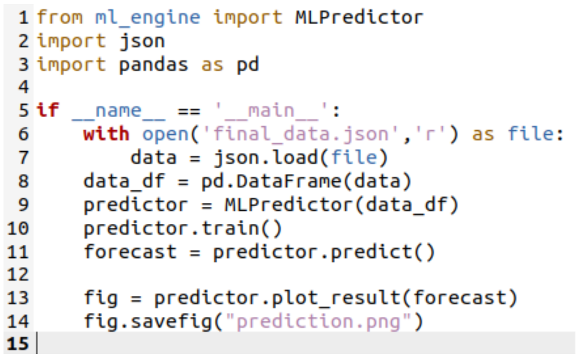
Output:



(d) Feed averaged PM2.5 data to machine learning model to predict the trend of PM2.5 for the next 15 days.

- The averaged data which was stored previously in a json file is fed to the machine learning model to predict.

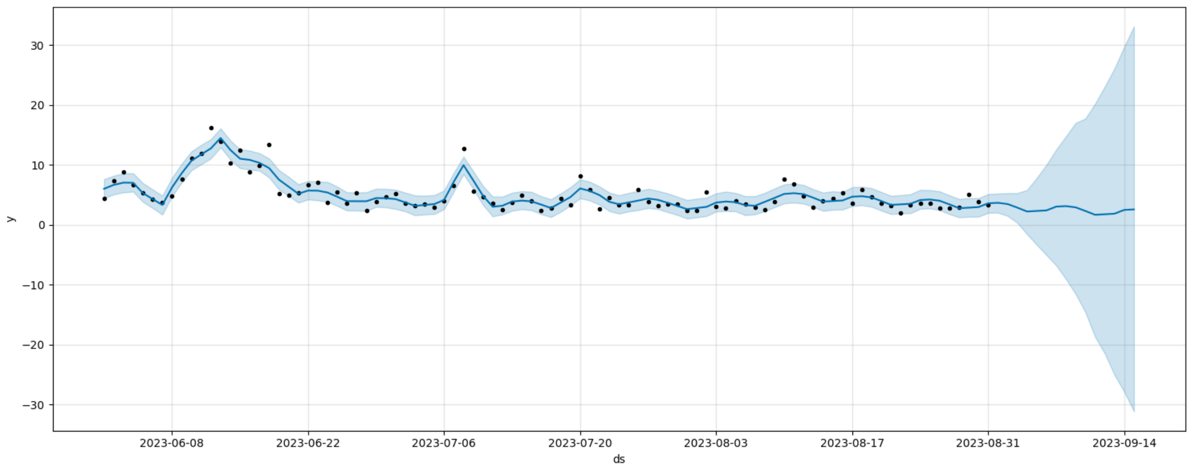
(e) Visualize predicted results from Machine Learning predictor/classifier model, directly display the figure or save it as a file.



```python
from ml_engine import MLPredictor
import json
import pandas as pd

if __name__ == '__main__':
    with open('final_data.json','r') as file:
        data = json.load(file)
    data_df = pd.DataFrame(data)
    predictor = MLPredictor(data_df)
    predictor.train()
    forecast = predictor.predict()

    fig = predictor.plot_result(forecast)
    fig.savefig("prediction.png")
```

- Importing the pre-defined Machine Learning engine code that was downloaded previously.
- Using this pre-defined Machine Learning engine code we predict the trend of PM2.5 for the next 15 days.

Output:



III. Analytical Discussion

Task 1

Using HTTP requests, we fetched data from Newcastle Urban Observatory and by focusing on the PM2.5 data. First task mainly focuses on requesting data from the Urban Observatory and sorting out the required data from it. Once all the requirements are satisfied, we send the data to Edge virtual machine to pre-process the data.

Task 2

To send the data to Edge virtual machine we make use of the service called EMQX which uses the MQTT protocol to transmit data, integration of Message Queuing Telemetry Transport (MQTT) ensures low latency and efficient data transferring. Once the data has been received in the Edge virtual machine, data preprocessing takes place to make the data ready for further processing. Once the requirements are satisfied, making use of RabbitMQ the data is transmitted to Cloud virtual machine from the edge.

Task 3

Once the data is received using RabbitMQ. High computing tasks takes place i.e., tasks like machine learning, predictions take place. Using matplotlib module that is available in python, plotting a graph on the averaged PM2.5 data. Furthermore, a pre-defined machine learning model is used to predict the data for 15 days ahead using the current data is performed. Once all the tasks are completed, an understanding of how to process Internet of Things (IoT) sensor data in the edge-cloud setting is achieved.

IV. Conclusion

This project has provided a comprehensive usage of Docker, MQTT using EMQX, RabbitMQ, and Machine learning to create an IoT data processing pipeline. The data flow from the device to edge layer and cloud layer has demonstrated a real time example of how an IoT device's data is transmitted and processed along different layers. The raw data that was collected by the device which was sorted and transmitted to the edge layer explains how sensors work in real time, as well as the pre-processing that was done in the edge layer helps us understand that we don't require something that is as powerful as cloud layer in terms of computational power to pre-process the data in the edge layer itself explains why we can save bandwidth instead of sending the data directly to the cloud without pre-processing it in the edge layer. Finally, the cloud layer which has high computational power is used to perform heavy tasks like plotting graphs and prediction of data using machine learning. Overall, the project has given a comprehensive solution for interpreting and processing IoT sensor data in an edge-cloud environment.