

Module: Internet of Things - CSC8112

Yash Gadodia - 240679307

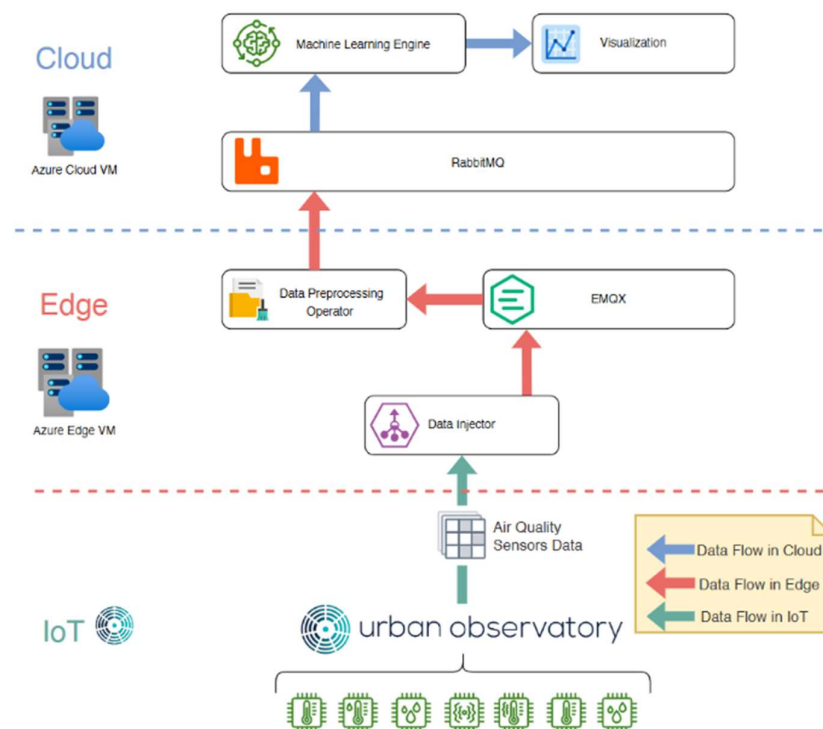
Aim

The aim of this coursework is to develop an Internet of Things (IoT) data processing pipeline that processes sensor data in an edge-cloud environment. The pipeline will be designed to perform operations such as data collection, preprocessing, visualization, and prediction by leveraging lightweight virtualization technology, specifically Docker. This setup will enable the integration of edge and cloud resources to support the implementation of a machine learning-based IoT pipeline, ultimately facilitating air quality predictions from urban data.

Introduction

The rapid expansion of IoT technologies has revolutionized the way urban environments are monitored and managed, providing real-time data that can inform decision-making and enhance the quality of life in cities. This coursework focuses on developing an IoT data processing pipeline that operates across both edge and cloud tiers, utilizing data from the Newcastle Urban Observatory (NCT UO). The NCT UO serves as a pivotal source of diverse, real-time urban data, encompassing over 50 distinct data types collected from sensors distributed throughout Newcastle city.

The proposed pipeline is architected to facilitate efficient data flow from the IoT tier to the cloud tier, encompassing several key components and stages:



1. IoT Tier:

- **Newcastle Urban Observatory (NCT UO):** Acts as the primary data source, providing a rich set of real-time urban metrics.

2. Edge Tier:

- **Data Injector:** A custom-developed software component responsible for interfacing with the NCT UO API to ingest data and transmit it to the machine learning pipeline.
- **EMQX Broker:** Utilizes the MQTT protocol to enable asynchronous communication between services, ensuring reliable message queuing within the pipeline.
- **Data Pre-processing Operator:** Processes raw data to prepare it for machine learning model training, ensuring data quality and relevance.

3. Cloud Tier:

- **RabbitMQ:** Serves as a cloud-based message queuing system to manage data flow and communication between cloud services.
- **Machine Learning Model/Classifier/Engine:** Trains on pre-processed data to predict future events, leveraging real-world data streams from the NCT UO.
- **Visualization Component:** Generates graphical representations of both raw time-series data and predictive analytics, facilitating intuitive data interpretation.

The integration of Docker as the virtualization technology stack enables the deployment of these components as microservices, promoting scalability, flexibility, and ease of management. Throughout this coursework, emphasis is placed on hands-on implementation using command-line interfaces and Python programming to configure Docker-based services, pull necessary images from Docker Hub, and develop machine learning models tailored to urban data streams.

Implementation

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). Perform this task first using the command line interface (CLI).

To initiate the EMQX broker on the Edge layer, we use the command:

docker pull emqx/emqx

```
docker run -d --name emqx -p 18083:18083 -p 1883:1883 emqx/emqx:latest
```

This command pulls the latest EMQX image from Docker Hub and runs it as a container in detached mode, mapping the default MQTT port (1883) on the host.

```
student@edge:~$ docker pull emqx/emqx
Using default tag: latest
latest: Pulling from emqx/emqx
302e3ee49805: Pull complete
30be1d7e6718: Pull complete
9f4dfdb00648: Pull complete
3cc4f9bc05be: Pull complete
2b21534bdbd1: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:9cfe9fb13c95306b4bc5eec1dde5c9265d8e06d60a7ef08f56c9d24bc35769d9
Status: Downloaded newer image for emqx/emqx:latest
docker.io/emqx/emqx:latest
student@edge:~$ docker run -d --name emqx_broker -p 1883:1883 -p 8083:8083 emqx/emqx
f498e55ba18c664d63c56e18af7d1653a937d333e46b3176aa256a450abe4680
student@edge:~$
```

Figure 1.1

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

- a) Collect data from Urban Observatory platform by sending HTTP request to the following url ([Urbanobservatory](#)). Following that, please print out the raw data streams that you collected on the console.

The figure below displays the output after executing the data collector, showing that raw data has been successfully retrieved from the Newcastle Urban Observatory.

```
Raw data collected from API:
```

```
{'sensors': [{'Third Party': {'0': False}, 'Ground Height Above Sea Level': {'0': 57.6699981689}, 'Raw ID': {'0': '79525'},  
    'sensor Name': {'0': 'PER AIRMON MONITOR1135100'}, 'Sensor Height Above Ground': {'0': 2.0}, 'Broker Name': {'0': 'aq mesh api'},  
    'Sensor Centroid Latitude': {'0': 54.979118}, 'Location (WKT)': {'0': 'POINT (-1.617676 54.979118)'}, 'data': {'NO2': {'Tt'  
estamp': 1685577600000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect  
Reading': False, 'Value': 6.20588}, {'Timestamp': 1685578500000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AI  
NOM MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 5.54224}, {'Timestamp': 1685579400000, 'Units': 'ugm -3',  
Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 4.20368}, {'Times  
stamp': 1685580300000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Re  
ading': False, 'Value': 3.48364}, {'Timestamp': 1685581200000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMO  
N MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 3.53816}, {'Timestamp': 1685582100000, 'Units': 'ugm -3', 'Va  
lue': 3.03808}, {'Timestamp': 1685583000000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Readin  
g': False, 'Value': 2.59398}, {'Timestamp': 1685583900000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MO  
TOR1135100', 'Flagged as Suspect Reading': False, 'Value': 4.51764}, {'Timestamp': 1685584800000, 'Units': 'ugm -3', 'Variab  
le': 'NO2', 'Sensor Name': 'PER_AIRMON_MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 2.8106}, {'Timestamp': 1  
85585700000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading'  
False, Value': 2.78428}, {'Timestamp': 1685586600000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITO  
R1135100', 'Flagged as Suspect Reading': False, 'Value': 2.42144}, {'Timestamp': 1685587500000, 'Units': 'ugm -3', 'Variable':  
'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 3.68668}, {'Timestamp': 1685  
88400000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': Fal  
se, Value': 3.87656}, {'Timestamp': 1685589300000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR11  
5100', 'Flagged as Suspect Reading': False, 'Value': 4.4316}, {'Timestamp': 1685590200000, 'Units': 'ugm -3', 'Variable': 'N  
O2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 3.92544}, {'Timestamp': 168555  
00000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': False,  
Value': 3.63216}, {'Timestamp': 1685592000000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR11351  
0', 'Flagged as Suspect Reading': False, 'Value': 4.53456}, {'Timestamp': 1685592900000, 'Units': 'ugm -3', 'Variable': 'NO2',  
Sensor Name': 'PER_AIRMON_MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 4.59848}, {'Timestamp': 1685593800  
00, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': False, V  
alue': 4.42364}, {'Timestamp': 1685594700000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100',  
Flagged as Suspect Reading': False, 'Value': 6.20964}, {'Timestamp': 1685595600000, 'Units': 'ugm -3', 'Variable': 'NO2',  
ensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': False, 'Value': 5.49524}, {'Timestamp': 1685596500000  
, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', 'Flagged as Suspect Reading': False, Valu  
' : 7.379}, {'Timestamp': 1685597400000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sensor Name': 'PER AIRMON MONITOR1135100', Fl  
agged as Suspect Reading': False, 'Value': 9.11612}, {'Timestamp': 1685598300000, 'Units': 'ugm -3', 'Variable': 'NO2', 'Sen
```

Figure 1.2

- b) Although the raw air quality data you collected from the Urban Observatory API contains many metrics including N O₂, NO, CO₂, PM_{2.5}, and PM₁₀, among others, for the purpose of this coursework, you only need to store and analyze PM_{2.5} data. While many meta-data are available for PM_{2.5} data, such as sensor name, timestamp, value, and location, you only need to store the metrics related to the Timestamp and Value meta-data fields.

We filter the data obtained from the Urban Observatory API to retain only the timestamp and value metadata fields, which are then displayed on the console.

```
{'Value': 4.273, 'Timestamp': 1685577600000}, {'Value': 3.483, 'Timestamp': 1685578500000}, {'Value': 3.713, 'Timestamp': 1685579400000}, {'Value': 3.813, 'Timestamp': 1685580300000}, {'Value': 3.885, 'Timestamp': 1685581200000}, {'Value': 3.809, 'Timestamp': 1685582100000}, {'Value': 3.869, 'Timestamp': 1685583000000}, {'Value': 3.839, 'Timestamp': 1685583900000}, {'Value': 3.805, 'Timestamp': 1685584800000}, {'Value': 4.094, 'Timestamp': 1685585700000}, {'Value': 4.474, 'Timestamp': 1685586600000}, {'Value': 4.918, 'Timestamp': 1685587500000}, {'Value': 5.56, 'Timestamp': 1685588400000}, {'Value': 5.637, 'Timestamp': 1685589300000}, {'Value': 5.71, 'Timestamp': 1685590200000}, {'Value': 6.004, 'Timestamp': 1685591100000}, {'Value': 5.322, 'Timestamp': 1685592000000}, {'Value': 4.961, 'Timestamp': 1685592900000}, {'Value': 4.679, 'Timestamp': 1685593800000}, {'Value': 4.815, 'Timestamp': 1685594700000}, {'Value': 4.498, 'Timestamp': 1685595600000}, {'Value': 4.393, 'Timestamp': 1685596500000}, {'Value': 4.433, 'Timestamp': 1685597400000}, {'Value': 4.467, 'Timestamp': 1685598300000}, {'Value': 5.76, 'Timestamp': 1685599200000}, {'Value': 4.549, 'Timestamp': 1685600100000}, {'Value': 4.499, 'Timestamp': 1685601000000}, {'Value': 4.946, 'Timestamp': 1685601900000}, {'Value': 4.864, 'Timestamp': 1685602800000}, {'Value': 6.079, 'Timestamp': 1685603700000}, {'Value': 5.902, 'Timestamp': 1685604600000}, {'Value': 6.025, 'Timestamp': 1685605500000}, {'Value': 3.99, 'Timestamp': 1685606400000}, {'Value': 3.223, 'Timestamp': 1685607300000}, {'Value': 2.434, 'Timestamp': 1685608200000}, {'Value': 2.07, 'Timestamp': 1685609100000}, {'Value': 1.968, 'Timestamp': 1685610000000}, {'Value': 2.497, 'Timestamp': 1685610900000}, {'Value': 3.192, 'Timestamp': 1685611800000}, {'Value': 3.166, 'Timestamp': 1685612700000}, {'Value': 2.994, 'Timestamp': 1685613600000}, {'Value': 3.023, 'Timestamp': 1685614500000}, {'Value': 3.652, 'Timestamp': 1685615400000}, {'Value': 2.243, 'Timestamp': 1685616300000}, {'Value': 2.652, 'Timestamp': 1685617200000}, {'Value': 3.137, 'Timestamp': 1685618100000}, {'Value': 3.075, 'Timestamp': 1685619000000}, {'Value': 3.376, 'Timestamp': 1685619900000}, {'Value': 3.494, 'Timestamp': 1685620800000}, {'Value': 3.639, 'Timestamp': 1685621700000}, {'Value': 3.826, 'Timestamp': 1685622600000}, {'Value': 3.888, 'Timestamp': 1685623500000}, {'Value': 4.861, 'Timestamp': 1685624400000}, {'Value': 5.634, 'Timestamp': 1685625300000}, {'Value': 5.097, 'Timestamp': 1685626200000}, {'Value': 5.414, 'Timestamp': 1685627100000}, {'Value': 6.029, 'Timestamp': 1685628000000}, {'Value': 5.582, 'Timestamp': 1685628900000}, {'Value': 5.885, 'Timestamp': 1685629800000}, {'Value': 4.228, 'Timestamp': 1685630700000}, {'Value': 4.729, 'Timestamp': 1685631600000}, {'Value': 4.843, 'Timestamp': 1685632500000}, {'Value': 5.195, 'Timestamp': 1685633400000}, {'Value': 4.848, 'Timestamp': 1685634300000}, {'Value': 5.279, 'Timestamp': 1685635200000}, {'Value': 4.905, 'Timestamp': 1685636100000}, {'Value': 3.846, 'Timestamp': 1685637000000}, {'Value': 3.344, 'Timestamp': 1685637900000}, {'Value': 3.533, 'Timestamp': 1685638800000}, {'Value': 3.769, 'Timestamp': 1685639700000}, {'Value': 3.571, 'Timestamp': 1685640600000}, {'Value': 4.7, 'Timestamp': 1685641500000}, {'Value': 4.365, 'Timestamp': 1685642400000}, {'Value': 3.379, 'Timestamp': 1685643300000}, {'Value': 3.65, 'Timestamp': 1685644200000}, {'Value': 4.062, 'Timestamp': 1685645100000}, {'Value': 4.384, 'Timestamp': 1685646000000}, {'Value': 4.546, 'Timestamp': 1685646900000}, {'Value': 4.785, 'Timestamp': 1685647800000}, {'Value': 4.646, 'Timestamp': 1685648700000}, {'Value': 3.858, 'Timestamp': 1685649600000}, {'Value': 4.017, 'Timestamp': 1685650500000}, {'Value': 4.1, 'Timestamp': 1685651400000}, {'Value': 4.372, 'Timestamp': 1685652300000}, {'Value': 4.599, 'Timestamp': 1685653200000}, {'Value': 5.043, 'Timestamp': 1685654100000}, {'Value': 5.614, 'Timestamp': 1685655000000}, {'Value': 5.468, 'Timestamp': 1685655900000}, {'Value': 5.435, 'Timestamp': 1685656800000}, {'Value': 5.385, 'Timestamp': 1685657700000}, {'Value': 4.732, 'Timestamp': 1685658600000}, {'Value': 5.029, 'Timestamp': 1685659500000}, {'Value': 5.128, 'Timestamp': 1685660400000}, {'Value': 5.514, 'Timestamp': 1685661300000}, {'Value': 5.505, 'Timestamp': 1685662200000}, {'Value': 5.388, 'Timestamp': 1685663100000}
```

Figure 1.3

- c) Send all PM_{2.5} data to be used by Task 2.2 (a) to EMQX service of Azure lab (Edge).

```
mqtt_subscriber.py:23: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
client = mqtt_client.Client()
PM2.5 Data Extracted Successfully!
PM2.5 Data Published to Topic: pm25_data
Connected to MQTT Broker!
```

Figure 1.4

Screenshots:

```

from paho.mqtt import client as mqtt_client
from urllib.request import urlopen

if __name__ == '__main__':
    mqtt_ip = "192.168.0.102"
    mqtt_port = 1883
    topic = "CSC8112"

    # Getting the data from the URL
    url = "http://uoweb3.nd.ac.uk/ap/v1.1/sensors/PER_AIRMON_MONITOR1135100/data/json/?starttime=20220601&endtime=20220831"

    # Storing the data from the Urban Observatory into the variable response
    response = urlopen(url)

    # data_json variable reads the data from the response and stores it as a JSON object
    data_json = json.loads(response.read())

    # Filter the JSON data to store only the PM2.5 variable
    data = data_json["sensors"][0]["data"]["PM2.5"]

    # Using a for loop to remove unnecessary items, keeping only Timestamp and Value
    for item in data:
        del item['Sensor Name']
        del item['Flagged as Suspect Reading']
        del item['Units']
        del item['Variable']

    # Create an MQTT client object
    client = mqtt_client.Client()

    # Callback function for MQTT connection
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT broker successfully")
        else:
            print("Failed to connect, return code %d\n", rc)

    # Connect to MQTT service
    client.on_connect = on_connect
    client.connect(mqtt_ip, mqtt_port)

    # Publish message to MQTT
    # Note: MQTT payload must be a string, bytearray, int, float, or None
    msg = json.dumps(data)
    client.publish(topic, msg)

```

```

import paho.mqtt.client as mqtt
import json
import time
import data_collector # Assuming this is the module where `extract_pm25_data()` is defined

# Function to confirm message publish
def on_publish(client, userdata, mid):
    print(f"Message {mid} published.") # Log successful message publication

# Function to send PM2.5 data to the EMQX MQTT broker
def send_data_to_emqx(pm25_data, broker_address="192.168.0.102"):
    if not pm25_data:
        print("No PM2.5 data to send.")
        return

    topic = 'iot/pm25'
    client = mqtt.Client()
    client.on_publish = on_publish

    try:
        # Connect to the broker
        print(f"Connecting to {broker_address}...")
        client.connect(broker_address, 1883, 60)
        client.loop_start() # Start the loop in the background

        # Publish each PM2.5 data point
        message = json.dumps(pm25_data)
        client.publish(topic, message)
        print(f"Successfully sent {len(pm25_data)} PM2.5 data points to {topic}.")
    except Exception as e:
        print(f"Error while sending data to EMQX: {e}")
    finally:
        client.loop_stop() # Stop the loop after publishing all data
        client.disconnect() # Disconnect from the broker

# Fetch and send PM2.5 data to EMQX broker
pm25_data = data_collector.extract_pm25_data()
send_data_to_emqx(pm25_data)

```

In this task, I pulled and ran the EMQX Docker image on an Azure VM using CLI commands, setting up a scalable MQTT broker for IoT data handling (Figure 1.1). I developed two Python scripts: data collector to fetch air quality data from the Urban Observatory API and print the collected data, and emqx publisher to filter only the PM2.5 data, extracting timestamp and value fields, and publishing this filtered data to EMQX using the paho-mqtt library. Figures 1.3 and 1.4 show the filtered data output and successful publishing

Task 2: Data preprocessing operator design

- 1) Define a Docker compose file which contains the following necessary configurations and instructions for deploying and instantiating the following set of Docker images on Azure lab (Cloud):

The figure below shows the Docker file created to run the RabbitMQ Docker image with specified port configurations.

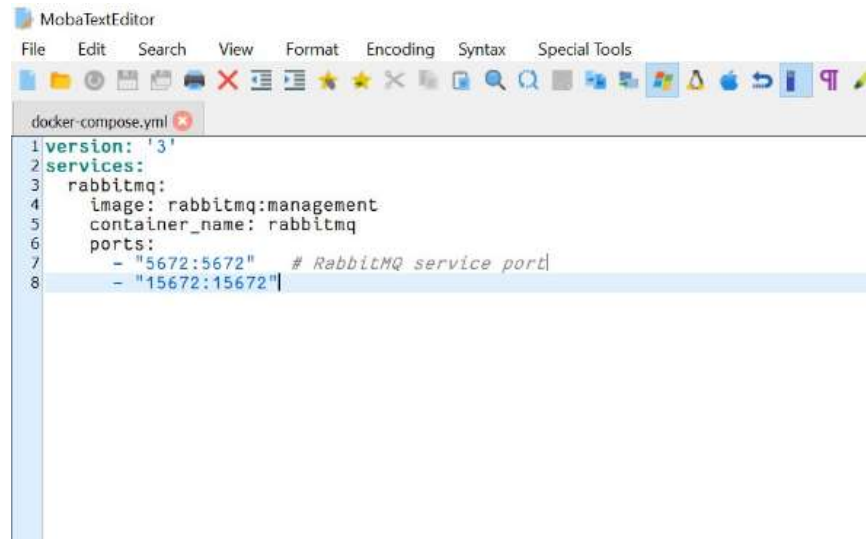


Figure 2.1

- a) Download and run RabbitMQ image (rabbitmq: management)

The image below displays the RabbitMQ container running on Azure Lab (Cloud) after executing the `docker-compose up` command in the cloud environment.

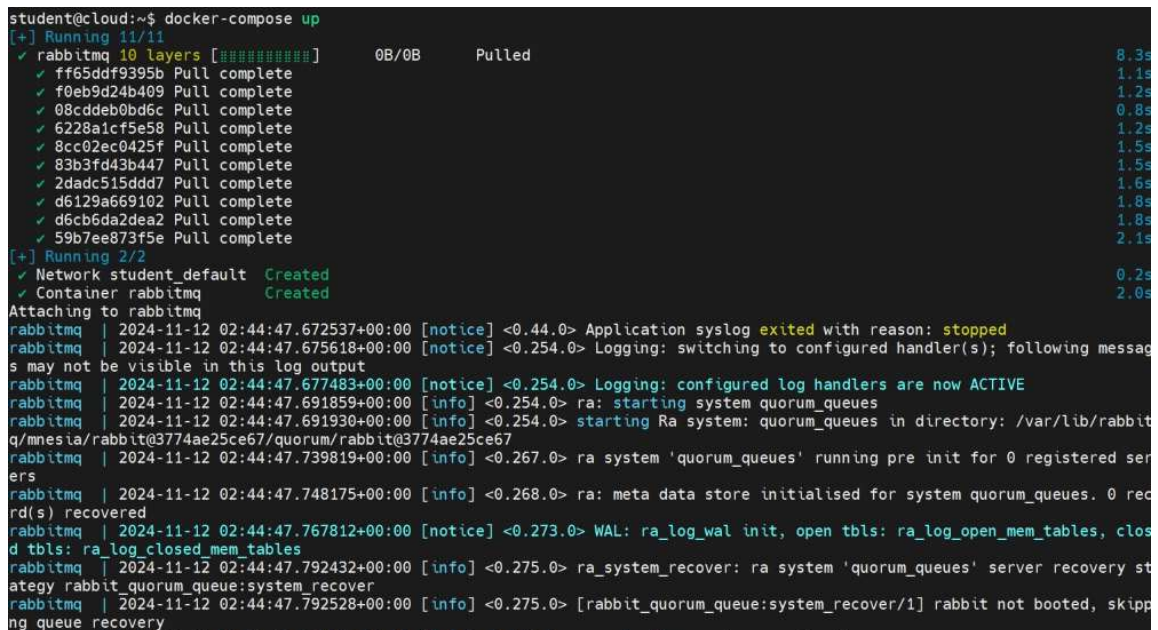


Figure 2.2

- 2) Design a data preprocessing operator with the following functions (code) in Azure Lab (Edge):
- Collect all PM2.5 data published by Task 1.2 from EMQX service, and please print out the PM2.5 data to the console (this operator will run as a Docker container, so the logs can be seen in the docker logs console automatically).

Connects to the EMQX broker in order to use paho-mqtt to gather the PM2.5 data. After connecting to the broker, the subscriber will print the necessary dataset.

```
{
  'Value': 6.003, 'Timestamp': 1693322100000,
  'Value': 5.924, 'Timestamp': 1693323000000,
  'Value': 6.617, 'Timestamp': 1693323900000,
  'Value': 7.044, 'Timestamp': 1693324800000,
  'Value': 6.475, 'Timestamp': 1693325700000,
  'Value': 5.816, 'Timestamp': 1693326600000,
  'Value': 6.093, 'Timestamp': 1693327500000,
  'Value': 5.555, 'Timestamp': 1693328400000,
  'Value': 4.755, 'Timestamp': 1693329300000,
  'Value': 4.125, 'Timestamp': 1693330200000,
  'Value': 4.279, 'Timestamp': 1693331100000,
  'Value': 4.437, 'Timestamp': 1693332000000,
  'Value': 4.308, 'Timestamp': 1693332900000,
  'Value': 4.38, 'Timestamp': 1693333800000,
  'Value': 4.638, 'Timestamp': 1693334700000,
  'Value': 4.571, 'Timestamp': 1693335600000,
  'Value': 4.052, 'Timestamp': 1693336500000,
  'Value': 3.669, 'Timestamp': 1693337400000,
  'Value': 3.894, 'Timestamp': 1693338300000,
  'Value': 5.048, 'Timestamp': 1693339200000,
  'Value': 6.893, 'Timestamp': 1693340100000,
  'Value': 4.941, 'Timestamp': 1693341000000,
  'Value': 3.315, 'Timestamp': 1693341900000,
  'Value': 3.35, 'Timestamp': 1693342800000,
  'Value': 3.187, 'Timestamp': 1693343700000,
  'Value': 2.807, 'Timestamp': 1693344600000,
  'Value': 2.747, 'Timestamp': 1693345500000,
  'Value': 2.699, 'Timestamp': 1693346400000,
  'Value': 2.647, 'Timestamp': 1693347300000,
  'Value': 2.936, 'Timestamp': 1693348200000,
  'Value': 3.132, 'Timestamp': 1693349100000,
  'Value': 3.281, 'Timestamp': 1693350000000,
  'Value': 3.21, 'Timestamp': 1693350900000,
  'Value': 3.158, 'Timestamp': 1693351800000,
  'Value': 3.392, 'Timestamp': 1693352700000,
  'Value': 3.171, 'Timestamp': 1693353600000,
  'Value': 3.183, 'Timestamp': 1693354500000,
  'Value': 3.334, 'Timestamp': 1693355400000,
  'Value': 3.3, 'Timestamp': 1693356300000,
  'Value': 3.25, 'Timestamp': 1693357200000,
  'Value': 3.401, 'Timestamp': 1693358100000,
  'Value': 3.474, 'Timestamp': 1693359000000,
  'Value': 3.367, 'Timestamp': 1693359900000,
  'Value': 3.281, 'Timestamp': 1693360800000,
  'Value': 3.235, 'Timestamp': 1693361700000,
  'Value': 3.19, 'Timestamp': 1693362600000,
  'Value': 3.367, 'Timestamp': 1693363500000,
  'Value': 3.81, 'Timestamp': 1693364400000,
  'Value': 3.512, 'Timestamp': 1693365300000,
  'Value': 3.617, 'Timestamp': 1693366200000,
  'Value': 3.524, 'Timestamp': 1693367100000,
  'Value': 3.419, 'Timestamp': 1693368000000,
  'Value': 3.579, 'Timestamp': 1693368900000,
  'Value': 3.69, 'Timestamp': 1693369800000,
  'Value': 5.248, 'Timestamp': 1693370700000,
  'Value': 5.465, 'Timestamp': 1693371600000,
  'Value': 5.334, 'Timestamp': 1693372500000,
  'Value': 3.834, 'Timestamp': 1693373400000,
  'Value': 3.864, 'Timestamp': 1693374300000,
  'Value': 4.045, 'Timestamp': 1693375200000,
  'Value': 4.146, 'Timestamp': 1693376100000,
  'Value': 3.371, 'Timestamp': 1693377000000,
  'Value': 3.438, 'Timestamp': 1693377900000,
  'Value': 2.943, 'Timestamp': 1693378800000,
  'Value': 2.954, 'Timestamp': 1693379700000,
  'Value': 5.547, 'Timestamp': 1693380600000,
  'Value': 2.406, 'Timestamp': 1693381500000,
  'Value': 2.709, 'Timestamp': 1693382400000,
  'Value': 2.211, 'Timestamp': 1693383300000,
  'Value': 2.569, 'Timestamp': 1693384200000,
  'Value': 2.584, 'Timestamp': 1693385100000,
  'Value': 2.731, 'Timestamp': 1693386000000,
  'Value': 2.995, 'Timestamp': 1693386900000,
  'Value': 2.373, 'Timestamp': 1693387800000,
  'Value': 2.741, 'Timestamp': 1693388700000,
  'Value': 2.907, 'Timestamp': 1693389600000,
  'Value': 2.477, 'Timestamp': 1693390500000,
  'Value': 2.293, 'Timestamp': 1693391400000,
  'Value': 2.438, 'Timestamp': 1693392300000,
  'Value': 2.706, 'Timestamp': 1693393200000,
  'Value': 2.581, 'Timestamp': 1693394100000,
  'Value': 2.766, 'Timestamp': 1693395000000,
  'Value': 2.868, 'Timestamp': 1693395900000,
  'Value': 2.686, 'Timestamp': 1693396800000,
  'Value': 4.479, 'Timestamp': 1693397700000,
  'Value': 3.288, 'Timestamp': 1693400400000,
  'Value': 3.1, 'Timestamp': 1693401300000,
  'Value': 3.494, 'Timestamp': 1693402200000,
  'Value': 3.186, 'Timestamp': 1693403100000,
  'Value': 3.36, 'Timestamp': 1693404000000,
  'Value': 3.644, 'Timestamp': 1693404900000,
  'Value': 3.457, 'Timestamp': 1693405800000,
  'Value': 3.218, 'Timestamp': 1693406700000,
  'Value': 4.1674, 'Timestamp': 1693407600000,
  'Value': 3.315, 'Timestamp': 1693408500000,
  'Value': 3.801, 'Timestamp': 1693409400000,
}
```

Figure 2.1

- Filter out outliers (the value greater than 50), and please print out outliers to the console (docker logs console).

We filter out any PM2.5 data values greater than 50 as outliers and print the remaining values to the Docker logs console.

```
Outliers:
Timestamp: 1654542900000, Value: 170.7
Timestamp: 1654543800000, Value: 72.13
Timestamp: 1654545600000, Value: 92.52
Timestamp: 1660051800000, Value: 98.11
```

Figure 2.2

- c) Since the original PM2.5 data readings are collected every 15 mins, so please implement a python code to calculate the averaging value of PM2.5 data on daily basis (every 24 hours), please pick the first start date of a day or a 24 hours interval as the new timestamp of averaged PM2.5 data, and please print out the result to the console (docker logs console).

The collected data is then averaged over a 24-hour period to generate a new timestamp for the PM2.5 data, and the output is displayed on the Docker console.

```
Daily average for 1/6/2022 is 5.62
Daily average for 2/6/2022 is 6.28
Daily average for 3/6/2022 is 5.62
Daily average for 4/6/2022 is 4.23
Daily average for 5/6/2022 is 5.39
Daily average for 6/6/2022 is 6.59
Daily average for 7/6/2022 is 9.99
Daily average for 8/6/2022 is 4.75
Daily average for 9/6/2022 is 3.37
Daily average for 10/6/2022 is 8.56
Daily average for 11/6/2022 is 5.66
Daily average for 12/6/2022 is 4.34
Daily average for 13/6/2022 is 4.92
Daily average for 14/6/2022 is 4.93
```

Figure 2.3

- d) Transfer all results (averaged PM2.5 data) to be used by Task 3.2 (a) into RabbitMQ service on Azure lab (Cloud).

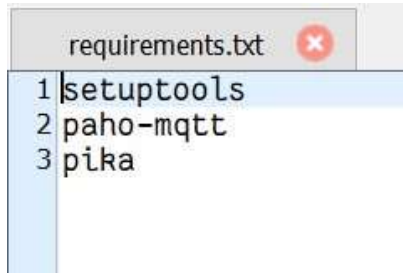
The output generated from the pre-processor code is then transmitted to the Azure Lab (Cloud) via the RabbitMQ publisher

```
'Value': 6.617}, {'Timestamp': 1693324800000, 'Value': 7.044}, {'Timestamp': 1693325700000, 'Value': 6.475}, {'Timestamp': 1693326600000, 'Value': 5.816}, {'Timestamp': 1693327500000, 'Value': 6.093}, {'Timestamp': 1693328400000, 'Value': 5.555}, {'Timestamp': 1693329300000, 'Value': 4.755}, {'Timestamp': 1693330200000, 'Value': 4.125}, {'Timestamp': 1693331100000, 'Value': 4.279}, {'Timestamp': 1693332000000, 'Value': 4.437}, {'Timestamp': 1693332900000, 'Value': 4.308}, {'Timestamp': 1693333800000, 'Value': 4.38}, {'Timestamp': 1693334700000, 'Value': 4.638}, {'Timestamp': 1693335600000, 'Value': 4.571}, {'Timestamp': 1693336500000, 'Value': 4.052}, {'Timestamp': 1693337400000, 'Value': 3.669}, {'Timestamp': 1693338300000, 'Value': 3.894}, {'Timestamp': 1693339200000, 'Value': 5.048}, {'Timestamp': 1693340100000, 'Value': 6.893}, {'Timestamp': 1693341000000, 'Value': 4.941}, {'Timestamp': 1693341900000, 'Value': 3.315}, {'Timestamp': 1693342800000, 'Value': 3.35}, {'Timestamp': 1693343700000, 'Value': 3.187}, {'Timestamp': 1693344600000, 'Value': 2.807}, {'Timestamp': 1693345500000, 'Value': 2.747}, {'Timestamp': 1693346400000, 'Value': 2.693}, {'Timestamp': 1693347300000, 'Value': 2.647}, {'Timestamp': 1693348200000, 'Value': 2.936}, {'Timestamp': 1693349100000, 'Value': 3.132}, {'Timestamp': 1693350000000, 'Value': 3.281}, {'Timestamp': 1693350900000, 'Value': 3.21}, {'Timestamp': 1693351800000, 'Value': 3.158}, {'Timestamp': 1693352700000, 'Value': 3.392}, {'Timestamp': 1693353600000, 'Value': 3.171}, {'Timestamp': 1693354500000, 'Value': 3.183}, {'Timestamp': 1693355400000, 'Value': 3.334}, {'Timestamp': 1693356300000, 'Value': 3.33}, {'Timestamp': 1693357200000, 'Value': 3.25}, {'Timestamp': 1693358100000, 'Value': 3.401}, {'Timestamp': 1693359000000, 'Value': 3.474}, {'Timestamp': 1693359900000, 'Value': 3.367}, {'Timestamp': 1693360800000, 'Value': 3.281}, {'Timestamp': 1693361700000, 'Value': 3.235}, {'Timestamp': 1693362600000, 'Value': 3.19}, {'Timestamp': 1693363500000, 'Value': 3.367}, {'Timestamp': 1693364400000, 'Value': 3.81}, {'Timestamp': 1693365300000, 'Value': 3.512}, {'Timestamp': 1693366200000, 'Value': 3.617}, {'Timestamp': 1693367100000, 'Value': 3.524}, {'Timestamp': 1693368000000, 'Value': 3.419}, {'Timestamp': 1693368900000, 'Value': 3.579}, {'Timestamp': 1693369800000, 'Value': 3.69}, {'Timestamp': 1693370700000, 'Value': 5.248}, {'Timestamp': 1693371600000, 'Value': 5.465}, {'Timestamp': 1693372500000, 'Value': 5.334}, {'Timestamp': 1693373400000, 'Value': 3.834}, {'Timestamp': 1693374300000, 'Value': 3.864}, {'Timestamp': 1693375200000, 'Value': 4.045}, {'Timestamp': 1693376100000, 'Value': 4.146}, {'Timestamp': 1693377000000, 'Value': 3.371}, {'Timestamp': 1693377900000, 'Value': 3.438}, {'Timestamp': 1693378800000, 'Value': 2.943}, {'Timestamp': 1693379700000, 'Value': 2.954}, {'Timestamp': 1693380600000, 'Value': 5.547}, {'Timestamp': 1693381500000, 'Value': 2.406}, {'Timestamp': 1693382400000, 'Value': 2.709}, {'Timestamp': 1693383300000, 'Value': 2.211}, {'Timestamp': 1693384200000, 'Value': 2.569}, {'Timestamp': 1693385100000, 'Value': 2.584}, {'Timestamp': 1693386000000, 'Value': 2.731}, {'Timestamp': 1693386900000, 'Value': 2.995}, {'Timestamp': 1693387800000, 'Value': 2.373}, {'Timestamp': 1693388700000, 'Value': 2.741}, {'Timestamp': 1693389600000, 'Value': 2.907}, {'Timestamp': 1693390500000, 'Value': 2.477}, {'Timestamp': 1693391400000, 'Value': 2.293}, {'Timestamp': 1693392300000, 'Value': 2.438}, {'Timestamp': 1693393200000, 'Value': 2.706}, {'Timestamp': 1693394100000, 'Value': 2.581}, {'Timestamp': 1693395000000, 'Value': 2.766}, {'Timestamp': 1693395900000, 'Value': 2.868}, {'Timestamp': 1693396800000, 'Value': 2.686}, {'Timestamp': 1693397700000, 'Value': 4.479}, {'Timestamp': 1693398600000, 'Value': 3.288}, {'Timestamp': 1693400000000, 'Value': 3.1}, {'Timestamp': 1693401000000, 'Value': 3.494}, {'Timestamp': 1693402000000, 'Value': 3.186}, {'Timestamp': 1693403000000, 'Value': 3.36}, {'Timestamp': 1693404000000, 'Value': 3.644}, {'Timestamp': 1693405000000, 'Value': 3.457}, {'Timestamp': 1693406000000, 'Value': 3.218}, {'Timestamp': 1693407000000, 'Value': 4.674}, {'Timestamp': 1693408000000, 'Value': 3.315}, {'Timestamp': 1693409000000, 'Value': 3.801}, {'Timestamp': 1693410000000, 'Value': 3.685}, {'Timestamp': 1693411000000, 'Value': 3.245}, {'Timestamp': 1693412000000, 'Value': 2.773}, {'Timestamp': 1693413000000, 'Value': 2.365}, {'Timestamp': 1693413900000, 'Value': 2.849}, {'Timestamp': 1693414800000, 'Value': 3.078}, {'Timestamp': 1693415700000, 'Value': 3.111}, {'Timestamp': 1693416600000, 'Value': 2.737}, {'Timestamp': 1693417500000, 'Value': 2.841}, {'Timestamp': 1693418400000, 'Value': 2.911}, {'Timestamp': 1693419300000, 'Value': 5.524}
```

Figure 2.4

- 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).

Screenshots:



```
requirements.txt
1|setuptools
2|paho-mqtt
3|pika
```

```
1 # Use the python:3.8.12 image as the base image
2 FROM python:3.8.12
3
4 # Switch to root user to install dependencies
5 USER root
6
7 # Copy all source files from the local machine to the workdir inside the container
8 ADD . /usr/local/source
9
10 # Change the working directory inside the container
11 WORKDIR /usr/local/source
12
13 # Install dependencies listed in requirements.txt
14 RUN pip3 install -r requirements.txt
15
16 # Start the data preprocessing task by running the main Python file
17 CMD ["python3", "subscriber.py"]
18
```

```
import pika
import json

def rabbitmq_publisher(data):
    # Configuration for RabbitMQ
    rabbitmq_ip = "192.168.0.100"
    rabbitmq_port = 5672
    rabbitmq_queue = "pm25_daily_averages"

    # Establish a connection to RabbitMQ
    connection = pika.BlockingConnection(pika.ConnectionParameters(host=rabbitmq_ip, port=rabbitmq_port))
    channel = connection.channel()

    # Declare the queue to ensure it exists
    channel.queue_declare(queue=rabbitmq_queue)

    # Publish the data
    channel.basic_publish(
        exchange="",
        routing_key=rabbitmq_queue,
        body=json.dumps(data),
        properties=pika.BasicProperties(delivery_mode=2) # Make message persistent
    )

    print(f"Sent daily average to RabbitMQ: {data}", flush=True)

    # Close the connection
    connection.close()
```

```

import json
import datetime
from paho.mqtt import client as mqtt_client
from rabbitmq_publisher import rabbitmq_publisher

def process_data(data):
    outliers = []
    daily_averages = []

    daily_total = 0
    daily_count = 0
    current_date = None

    filtered_data = []
    for d in data:
        if 'value' in d:
            if d['value'] > 50:
                outliers.append(d)
            else:
                filtered_data.append(d)
        else:
            print(f"Missing 'value' key in data: {d}", flush=True)

    Calculate daily averages
    for entry in filtered_data:
        timestamp = entry['timestamp']
        value = entry['value']

        Convert timestamp to datetime and isolate the date
        date_time = datetime.datetime.fromtimestamp(timestamp / 1000)
        date = date_time.date()

        Check if date has changed to reset daily calculation
        if date != current_date:
            if current_date is not None and daily_count > 0:
                Save the average for the previous day
                daily_averages.append({
                    'timestamp': current_date.strftime('%Y-%m-%d'),
                    'value': daily_total / daily_count
                })

            Reset totals for the new day
            current_date = date
            daily_total = value
            daily_count = 1
        else:
            daily_total += value
            daily_count += 1

    Final day's data, if any
    if current_date is not None and daily_count > 0:
        daily_averages.append({
            'timestamp': current_date.strftime('%Y-%m-%d'),
            'value': daily_total / daily_count
        })

    Print calculated daily averages to Docker logs
    print("Daily Averages:", daily_averages, flush=True)

    Publish the daily averages to RabbitMQ
    rabbitmq_publisher(daily_averages)

```

```

student@edge:~/data_preprocessor$ docker-compose up --build -d
Creating network "data_preprocessor_default" with the default driver
Building mqtt_subscriber
Step 1/6 : FROM python:3.8.12
----> 52bb9574949f
Step 2/6 : USER root
----> Using cache
----> dfd3baa5a8c0
Step 3/6 : ADD . /usr/local/source
----> Using cache
----> 942ef71d6aeb
Step 4/6 : WORKDIR /usr/local/source
----> Using cache
----> 67173d41c2a0
Step 5/6 : RUN pip3 install -r requirements.txt
----> Using cache
----> eea0c7ea8a23
Step 6/6 : CMD ["python3", "mqtt_subscriber.py"]
----> Using cache
----> ef8eeca55d83
Successfully built ef8eeca55d83
Successfully tagged mqtt_subscriber:latest
Creating data_preprocessor_mqtt_subscriber_1 ... done

```

MobaTextEditor

File Edit Search View Format Encoding Syntax Sp

* docker-compose.yml

```

1 version: "3"
2
3 services:
4   mqtt_subscriber:
5     build: ./subscriber
6     image: mqtt_subscriber:latest
7     environment:
8       RABBITMQ_BROKER: "192.168.0.100"
9       EMQX_BROKER: "192.168.0.102"
10

```

```

import json
import time
from paho.mqtt import client as mqtt_client
from data_preprocessor import process_data

# Callback when a message is received from the MQTT broker
def on_message(client, userdata, msg):
    print(f"Received message from topic '{msg.topic}': {msg.payload.decode()}", flush=True)
    data = json.loads(msg.payload.decode())
    data_preprocessor.process_data(data)

# Running the subscriber
if __name__ == '__main__':
    print("Starting MQTT Subscriber...")
    mqtt_ip = "192.168.0.102"
    mqtt_port = 1883
    topic = "iot/pm25"

    # Create a mqtt client object
    client = mqtt_client.Client()

    # Callback function for MQTT connection
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT OK!")
        else:
            print(f"Failed to connect, return code (rc){n}")

    # Set the callback functions
    client.on_connect = on_connect
    client.on_message = on_message

    # Connect to MQTT service
    client.connect(mqtt_ip, mqtt_port)

    # Subscribe to the topic
    client.subscribe(topic)

    # Start a thread to monitor messages from the publisher
    client.loop_forever()

```

Task 3: Time-series data prediction and visualization

- 1) Download a pre-defined Machine Learning (ML) engine code from [https://github.com/ncl-iot-team/CSC8112_MLEngine].

We download and utilize a pre-existing machine learning code to make predictions on the data.

```
'''
Author: Rui Sun
Date: 2022-09-25
Predict timeseries data
Official guide book of Prophet: https://facebook.github.io/prophet/docs/quick_start.html#python-api
'''

from prophet import Prophet

class MLPredictor(object):
    '''
    Example usage method:
    from ml_engine import MLPredictor
    predictor = MLPredictor(pm25_df)
    predictor.train()
    forecast = predictor.predict()
    fig = predictor.plot_result(forecast)
    fig.savefig(os.path.join("Your target dir path", "Your target file name"))
    '''

    def __init__(self, data_df):
        '''
        :param data_df: Dataframe type dataset
        '''
        self.__train_data = self.__convert_col_name(data_df)
        self.__trainer = Prophet(Changepoint_prior_scale=12)

    def train(self):
        self.__trainer.fit(self.__train_data)

    def __convert_col_name(self, data_df):
        data_df.rename(columns={"timestamp": "ds", "Value": "y"}, inplace=True)
        print(f"After rename columns \n{data_df.columns}")
        return data_df
```

- 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 by Task 2.2 (d) from RabbitMQ service, and please print out them to the console.

We execute the rabbitmq_consumer.py code, which collects all the data published by the RabbitMQ service and prints the results to the console.

```
import pika
import json
import pandas as pd
from plot_data import plot_example_data
from ml_predictor import ml_prediction

rabbitmq_ip = "192.168.0.100"
rabbitmq_port = 5672
rabbitmq_queue = "pm25_daily_averages"

connection = pika.BlockingConnection(pika.ConnectionParameters(host=rabbitmq_ip, port=rabbitmq_port))
channel = connection.channel()

channel.queue_declare(queue=rabbitmq_queue)

processed_data = []

def callback(ch, method, properties, body):
    pm25_data = json.loads(body)
    data_df = pd.DataFrame(pm25_data)

    plot_example_data(pm25_data)
    ml_prediction(data_df)

channel.basic_consume(queue=rabbitmq_queue, on_message_callback=callback, auto_ack=True)

print('Waiting for data...')
channel.start_consuming()
```


- b) Convert timestamp to date time format (year-month-day hour:minute:second), and please print out the PM2.5 data with the reformatted timestamp to the console.

We convert the obtained data into a date-time format by passing it to the process function, as outlined in the below above.

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import pandas as pd
import os

def plot_example_data(processed_data):
    data_df = pd.DataFrame(processed_data)

    plt.figure(figsize=(8, 4), dpi=200)
    plt.plot(data_df["Timestamp"], data_df["Value"], color="#FF3B1D", marker='.', linestyle="-")

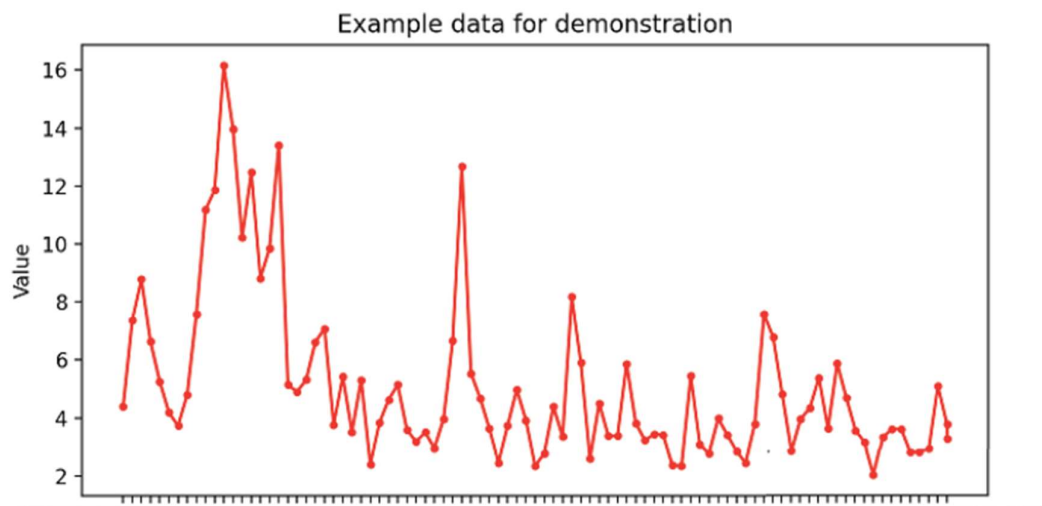
    plt.title("Example Data for Demonstration")
    plt.xlabel("DateTime")
    plt.ylabel("Value")

    plt.savefig("figure1.png")

if __name__ == "__main__":
    # Example of how to call plot_example_data
    sample_data = [
        {"Timestamp": "2024-11-01 10:00:00", "Value": 25},
        {"Timestamp": "2024-11-01 11:00:00", "Value": 30},
        {"Timestamp": "2024-11-01 12:00:00", "Value": 28}
    ]
    plot_example_data(sample_data)
```

- 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.

The averaged PM2.5 daily data is passed to the function, where it is represented visually using matplotlib.

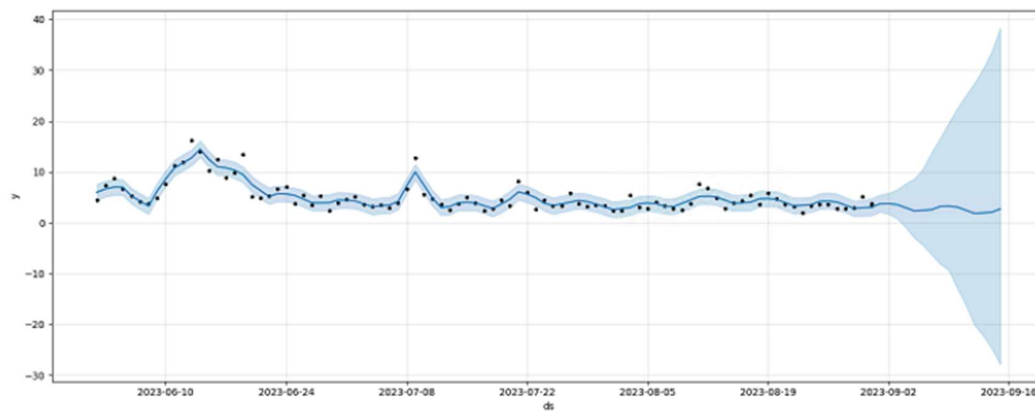


- d) Feed averaged PM2.5 data to machine learning model to predict the trend of PM2.5 for the next 15 days (this predicted time period is a default setting of provided machine learning predictor/classifier model).

The average daily PM2.5 data is then input into the provided machine learning model (mentioned in Task 3, Part 1) to predict the PM2.5 levels for the next 15 days.

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

Once the machine learning model was implemented, we were able to generate a visual representation of the predicted PM2.5 data for the next 15 days.



Screenshots:

```
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import pandas as pd
from ml_engine import MLPredictor

def ml_prediction(data_df):
    if 'Timestamp' in data_df.columns and 'Value' in data_df.columns:
        predictor = MLPredictor(data_df)

        predictor.train()
        forecast = predictor.predict()
        fig = predictor.plot_result(forecast)
        fig.savefig("predicted_pm25_trend.png")
        fig.show()

    plt.close(fig)
    else:
        print("Error: The data doesn't contain required columns 'Timestamp' and 'Value'")

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import pandas as pd
import os

def plot_example_data(processed_data):
    data_df = pd.DataFrame(processed_data)

    plt.figure(figsize=(8, 4), dpi=200)
    plt.plot(data_df["Timestamp"], data_df["value"], color="#FF3B1D", marker='.', linestyle="-")
    plt.title("Example data for demonstration")
    plt.xlabel("DateTime")
    plt.ylabel("Value")
    plt.savefig("Figure1.png")
    plt.show()

# Ensure the if block is properly closed if you are running this file as a script
if __name__ == '__main__':
```

Analytical Discussion:

In this project, we gained valuable insights into the PM2.5 data over time and generated predictions showing a decline in PM2.5 values over the next 15 days. The PM2.5 graph revealed that the highest concentration occurred in June, followed by a gradual decrease through November.

Conclusion:

This project allowed us to implement an IoT pipeline using two different messaging systems (EMQX and RabbitMQ), functioning across the Azure Edge and Azure Cloud layers. It provided a deeper understanding of IoT principles and demonstrated the use of various tools, such as Docker, to containerize code into lightweight images that run on the edge layer for efficient computation. The project has enhanced my knowledge of different modules and techniques needed to implement a basic IoT pipeline.

References:

- [Stackoverflow](#)– For resolving implementation-related queries.
- [Docker](#)– Docker documentation.
- [Github](#)- For various code references and set-up of pipeline