# Report on Azure Data Pipeline Implementation

**Introduction**

This report details the steps taken to create an Azure Data Pipeline, starting from the creation of an Azure account, setting up a virtual machine, and developing a data producer that generates sample data in JSON format. Additionally, it covers the integration of Azure Storage Queue for message queuing.

**Step 1: Azure Account Creation**

The first step involved creating an Azure account, which serves as the foundation for accessing various Azure services, including virtual machines, data storage, and data processing tools.

**Step 2: Virtual Machine Setup**

After successfully creating the Azure account, a virtual machine (VM) was set up. This VM will be used to run the data producer script and facilitate the data pipeline process. The VM configuration was chosen based on the requirements of the data processing tasks.

**Step 3: Data Producer Development**

The next step was to develop a data producer that generates sample data in JSON format. This was accomplished by creating a Python script on my laptop. Below are the details of the implementation:

**1. Data Producer**

 **Objective**: To generate sample data in JSON format for testing and demonstration purposes.

 **Libraries Used**:

 json: For handling JSON data.

 random: For generating random values.

 datetime: For generating timestamps.

**Python Script Example**

Here is the Python script used to generate the sample data:

import json

import random

from datetime import datetime

# Function to generate random sample data

def generate\_data():

    # Sample data fields

    data = {

        "id": random.randint(1000, 9999),  # Random ID between 1000 and 9999

        "temperature": round(random.uniform(10.0, 35.0), 2),  # Random temperature between 10 and 35

        "humidity": round(random.uniform(20.0, 80.0), 2),     # Random humidity between 20% and 80%

        "timestamp": datetime.now().strftime('%Y-%m-%d %H:%M:%S')  # Current date and time

    }

    return data

# Generate data continuously (if needed)

def produce\_data():

  with open('data\_output.json', 'a') as f:

    for \_ in range(10):  # Produces 10 data entries

        data = generate\_data()

        json\_data = json.dumps(data)  # Convert data to JSON format

        f.write(json\_data + '\n')

        print(json\_data)  # Print the JSON data (or you could save it to a file)

# Call the data producer function

if \_\_name\_\_ == "\_\_main\_\_":

    produce\_data()

**Explanation of the Script**

 The script defines a function generate\_data that creates a dictionary with random values for id, temperature, humidity, and timestamp.

 The produce\_data function generates 10 entries of data, converts each entry to JSON format, and appends it to a file named data\_output.json.

**Step 4: Message Queue Integration**

After developing the data producer, the next step was to integrate Azure Storage Queue for message queuing.

**2. Message Queue**

 **Objective**: To send generated data to an Azure Storage Queue for further processing.

 **Azure Service Used**: Azure Storage Account with Queue Service.

 **Library Used**: azure-storage-queue.

**Updated Python Script Example**

Here is the updated Python script that integrates the message queue:

import json

import random

from datetime import datetime

from azure.storage.queue import QueueClient

# Function to generate random sample data

def generate\_data():

    # Sample data fields

    data = {

        "id": random.randint(1000, 9999),  # Random ID between 1000 and 9999

        "temperature": round(random.uniform(10.0, 35.0), 2),  # Random temperature between 10 and 35

        "humidity": round(random.uniform(20.0, 80.0), 2),     # Random humidity between 20% and 80%

        "timestamp": datetime.now().strftime('%Y-%m-%d %H:%M:%S')  # Current date and time

    }

    return data

# Function to send data to Azure Queue

def send\_data\_to\_queue(data):

    # Azure Storage Queue connection string and queue name

    connection\_string = -----------

    queue\_name = "data-queue"  # Your queue name here

    # Create a QueueClient

    queue\_client = QueueClient.from\_connection\_string(conn\_str=connection\_string, queue\_name=queue\_name)

    # Send the message to the queue

    message = json.dumps(data)

    queue\_client.send\_message(message)

    print(f"Sent message to queue: {message}")

# Generate data and send to the queue

def produce\_data():

    for \_ in range(10):  # Produces 10 data entries

        data = generate\_data()

        send\_data\_to\_queue(data)  # Send data to the Azure queue

# Call the data producer function

if \_\_name\_\_ == "\_\_main\_\_":

    produce\_data()

I installed the required python libraries:

pip install azure-eventhub azure-storage-queue.

But I got error message about Virtual Environment. Then I created a virtual envioronment.

Python3 –m venv myvenv

Then I activated the virtual environment:

Source myenv/bin/activate

Then I again installed PIP. Then run my python script: python3 data\_queue.py and it work.

**Step 5: Serverless SQL Solution**

In this step I created Azure SQL server and database. My server name is “sql---------” and database name is “------“. Then I created a python send to send data in sql server. Here is my python script:

import json

import pyodbc

from azure.storage.queue import QueueClient

def get\_data\_from\_queue():

    queue\_name = "data-queue"

    connection\_string = ---------Replace with your connection string

    queue\_client = QueueClient.from\_connection\_string(conn\_str=connection\_string, queue\_name=queue\_name)

    messages = queue\_client.receive\_messages()

    for message in messages:

        print(f"Raw message content: {message.content}")  # Log raw message for debugging

        process\_message(message)

        queue\_client.delete\_message(message)  # Remove message from queue after processing

def process\_message(message):

    data = json.loads(message.content)

    print(f"Parsed data: {data}")  # Log parsed data for debugging

    # Check if required fields are present

    if 'id' in data and 'temperature' in data and 'humidity' in data and 'timestamp' in data:

        insert\_data\_to\_sql(data)

    else:

        print(f"Skipped inserting due to missing critical data: {data}")

def insert\_data\_to\_sql(data):

    server = '-------------------------'

    database = '-------'

    username = 'sql'

    password = '-------'  # Replace with your actual password

    driver = '{ODBC Driver 17 for SQL Server}'

    conn\_str = (

        'DRIVER=' + driver + ';'

        'SERVER=' + server + ';'

        'PORT=1433;'

        'DATABASE=' + database + ';'

        'UID=' + username + ';'

        'PWD=' + password + ';'

        'Encrypt=yes;'

        'TrustServerCertificate=no;'

        'Connection Timeout=30;'

    )

    conn = pyodbc.connect(conn\_str)

    cursor = conn.cursor()

    # Adjust the SQL insert statement to match the data structure

    insert\_query = """

    INSERT INTO SensorData (id, temperature, humidity, timestamp)

    VALUES (?, ?, ?, ?)

    """

    cursor.execute(insert\_query, (data['id'], data['temperature'], data['humidity'], data['timestamp']))

    conn.commit()

    cursor.close()

    conn.close()

    print(f"Inserted data: {data}")

if \_\_name\_\_ == "\_\_main\_\_":

    get\_data\_from\_queue()

Before running my data consumer script, ensure that I have the necessary Python libraries installed on your VM. I’ll need pyodbc for SQL Server connectivity and azure-storage-queue for reading messages from Azure Queue Storage. I installed the required python libraries:

pip install pyodbc azure-storage-queue .

**Check ODBC Driver Installation:** I hadeEnsure that the ODBC driver is properly installed by running the following command: **dpkg -l | grep odbc**

After that I had to install ODBC driver by running the following command:

sudo su

curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -

curl https://packages.microsoft.com/config/ubuntu/22.04/prod.list > /etc/apt/sources.list.d/mssql-release.list

exit

sudo apt-get update

sudo ACCEPT\_EULA=Y apt-get install -y msodbcsql17

Then I have executed my python script by following command: Python3 sql\_data\_queue.py

It was successfully executed. Then I have run my SQL query to verify that data has been inserted into my SQL database.

**Step 6: Create Logic App:** I created Logic App . Here my Logic App Design.

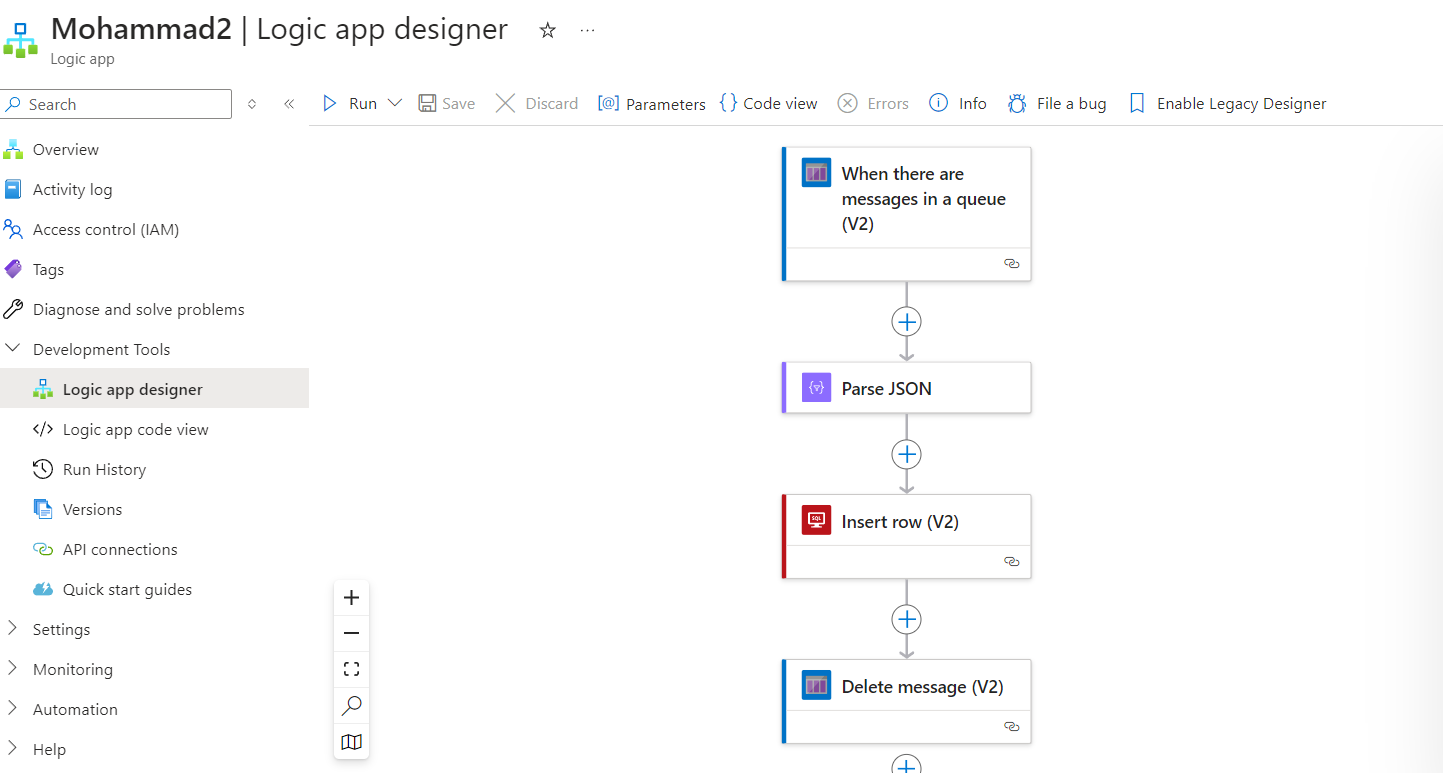
**Workflow:**

 **Trigger**: When a new message is added to the Azure Storage Queue.

 **Action**: Parse the message content.

 **Action**: Insert the parsed data into Azure SQL Database.

 **Action**: Send a notification (e.g., email or message) upon successful insertion.



**Step 7: Grafana Dashboard Integration**

**Grafana Dashboard**

 **Objective**: To visualize the data stored in Azure SQL Database using Grafana.

 **Installation**: Grafana was installed on the Ubuntu VM to create and manage dashboards.

 **Configuration**: Grafana was configured to connect to the Azure SQL Database, allowing it to query and visualize the data.

**Steps to Install and Configure Grafana**

#### ****1. Add the Grafana APT Repository****

First, make sure my system is updated and the Grafana APT repository is added. Run these commands: sudo apt-get update

sudo apt-get install -y software-properties-common

sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"

When I ran the command, got the error because the Grafana APT repository is missing its **GPG key.**

**Add the Grafana GPG Key:** I need to download and add the GPG key for the Grafana repository. Run the following command:

**sudo wget -q -O - https://packages.grafana.com/gpg.key | sudo tee /etc/apt/trusted.gpg.d/grafana.asc**

**1.Update the Package List**

Now that the GPG key is added, I should update your package list to refresh the repository information: sudo apt-get update

**2.Install Grafana**

Once the package list is updated, proceed with the installation of Grafana:

sudo apt-get install grafana

### 3.Reload the systemd daemon

Run the following command to reload the systemd configuration. This ensures that the system recognizes the newly installed Grafana service:

sudo /bin/systemctl daemon-reload

### 4.Enable the Grafana service

Enable the Grafana service to start automatically at boot:

sudo /bin/systemctl enable grafana-server

### 5.Start the Grafana service

Now, start the Grafana server: sudo /bin/systemctl start grafana-server

### 6. Verify the Grafana Service

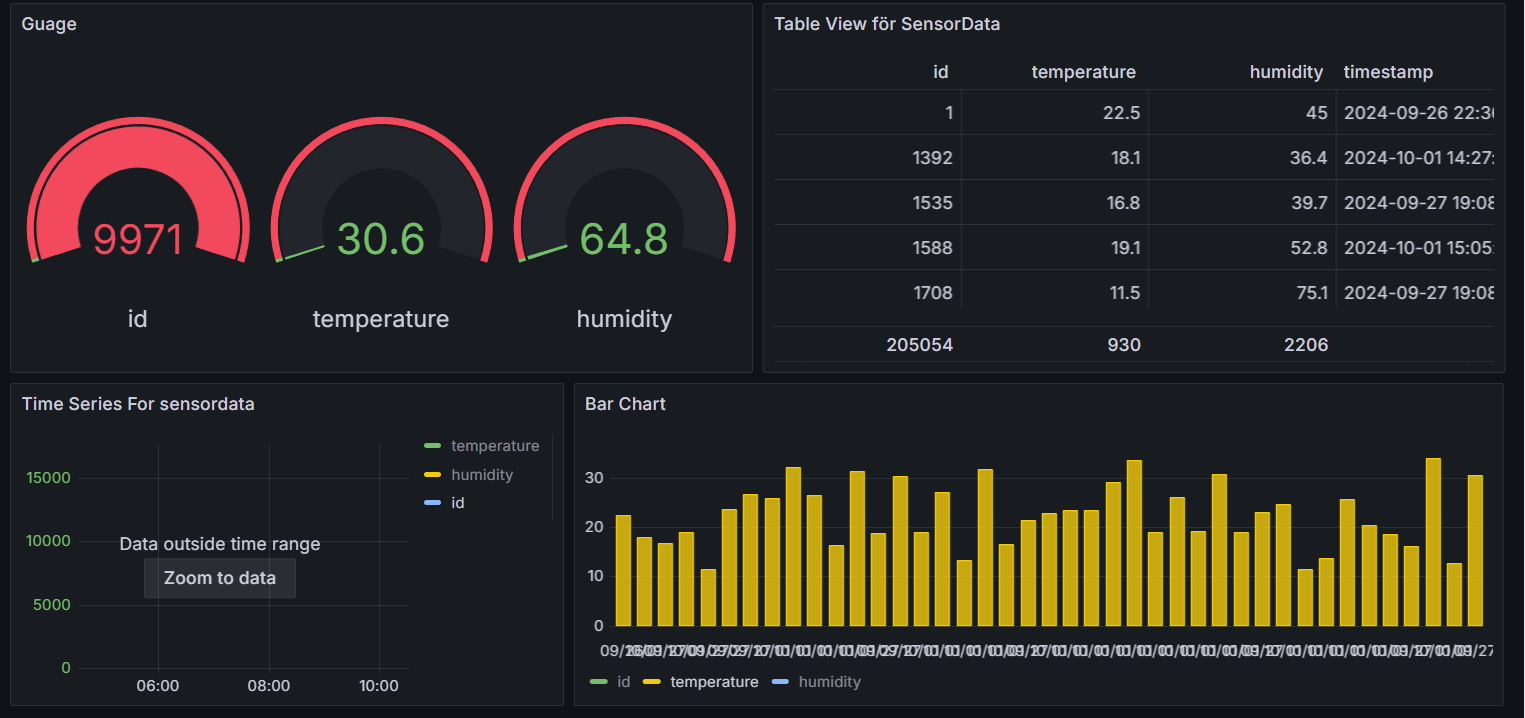
Check the status of the Grafana service to make sure it is running correctly:

sudo systemctl status grafana-server. Then I have seen output indicating that the service is **active (running)**.

### 7. Access Grafana

Once Grafana is running, open my web browser and go to: http:// 135.225.57.27:3000

**Grafana Dashboard:** Here is my grafana dashboard.



**Step 8: Power BI Dashboard Integration**

**Power BI Dashboard**

 **Objective**: To create an interactive dashboard using Power BI, part of Azure Fabric, for advanced data visualization and reporting.

 **Connection**: The Power BI dashboard was connected to the Azure SQL Database to retrieve and visualize the data.

**Steps to Create a Power BI Dashboard**

1. Open my Power BI Desktop.

2. **Connect to Azure SQL Database**:

Select **Azure SQL Database** as the data source.

 Enter the connection details:

 **Server Name**: sql-moh---------------

 **Database Name**: M--------------

3. **Import Data**:

 Once connected, selected the SensorData tableor

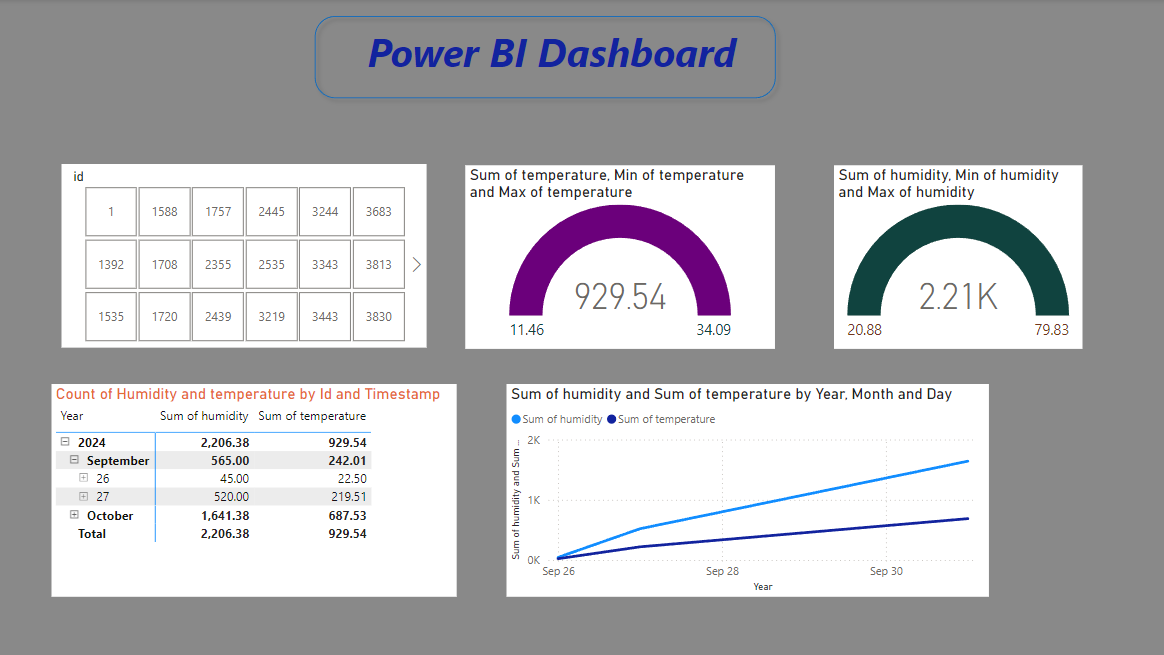
 Import the data into Power BI for analysis.

**4.Create Visualizations**:

 Use the Power BI interface to create Slicer, Gauge, Matrix and Line Chart visualizations based on the imported data.

 Drag and drop fields onto the report canvas to build my dashboard.

Here is my Power BI Dashboard:



**8. Create Matplotlib :** 1. Install required libraries:

First, make sure I have the necessary libraries installed. I'll need matplotlib, pyodbc, and pandas. I can install them using pip:

**pip install matplotlib pyodbc pandas.**

I got error message about virtual environment. Then I installed necessary tools by following command:

**1.First, ensure you have the necessary tools installed:**

**sudo apt update**

**sudo apt install python3-venv python3-pip**

**2.Create a virtual environment: python3 -m venv ~/myenv**

**3. Activate the virtual environment: source ~/myenv/bin/activate**

**4. Now I can install the required packages: pip install matplotlib pyodbc pandas**

**5. After installation, you can run your Python script within this virtual environment:**

**python data\_consumer1.py. Here is my python script**

import pyodbc

import pandas as pd

import matplotlib.pyplot as plt

from matplotlib.dates import DateFormatter  # Import DateFormatter

from datetime import datetime, timedelta

# Database connection function

def connect\_to\_database():

    server = '----------------'

    database = '---------'

    username = 'sql'

    password = '------------'

    driver = '{ODBC Driver 17 for SQL Server}'

    conn\_str = (

        'DRIVER=' + driver + ';'

        'SERVER=' + server + ';'

        'PORT=1433;'

        'DATABASE=' + database + ';'

        'UID=' + username + ';'

        'PWD=' + password + ';'

        'Encrypt=yes;'

        'TrustServerCertificate=no;'

        'Connection Timeout=60;'  # Increased timeout

    )

    try:

        print("Attempting to connect to the database...")

        conn = pyodbc.connect(conn\_str)

        print("Successfully connected to the database.")

        return conn

    except pyodbc.Error as e:

        print(f"Error connecting to database: {e}")

        print(f"Connection string used: {conn\_str}")

        return None

# Function to fetch data from the database

def fetch\_data(conn, hours=72):

    cursor = conn.cursor()

    try:

        time\_threshold = datetime.now() - timedelta(hours=hours)

        query = """

        SELECT id, temperature, humidity, timestamp

        FROM SensorData

        WHERE timestamp > ?

        ORDER BY timestamp

        """

        print(f"Executing query: {query}")

        print(f"With time threshold: {time\_threshold}")

        cursor.execute(query, (time\_threshold,))

        data = cursor.fetchall()

        if not data:

            print(f"No data found in the last {hours} hours. Fetching all available data.")

            query = """

            SELECT TOP 1000 id, temperature, humidity, timestamp

            FROM SensorData

            ORDER BY timestamp DESC

            """

            print(f"Executing query: {query}")

            cursor.execute(query)

            data = cursor.fetchall()

        if data:

            print("First row of data:")

            print(data[0])

            print("Number of columns:", len(data[0]))

            print("Column types:", [type(item) for item in data[0]])

            print(f"Total rows fetched: {len(data)}")

        else:

            print("No data retrieved from the database.")

        return data

    except pyodbc.Error as e:

        print(f"Error fetching data: {e}")

        return None

    finally:

        cursor.close()

# Function to print data summary

def print\_data\_summary(data):

    if not data:

        print("No data available.")

        return

    print("\nRaw data structure:")

    unpacked\_data = [list(row) for row in data]  # Convert to list of lists

    for i, row in enumerate(unpacked\_data[:5]):  # Print first 5 rows

        print(f"Row {i}: {row}")

        print(f"Row {i} type: {type(row)}")

        print(f"Row {i} length: {len(row)}")

        print()

    try:

        df = pd.DataFrame(unpacked\_data)  # Create DataFrame from unpacked data

        print("\nDataFrame created successfully.")

        print("\nDataFrame shape:", df.shape)

        print("\nColumn names:")

        print(df.columns)

        print("\nFirst few records:")

        print(df.head())

        print("\nData types:")

        print(df.dtypes)

        # Rename columns and convert data types

        if df.shape[1] == 4:

            df.columns = ['id', 'temperature', 'humidity', 'timestamp']

            df['timestamp'] = pd.to\_datetime(df['timestamp'], errors='coerce')

            df['temperature'] = pd.to\_numeric(df['temperature'], errors='coerce')

            df['humidity'] = pd.to\_numeric(df['humidity'], errors='coerce')

            df['id'] = pd.to\_numeric(df['id'], errors='coerce')

            print("\nAfter data type conversion:")

            print(df.dtypes)

            print("\nDescriptive statistics:")

            print(df.describe())

            print("\nDate range:")

            print(f"From: {df['timestamp'].min()}")

            print(f"To: {df['timestamp'].max()}")

        else:

            print(f"Unexpected number of columns: {df.shape[1]}")

    except Exception as e:

        print(f"Error in print\_data\_summary: {e}")

# Function to plot the data

# Function to plot the data

def plot\_data(data):

    if not data:

        print("No data available to plot.")

        return

    try:

        # Create DataFrame

        unpacked\_data = [list(row) for row in data]  # Convert to list of lists

        df = pd.DataFrame(unpacked\_data)

        if df.shape[1] != 4:

            print(f"Unexpected number of columns: {df.shape[1]}. Cannot create plot.")

            return

        df.columns = ['id', 'temperature', 'humidity', 'timestamp']

        df['timestamp'] = pd.to\_datetime(df['timestamp'])

        df['temperature'] = pd.to\_numeric(df['temperature'], errors='coerce')

        df['humidity'] = pd.to\_numeric(df['humidity'], errors='coerce')

        df['id'] = pd.to\_numeric(df['id'], errors='coerce')

        # Create a figure with three subplots

        fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(12, 15), sharex=True)

        # 1. Line chart for temperature and humidity by ID

        ax1.plot(df['id'], df['temperature'], label='Temperature', marker='o')

        ax1.plot(df['id'], df['humidity'], label='Humidity', marker='o')

        ax1.set\_ylabel('Value')

        ax1.set\_title('Temperature and Humidity by ID')

        ax1.legend()

        # 2. Bar chart for temperature and humidity

        bar\_width = 0.35

        index = range(len(df))

        ax2.bar(index, df['temperature'], width=bar\_width, label='Temperature', color='b', align='center')

        ax2.bar([i + bar\_width for i in index], df['humidity'], width=bar\_width, label='Humidity', color='g', align='center')

        ax2.set\_ylabel('Value')

        ax2.set\_title('Temperature and Humidity Bar Chart')

        ax2.set\_xticks([i + bar\_width / 2 for i in index])

        ax2.set\_xticklabels(df['id'])

        ax2.legend()

        # 3. Scatter plot for Temperature vs Humidity

        scatter = ax3.scatter(df['temperature'], df['humidity'], c=df['timestamp'], cmap='viridis')

        ax3.set\_xlabel('Temperature')

        ax3.set\_ylabel('Humidity')

        ax3.set\_title('Temperature vs Humidity')

        plt.colorbar(scatter, ax=ax3, label='Time')

        # Format x-axis for the first two subplots

        ax1.set\_xticks(df['id'])  # Set x-ticks to ID numbers

        ax1.set\_xticklabels(df['id'], rotation=45)  # Rotate x-tick labels for better visibility

        plt.tight\_layout()

        plt.savefig('sensor\_data\_visualization.png')

        print("Plot saved as sensor\_data\_visualization.png")

    except Exception as e:

        print(f"Error in plot\_data: {e}")

# Main function

def main():

    conn = connect\_to\_database()

    if not conn:

        print("Failed to connect to the database. Exiting.")

        return

    try:

        print("Fetching data from the database...")

        data = fetch\_data(conn, hours=72)  # Fetch last 72 hours of data

        if data:

            print("Data fetched successfully. Printing summary...")

            print\_data\_summary(data)

            print("Generating plot...")

            plot\_data(data)

            print("Script execution completed successfully.")

        else:

            print("No data available. Script execution completed.")

    except Exception as e:

        print(f"An unexpected error occurred: {e}")

    finally:

        print("Closing database connection...")

        conn.close()

        print("Database connection closed.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

I want to make the file accessible via a web browser, you set up a simple HTTP server on my VM. Here's how: python3 -m http.server 8000

Access the Visualization in my Browser

Open my Web Browser:

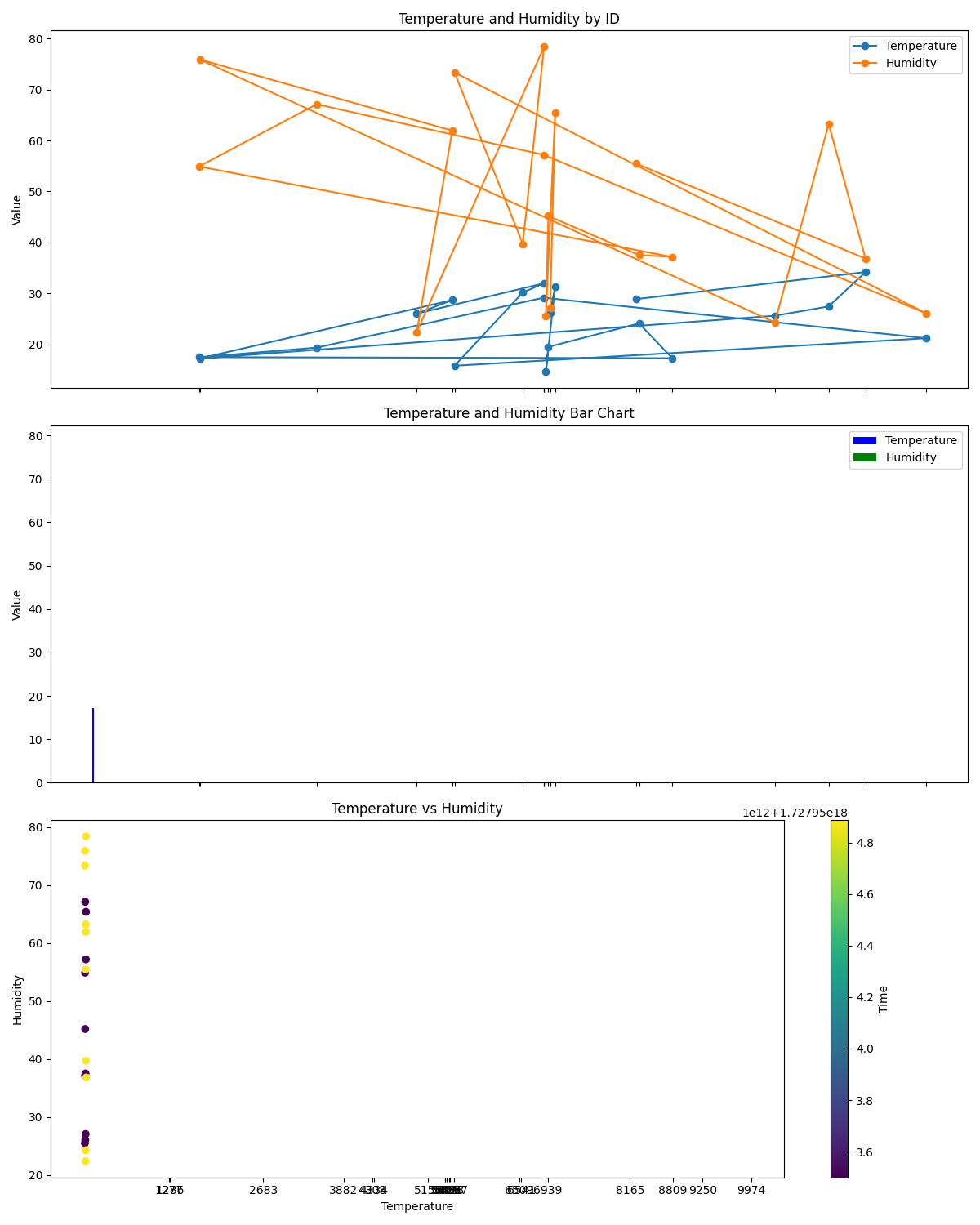
On my local machine, open a web browser.

Enter the URL:

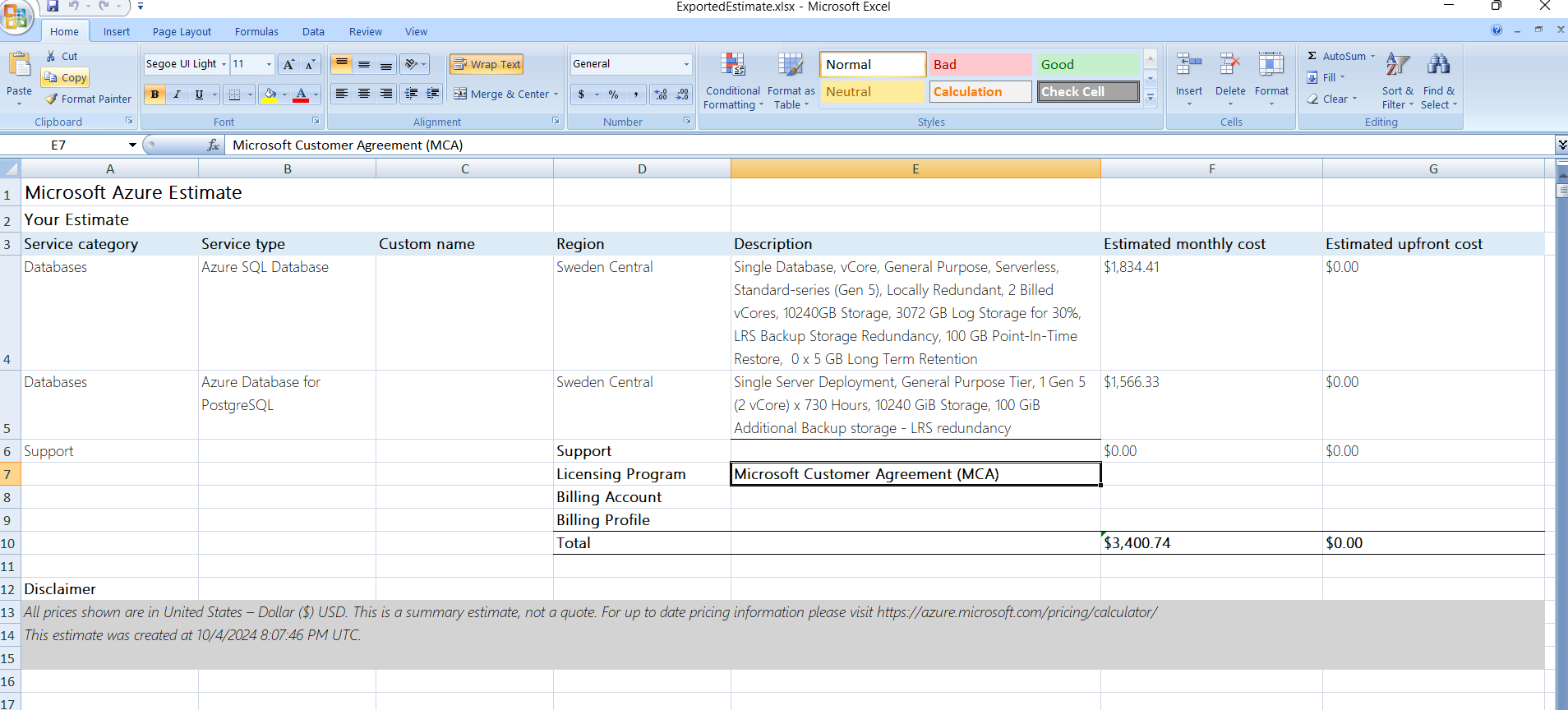
In the address bar, enter the URL to access the image.

http:// 135.225.57.27:8000/sensor\_data\_visualization.png

**View the Image: Here is my image.**



**Cost Analysis : Azure SQL Database (Serverless) vs. Azure PostgreSQL Database.**

****

**Conclusion: Overall, the integration of Azure services, including Azure SQL Database, Azure VMs, Grafana, and Power BI, Matplotlib provides a comprehensive data pipeline solution that can be tailored to meet various business needs while maintaining cost efficiency**