



Department of Information Engineering, Computer Science and Mathematics

Smart Traffic Monitoring System for Smart Cities

Project

By:

Salim Kamaludeen Lawal

Angelina Kovalinskaia

Vinay Sanga

Submitted to:

Prof. Davide Di Ruscio

As a part of the Software Engineering for IoT Course

GitHub repository: https://github.com/elsirleem/SE4IoT_project

Date Submitted: 29.01.2025

Contents

Contents	2
Introduction	3
Objective	3
Project Scope	3
Functional Requirements	4
Non-Functional Requirements	5
Technologies Used	6
Docker	6
Python + Paho	6
Mosquitto	7
Node-RED + WhatsApp	7
Telegraf	9
InfluxDB	10
Grafana	11
System Architecture	13
Conclusion	14
Suggestion For Future Work	15
References	16

Introduction

This project is a simulated IoT system designed to monitor urban traffic and environmental conditions. It integrates components such as sensors, middleware, databases, and dashboards to provide real-time insights, helping users observe traffic patterns and assess environmental impact.

Objective

Urban areas face increasing challenges with traffic congestion, rising carbon emissions, and deteriorating air quality. Research has shown that traffic factors such as speed limits significantly affect fuel consumption and carbon emissions, with higher speeds contributing to increased fuel use and environmental pollutants (Gao, 2024). These issues necessitate effective solutions for monitoring traffic conditions and environmental impact in real time to support data-driven decisions.

This project aims to develop a real-time traffic and environmental monitoring system to address these challenges by collecting and analyzing data on traffic density, speed, fuel consumption, CO₂, PM2.5, and PM10 levels. The system will provide city planners and traffic managers with actionable insights into the relationship between traffic patterns and environmental sustainability. It will enable the evaluation of traffic policies, such as speed limit regulations, by quantifying their impact on carbon emissions. With real-time alerts and data visualization, this system will help optimize traffic management strategies, reduce environmental harm, and support the creation of more sustainable urban environments.

Project Scope

- Simulates environmental metrics, including traffic density, average speed, CO₂, PM2.5, PM10, and noise levels.
- Processes data using Telegraf for data collection and transformation.
- Stores time-series data in InfluxDB for long-term analysis.
- Visualizes data in Grafana with customizable dashboards.
- Uses Mosquitto as the MQTT broker to manage message communication between IoT devices and the system
- Node-RED for handling alerting mechanisms, allowing for the creation of custom alerts based on real-time data streams.
- Fully containerized deployment using Docker Compose.

Functional Requirements

ID	REQUIREMENT	DESCRIPTION
1	Real-Time Traffic Monitoring	The system must monitor traffic density and vehicle speed in real-time, providing continuous updates to users.
2	Environmental Data Monitoring	The system must collect and display environmental data, including CO, CO ₂ levels, PM2.5, PM10, and noise levels, from sensors in the monitored area.
3	Data Aggregation and Display	The system must aggregate the collected data and display it on interactive dashboards with graphical visualizations, showing trends and real-time conditions in a user-friendly format.
4	Alerts	The system must generate real-time alerts based on predefined thresholds for traffic and environmental metrics and send notifications to relevant users.

Non-Functional Requirements

ID	REQUIREMENT	DESCRIPTION
1	Portability	The entire system should enable deployment on diverse platforms without reconfiguration.
2	Scalability	The design allows for easy integration of additional sensors, locations, and metrics.
3	Resilience	The system should be able to handle communication failures and ensure consistent data storage and processing.
4	User-friendly design	Intuitive dashboards with clear insights, summaries, and threshold indicators.
5	Security	The system should undertake security measures including authentication and database access controls to ensure data integrity.

Technologies Used

The technologies we have used in the project include:

Docker

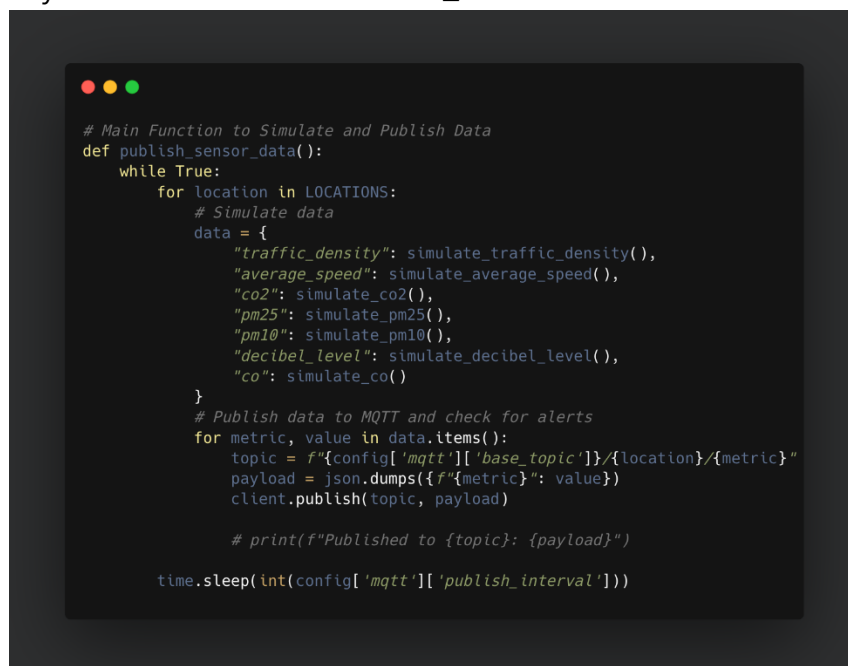
Docker was used to containerize the entire solution, ensuring a consistent runtime environment for all components. This allows for easy deployment, scalability, and management of the system across various infrastructure setups.

Python + Paho

Python, along with the Paho MQTT library, was used to simulate sensor data, including metrics like traffic density, speed, and environmental pollutants. Each metric is randomly generated within real-world constraints. The simulated data is published to the MQTT broker every 5 seconds using JSON data format for further processing in the system.

Topics to be published include:

- /smart city/traffic/<location>/traffic_density,
- /smart city/traffic/<location>/average_speed,
- /smart city/traffic/<location>/CO₂,
- /smart city/traffic/<location>/co,
- /smart city/traffic/<location>/pm25,
- /smart city/traffic/<location>/pm10,
- /smart city/traffic/<location>/decibel_level.

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function named 'publish_sensor_data' that simulates and publishes sensor data. It includes comments in English and uses various Python constructs like loops, dictionaries, and JSON formatting. The function simulates data for multiple locations and publishes it to an MQTT broker at regular intervals.

```
# Main Function to Simulate and Publish Data
def publish_sensor_data():
    while True:
        for location in LOCATIONS:
            # Simulate data
            data = {
                "traffic_density": simulate_traffic_density(),
                "average_speed": simulate_average_speed(),
                "co2": simulate_co2(),
                "pm25": simulate_pm25(),
                "pm10": simulate_pm10(),
                "decibel_level": simulate_decibel_level(),
                "co": simulate_co()
            }
            # Publish data to MQTT and check for alerts
            for metric, value in data.items():
                topic = f"{config['mqtt']['base_topic']}/{location}/{metric}"
                payload = json.dumps({"metric": metric, "value": value})
                client.publish(topic, payload)

            # print(f"Published to {topic}: {payload}")

            time.sleep(int(config['mqtt']['publish_interval']))
```

Figure 1: Main Function to Simulate and Publish Data

Mosquitto

Mosquitto serves as the MQTT broker, enabling efficient communication between simulated sensors and other system components. It handles real-time data transmission between publishers (sensors) and subscribers.

Node-RED + WhatsApp

Node-RED subscribes to the MQTT broker, performs threshold checks on incoming data, and generates alerts when thresholds are exceeded. Notifications are sent to WhatsApp CallMeBot API to ensure timely alerts and responses (Node-RED: Send Messages to WhatsApp, n.d.).

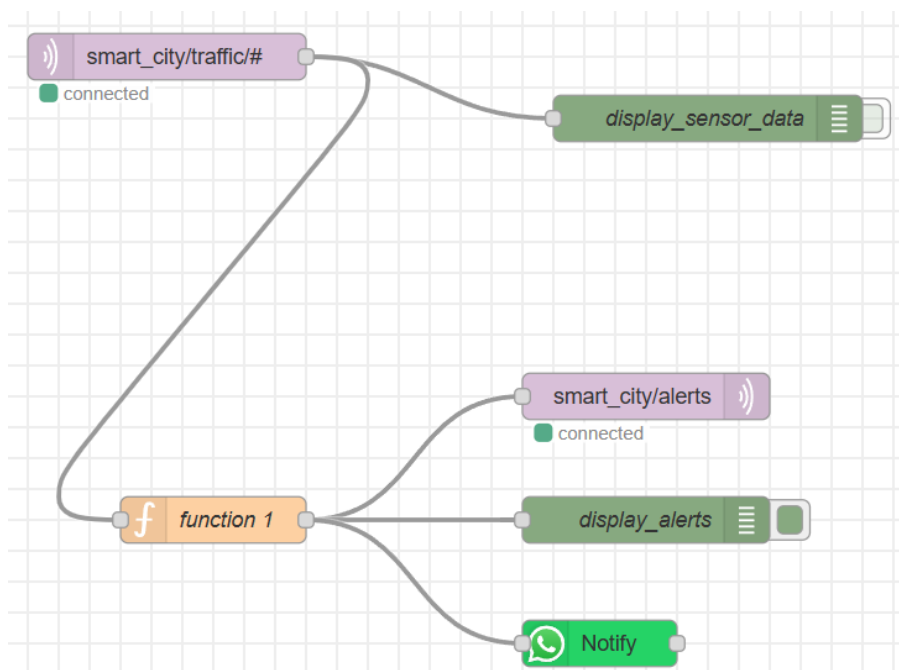


Figure 2: Node-RED flow for alert generation

```

const thresholds = {
  traffic_density: 99,
  average_speed: 59,
  co2: 499,
  pm25: 49,
  pm10: 69,
  decibel_level: 89,
  co: 4.9
};

const metric = msg.topic.split("/").pop(); // Extract the metric from the
const value = msg.payload[metric];

if (thresholds[metric] !== undefined && value > thresholds[metric]) {
  msg.alert = `Critical ${metric} level detected: ${value}`;
  return msg; // Forward message for alerts
}

return null; // Ignore if no threshold is exceeded

```

Figure 1. Node-RED threshold check for alert generation

Here are the steps to be followed for using the Whatsapp notifications in Node-RED.

1. **Obtain the API Key:** Add the CallMeBot phone number to WhatsApp contacts. Send the message "I allow callmebot to send me messages" to the contact. Wait for the response with the unique API key.
2. **Install the WhatsApp Node:** In Node-RED, go to the palette manager (Menu > Manage Palette > Install) and install the node-red-contrib-whatsapp-cmb package.
3. **Configure the Flow:** Create a flow with an "Inject" node, a "Send Message WhatsApp" node, and a "Debug" node. Configure the "Send Message WhatsApp" node with the phone number in international format and the API key. Set the message content in the "Inject" node's msg.text property.
4. **Deploy and Test:** Deploy the flow. Trigger the "Inject" node to send the message. Confirm successful delivery to the WhatsApp account.

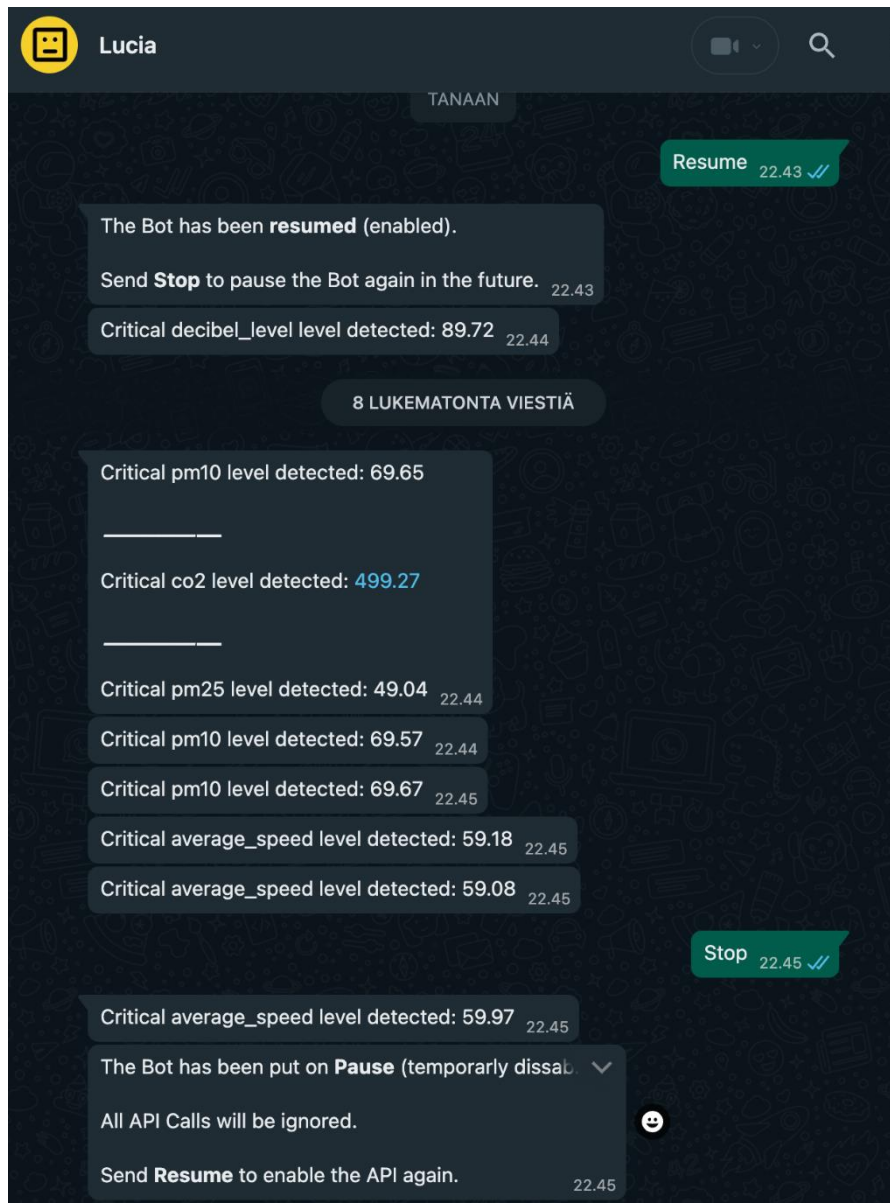


Figure 3: WhatsApp notification

Telegraf

Telegraf acts as a subscriber to the MQTT broker, collecting the sensor data and broadcasting it to the InfluxDB database for storage and further analysis. Telegraf recognizes dynamic tags and measurements based on the MQTT topic structure, with the topics being parsed to extract measurement names and tags directly from the topic name (MQTT Consumer Input Plugin for Telegraf, n.d.).

The used configuration is shown in the code snippet below:

```
[[inputs.mqtt_consumer.topic_parsing]]
topic = "smart_city/traffic/+/+"
measurement = "_/measurement/_/_/"
tags = "_/_/location/_/"
```

InfluxDB

We have used InfluxDB to store the time-series data received from Telegraf, providing a reliable and efficient database for long-term storage and querying of traffic and environmental metrics. The query shown below has been used to filter parameters according to specific or different locations for easy access through the influxDB and the Grafana dashboard interface.



```
from(bucket: "smart_city")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "traffic")
  |> filter(fn: (r) => r["location"] == "${location}")
  |> filter(fn: (r) => r["_field"] == "average_speed")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty:
false)
  |> yield(name: "mean")
```

Figure 4: Flux Query to filter data

Grafana

Grafana is used to create real-time dashboards that visualize the traffic and environmental data stored in InfluxDB. It also generates automatic alerts, which are configured to send notifications to Microsoft Teams, ensuring the relevant teams are informed of any critical situations (Configure Grafana-managed alert rules, n.d.).

[FIRING:1] Traffic Density (%) Density_alert (location1 smart_city/traffic/location1/traffic_density)

Firing

Value: C=1, density=98 Labels:

- alertname = Traffic Density (%)
- grafana_folder = Density_alert
- location = location1
- topic = smart_city/traffic/location1/traffic_density

Annotations:

- summary = Achtung! Traffic density exceeded the 80% threshold

Source: <http://localhost:3000/alerting/grafana/eea6l59cvre9sf/view?orgId=1>

Figure 5: Teams Notification

Grafana dashboard features:

- **Line charts for traffic trends:** These charts allow users of the system to identify different trends such as peak hours, daily traffic patterns, or unusual spikes that may reduce congestion and improve proper traffic management.
- **Gauges for air quality levels:** The the gauges display the CO₂, PM2.5, and PM10 levels in real-time, highlighting the different color-coded zones which make it easy to interpret air quality at a glance.
- **Threshold violations:** These alerts are highlighted in red for quick identification which helps prioritize urgent actions or interventions, such as adjusting traffic flow or notifying environmental agencies on necessary actions required.

For different levels of administration, there are 2 dashboards:

1. Regional level based on location

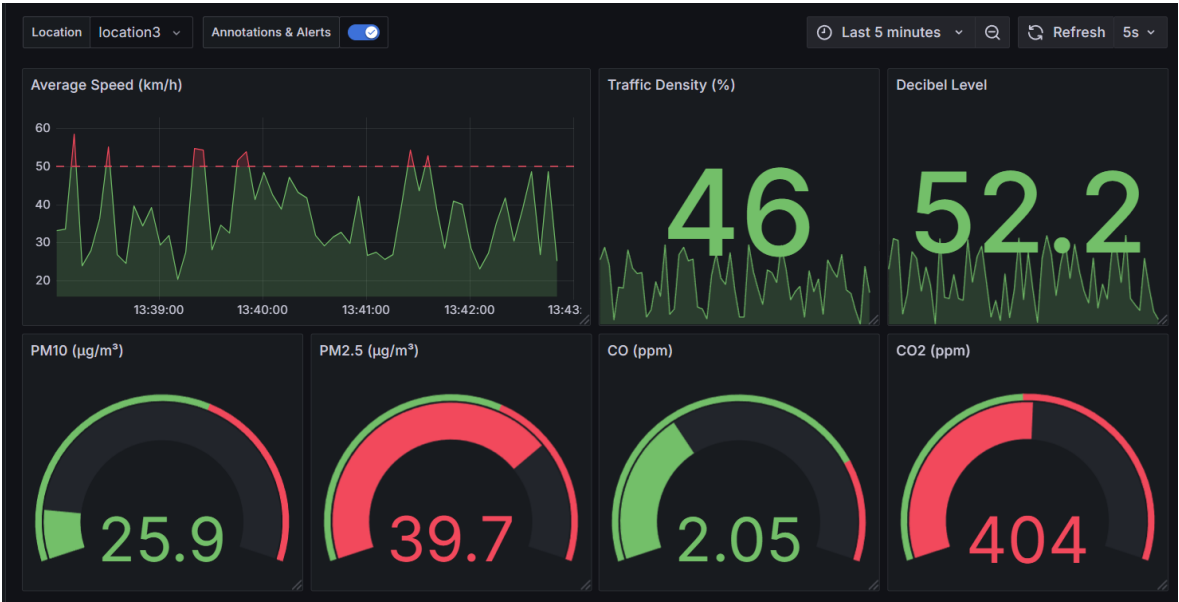


Figure 6: Grafana dashboard for regional monitoring in location 3

2. Global monitoring of specific parameters for all locations

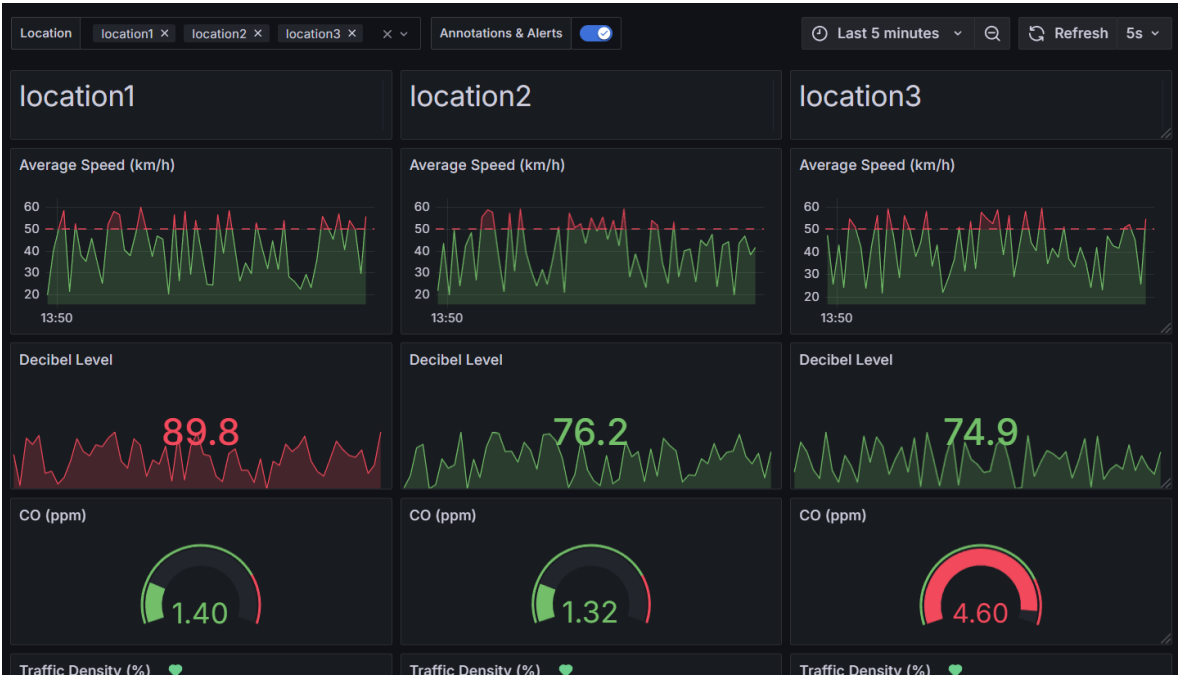


Figure 7: Grafana dashboard for global monitoring in locations 1, 2, 3

System Architecture

The system uses Docker for containerizing components. A sensor simulation developed in Python publishes data to the Mosquitto message broker via MQTT. The Telegraf agent collector subscribes to the broker to gather and broadcast data, which is then stored in an InfluxDB timeseries database. The data is queried and visualized through Grafana dashboards for real-time monitoring. Additionally, Grafana is configured to send notifications to Microsoft Teams when the traffic density measure exceeds a defined threshold. Meanwhile, alerts for every measure are managed by an alert system in Node-RED that subscribes to the Mosquitto broker and sends notifications to WhatsApp.

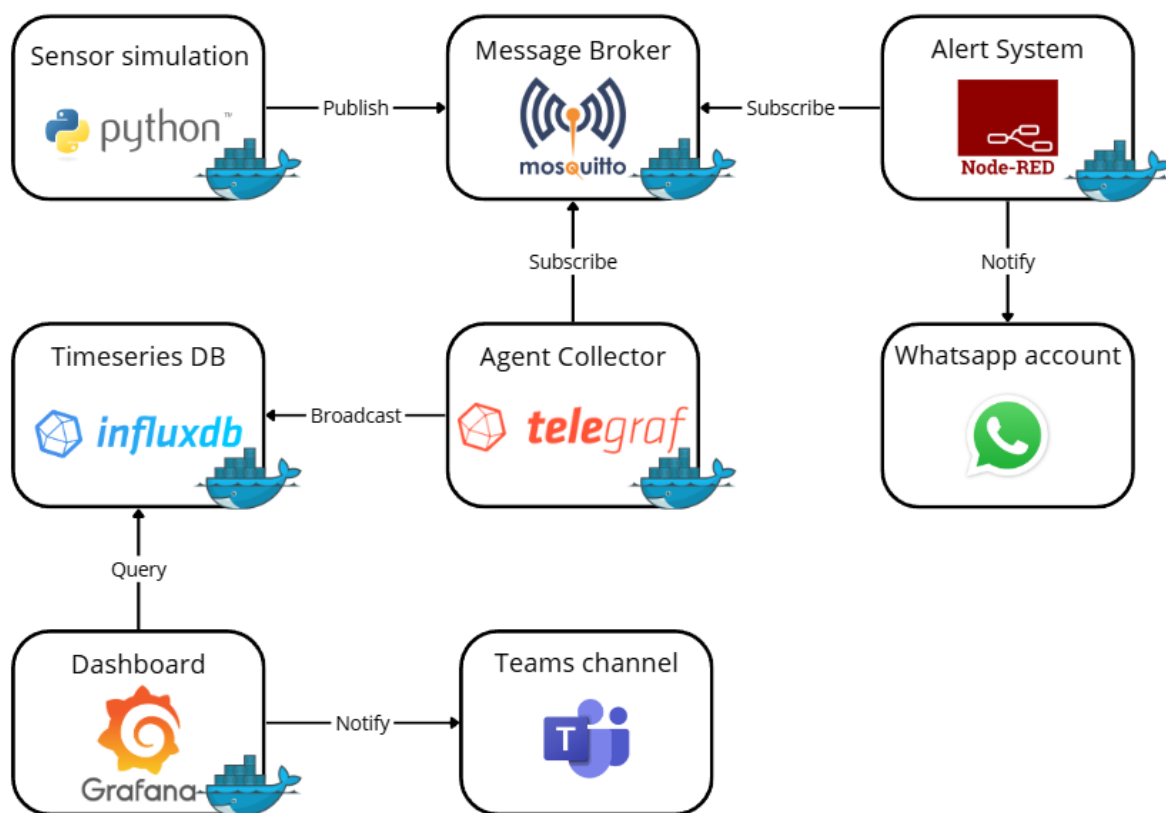


Figure 4. System architecture with technologies used

Conclusion

This project successfully developed a simulated IoT-based smart city traffic and environmental conditions monitoring system, integrating a range of technologies to provide real-time data insights.

Docker facilitated easy deployment and management of the entire solution. Python with Paho simulated sensor data published to an MQTT broker powered by Mosquitto. This real-time data was then processed by Node-RED for alerting via WhatsApp, and simultaneously collected by Telegraf for storage in InfluxDB.

The stored time-series data in InfluxDB was then visualized using Grafana, providing real-time dashboards for both regional and global monitoring. Grafana also enabled automated alerts sent to Microsoft Teams, ensuring timely notification of critical situations.

This integrated approach allows for effective monitoring of traffic and environmental conditions, enabling informed decision-making for smart city management. Through ongoing development, it holds great potential to improve the quality of life in cities and contribute to environmental sustainability.

Suggestion For Future Work

For future work, we advise to consider leveraging the information displayed on the Grafana dashboard to advance and dynamic decision-making for traffic management. Some of its application include:

1. **Rerouting Vehicles:** Utilizing metrics like average speed and CO₂ emissions to identify potential traffic congestion in specific locations. If the average speed falls below a defined threshold and CO₂ levels exceed limits, this can trigger rerouting strategies to alleviate congestion.
2. **Safety Warnings:** Monitoring the decibel levels and air quality attributes such as PM2.5 and PM10. If these exceed safe thresholds, warnings can be issue to road users to avoid routes that may pose health risks.
3. **Automated Alerts:** The Automated alert systems based on threshold violations developed in this project can be extended to notify relevant authorities of the current traffic situation and provide real-time route suggestions to drivers.

References

Configure Grafana-managed alert rules. (n.d.). Retrieved 01 10, 2025, from <https://grafana.com/docs/grafana/latest/alerting/alerting-rules/create-grafana-managed-rule/>

Gao, C. (2024). Correlation between carbon emissions, fuel consumption of vehicles and speed limit on expressway. *Journal of Traffic and Transportation Engineering (English Edition)*, 11(4), 631-642.

MQTT Consumer Input Plugin for Telegraf. (n.d.). Retrieved 01 10, 2025, from https://github.com/influxdata/telegraf/blob/master/plugins/inputs/mqtt_consumer/README.md

Node-RED: Send Messages to WhatsApp. (n.d.). Retrieved 01 10, 2025, from <https://randomnerdtutorials.com/node-red-send-messages-whatsapp/>