University of Applied Sciences

HOCHSCHULE EMDEN·LEER

Data Acquisition and Dashboarding-Report

Master Project (WS 2024-25)
Master Industrial Informatics Degree

Report By:

Student 1: Sai Sanjeev Saripalli

Matriculation-No.: 7023241

Student 2: Dheeraj Pal

Matriculation-No.: 7025303

Under supervision:

Jeffrey Wermann

Hochschule Emden/Leer · Industrial Informatics
Constantiaplatz 4 · 26723 Emden · http://www.hs-emden-leer.de

University of Applied Sciences

HOCHSCHULE EMDEN·LEER

Contents

1.Introduction

This project focuses on designing and implementing a real-time data acquisition and visualization system for a Digital Factory. The system integrates OPC UA, Telegraf, InfluxDB, and Grafana to enable efficient data collection, storage, and visualization. The OPC UA server, developed in Python, simulates industrial sensors, generating temperature, pressure, and humidity data. This data is then collected by Telegraf, stored in InfluxDB, and visualized using Grafana dashboards.

The system runs in a Dockerized environment, ensuring modularity and ease of deployment. By leveraging time-series databases and real-time monitoring, this project provides a scalable and efficient solution for Industry 4.0 applications. The report documents the system architecture, setup process, data flow, query execution, and dashboard creation, ensuring a comprehensive understanding of the implementation.

This project demonstrates the feasibility of integrating Industrial IoT (IIoT) technologies for enhanced monitoring, predictive maintenance, and digital transformation in manufacturing environments

1.1 System Architecture

The system is designed to collect, store, and visualize real-time industrial data using a modular, containerized approach. It consists of an OPC UA server, Telegraf for data collection, InfluxDB for time-series data storage, and Grafana for visualization.

1. Digital Factory:

The Digital Factory represents the real-world industrial environment where sensor data is generated. It simulates an industrial setting where key parameters such as temperature, pressure, and humidity are continuously monitored. These values are exposed through an OPC UA server implemented in Python, acting as the source of sensor data that reflects the operational conditions of the factory environment.

2. OPC UA Server (Python Script):

The OPC UA Server is a standalone Python application, not running inside Docker, that simulates sensor data. It acts as the source of truth for industrial sensor data, generating and making available simulated temperature, pressure, and humidity readings via the OPC UA protocol. This server plays a key role in exposing real-time data to external systems for processing and analysis.

3. Telegraf Container (Data Collection Agent):

Telegraf, running as a Docker container, serves as the data collection agent in this system. It collects real-time sensor data from the OPC UA server, processes it, and forwards the formatted data to InfluxDB for storage. This container ensures continuous data collection from the OPC UA server and is critical for integrating sensor data into the broader system for monitoring and analysis.

4. InfluxDB Container (Time-Series Database):

InfluxDB is a time-series database that runs inside a Docker container with persistent storage enabled. It stores the incoming sensor data collected by Telegraf, providing both historical and real-time data access for monitoring and analysis. InfluxDB efficiently handles the time-series data from sensors, allowing for easy querying and retrieval of sensor data for further processing or visualization.

5. Grafana Container (Visualization & Monitoring):

Grafana, also running in a Docker container, connects to InfluxDB and queries the stored sensor data. It visualizes the data on interactive dashboards, displaying real-time sensor readings.

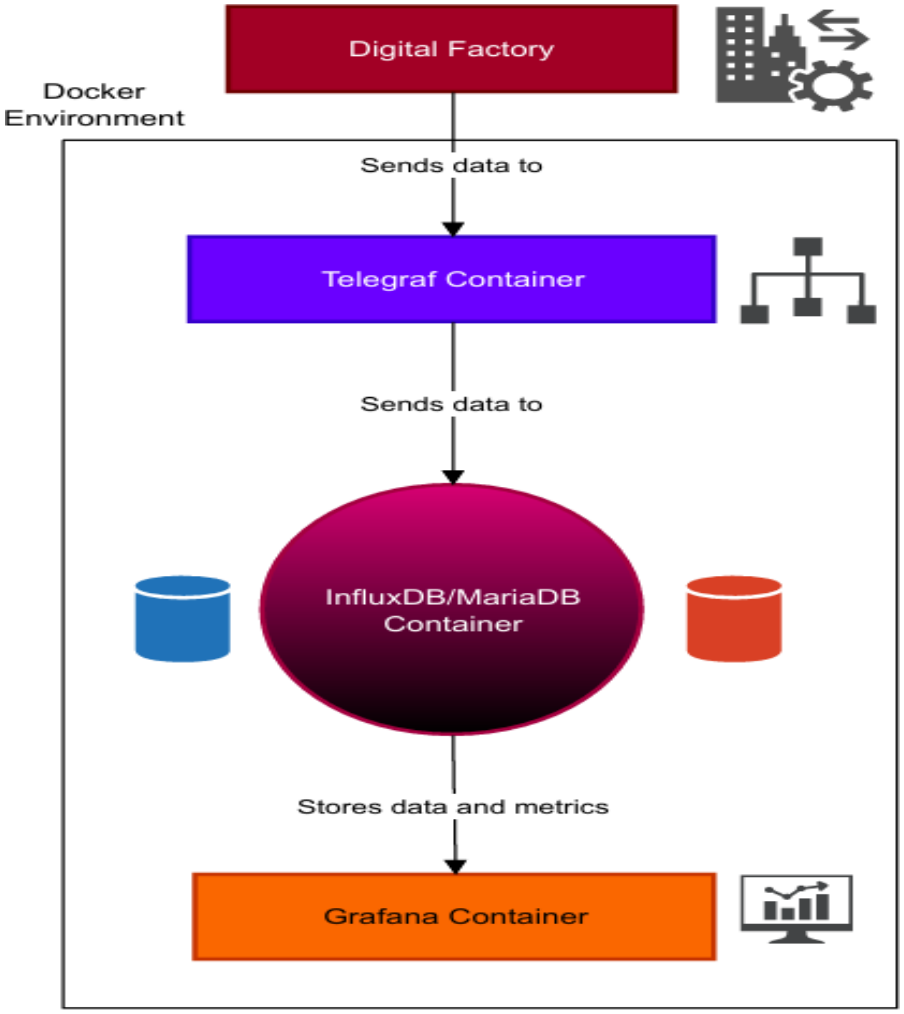
Grafana allows users to analyze time-series data through various graph types, set up alerts, and generate reports, providing a user-friendly interface for monitoring industrial conditions and trends.

6. Data Flow in the System:

The data flow starts with the OPC UA Server generating simulated sensor data. Telegraf collects this data from the OPC UA server and forwards it to InfluxDB, where the data is stored as time-series information. Grafana then queries InfluxDB to visualize and analyze the data in real-time, displaying it on interactive dashboards and providing insights into the sensor readings. This flow ensures that data is continuously collected, stored, and visualized for effective monitoring of industrial systems.

System Architecture Diagram

This architecture ensures scalability, flexibility, and real-time industrial data monitoring.



2.Implementation

The implementation of the system involves setting up and integrating various components, including the OPC UA server, Telegraf, InfluxDB, and Grafana. This chapter provides a step-by-step explanation of how each component was implemented and configured to enable real-time data acquisition, storage, and visualization.

2.1Setting Up the OPC UA Server

The OPC UA (Open Platform Communications Unified Architecture) server is responsible for simulating and providing sensor data in a structured format. This server acts as a data source that Telegraf will later collect and push into InfluxDB for storage.

The OPC UA server was implemented using Python and the opcua library to simulate three sensor values: temperature, pressure, and humidity. To set up the server, first, ensure Python is installed and install the required library using pip install opcua. Next, create a Python script that initializes an OPC UA server, defines three sensor nodes, and dynamically updates their values. This server acts as a simulated industrial data source, providing real-time sensor readings for further processing and visualization.Below is the python code.

```
python
from opcua import Server
from random import uniform
import time

# Initialize OPC UA Server
server = Server()

server.set_endpoint("opc.tcp://0.0.0.0:4840")  # Server URL

# Define a namespace
uri = "http://digitalfactory.opcua"
```

```
idx = server.register_namespace(uri)

# Create an object to hold sensor values
sensor_obj = server.nodes.objects.add_object(idx, "SensorData")

# Create variables (Temperature, Pressure, Humidity)
temp = sensor_obj.add_variable(idx, "Temperature", 0.0)
pressure = sensor_obj.add_variable(idx, "Pressure", 0.0)
humidity = sensor_obj.add_variable(idx, "Humidity", 0.0)

# Set writable to allow dynamic updates
temp.set_writable()
pressure.set_writable()
humidity.set_writable()

# Start the server
server.start()
print("OPC UA Server is running at opc.tcp://0.0.0.0:4840")

try:
    while True:
        # Simulate random sensor values
        temp.set_value(round(uniform(20.0, 30.0), 2))
        pressure.set_value(round(uniform(1.0, 2.0), 2))
```

```
        humidity.set_value(round(uniform(50.0, 70.0), 2))

        time.sleep(2)  # Update every 2 seconds

    finally:
        server.stop()
        print("Server Stopped")
```

Run the OPC UA Server

Save the script as opcua_server.py and run it using:

```
python opcua_server.py
```

This starts the OPC UA server on opc.tcp://0.0.0.0:4840

Verifying OPC UA Server with OPC UA Expert

Once the OPC UA server is running, it is essential to verify its operation and ensure that the sensor data is being generated correctly. This can be done using OPC UA Expert, a third-party tool for testing and interacting with OPC UA servers.

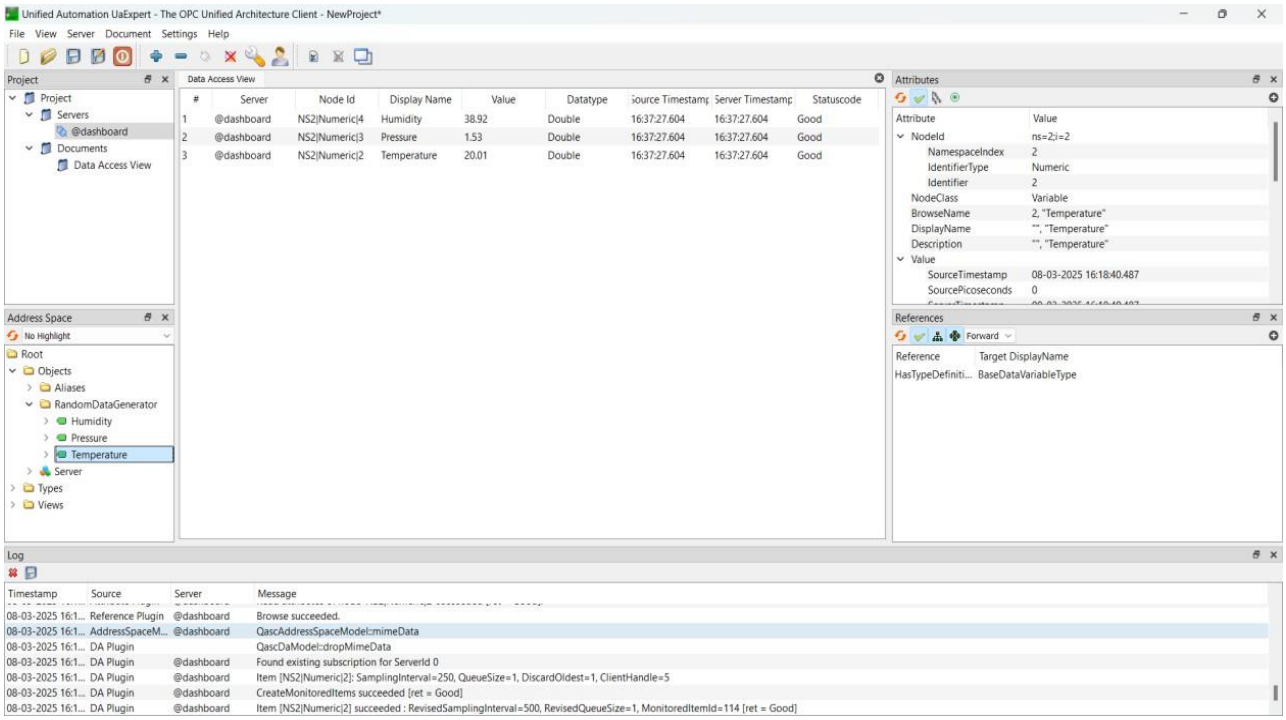
2.2.Installing and Setting Up OPC UA Expert

To use OPC UA Expert, first, download the tool from its official website, follow the installation instructions, and launch the application. To connect to the OPC UA server, open OPC UA Expert, click on "Add Server", and enter the Discovery URL as opc.tcp://localhost:4840. Then, click "Connect", select Anonymous (if security is disabled on the server), and click "OK" to establish the connection

Browsing and Monitoring

After successfully connecting to the OPC UA server in OPC UA Expert, expand the server's address space to explore available data nodes. Navigate to Objects → SensorData, where you will find the sensor variables: Temperature, Pressure, and Humidity, representing real-time industrial sensor readings.

Right-click on each variable (Temperature, Pressure, and Humidity) in OPC UA Expert and select "Monitor" to observe real-time updates. The values should dynamically change, reflecting the simulated sensor data generated by the OPC UA server. If the values are updating continuously, it confirms that the OPC UA server is functioning correctly and transmitting data as expected



2.3.Setting Up Docker for Containerized Deployment

Docker is used to containerize the different components of the system, ensuring scalability and ease of deployment. In this step, we will configure Docker to run InfluxDB, Telegraf, and Grafana inside containers while keeping the OPC UA server running in a Python script.

Installing Docker

Before setting up the containers, Docker needs to be installed on the system.

Download and Install Docker

To install Docker Desktop, first download it from Docker's official website and follow the installation steps. After completing the installation, restart your system. Then, open a terminal (or PowerShell for Windows) and verify the installation by running the command docker --version. If Docker is installed correctly, it will display the installed version.

Pulling Docker Images and Preparing the Environment

Before running the containers for InfluxDB, Telegraf, and Grafana, we need to ensure that the required Docker images are available on the system. Pulling images beforehand helps in avoiding delays during container deployment.

Pulling Required Docker Images

The following images need to be pulled from Docker Hub:

InfluxDB (Time-Series Database)

```
docker pull influxdb:latest
```

This command downloads the latest version of InfluxDB, which will store the sensor data collected from the OPC UA server.

Telegraf (Data Collection Agent)

```
docker pull telegraf:latest
```

This command downloads Telegraf, which is used to collect sensor data from the OPC UA server and send it to InfluxDB.

Grafana (Visualization Tool)

docker pull grafana/grafana:latest

This command downloads Grafana, which will be used to visualize the real-time data stored in InfluxDB

Verifying the Downloaded Images

Once all images are pulled, verify them using the following command:

docker images

The output should list:
influxdb

telegraf
grafana/grafana

Each image should have the latest tag along with its respective size.

Preparing the Docker Environment

Before launching the containers, it's essential to ensure that all necessary configurations and volumes are set up. First, create a Docker network by running docker network create monitoring to enable communication between containers within the same network. Next, confirm that Docker is running by checking its status with docker info. If Docker isn't running, start it manually based on your operating system, such as by launching Docker Desktop on Windows.

Automating Deployment with Docker Compose

To efficiently manage multiple services required for the system—OPC UA Server, Telegraf, InfluxDB, and Grafana—we use Docker Compose. Docker Compose allows us to define and run multi-container applications using a single docker-compose.yml file. This eliminates the need to manually start each container separately and ensures that all services communicate effectively.

Writing the docker-compose.yml File

The docker-compose.yml file is the heart of our deployment. It defines all containers, their configurations, networks, and dependencies. Below is a step-by-step breakdown of how to create and structure this file.

Creating the docker-compose.yml File

1. Open a terminal or command prompt in the project directory.
2. Create a new file named docker-compose.yml.
3. Open the file in a text editor such as VS Code or Notepad++.

Defining Services

The file consists of several services, each representing a container:

- InfluxDB (for time-series data storage)
- Telegraf (to collect and send data to InfluxDB)
- Grafana (for visualizing data)
- OPC UA Server (Python script that simulates industrial data)

Below is a sample docker-compose.yml file for this setup:

```
version: '3.8'

services:
  telegraf:
    image: telegraf
    container_name: telegraf
    restart: always
    volumes:
```

```
    - ./telegraf.conf:/etc/telegraf/telegraf.conf
  depends_on:
    - influxdb
  networks:
    - monitoring
  -
influxdb:
  image: influxdb
  container_name: influxdb
  restart: always
  ports:
    - "8086:8086"
  environment:
    - INFLUXDB_DB=dashboarding
    - INFLUXDB_ADMIN_USER=admin
    - INFLUXDB_ADMIN_PASSWORD=adminpassword
  networks:
    - monitoring
grafana:
  image: grafana/grafana
  container_name:
  grafana restart: always
  ports:
    - "3000:3000"
  depends_on:
    - influxdb
  networks:
```

```
    - monitoring
networks:

  monitoring:
```

Understanding the docker-compose.yml File

This file is structured into several key sections that define the configuration for a Docker Compose setup. The version declaration (version: '3.8') specifies the Docker Compose version in use. The services section (services:) outlines individual services like InfluxDB, Telegraf, Grafana, and OPC UA Server. The InfluxDB configuration section defines the use of the latest InfluxDB image, exposes port 8086, creates an initial database with admin credentials, and uses a volume for persistent storage. Telegraf is configured to collect data, uses a configuration file (telegraf.conf) for OPC UA data collection, depends on InfluxDB, and communicates over the monitoring network. Grafana is set up for data visualization, exposes port 3000 for the UI, uses admin credentials, stores configurations persistently, and depends on InfluxDB. The volumes section defines persistent storage for InfluxDB and Grafana, while the networks section defines a custom network (monitoring) to ensure communication between Telegraf, InfluxDB, and Grafana.

Running Docker Compose

Once the docker-compose.yml file is created, we can use the following commands to deploy and manage the system.

Starting the Services

Navigate to the directory containing the docker-compose.yml file and run:

```
docker-compose up -d
```

This command Pulls the required images.

Starts the services in the background (-d runs it in detached mode) and Ensures dependencies are handled correctly.

Checking Running Containers

To verify if all containers are running, use:

```
docker ps
```

This should display running containers for:

influxdb
telegraf
Grafana

2.4.Configuring and Running Telegraf for Data Collection

Telegraf is a key component in this setup as it collects real-time data from the OPC UA Server and sends it to InfluxDB for storage and further processing. In this section, we will go through the step-by-step process of configuring Telegraf, connecting it to the OPC UA server, and verifying data ingestion.

Understanding Telegraf

Telegraf is an agent for collecting, processing, and forwarding metrics from various input sources to different output destinations. In our system:
Input Plugin: opcua (collects data from the OPC UA server)
Output Plugin: influxdb_v2 (sends collected data to InfluxDB)

Configuring Telegraf

The main configuration file for Telegraf, telegraf.conf, defines several key settings, including the OPC UA server connection details, the specific nodes (data points) to collect, and the InfluxDB endpoint where Telegraf sends the collected data. To create or update telegraf.conf, locate the file in your working directory. If it does not already exist, create a new file named telegraf.conf. Open it in a text editor and add the necessary configuration to define the OPC UA connection, data points, and InfluxDB endpoint.

Telegraf Configuration for OPC UA Input Plugin

Telegraf Configuration File for OPC UA Data Collection

[agent]

interval = "1s" # Collect data every second

flush_interval = "10s"

 # Input Plugin: OPC UA

[[inputs.opcua]]

```
# OPC UA server endpoint
endpoint = "opc.tcp://host.docker.internal:4840"

# Disable certificate verification
security_policy = "None"
security_mode = "None"
certificate = ""
private_key = ""

# Nodes to read
nodes = [
  {name="Temperature", namespace="2", identifier_type="i", identifier="2"},
  {name="Pressure", namespace="2", identifier_type="i", identifier="3"},
  {name="Humidity", namespace="2", identifier_type="i", identifier="4"}
]

# Output Plugin: InfluxDB v2
[[outputs.influxdb_v2]]
  urls = ["http://127.0.0.1:8086"]
  token =
"y0ZxDSF8scLqkkAm8M_eQfL86KCNbXKpPeB3MdbnWRYDYf2YscXliX5ZYtzVP_bMxami
WQ_YKkyF1ukwq0WX0g=="
  organization = "EmdenLeer"
  bucket = "datadb"
```

Explanation of the telegraf.conf File

The configuration file for Telegraf consists of three main sections. The Agent Settings section ([agent]) defines how often data is collected (interval = "1s" for every second) and how often it is sent to InfluxDB (flush_interval = "10s" every 10 seconds). The OPC

UA Input Plugin section ([[inputs.opcua]]) sets up the connection to the OPC UA server running in Python, specifies the security settings (None, meaning no encryption), and identifies the nodes to monitor (Temperature, Pressure, Humidity), using integer-based identifiers (identifier_type="i"). The InfluxDB Output Plugin section ([[outputs.influxdb_v2]]) configures where to send the collected data (to InfluxDB), uses token authentication for secure access, and stores the data in the datadb bucket under the EmdenLeer organization.

Copying telegraf.conf into the Docker Container

Once the telegraf.conf file is ready, we need to place it inside the Telegraf container.

Stop the Running Telegraf Container

Since Telegraf locks its config file when running, **stop it first**:

docker stop telegraf

Copy telegraf.conf into the Telegraf Container

docker cp telegraf.conf telegraf:/etc/telegraf/telegraf.conf

This command Copies telegraf.conf from your host system to

the Telegraf container.

Restart the Telegraf Container

docker start telegraf

or

docker restart telegraf

Verifying Telegraf's Connection to OPC UA and InfluxDB

To ensure Telegraf is collecting and forwarding data correctly, we need to check its logs.

Checking Logs

Run the following command to view Telegraf's logs:

docker logs telegraf --tail 50

If Telegraf successfully connects to OPC UA, you should see messages like:

I! Loaded inputs: opcuaI! Loaded outputs: influxdb_v2

If data is not being collected, you may see errors

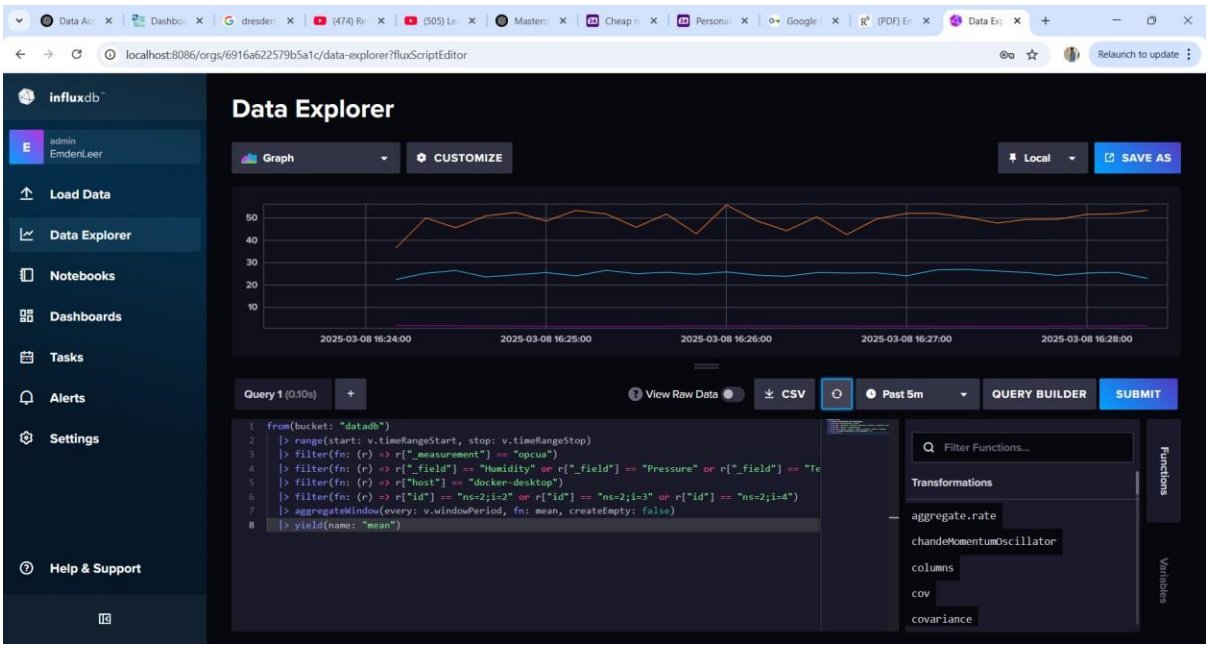
like:

**[E! [inputs.opcua] status not OK for node Temperature (ns=2;i=2): The node id refers to a node that does not exist*

This means the node identifier is incorrect or the OPC UA server is not running.

2.5.Storing Data in InfluxDB

Once Telegraf is successfully collecting data from the OPC UA Server and sending it to InfluxDB, we need to verify that the data is being properly stored in the database. This step is crucial before integrating Grafana for visualization.



In this section, we will:

1. Access the InfluxDB Web Interface
2. Verify the Database (Bucket) and Data
3. Query Data using the InfluxDB UI
4. Use Flux Queries to Inspect Data

Access the InfluxDB Web Interface

InfluxDB provides a built-in web UI where you can explore stored data using queries.

a. Open the InfluxDB UI

To access the default InfluxDB 2 web interface, open your web browser and go to

http://localhost:8086. If InfluxDB is running inside Docker and you're trying to access it from another system, make sure to use the correct IP address of the machine where Docker is running instead of localhost.

b.Log in to InfluxDB

To log in to the InfluxDB web interface, enter the username and password you set up during the InfluxDB initialization. If you don't remember the credentials, you can check your InfluxDB Docker logs by running docker logs *influxdb --tail 50* to find them. If you encounter an authentication error, you may need to create a new token. To do this, navigate to Load Data → API Tokens → Generate New Token in the InfluxDB UI, if necessary.

c.Navigate to Data Explorer

Once logged in, follow these steps to view data:
Go to the "Data Explorer" tab
This is the built-in query editor for InfluxDB 2.

d.Select the "Bucket" where data is stored

In our case, select the bucket datadb, which we configured in Telegraf.

Check if measurement names are available
Click on "Filter by measurement" to see if any data is available.If you see entries like: *opcua_metrics*
then data is being successfully collected!

e.Running Queries in InfluxDB UI

InfluxDB uses Flux, a powerful query language, to fetch data.

Basic Query

To check if Telegraf is writing data, run:

```
from(bucket: "datadb")|> range(start: -1h)   // Fetch last 1 hour of data
```
This query retrieves all data from the last hour.

Filtering by Measurement

To see only OPC UA data, use:

```
from(bucket: "datadb")
  |> range(start: -1h)
```

```
|> filter(fn: (r) => r._measurement == "opcua_metrics")
```

This filters data only for the opcua_metrics measurement.

To view only Temperature values, modify the query:

```
from(bucket: "datadb")
  |> range(start: -1h)
|> filter(fn: (r) => r._measurement == "opcua_metrics" and r._field == "Temperature
```

2.6. Configuring Grafana for Real-Time Data Visualization

Now that we have successfully stored data in InfluxDB, the next step is to visualize it in Grafana. This section provides a detailed step-by-step guide to configuring Grafana and setting up dashboards for real-time monitoring.

a.Accessing Grafana Web UI

Grafana provides a web-based interface where we can configure data sources and create dashboards.

Open a **web browser** and go to:

http://localhost:3000

If Grafana is running inside Docker, ensure that the correct IP address is used.

Login to Grafana

Default credentials:

Username: admin

Password: admin

If you changed your password, enter the new one.
Once logged in, you will see the Grafana Home Dashboard.

b.Adding InfluxDB as a Data Source

To allow Grafana to fetch data from InfluxDB, we need to configure it as a data source.

1.Click on "Configuration" (Gear Icon) → Data Sources.
2.Click on "Add data source".

3.Scroll down and select InfluxDB.

4.In the InfluxDB configuration settings, fill in the following details:
Query Language: Flux (InfluxQL is only for older versions)
URL: http://influxdb:8086
Organization: EmdenLeer
Bucket: datadb
Token:

Use the admin token generated in the InfluxDB UI.
If unsure, navigate to InfluxDB UI → API Tokens and copy the correct token.

5.Click "Save s Test".
If everything is correct, you will see a green "Data source is working" message.

c.Creating a New Dashboard in Grafana

Now that InfluxDB is connected, we will create a dashboard to visualize OPC UA data.

Creating a Dashboard

1. Click on "Create" (Plus icon) → Dashboard.
2. Click on "Add a new panel".

d. Writing a Query in Grafana

Grafana uses Flux to retrieve data from InfluxDB.

Basic Query for Temperature

1. In the Query Editor, write the followingquery:

```
from(bucket: "datadb")|> range(start: -1h)|>
filter(fn: (r) => r._measurement == "opcua_metrics" and
r._field == "Temperature")
```

This retrieves the Temperature data from the last 1 hour.

2. Click "Run Query" to preview the data.

If the data is displayed correctly, proceed to visualization

- .

e. Choosing a Visualization Type

1. Click on "Graph" to display temperature readings over time.
2. Change the panel title to "Temperature Monitoring".
3. Click "Save" and name your dashboard.

f. Adding More Panels (Pressure s Humidity)

To monitor **Pressure** and **Humidity**, repeat **Step 4** with the following queries:

Query for Pressure

```
from(bucket: "datadb")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "opcua_metrics" and r._field == "Pressure")
```

Query for Humidity

```
from(bucket: "datadb")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "opcua_metrics" and r._field == "Humidity")
```

g. Customizing the Dashboard

1. Changing Panel Appearance

- Click on each panel and go to **"Panel Settings"**.
- Adjust:
  - **Line Colors**
  - **Time Ranges**
  - **Labels**

2. Setting Refresh Rate

- In the top-right corner, set auto-refresh to "5s" or "10s".

Saving the Dashboard

- Click "Save" and name it " Data Monitoring".

H: Troubleshooting Grafana

If no data is shown:
Check Data Source Configuration
Go to **Configuration → Data Sources** and ensure InfluxDB is properly connected.
Verify Queries

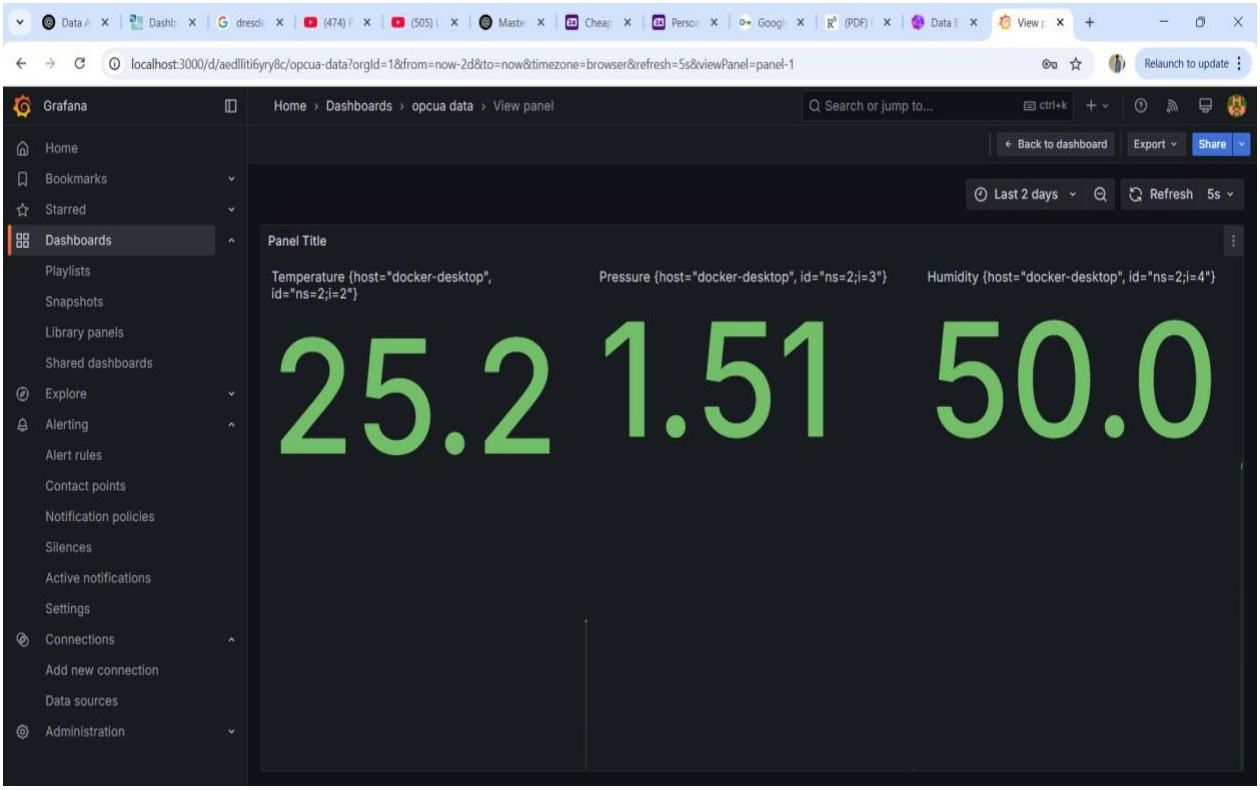
Use **InfluxDB UI → Data Explorer** to ensure data is available.

Check Logs

Run:

```
docker logs grafana --tail 50
```

Look for any authentication errors.



3.Conclusion

This project successfully implemented a data acquisition and visualization system for the Digital Factory at the university using an OPC UA server, Telegraf, InfluxDB, and Grafana within a Dockerized environment. The system enables real-time monitoring and analysis of key industrial parameters, such as temperature, pressure, and humidity, ensuring efficient data collection, storage, and visualization.

By integrating an OPC UA server, the project established a reliable communication channel between industrial devices and the data processing pipeline. Telegraf was configured as an intermediary to collect sensor data from the OPC UA server and forward it to InfluxDB, a time-series database optimized for storing and managing industrial data. Finally, Grafana was used to create dynamic dashboards, providing an intuitive and real-time graphical representation of the collected data.

The entire solution was containerized using Docker and managed through Docker Compose, ensuring a streamlined deployment process, improved scalability, and portability. The use of Dockerized services also contributed to enhanced system reliability by isolating components while maintaining efficient data flow.

4.Key Achievements of the Project:

1. Automated Data Acquisition:

Successfully connected an OPC UA server, running in Python, to collect real-time industrial data. The connectivity and data transmission were verified using OPC UA Expert, ensuring that the data was being accurately transferred from the server.

2. Efficient Data Handling with Telegraf s InfluxDB:

Configured Telegraf to fetch data from the OPC UA server and store it in InfluxDB. The data integrity and correctness were verified by running InfluxDB queries, ensuring that the data collected and stored was accurate and aligned with the expected values.

3. Real-Time Visualization with Grafana:

Created dashboards to display live sensor readings dynamically, incorporating time-series graphs, gauges, and analytical views for effective industrial monitoring. These visualizations provide real-time insights, allowing for easy tracking of sensor data and performance metrics.

4. Docker-Based Deployment:

Used Docker Compose to efficiently manage and deploy multiple services, ensuring modularity and streamlining the process of updates and maintenance. This approach also enhanced portability, making it easier to run the services across different environments without configuration issues.

5.Challenges s Solutions:

1.OPC UA Connectivity Issues:

Initially faced connection errors while retrieving data from the OPC UA server, but the issue was resolved by carefully verifying the correct namespace, node IDs, and security settings, ensuring proper communication between the server and the data collection system.

2.Telegraf Configuration Errors:

Encountered issues in properly mapping OPC UA nodes, but the problem was solved by debugging the telegraf.conf file and verifying the OPC UA namespace identifiers, ensuring the correct mapping of the nodes for data collection.

3.Grafana Data Source Authorization:

Faced authentication issues when integrating InfluxDB with Grafana, but the problem was resolved by generating and using the correct API token for authentication, ensuring smooth communication between the two systems.

6.Future Scope s Improvements:

This project serves as a foundation for implementing industrial data monitoring solutions in smart factories. Some potential improvements include:

1.Integration of AI-based Predictive Analytics:

Using machine learning models on the collected data to predict failures and optimize operations.

2.Expanding the Data Sources:

Connecting more industrial sensors and PLCs for comprehensive monitoring.

3.Alerting s Notifications:

Implementing automatic alerts in Grafana for abnormal conditions.
Edge Computing Integration. Processing data at the edge for faster decision-making before sending it to the cloud.