Energy Monitoring Consumption

Internet Of Things project 2023/2024

Dario D'Ercole	288643
Giovanni Spaziani	295397
Nimrah Yousuf	293654

Index

1. Introduction

This Energy Monitoring Dashboard is a real-time visualization tool that tracks and displays energy usage metrics, including total consumption in megawatt-hours and current energy trends over time. It's designed to help users quickly understand their power usage and its cost implications, ensuring efficient energy management and cost savings.

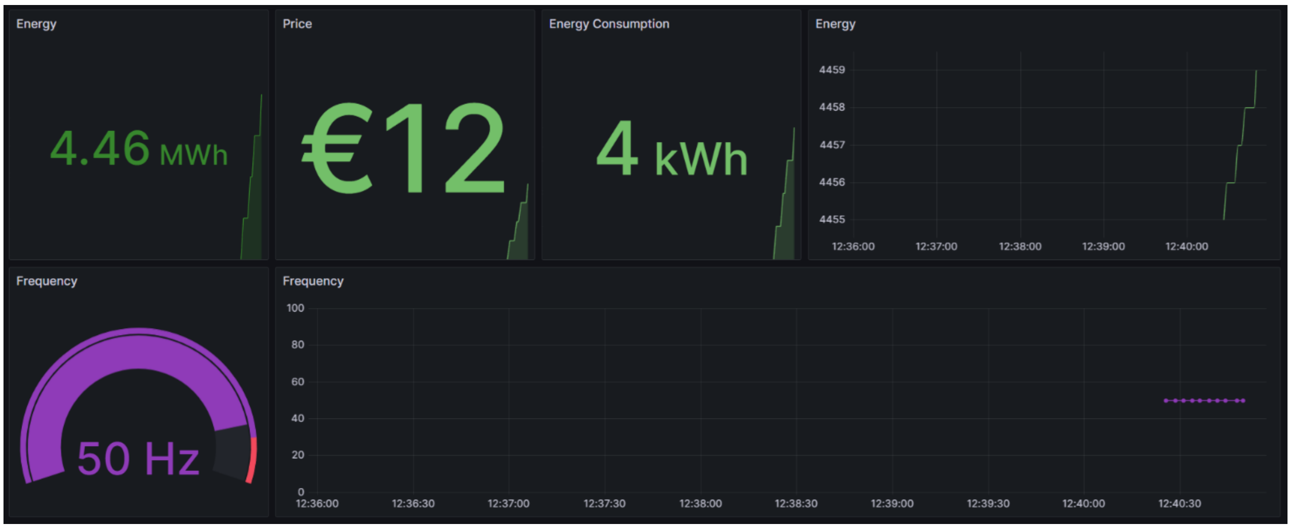


Figure 1 - Dashboard

1.1 Objective of our project

The following list shows the goals we set ourselves before creating this project:

- **Enhance Energy Efficiency**: Provide precise energy consumption data to enable users to identify areas for improving energy usage and reduce wastage.
- **Cost Reduction:** Aid residential and commercial users in minimizing their energy costs by offering detailed insights into their consumption patterns.
- **Real-Time Monitoring:** Deliver up-to-the-minute information on energy usage to facilitate immediate response to any irregularities or system inefficiencies.
- **User Engagement:** Develop an intuitive and interactive dashboard that encourages users to engage regularly with their energy usage data.
- **Sustainability Goals:** Support the pursuit of sustainability by providing tools that can track and report on energy savings and efficiency improvements over time.

1.2 Functional requirements

Functional Requirement	Description
Data Acquisition	The system should continuously acquire real-time data on voltage, current, power, and frequency from the connected energy meters or sensors.
Data Processing	It should process the raw data to calculate energy consumption in kilowatt-hours (kWh) and total energy usage in megawatt-hours (MWh).

Data Display	The dashboard should display current energy consumption, total energy consumption, and all other values in an easy-to-read format.
Historical Data Analysis	The system should provide the ability to view historical energy consumption data over selectable time periods.
Cost Calculation	It should calculate the cost of energy consumed using predefined energy rates, displaying the cost in real-time.
Notifications	The system should generate emails for energy consumption.

Table 1 - Functional requirements

1.3 Non-functional requirements

Non-functional Requirement	Description
Performance	The system should be able to process and display data in real-time with minimal latency.
Availability	The dashboard should be always available for use, with a defined uptime percentage (e.g., 99.9% availability), except during scheduled maintenance.
Scalability	The system must be scalable both vertically and horizontally to accommodate an increasing amount of data and number of users without requiring a complete redesign.
Usability	The user interface should be intuitive and user-friendly, allowing users with varying levels of expertise to operate the dashboard with ease.
Interoperability	The dashboard should be able to integrate and operate with different types of sensors and devices, as well as other systems and software through well-defined APIs.

Table 2 - Non-functional requirements

2. System architecture

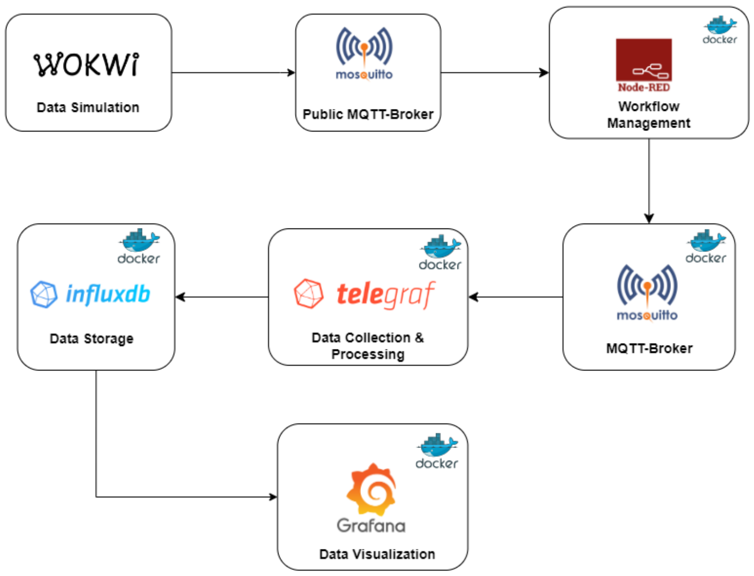


Figure 2 - System architecture

This is the structure of our project architecture, with the technologies used. We designed the whole flow, starting from Wokwi for the data simulation, up to Grafana, where there is a dashboard for viewing data.

3. Wokwi for simulating data

The primary purpose of this simulation is to mimic the behavior of an IoT device that reads various electrical parameters such as voltage, current, power, frequency, and power factor and publishes this data to an MQTT broker at regular intervals.

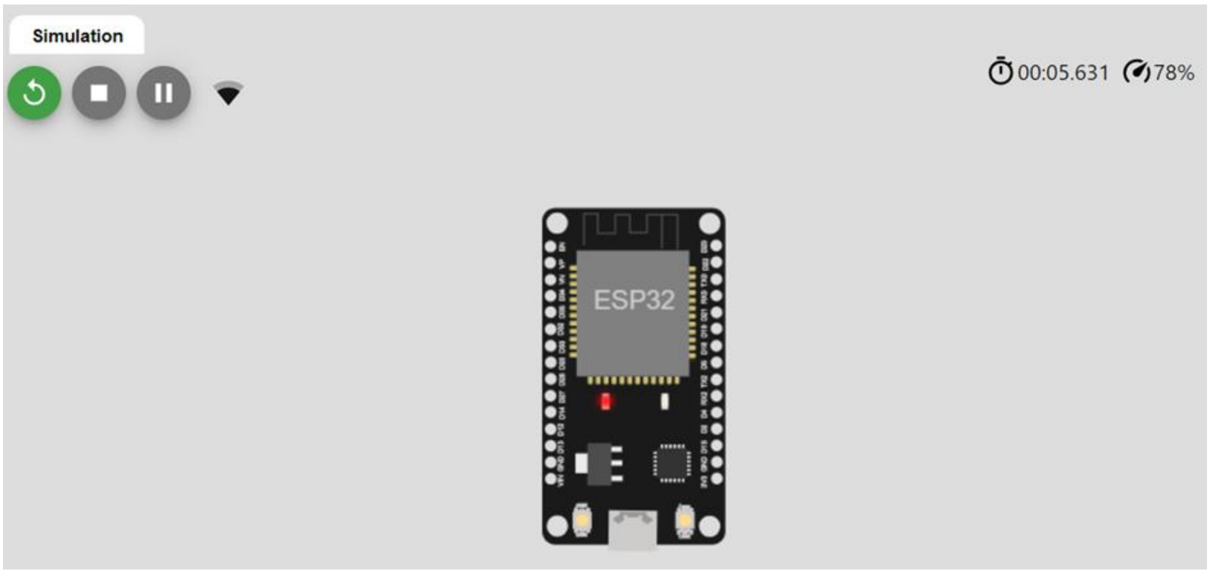


Figure 3 - Wokwi

3.1 Pre-requisites

- ESP32 development board (simulated on platforms such as Wokwi).

- Access to Wi-Fi network with internet connectivity.

- MQTT broker (public or private, e.g., test.mosquitto.org).

3.2 Wi-Fi Connection

The ESP32 is programmed to connect to the specified Wi-Fi network using the SSID and password variables. Upon successful connection, the device will print its IP address to the serial monitor.

3.3 MQTT Configuration

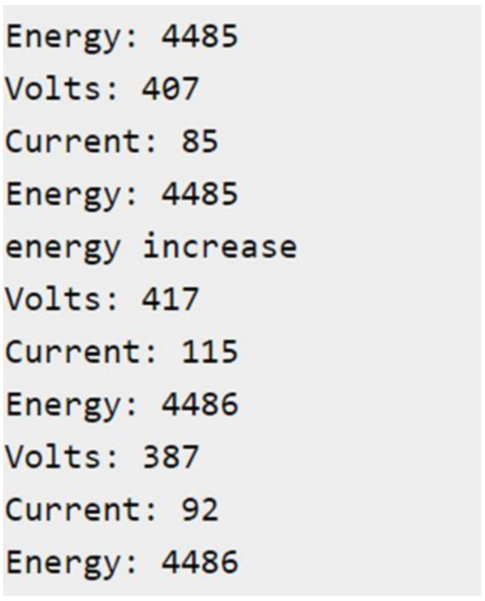
The sketch uses the PubSubClient library for MQTT communication. The ESP32 will attempt to establish a connection with the MQTT broker using the specified server URL. It will publish an introductory message upon connection and subscribe to an MQTT topic.

3.4 Data Simulation and Publishing

The ESP32 simulates sensor readings for various electrical parameters:

Voltage	Simulated within 380 to 420 volts
Current	Simulated within 50 to 120 amperes
Power Load	Simulated within 15 to 50 kW
Frequency	Fixed at 50 Hz
Power Factor	Simulated within 90 to 95%

Table 3 – Data simulation



```
Energy: 4485
Volts: 407
Current: 85
Energy: 4485
energy increase
Volts: 417
Current: 115
Energy: 4486
Volts: 387
Current: 92
Energy: 4486
```

Figure 4 - Output of Wokwi

The simulation publishes data to the MQTT broker every 2 seconds on different topics corresponding to each parameter.

3.5 Topics

The following MQTT topics are used for publishing data:

Voltage	/Thinkitive/v1
Current	/Thinkitive/i1
Energy	/Thinkitive/kwh1
Power Load	/Thinkitive/kw1
Frequency	/Thinkitive/hz1
Power Factor	/Thinkitive/pF1

Table 4 - Topics

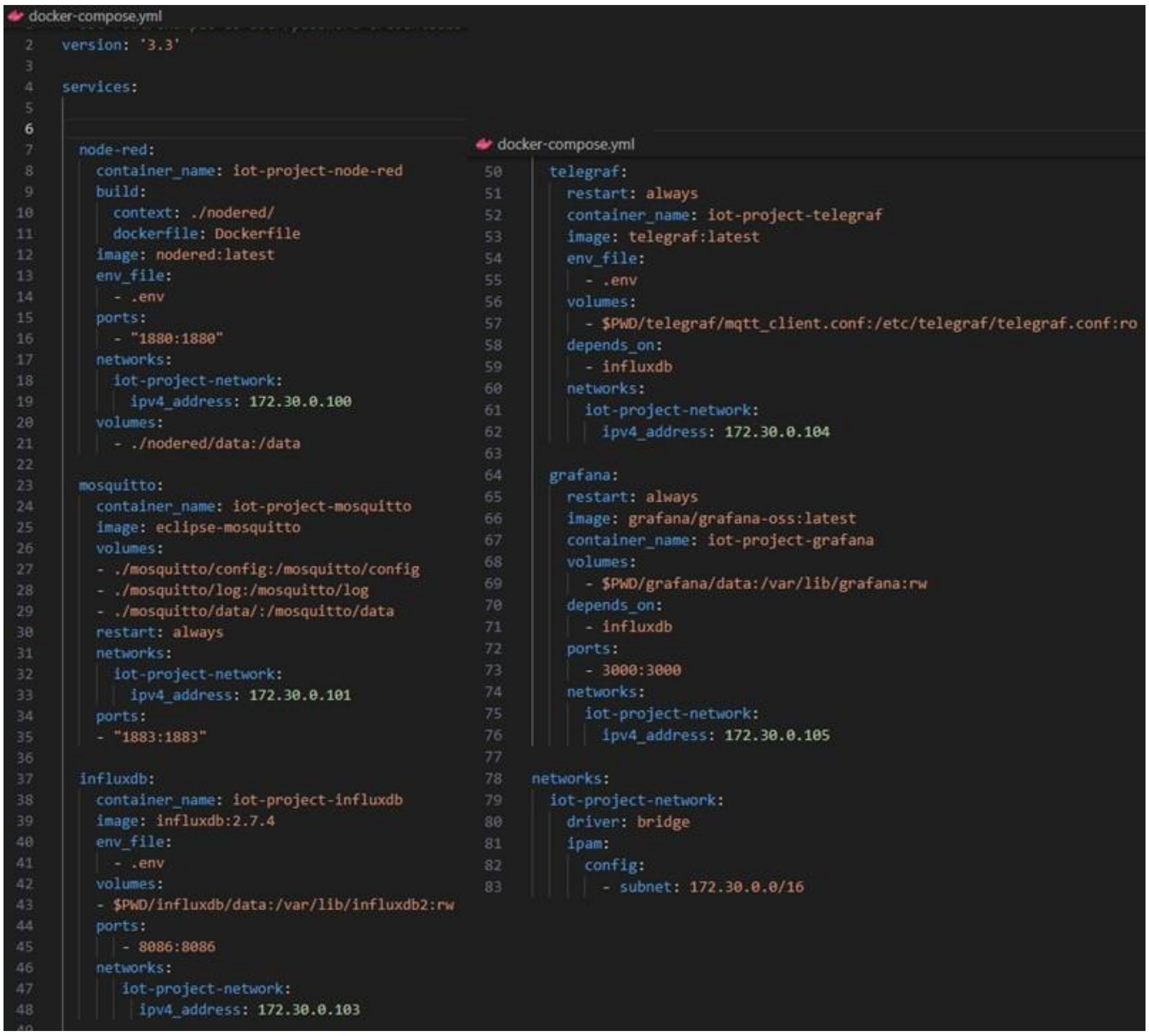
3.6 Notes

Wokwi simulation environment does not include an electrical analyzer sensor; thus, actual sensing of electrical parameters like voltage, current, and power is not performed. The values are purely fictional and for illustrative purposes.

4. Part of the code made in Visual Studio Code

In this part, the part of the docker-compose file will be explained, where all the steps for dockerization of the various components used in our project have been performed. We dockerized:

1. Node-red.
2. Mosquitto.
3. InfluxDB.
4. Telegraf.
5. Grafana.
6. Networks.



docker-compose.yml

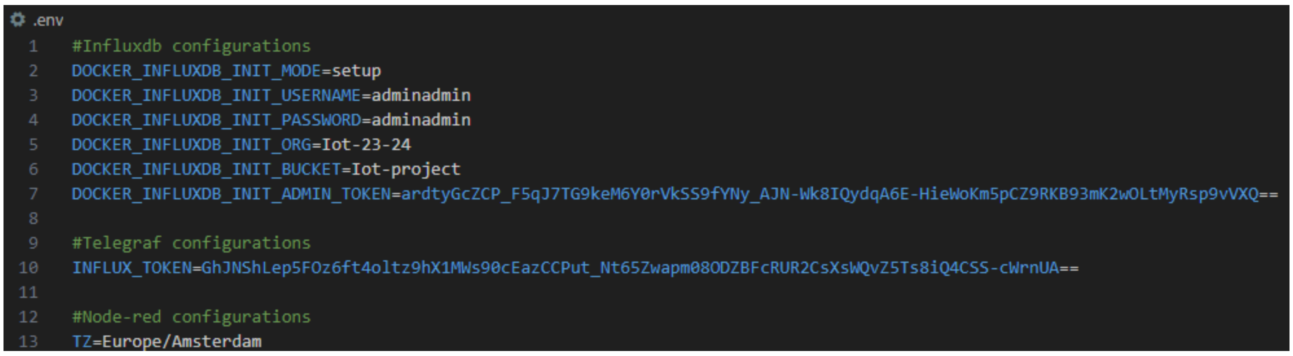
```yaml
version: '3.3'

services:


  node-red:
    container_name: iot-project-node-red
    build:
      context: ./nodered/
      dockerfile: Dockerfile
    image: nodered:latest
    env_file:
      - .env
    ports:
      - "1880:1880"
    networks:
      iot-project-network:
        ipv4_address: 172.30.0.100
    volumes:
      - ./nodered/data:/data

  mosquitto:
    container_name: iot-project-mosquitto
    image: eclipse-mosquitto
    volumes:
    - ./mosquitto/config:/mosquitto/config
    - ./mosquitto/log:/mosquitto/log
    - ./mosquitto/data/:/mosquitto/data
    restart: always
    networks:
      iot-project-network:
        ipv4_address: 172.30.0.101
    ports:
    - "1883:1883"

  influxdb:
    container_name: iot-project-influxdb
    image: influxdb:2.7.4
    env_file:
      - .env
    volumes:
    - $PWD/influxdb/data:/var/lib/influxdb2:rw
    ports:
       - 8086:8086
    networks:
       iot-project-network:
         ipv4_address: 172.30.0.103
```

docker-compose.yml

```yaml
  telegraf:
    restart: always
    container_name: iot-project-telegraf
    image: telegraf:latest
    env_file:
      - .env
    volumes:
      - $PWD/telegraf/mqtt_client.conf:/etc/telegraf/telegraf.conf:ro
    depends_on:
      - influxdb
    networks:
      iot-project-network:
        ipv4_address: 172.30.0.104

  grafana:
    restart: always
    image: grafana/grafana-oss:latest
    container_name: iot-project-grafana
    volumes:
      - $PWD/grafana/data:/var/lib/grafana:rw
    depends_on:
      - influxdb
    ports:
      - 3000:3000
    networks:
      iot-project-network:
        ipv4_address: 172.30.0.105

networks:
  iot-project-network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.30.0.0/16
```

Figure 5 - Docker compose file on VS

We first set all the container names, then we took the official images from Docker-Hub, which serve as a set of instructions to create a Docker container, as a template. We saved the configurations of the various containers with a folder inside the repository. We set up a network using a driven bridge, we set a 16-bit subnet mask to connect the containers. Subsequently we created a ".env" file to keep all the configurations of the various components under control. They can be seen in the following image.

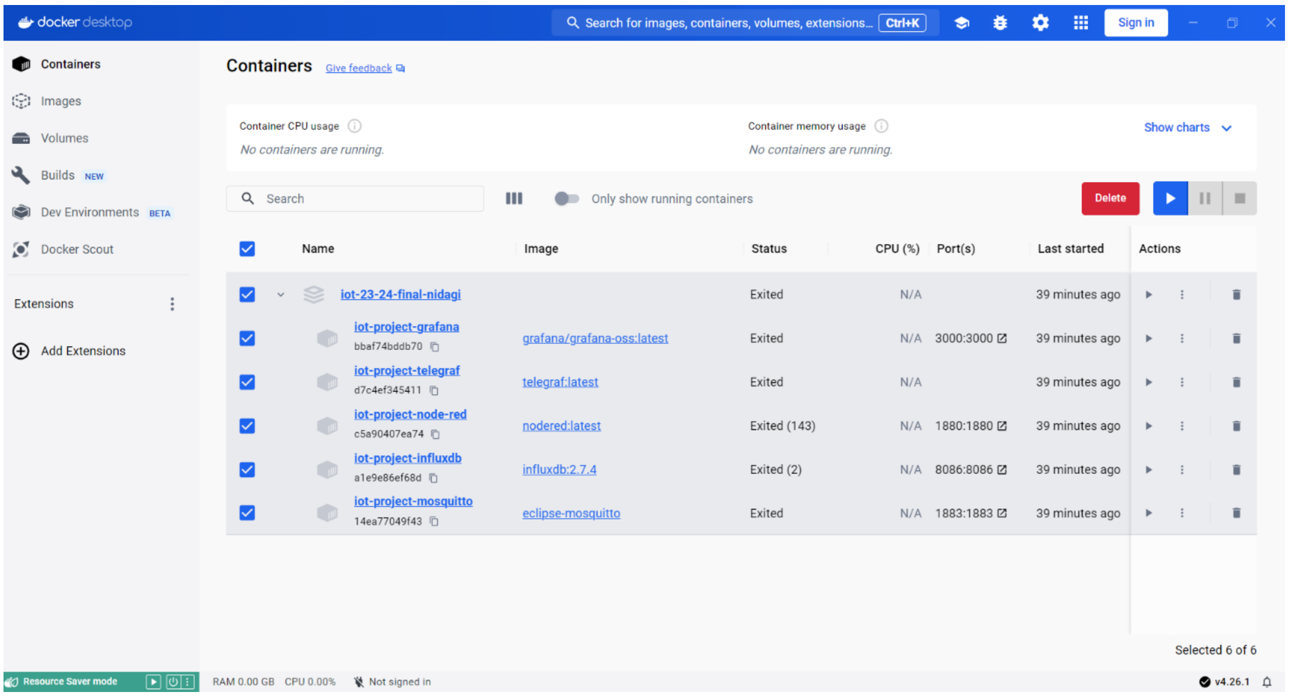


```
.env
#Influxdb configurations
DOCKER_INFLUXDB_INIT_MODE=setup
DOCKER_INFLUXDB_INIT_USERNAME=adminadmin
DOCKER_INFLUXDB_INIT_PASSWORD=adminadmin
DOCKER_INFLUXDB_INIT_ORG=Iot-23-24
DOCKER_INFLUXDB_INIT_BUCKET=Iot-project
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=ardtyGcZCP_F5qJ7TG9keM6Y0rVkSS9fYNy_AJN-Wk8IQydqA6E-HieWoKm5pCZ9RKB93mK2wOLtMyRsp9vVXQ==

#Telegraf configurations
INFLUX_TOKEN=GhJNShLep5FOz6ft4oltz9hX1MWs90cEazCCPut_Nt65Zwapm08ODZBFcRUR2CsXsWQvZ5Ts8iQ4CSS-cWrnUA==

#Node-red configurations
TZ=Europe/Amsterdam
```

Figure 6 - .env file

5. Docker desktop

We used Docker Desktop, a platform that provides an intuitive graphical interface (GUI) to manage containers, applications, and images directly on your computer. It is very intuitive to view all the running containers, the related images, and finally the related ports in localhost.



Containers

Container CPU usage
No containers are running.

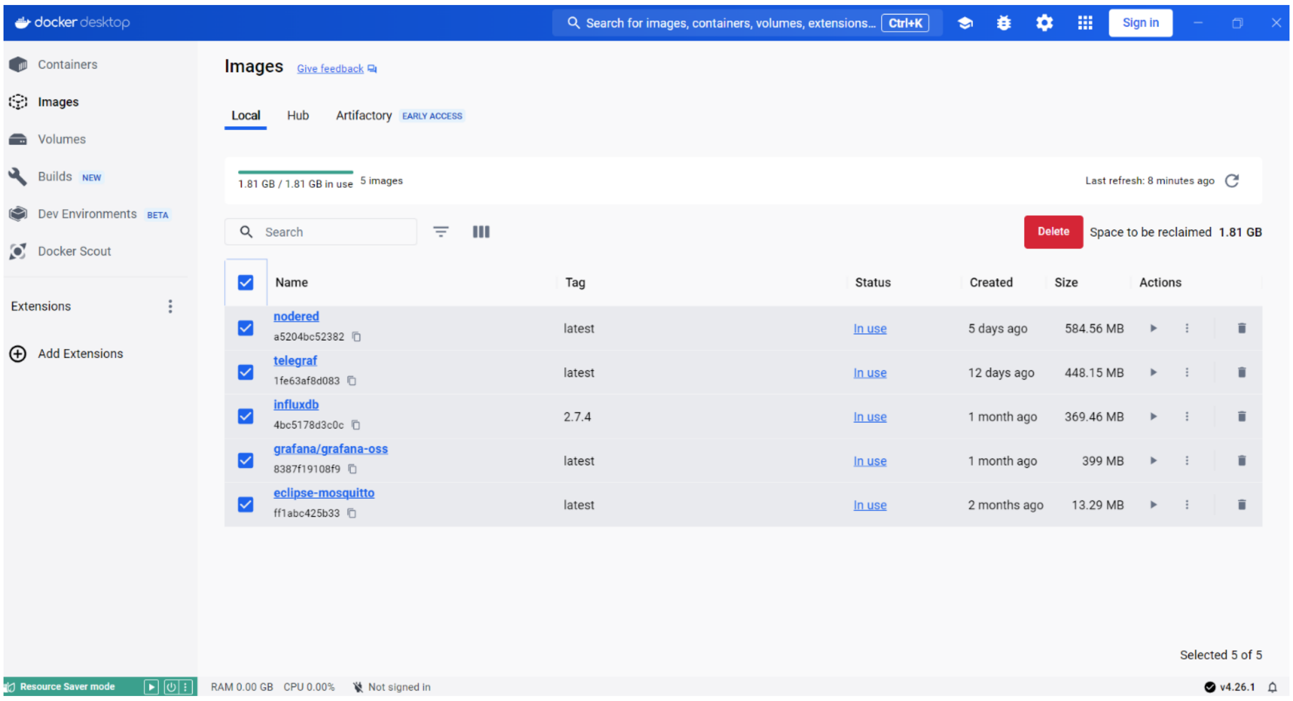
Container memory usage
No containers are running.

Name	Image	Status	CPU (%)	Port(s)	Last started
iot-23-24-final-nidagi		Exited	N/A		39 minutes ago
iot-project-grafana bbaf74bddb70	grafana/grafana-oss:latest	Exited	N/A	3000:3000	39 minutes ago
iot-project-telegraf d7c4ef345411	telegraf:latest	Exited	N/A		39 minutes ago
iot-project-node-red c5a90407ea74	nodered:latest	Exited (143)	N/A	1880:1880	39 minutes ago
iot-project-influxdb a1e9e86ef68d	influxdb:2.7.4	Exited (2)	N/A	8086:8086	39 minutes ago
iot-project-mosquitto 14ea77049f43	eclipse-mosquitto	Exited	N/A	1883:1883	39 minutes ago

Selected 6 of 6

RAM 0.00 GB CPU 0.00% v4.26.1

Figure 7 - Containers section


docker desktop

Search for images, containers, volumes, extensions… Ctrl+K

Sign in

Containers
Images
Volumes
Builds NEW
Dev Environments BETA
Docker Scout

Extensions

Add Extensions

Images Give feedback

Local | Hub | Artifactory EARLY ACCESS

1.81 GB / 1.81 GB in use 5 images

Last refresh: 8 minutes ago

Search

Delete Space to be reclaimed 1.81 GB

	Name	Tag	Status	Created	Size	Actions
✓	nodered a5204bc52382	latest	In use	5 days ago	584.56 MB	
✓	telegraf 1fe63af8d083	latest	In use	12 days ago	448.15 MB	
✓	influxdb 4bc5178d3c0c	2.7.4	In use	1 month ago	369.46 MB	
✓	grafana/grafana-oss 8387f19108f9	latest	In use	1 month ago	399 MB	
✓	eclipse-mosquitto ff1abc425b33	latest	In use	2 months ago	13.29 MB	

Selected 5 of 5

Resource Saver mode RAM 0.00 GB CPU 0.00% Not signed in v4.26.1

Figure 8 - Images section

6. Node-RED

Our Node-RED flowchart can be split in two parts:

1. Flows for sending data coming from the public MQTT broker to our local broker and dashboards.
2. Flow for collecting energy readings for a set interval and send a report by email.

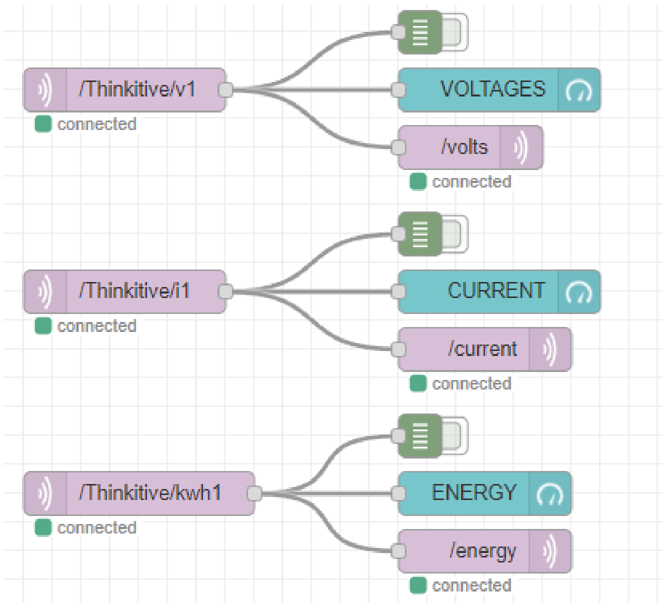


Figure 9 – Sensoring data flow pt. 1

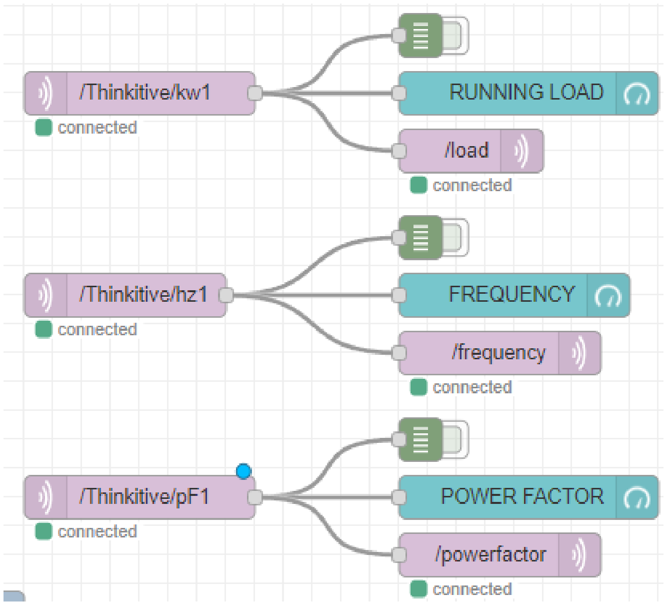


Figure 10 - Sensoring data flow pt. 2

6.1 Data flows

A set of MQTT input nodes subscribe to the topics corresponding to our simulated data on the public Mosquitto test server and it's passed along to a Node-RED dashboard and MQTT output nodes that publishes to our local, dockerized Mosquitto broker.

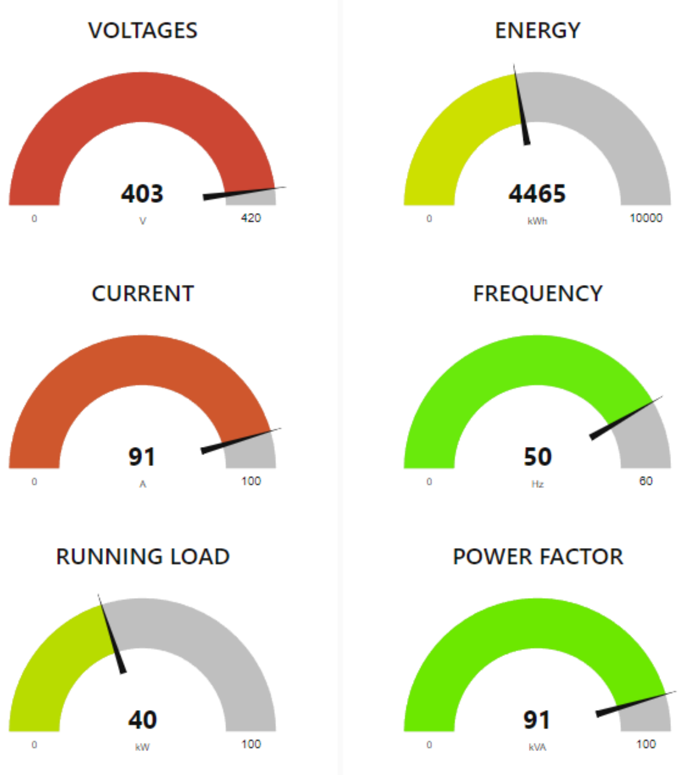
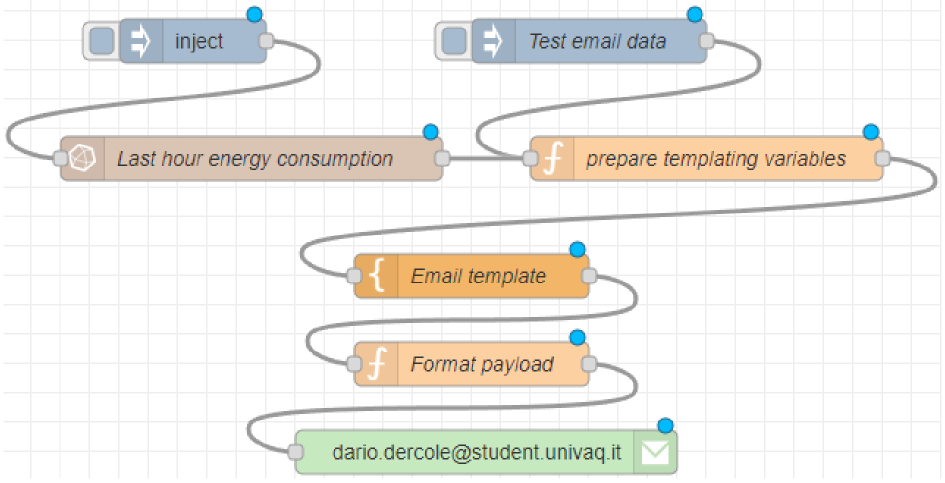


Figure 11 - Node-RED dashboard

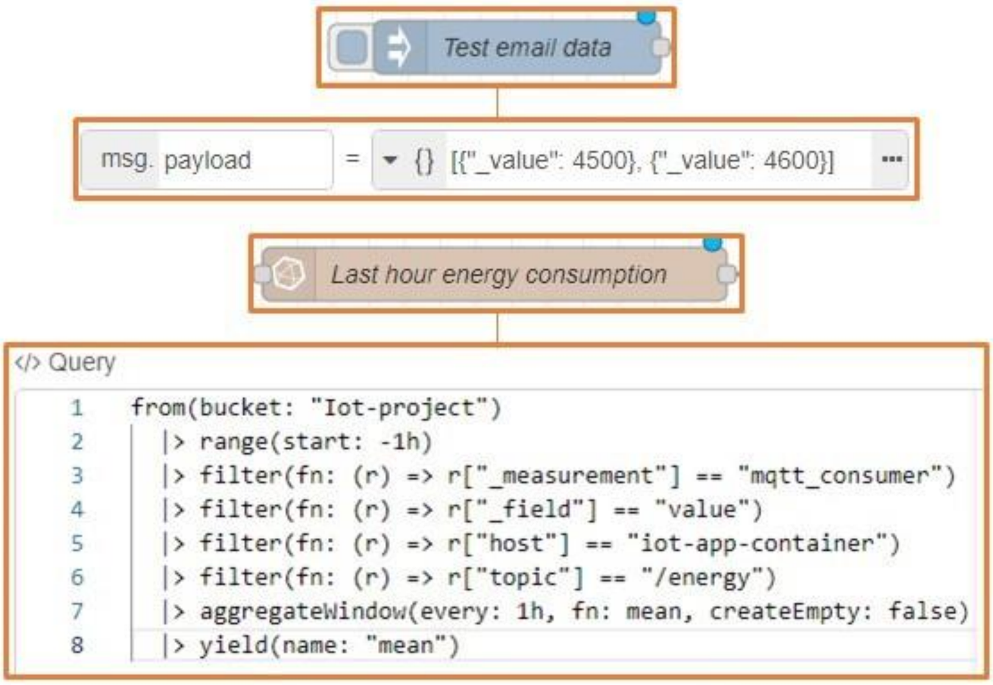
6.2 Email report flow



inject
Test email data
Last hour energy consumption
prepare templating variables
Email template
Format payload
dario.dercole@student.univaq.it

Figure 12 - Email report flow

In this flow we query data about energy consumptions in the last hour. For testing purposes, we may change the range values in the query to take a smaller range or trigger the "Test email data" inject to use an hardcoded JavaScript object.

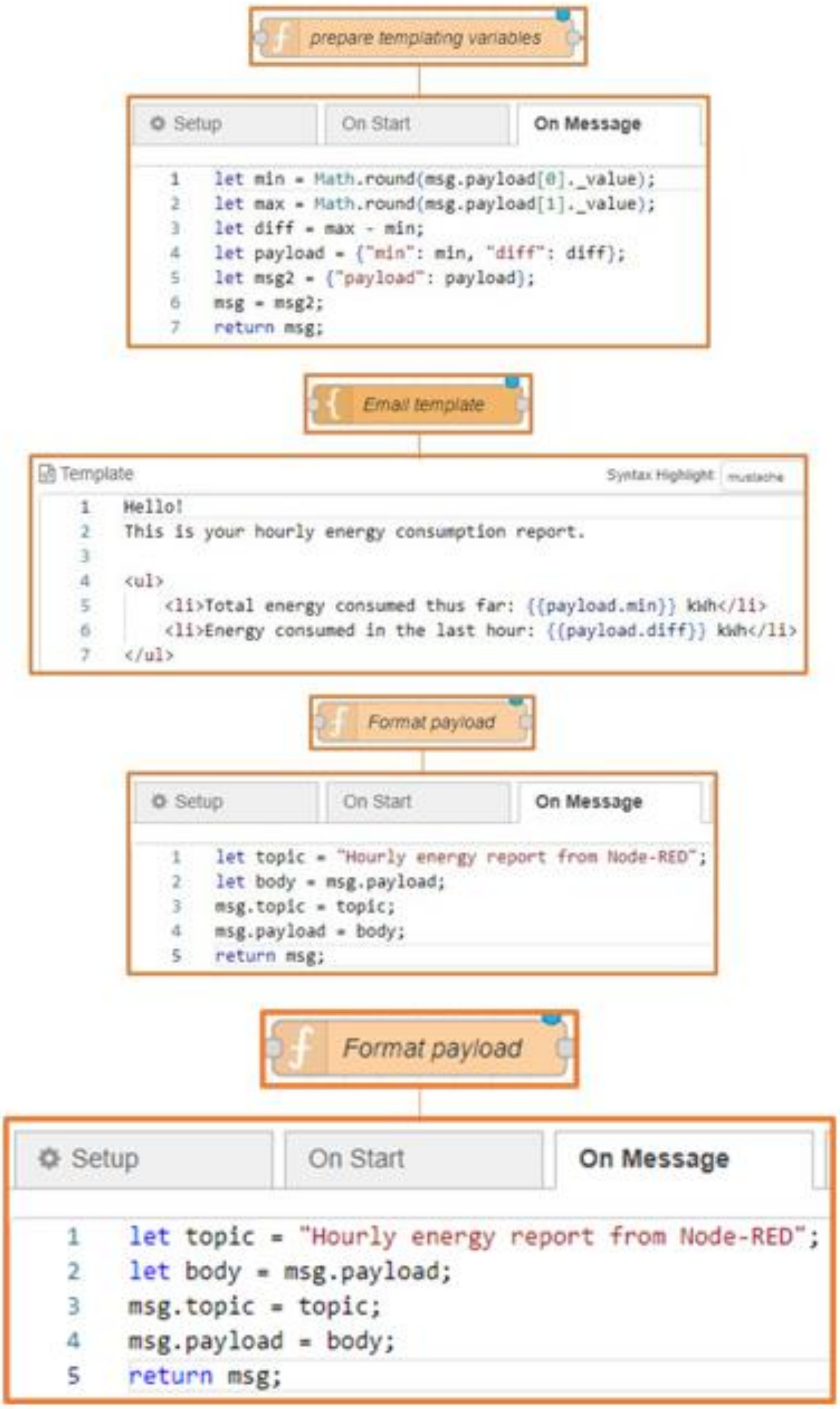


```
msg. payload = {} [{"_value": 4500}, {"_value": 4600}]
```

Query

```
from(bucket: "Iot-project")
  |> range(start: -1h)
  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")
  |> filter(fn: (r) => r["_field"] == "value")
  |> filter(fn: (r) => r["host"] == "iot-app-container")
  |> filter(fn: (r) => r["topic"] == "/energy")
  |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Figure 13 - Some details from the flow pt. 1



```
let min = Math.round(msg.payload[0]._value);
let max = Math.round(msg.payload[1]._value);
let diff = max - min;
let payload = {"min": min, "diff": diff};
let msg2 = {"payload": payload};
msg = msg2;
return msg;
```

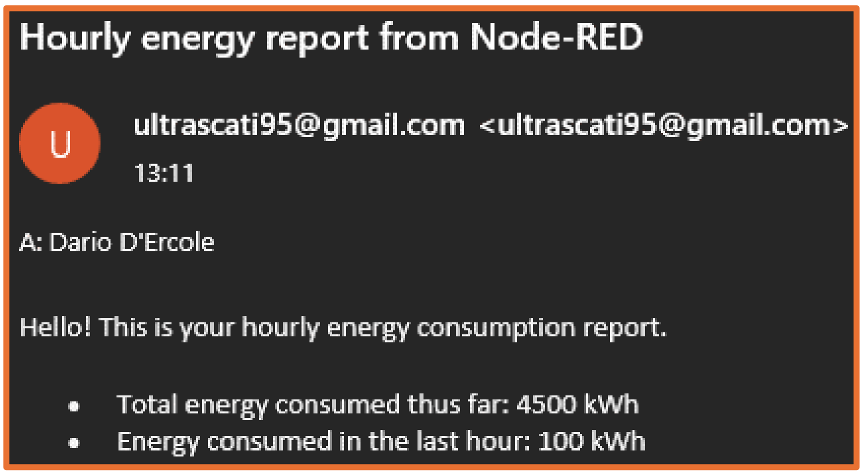
```
Hello!
This is your hourly energy consumption report.

<ul>
    <li>Total energy consumed thus far: {{payload.min}} kWh</li>
    <li>Energy consumed in the last hour: {{payload.diff}} kWh</li>
</ul>
```

```
let topic = "Hourly energy report from Node-RED";
let body = msg.payload;
msg.topic = topic;
msg.payload = body;
return msg;
```

```
let topic = "Hourly energy report from Node-RED";
let body = msg.payload;
msg.topic = topic;
msg.payload = body;
return msg;
```

Figure 14 - Some details from the flow pt. 2

And here is the resulting email using the “Test email data” trigger.


Hourly energy report from Node-RED

ultrascati95@gmail.com <ultrascati95@gmail.com>
13:11

A: Dario D'Ercole

Hello! This is your hourly energy consumption report.

- Total energy consumed thus far: 4500 kWh
- Energy consumed in the last hour: 100 kWh

Figure 15 - Email sent from Node-RED

7. InfluxDB

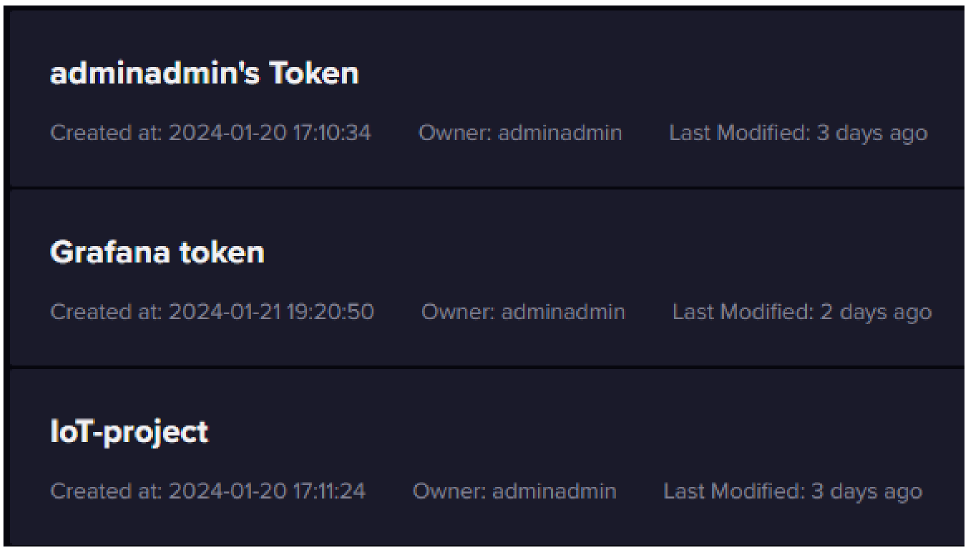
InfluxDB is an open-source time series database developed by the company InfluxData. It is used for storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics.

7.1 Setup and configurations

In our project InfluxDB is deployed in a Docker container using the official docker image (v2.7.4) and environment variables have been set into the .env file to enable automatic configuration (see Figure 5).

7.2 API tokens

In addition to the default token (adminadmin), we generated two custom tokens to use in other applications that communicate with InfluxDB (see the following image).


adminadmin's Token

Created at: 2024-01-20 17:10:34 Owner: adminadmin Last Modified: 3 days ago

Grafana token

Created at: 2024-01-21 19:20:50 Owner: adminadmin Last Modified: 2 days ago

IoT-project

Created at: 2024-01-20 17:11:24 Owner: adminadmin Last Modified: 3 days ago

Figure 16 - Tokens available on InfluxDB

- Grafana token provides read-only access to the bucket and is provided to Grafana as part of the InfluxDB data source configuration.
- IoT-project is a token that we provide to Telegraf so that it can write into our "IoT-project" bucket when it consumes data from the local MQTT broker.

7.3 Sample queries to our bucket

Below are some example images of the data that is being written in our bucket.

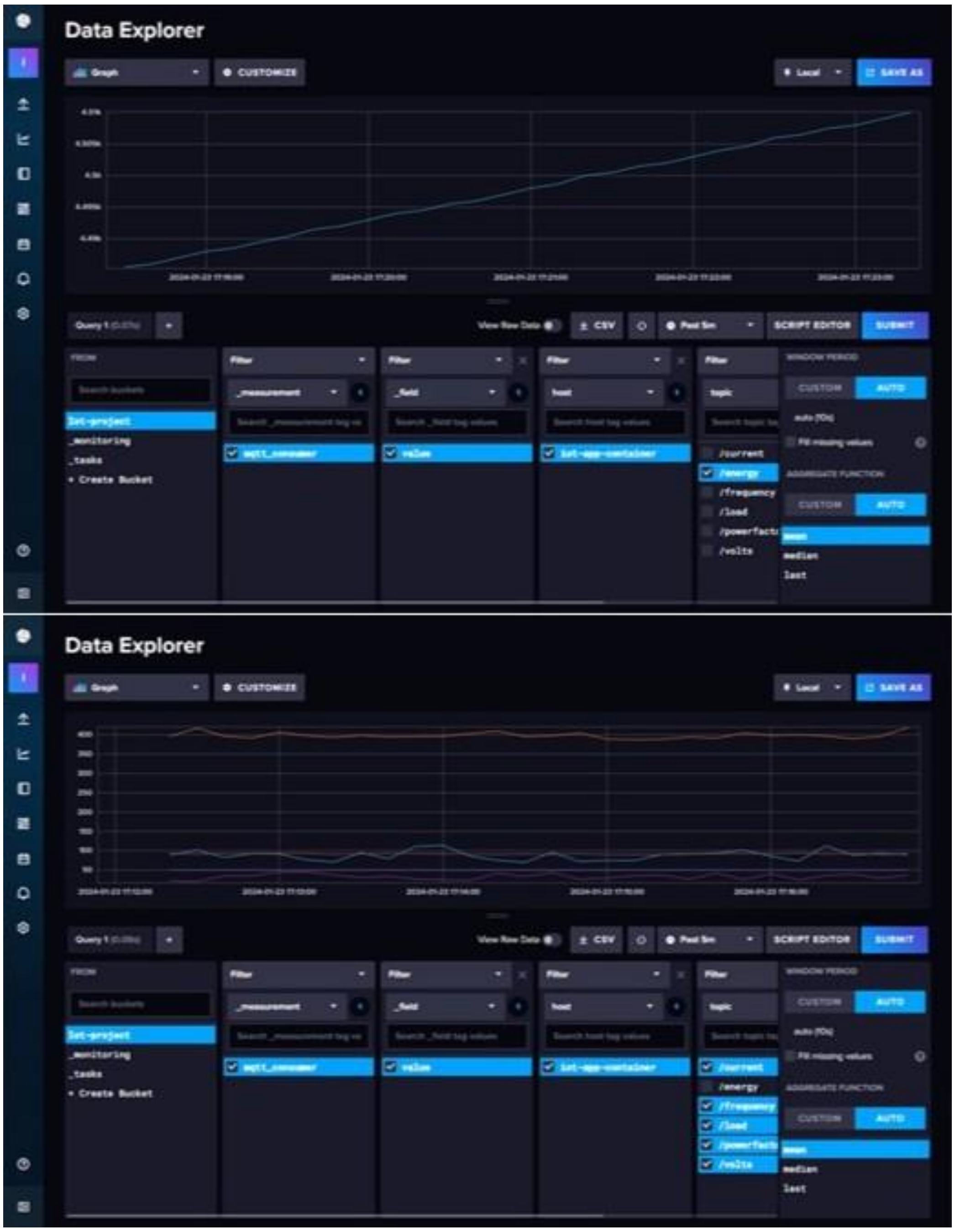
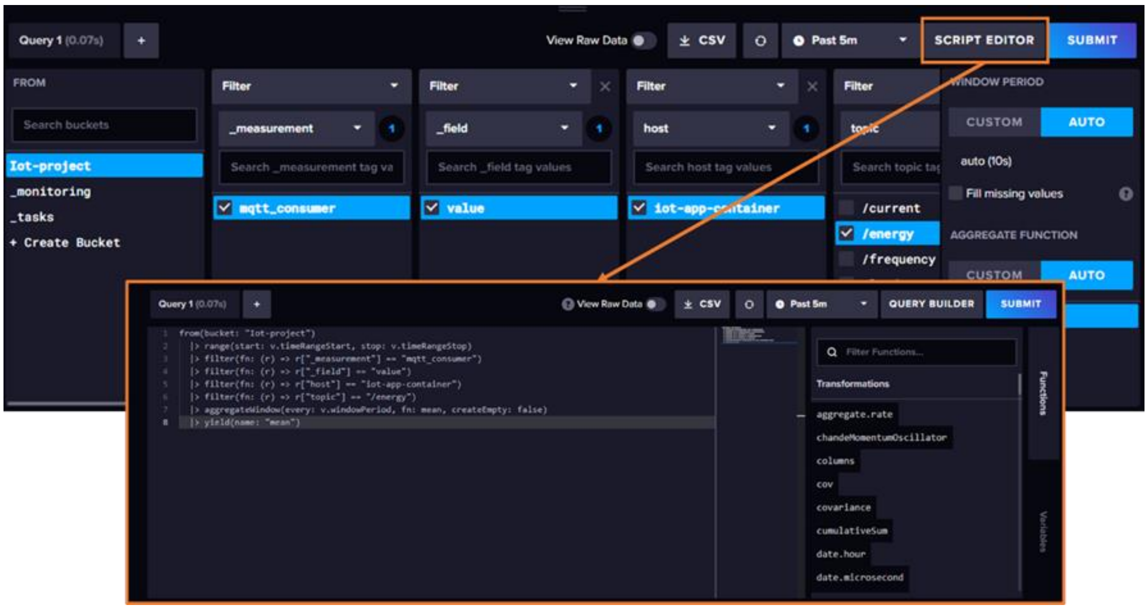


Figure 17 - Data written in our bucket

Then, we made use of the script editor view to easily obtain queries to use later inside our Grafana dashboard.



```
from(bucket: "Iot-project")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")
  |> filter(fn: (r) => r["_field"] == "value")
  |> filter(fn: (r) => r["host"] == "iot-app-container")
  |> filter(fn: (r) => r["topic"] == "/energy")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Figure 18 - Script editor from InfluxDB to get queries

8. Grafana

Grafana is a web application for interactive data visualization and analysis. From the "Dashboards" section we select our personal dashboard, called "Energy monitoring IoT project".

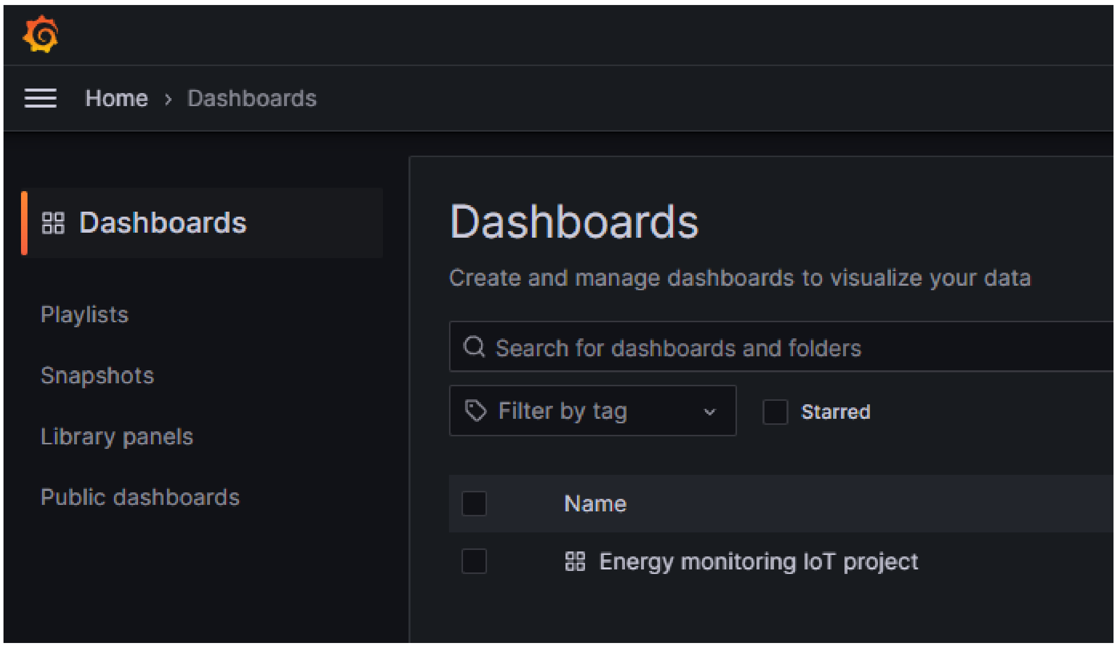
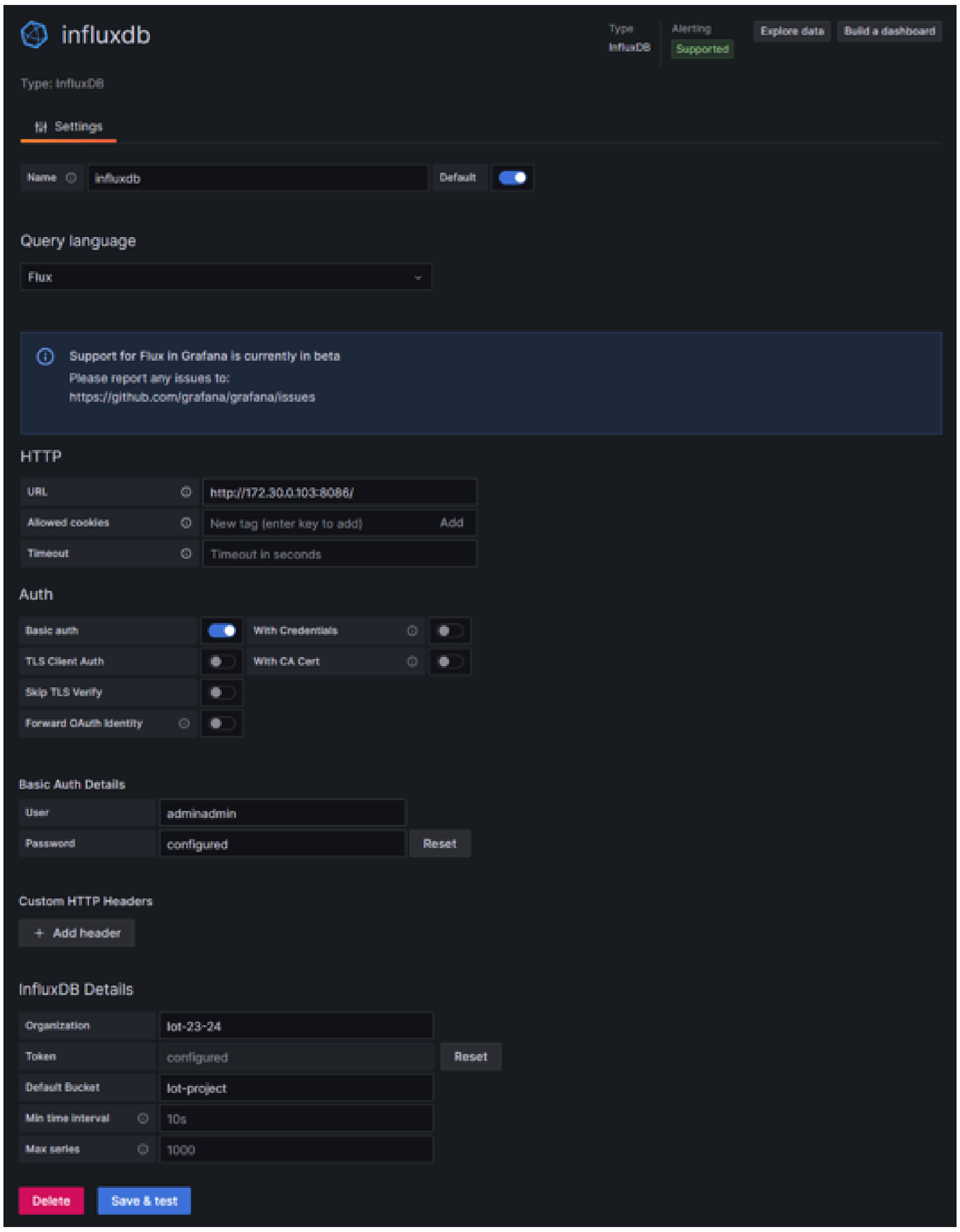

Home › Dashboards
Dashboards
Playlists
Snapshots
Library panels
Public dashboards
Dashboards
Create and manage dashboards to visualize your data
Search for dashboards and folders
Filter by tag
Starred
Name
Energy monitoring IoT project

Figure 19 - Dashboard section on Grafana

8.1 Configuration of data source

We setup communication to our InfluxDB bucket by using InfluxDB credentials and "Grafana token" API token.



Name	influxdb	Default	●

Query language

Flux

Support for Flux in Grafana is currently in beta
Please report any issues to:
https://github.com/grafana/grafana/issues

HTTP

URL	http://172.30.0.103:8086/
Allowed cookies	New tag (enter key to add) Add
Timeout	Timeout in seconds

Auth

Basic auth	●	With Credentials	○
TLS Client Auth	○	With CA Cert	○
Skip TLS Verify	○		
Forward OAuth Identity	○		

Basic Auth Details

User	adminadmin	
Password	configured	Reset

Custom HTTP Headers

+ Add header

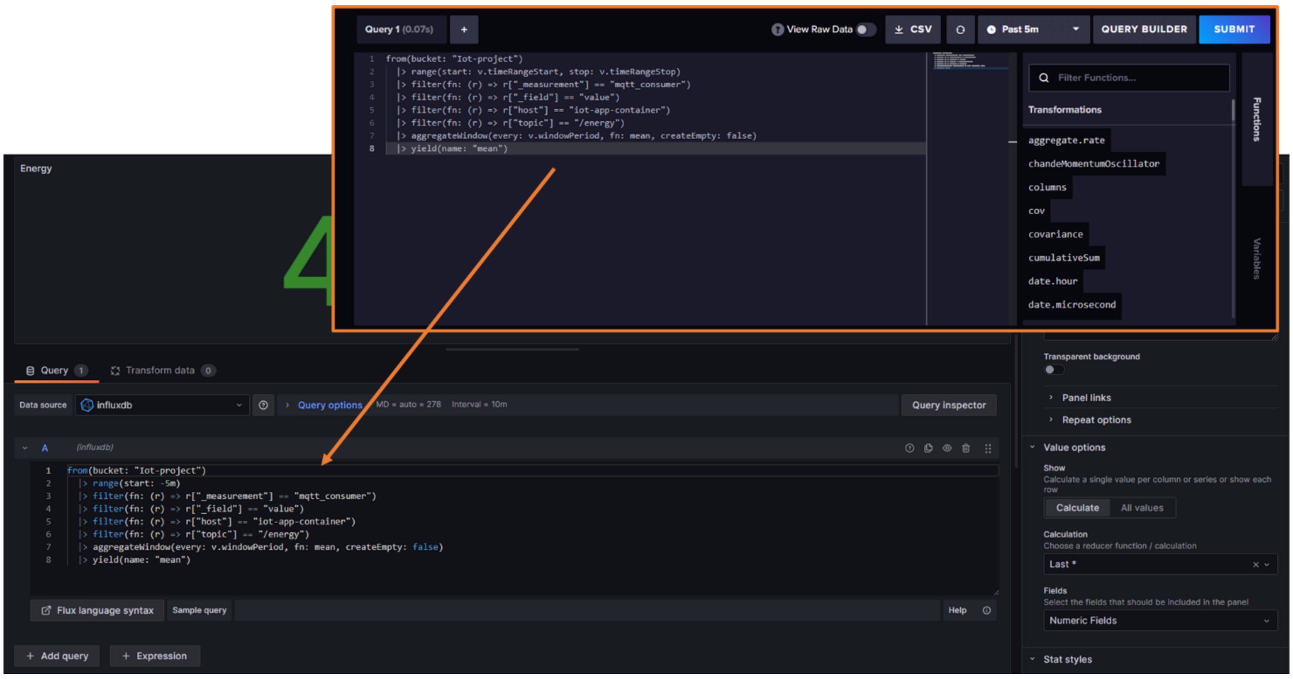
InfluxDB Details

Organization	Iot-23-24	
Token	configured	Reset
Default Bucket	Iot-project	
Min time interval	10s	
Max series	1000	

Figure 20 - InfluxDB data-source configuration in Grafana

8.2 Sample query for a dashboard panel

For each dashboard panel we used the query generated by influx (applying in some cases an edit on the selected range in the query, like in the image).



```
from(bucket: "Iot-project")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")
  |> filter(fn: (r) => r["_field"] == "value")
  |> filter(fn: (r) => r["host"] == "iot-app-container")
  |> filter(fn: (r) => r["topic"] == "/energy")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

```
from(bucket: "Iot-project")
  |> range(start: -5m)
  |> filter(fn: (r) => r["_measurement"] == "mqtt_consumer")
  |> filter(fn: (r) => r["_field"] == "value")
  |> filter(fn: (r) => r["host"] == "iot-app-container")
  |> filter(fn: (r) => r["topic"] == "/energy")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Figure 21 - Sample query

8.3 Data visualization

After clicking on the dashboard, and after starting the simulation with Wokwi, the data begins to arrive. The dashboard after these procedures is as follows. you can specify a time interval to be considered for obtaining the data, and you can also select the auto-refresh option for the data.

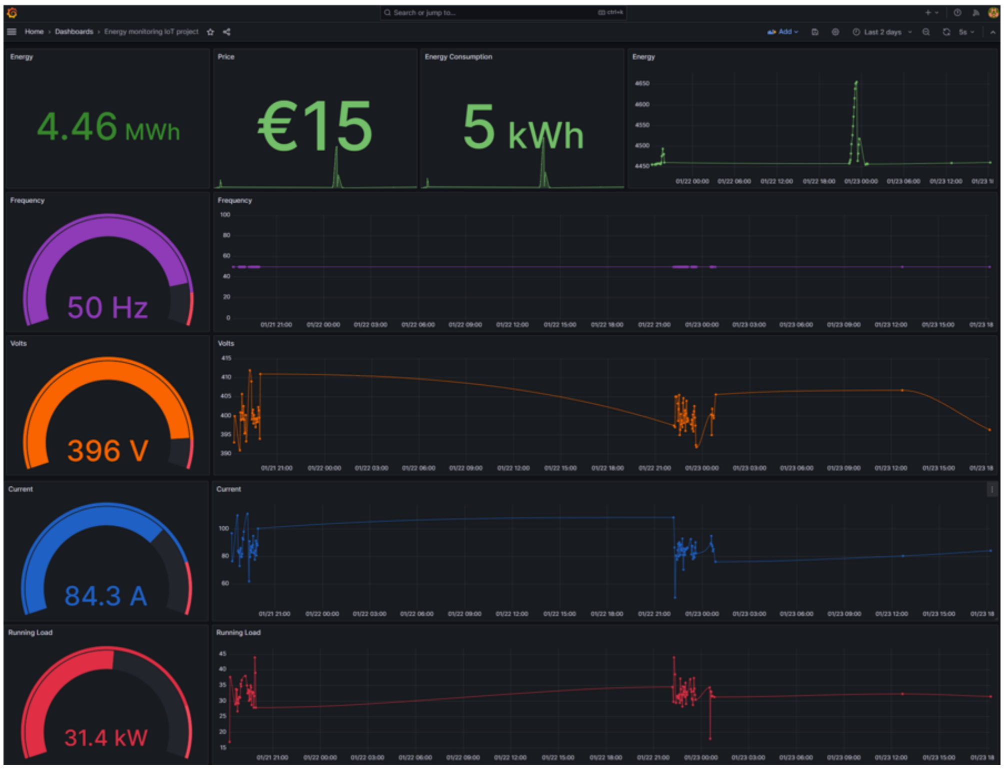
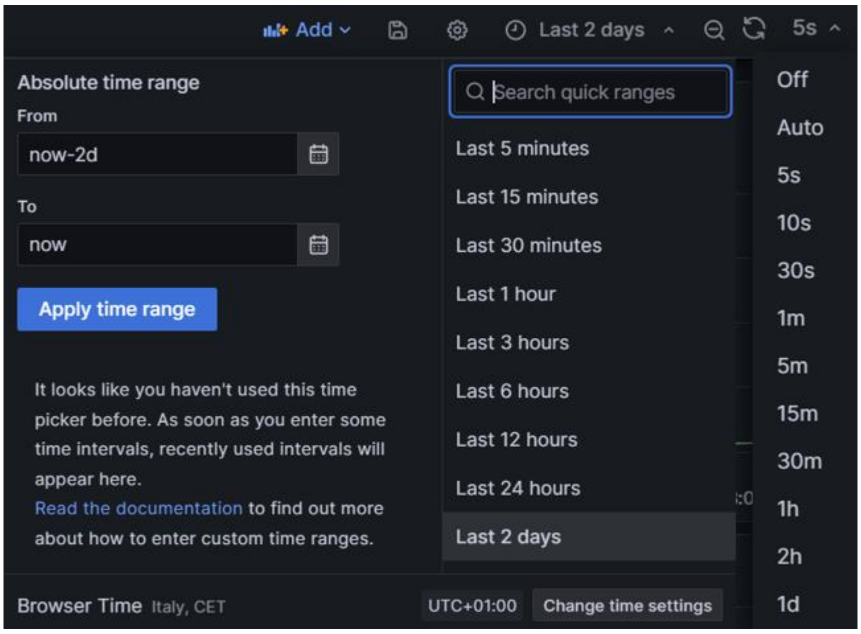


Figure 22 - Data visualization on the Grafana dashboard



Add | Last 2 days | 5s

Absolute time range

From

now-2d

To

now

Apply time range

It looks like you haven't used this time picker before. As soon as you enter some time intervals, recently used intervals will appear here.
Read the documentation to find out more about how to enter custom time ranges.

Search quick ranges

Last 5 minutes
Last 15 minutes
Last 30 minutes
Last 1 hour
Last 3 hours
Last 6 hours
Last 12 hours
Last 24 hours
Last 2 days

Off
Auto
5s
10s
30s
1m
5m
15m
30m
1h
2h
1d

Browser Time Italy, CET UTC+01:00 Change time settings

Figure 23 - Time interval and auto-refresh section