Grant Agreement N° 872592

PLATOON
Digital platform and analytic tools for energy

Deliverable D5.4
Energy Analytics dashboard for business analysts

Contractual delivery date:
M27

Actual delivery date:
31/03/2022

Responsible partner:
P02: TECN, SPAIN

Project Title	PLATOON – Digital platform and analytic tools for energy
Deliverable number	D5.4
Deliverable title	Energy Analytics dashboard for business analysts
Author(s):	TECN, UBO, ENGIE, CS, IND
Responsible Partner:	TECN (Partner nº2)
Date:	31/03/2022
Nature	Other
Distribution level (CO, PU):	PU
Work package number	WP5 – WP5-Big data sharing and analysis reference implementations
Work package leader	UBO, Germany
Abstract:	This deliverable explains in detail two open source alternatives of analytics dashboards have been developed and implemented as part of PLATOON project. The document contains the complementary information for the actual developed open source visualisation dashboards publicly available in the PLATOON GitHub repository
Keyword List:	Visualisation Dashboard, Batch, Realtime, Open Source, Visual Analytics.
Reviewer(s):	Valentina Janev (IMP) Philippe Calvez (ENGIE)
Approved by:	Valentina Janev (IMP)
Recommended/mandatory readers:	WP5 and WP6 partners

The research leading to these results has received funding from the European Community's Horizon 2020 Work Program (H2020) under grant agreement no 872592.

Document Revision History

Version	**Date**	**Modifications Introduced**	
		Modification Reason	Modified by
v0.1	16/02/2022	1st version of TOC	TECN
v0.2	09/03/2022	Completed 1st version of sections 2 and 3.	TECN
v0,3	17/03/2022	Contributions from IND, UBO and TECN to sections 3,4 and 5.	TECN, IND, UBO
V0.4	23/03/2022	Amendments after internal review from Valentina Janev	TECN

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

Table 1: Terms and Abbreviations

API	Application Programming Interface
CA	Consortium Agreement / Certification Authority
CO	Confidential

DAPS	Dynamic Attribute Provisioning Service
DM	Dissemination Manager
EC	European Commission
EM	Exploitation Manager
GA	Grant Agreement
GAM	General Assembly Meeting
GUI	Graphical User Interface
HW	Hardware
IOT	Internet of Things
IDS	International Data Spaces
IT	Information Technology
PaaS	Platform as a Service
PKI	Public Key Infrastructure
PM	Project Manager
PU	Public
QA	Quality Assurance
RE	Restricted
SC	Steering Committee
SLA	Service Level Agreement
SaaS	Software as a Service
SW	Software
TM	Technical Manager
WP	Work package
WPL	Work package Leader

Executive Summary

The analytics dashboard (also known as data analytics dashboard) is one of the northbound interfaces of the PLATOON reference architecture. The analytics dashboard is part of the intelligence layer which is responsible for processing and visualising the data from the lower levels in order to get valuable insights.

This deliverable is the complementary technical report which supports the actual developed open source visualisation dashboards that are publicly in the PLATOON GitHub repository (https://github.com/PLATOONProject).

This deliverable explains in detail two open source alternatives of data analytics dashboards that have been developed and implemented as part of PLATOON project: 1) Batch-data Visualisation Dashboard 2) Real-time Visualisation Dashboard. Each of the alternatives have their own advantages and disadvantages and are suited to each of the two scenarios defined in deliverable D5.1.

On the one hand, the Batch visualisation dashboard is an open-source dashboard based on the Generic Visualisation Toolbox developed in task T4.6. The target users of the tool are energy experts with high domain knowledge but low coding skills. Thus, an intuitive and simple Graphical User Interface (GUI) has been defined in order to configure and visualize the dashboards.

On the other hand, the Real-time Visualisation Dashboard is an open-source solution which integrates a series of existing open source tools that allow users to view graphics on the screen, representations of real time data, events and process prompts, as well as historical data, offering a set of standard display panels, screens, and charts.

A description of the different visual analytics pipelines and templates implemented for each of the scenarios are shown in section 5.

1 Introduction

This deliverable explains in detail the two open source alternatives of analytics dashboards that have been developed and implemented as part of PLATOON project. The document contains the complementary information for the actual developed open source visualisation dashboards publicly available in the PLATOON GitHub repository (https://github.com/PLATOONProject). The report is structured in 6 sections as follows:

Section 2 explains the link of the developed analytics dashboards with the PLATOON Reference architecture defined in D2.1 and the different scenarios defined in D5.1.

Section 3 explains in detail the developed visualisation dashboard for batch data (non-edge scenario) including a description of the main components, configuration instructions and user guide.

Section 4 explains in detail the developed visualisation dashboard for real-time data (edge scenario) including a description of the main components, configuration instructions and user guide.

Section 5, contains a description of the different visual analytics pipelines and visualisation templates implemented for each of the scenarios.

Finally, in the conclusions section the main advantages and limitations of each of the alternatives are summarized.

2 Analytics Dashboard

The analytics dashboard (also known as data analytics dashboard) is one of the northbound interfaces of the PLATOON reference architecture shown in Figure 1. The analytics dashboard is part of the intelligence layer which is responsible for processing and visualising the data from the lower levels in order to get valuable insights.

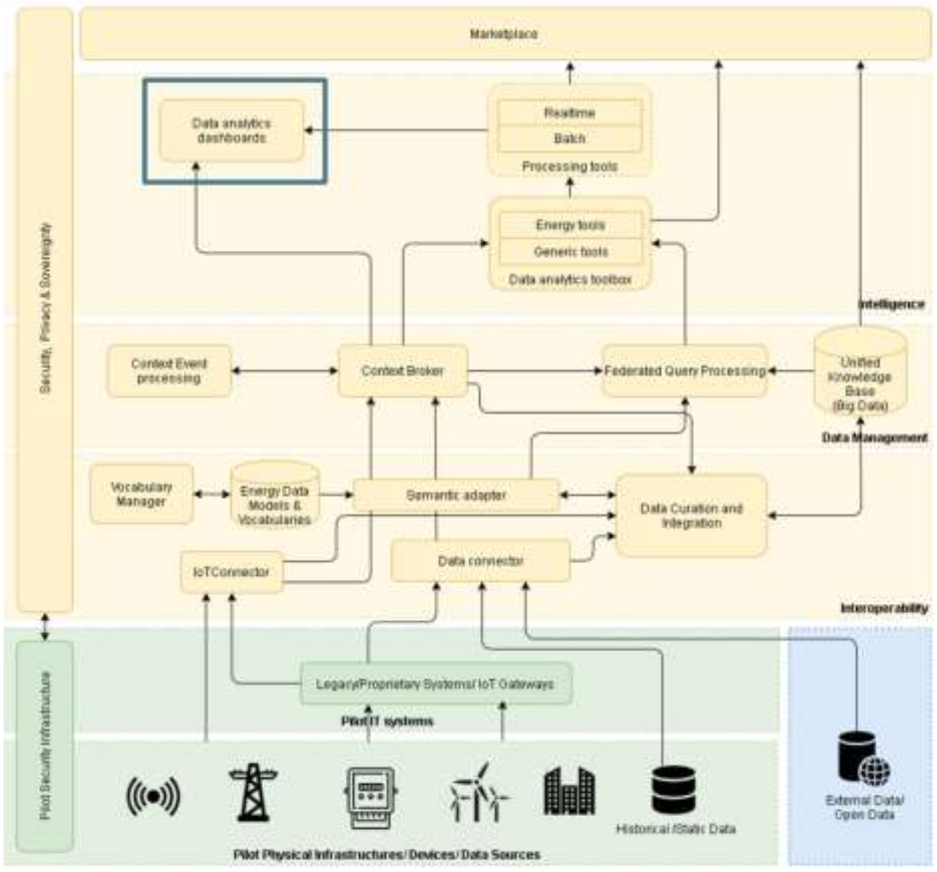

Marketplace
Security, Privacy & Sovereignty
Data analytics dashboards
Realtime
Batch
Processing tools
Energy tools
Generic tools
Data analytics toolbox
Intelligence
Context Event processing
Context Broker
Federated Query Processing
Unified Knowledge Base (Big Data)
Data Management
Vocabulary Manager
Energy Data Models & Vocabularies
Semantic adapter
Data Curation and Integration
IoTConnector
Data connector
Interoperability
Pilot Security Infrastructure
Legacy/Proprietary Systems/ IoT Gateways
Pilot IT systems
Historical /Static Data
External Data/ Open Data
Pilot Physical Infrastructures/ Devices/ Data Sources

Figure 1: PLATOON reference architecture

The following figure shows the connection of the Data Analytics Toolbox with the rest of the components of the PLATOON Reference Architecture. On the one hand, the data analytics dashboard can be connected to data management layer components, specifically to Context Broker, Complex Event Processing and Federated Query Processing for visualising raw data. On the other hand, the data analytics dashboard can be connected to the processing tools for visualising processed data produced by the Data Analytics Toolbox.

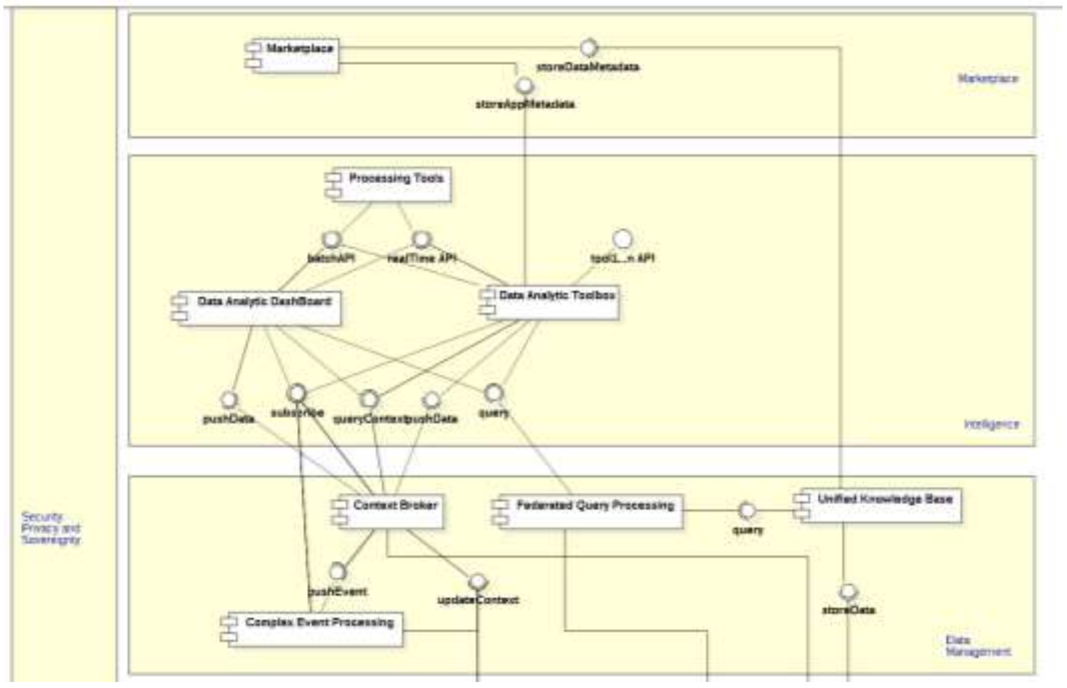


Figure 2: Data Analytics Dashboard - connection with other components

In this task two open source alternatives of data analytics dashboards have been developed and implemented:

1. Batch-data Visualisation Dashboard based on the Open Source Generic Visualisation Toolbox developed in task T4.6.
2. Real-time Visualisation Dashboard based on existing open source solutions such as Grafana and Chronograf.

Each of the alternatives have their own advantages and disadvantages and are suited to each of the two scenarios defined in deliverable D5.1. On the one hand, the Batch-data Visualisation Dashboard has been implemented and tested for the non-edge (batch) scenario. On the other hand, the Real-time Visualisation Dashboard has been implemented and validated for the edge (real-time) scenario.

3 Batch-data Visualisation Dashboard

3.1 Description

The Batch visualisation dashboard is an open-source dashboard that provides a collection of reusable visualization types that can be easily integrated into customer-oriented cloud-based dashboard for predictive analytics and insights analysis.

The visualisation dashboard includes different type of charts based on the Generic Visualisation Toolbox developed in task T4.6.

The target user of the tool are energy experts with high domain knowledge but low coding skills. Thus, an intuitive and simple Graphical User Interface (GUI) has been defined in order to configure and visualize the dashboards.

Summing up, the developed dashboard is formed of 3 main components:

1. Generic Visualisation Tools
2. API service
3. Graphical User interface.

All the different components have been integrated into separate dockers. A FAST API service has been built on top to allow Graphical User Interface to interact with the Generic Visualization Tools. This provides the required elasticity and flexibility to customize different type of dashboards and be able to scale them according to the user needs.

Each of the components are explained in more detail in the following subsections.

3.1.1 Generic Visualisation Toolbox

As part of Task T4.6, a set of Open Source Generic Visualization tools were developed in Python. The corresponding code is openly available in: https://github.com/PLATOONProject/Generic_Visualisation_Toolbox

These tools make use of existing Open Source Python libraries such as Pandas[i] and Bokeh[ii]. In this project we have developed a top-level soft layer formed of several functions that facilitates the configuration and implementation of Bokeh[ii] based dynamic dashboards.

In this case, the resulting plots are served as dynamic HTML file allowing the combination of different charts that allows to generate dynamic dashboards. This solution results especially suitable for static batch data and facilitates the integration with the GUI.

The Generic Visualization Toolbox is formed of 13 different tools listed below that provide a separate functionality:

1. Heatmap: heatmap plot.
2. Histograms: histogram plot.
3. Correlation: correlation matrix plot.
4. Time_lines: time series line plot.
5. Time_lines_and_dots: combination of time series line and scatter graph.
6. Time_bars: time series bar plot.
7. Time_scatter: time series scatter plot.
8. Time_range_tool: Plots a range selection tool for time series plots with xpan tools.
9. Scatter: Scatter plot with histograms on x and y axis and regression line/curve.
10. Lines: line graph
11. Plot_blanks: plots data frame values as pixel intensity, using index as y axis values.
12. Plot_table: plot column data table.
13. Plot_text: plots a text box.

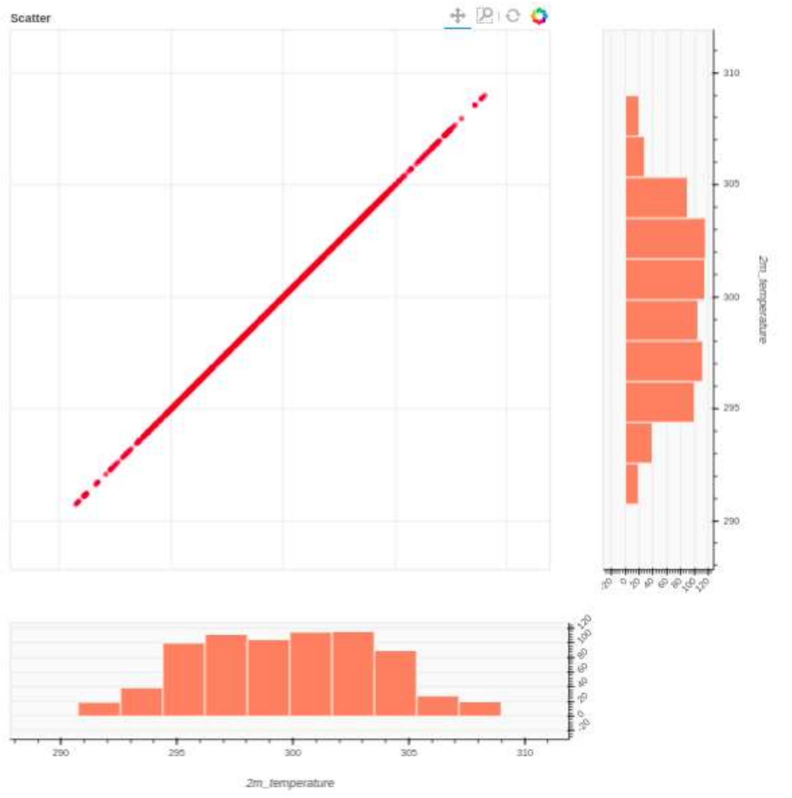


Figure 3: Batch-data Visualisation Dashboard - Example of Plot Types

In addition, these tools are enhanced by functionalities for data exploration such as range selection, zoom, group selection, etc.

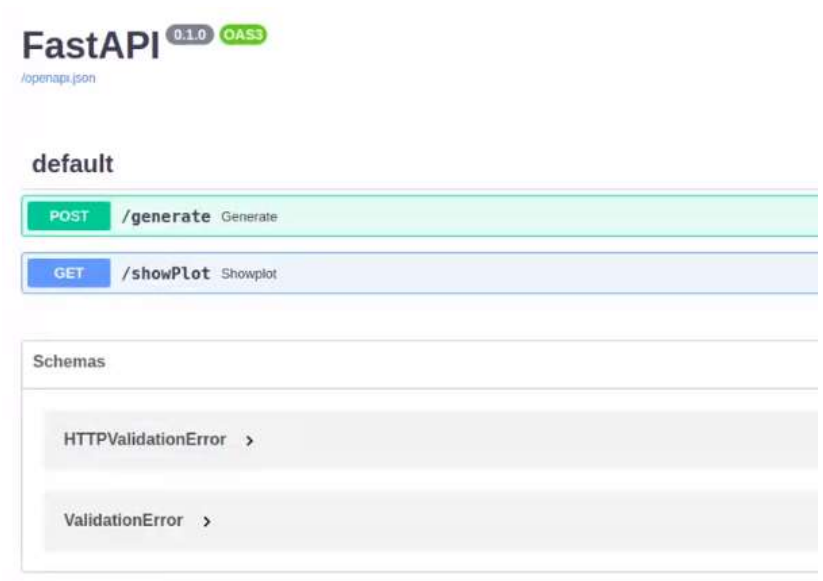
In this task, each of the visualization tools that are part of the toolbox have been embedded into separate dockers. This provides the required elasticity and flexibility to customize different type of dashboards and be able to scale them according to the user needs.

3.1.2 API Service

On top of the independently dockerized visualization tools explained in the previous section, a REST API wrapper has been developed which enables the connection between the dockerized visualization tools and the dockerized Graphical User Interface explained in the next section. The API wrapper has been built using FastAPI[iii]. FastAPI is a modern, high-performance, web framework for building APIs with Python 3.6+ based on standard Python type hints which has the following advantages:

- Very high performance. One of the fastest Python frameworks available.
- Robust with minimum code duplication and fewer bugs.
- Standards-based: Based on and fully compatible with OpenAPI (previously known as Swagger) and JSON Schema as defined in the PLATOON common API specification in deliverable D2.2.

Figure 4 shows the OpenAPI specification for the developed FastAPI- Wrapper.



FastAPI 0.1.0 OAS3
/openapi.json

default

POST /generate Generate

GET /showPlot Showplot

Schemas

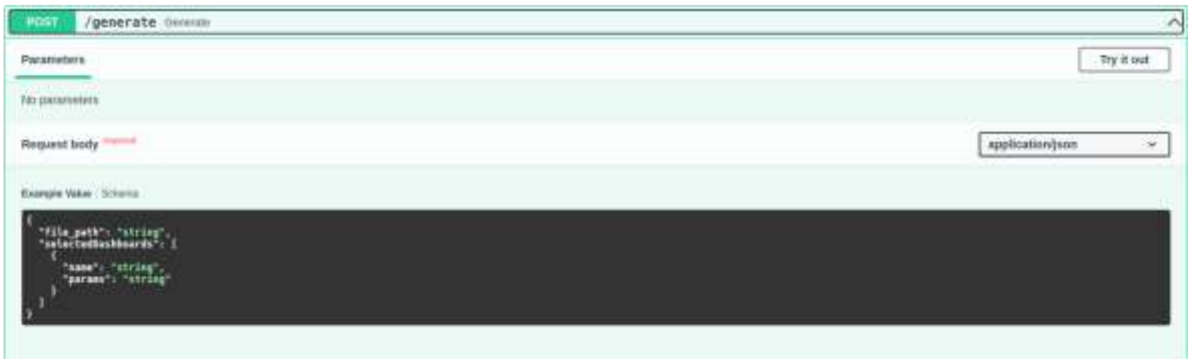
HTTPValidationError >

ValidationError >

Figure 4: Batch-data Visualisation Dashboard - FastAPI- Wrapper

As it can be seen two methods have been defined:

- generate: The function of this method goes through the user's request and generates an html with the plots that come in this request. This request will contain the url of the json to be processed, the plots to generate on that json and the necessary parameters for each plot



POST /generate Generate

Parameters

Try it out

No parameters

Request body required

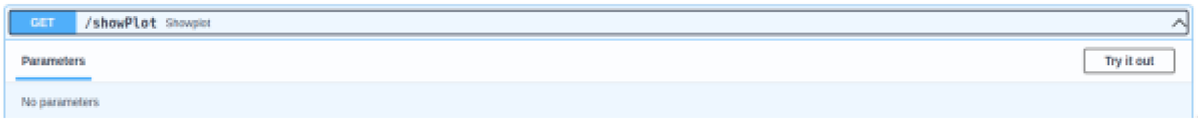
application/json

Example Value | Schema

```
{
  "file_path": "string",
  "selectedDashboards": [
    {
      "name": "string",
      "params": "string"
    }
  ]
}
```

Figure 5: Batch-data Visualisation Dashboard - FastAPI - generate method

- showplot: This method displays the previously generated html



GET /showPlot Showplot

Parameters

Try it out

No parameters

Figure 6: Batch-data Visualisation Dashboard - FastAPI - showplot method

3.1.3 Graphical User Interface

Finally, a Graphical User Interface (GUI) also known as Front-End has been defined using Angular[iv]. Angular is an open source application design framework and development platform for creating for building scalable web applications. Angular is built on TypeScript[v] and includes a collection of well-integrated libraries that cover a wide variety of features, including routing, client-server communication, and more.

Angular completely separates the frontend and backend in the application, avoids writing repetitive code and keeps everything more organized thanks to its MVC (Model-View-Controller) pattern, ensuring rapid development, while allowing modifications and updates.

The front-end reads the input data from a url that will provide a JSON file with the following schema:

```
{
"$schema": "http://json-schema.org/draft-04/schema#",
"type": "object",
"properties": {
"column_name1": {
"type": "object",
"properties": {
"1498867200000": {
"type": "number"
},
"1498870800000": {
"type": "number"
},
...........
},
"required": [
"1498867200000",
"1498870800000",
...........
]
},
"column_name2": {
"type": "object",
"properties": {
"1498867200000": {
"type": "number"
},
"1498870800000": {
"type": "number"
},
...........
},
"required": [
"1498867200000",
"1498870800000",
...........
]
},
"column_name3": {
"type": "object",
"properties": {
"1498867200000": {
"type": "number"
},
"1498870800000": {
"type": "number"
},
...........
},
"required": [
"1498867200000",
"1498870800000",
............
]
}
},
"required": [
"column_name1",
"column_name2",
"column_name3"
]
}
```

The frond end has the following functionalities:

- Select JSON URL path: where the user can include the url to the JSON file that will be used to generate the plots
- Plot type selection dropdown: where the user can select the specific plots listed in the Generic Visualisation Toolbox explained in section 3.1.1.
- Variable selection box: free text box where the user can select the input variables for the different type of charts. It simply accepts the names of the columns to display based on the type of chart.
- Add Plot Button: It will add the plot selected in the combo along with its parameters to the list of plots below.
- List of plots: It will be able to see the list of plots that we want to visualize. It is allowed to delete some plot
- Generate Button: The generate button will send to the server all the necessary information to generate the plots that will be shown

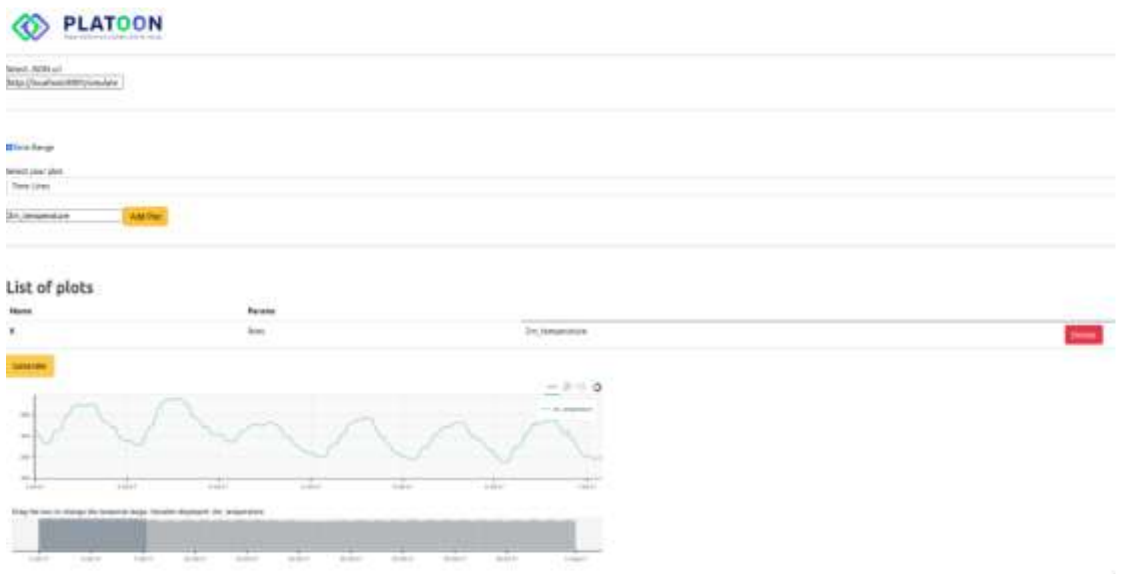


Figure 7: Batch-data Visualisation Dashboard - GUI

3.2 Configuration

The developed batch data analytics visualisation dashboard is open source and is openly available in: https://github.com/PLATOONProject/Analytics-Dashboard-WP5.

The following subsections contain the details regarding the software/hardware prerequisites and configuration instructions for the developed dashboard.

3.2.1 Software and Hardware Prerequisites

Being a web application, we have to take into account that the requirements are divided into client and server.

On the client side, the requirements are those set by the angular itself. Angular supports most recent browsers. This includes the following specific versions:

Table 2: Angular Supported Browsers[vi]

Browser	**Supported versions**
Chrome	latest
Firefox	latest and extended support release (ESR)
Edge	2 most recent major versions
Safari	2 most recent major versions
iOS	2 most recent major versions
Android	2 most recent major versions

In the server part, everything will be deployed in Docker Swarm, with which the requirements part is delegated to it. Docker Swarm is an orchestration management tool that runs on Docker applications. It helps end-users in creating and deploying a cluster of Docker nodes. Each node of a Docker Swarm is a Docker daemon, and all Docker daemons interact using the Docker API.

Hardware requirements:

- Minimum hardware: 4 core / 8 GB RAM / 200 GB hard disk for three nodes.
- Recommended hardware: 8 core / 16 GB RAM / 200 GB hard disk for three nodes.
- Optional hardware: 8 core+ / 16 GB RAM+ / 200 GB+ hard disk for six nodes.

3.2.2 Configuration Instructions

Two (2) separate modules have been generated. On the one hand, a module containing the Python code with the Generic Visualisation Tools and the API wrapper explained in sections 3.1.1 and 3.1.2. On the other hand the module with the Angular front-end (GUI) explained in section 3.1.3.

Each module has its Dockerfile and later inside the “dev-ops” folder there is a docker-compose for the integration of both.

The module containing the Python code with the Generic Visualisation Tools and the API wrapper has been dockerized with the following Dockerfile:

```
# Pull base image
FROM python:3.6
# Set environment varibles
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN pip install --upgrade pip
WORKDIR /code/
# Install dependencies
COPY code/ ./
COPY requirements.txt ./
RUN pip install -r requirements.txt

EXPOSE 8000
CMD ["python", "main.py"]
```

The module with the Angular front-end (GUI) has been dockerized with the following Dockerfile:

```
# Nodejs Base image
FROM node
WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH
# install and app dependencies
COPY pynalia-front/ ./
RUN npm install
RUN npm install -g @angular/cli
# start app
CMD ng serve --host 0.0.0.0
```

To execute this project, we will launch a docker-compose with the following instruction:

docker-compose -f dev-ops/docker-compose.yml up -d –build

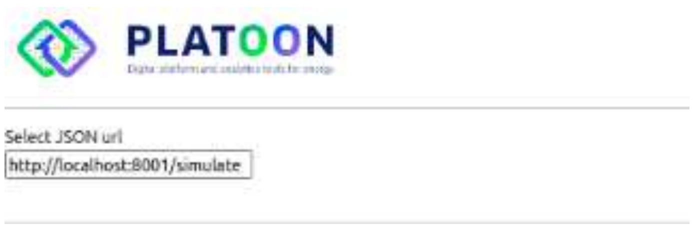
Once this instruction is executed there will have the 2 dockers uploaded in the following paths:

- FastApi --> http://localhost:8000/docs
- Front End --> http://localhost:4200/

3.3 User guide

The target user of the tool are energy experts with high domain knowledge but low coding skills. Thus, an intuitive and simple Graphical User Interface (GUI) has been defined in order to configure and visualize the dashboards. In order to configure the dashboard the following steps should be followed:

1. Type the url of the JSON file containing the input data:

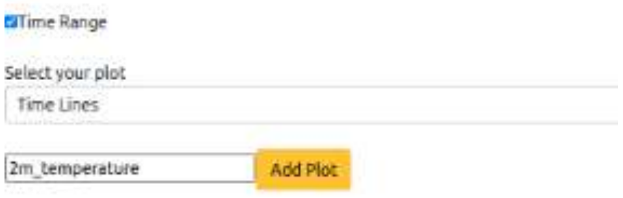


Select JSON url

http://localhost:8001/simulate

Figure 8: Batch-data Visualisation Dashboard – JSON input

2. Select the corresponding plot type (e.g. Time Lines) and type the name of the corresponding variable(s) you want to visualize (e.g. “2m_temperature”). In case you want to add the time range selection functionality click on “Time range”. Finally, click on “Add Plot”.



☑Time Range

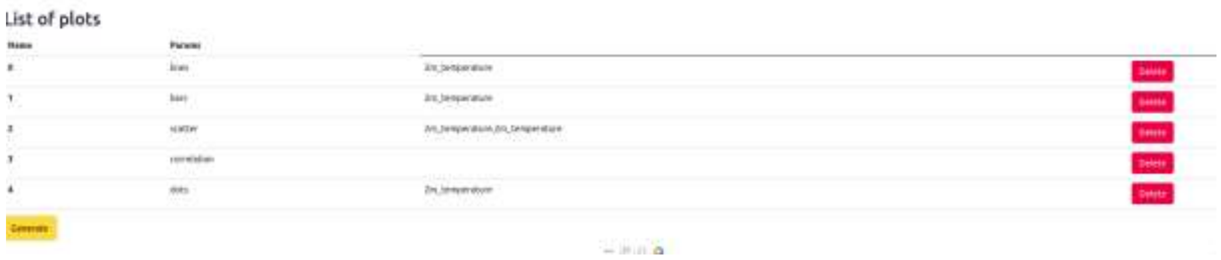
Select your plot

Time Lines

2m_temperature

Figure 9: Batch-data Visualisation Dashboard – Add plot

Once you click on “Add plot” the new plot will be added to the “List of Plots” where you can delete them if necessary.

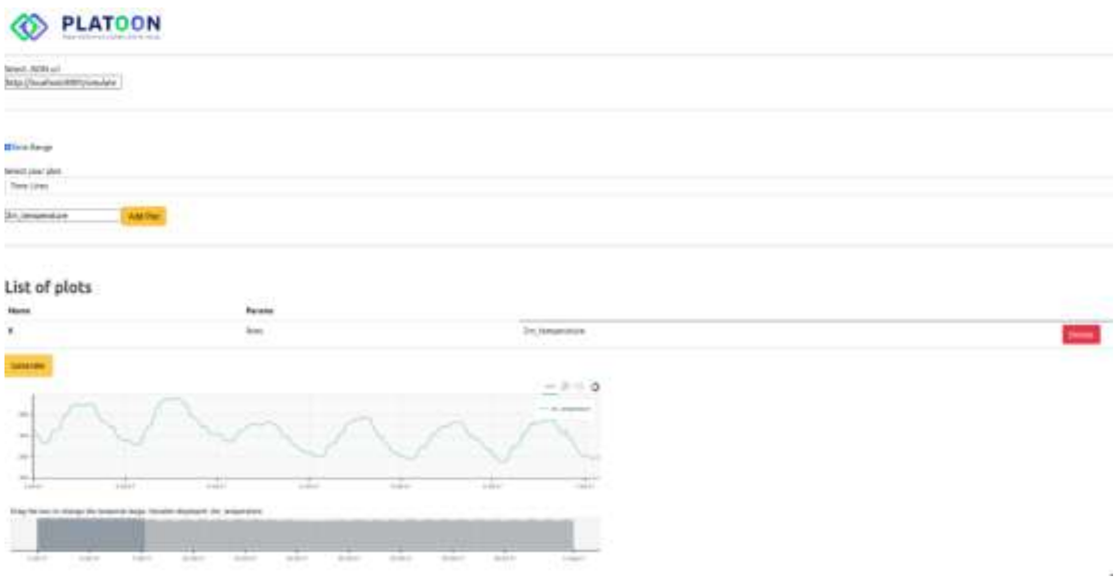


List of plots

Name	Params		
0	lines	2m_temperature	Delete
1	bars	2m_temperature	Delete
2	scatter	2m_temperature,2m_temperature	Delete
3	correlation		Delete
4	dots	2m_temperature	Delete

Figure 10: Batch-data Visualisation Dashboard – List of Plots

3. Once all the plot types have been added, click on generate. This will generate the dashboard with all the selected plots interlinked with each other.



List of plots

Figure 11: Batch-data Visualisation Dashboard – Generated Dashboard

For each of the plots you can use the additional functionalities such as plot, time range, etc.

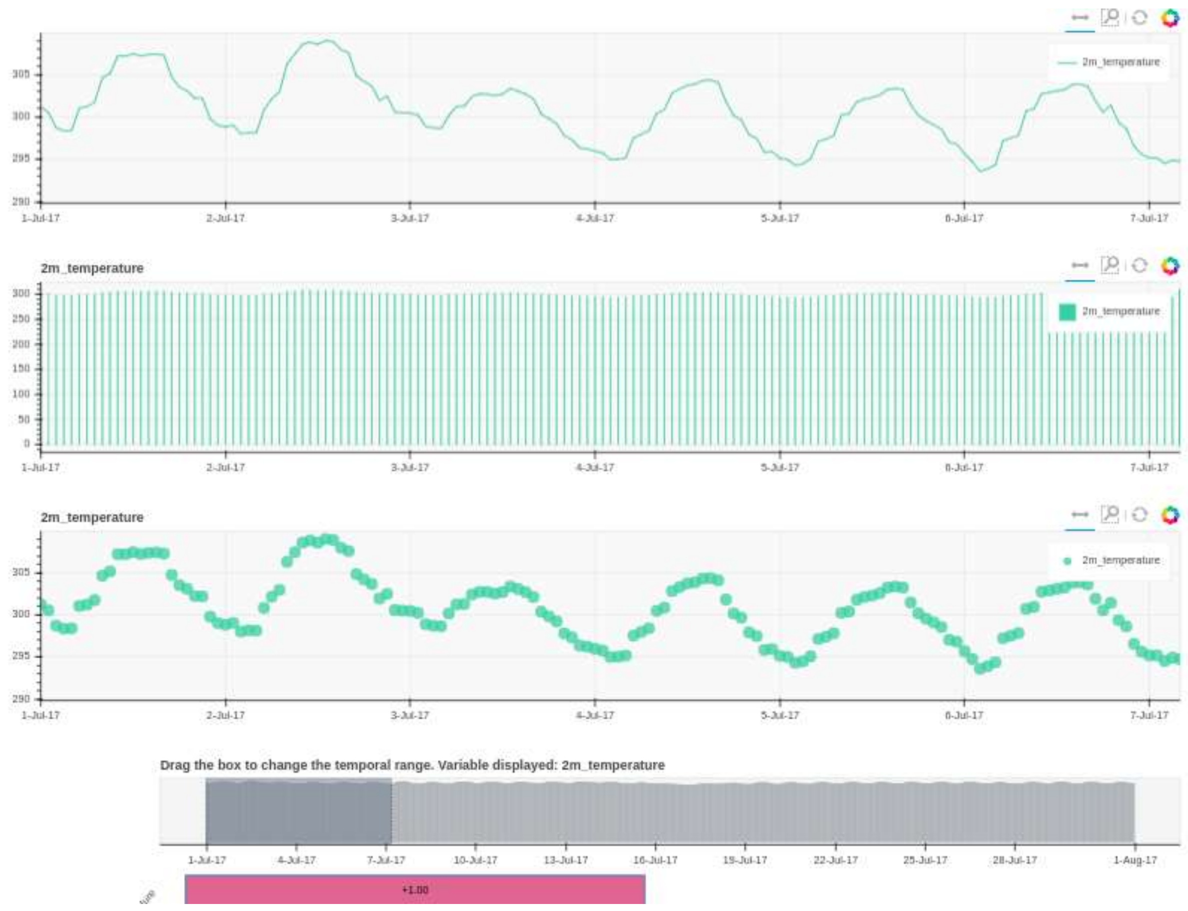


Figure 12: Batch-data Visualisation Dashboard – Generated Plots with additional functionalities

4. Real-time Visualisation Dashboard

A real-time dashboard is a tool that is automatically updated with the most current data available. The visualizations feature a combination of historic data and real-time information that is useful for identifying emerging trends and monitoring efficiency. Real time dashboards usually contain data that is time-sensitive.

In order to monitor the acquisition of data by the edge-node from different data sources, the Chronograf tool is being used. Chronograf is the administrative user interface and visualization and graph engine for the real-time series data InfluxDB.

InfluxDB is a high-performance data store written specifically for time series data. It allows for high throughput ingest, compression and real-time querying. InfluxDB is written entirely in Go and compiles into a single binary with no external dependencies. It provides write and query capabilities with a command-line interface, a built-in HTTP API, a set of client libraries (e.g., Go, Java, and JavaScript)

Chronografh is very easy to use and includes templates and libraries that allow you to quickly create dashboards with real-time views of data and easily create automation rules and alerts. In the following figure you can see how the node is acquiring the active energy measurements of a given meter and read them from the distributor's database.

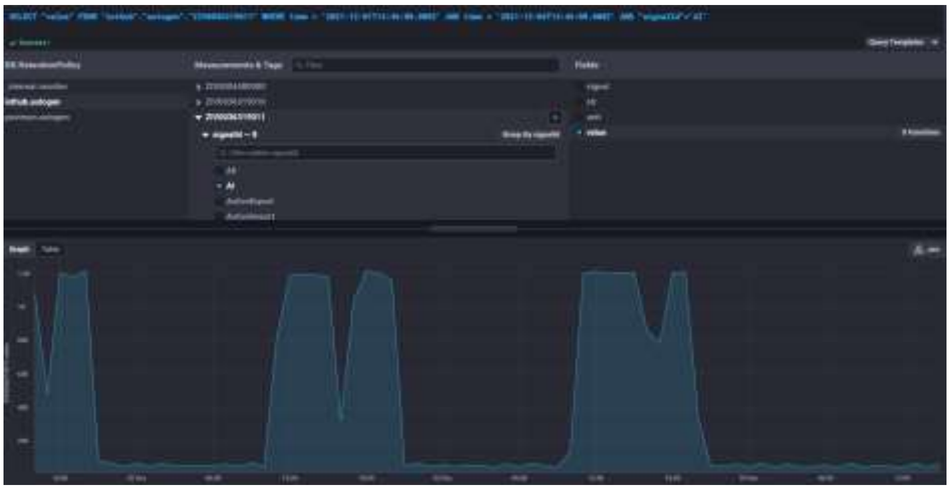


Figure 13: Active Energy for a meter using Chronograf

4.1 Description

The input flow of the information necessary for the execution of the different implemented algorithms and their outputs are integrated into the monitoring portal, which integrates a series of utilities or tools that allow users to view graphics on the screen, representations of data, events and process prompts, as well as historical data, offering a set of standard display panels, screens, and charts.

The tools that make up this monitoring portal are the following:

- Tick Stack Platform[vii]
- Grafana[viii]
- Node Network

TICK Stack is a collection of open source components that combine to provide a platform for easily storing, visualizing, and monitoring time-series data such as measures, metrics and events. Its components are: Telegraf[ix], a server agent for collecting and reporting metrics; InfluxDB[vii], a high-performance time series database; Chronograf[x], a user interface for the platform, the dashboard tool; and Kapacitor[xi], a data processing engine that can process, stream, and pool InfluxDB data.

The heart of the system is the InfluxDB component, which is one of the best time series databases around.

Of these components, the project uses Chronograf and InfluxDB as fundamental elements for the storage and monitoring of information.

As previously detailed, InfluxDB is a time-series-oriented database, and given the computing capabilities of the Edge Node are limited, its use is highly recommended since it shows good performance when working with sensors and for data analysis. It has an SQL-like query language, it is compatible with JSON and it is specifically designed for metrics and events, the type of information that the Edge Node will process. InfluxDB can handle millions of data points per second. Working with that much data over a long period can lead to storage concerns. InfluxDB automatically compacts data to minimize your storage space. In addition, you can easily downsample the data; keeping high-precision raw data for a limited time and storing the lower-precision, summarized data for much longer or until the end of time. InfluxDB has two features that help to automate the downsampling and data expiration processes, namely, Continuous Queries and Retention Policies.

As the computing capacity of the Edge Node is usually very limited, it is recommended to use InfluxDB as a database: it shows good performance when working with sensors and for data analysis (R and Python language support), it has a SQL-like query, is JSON compliant, and is designed specifically for metrics and events, which are the type of information that the Edge Node will process.

In cases where visualization via “Chronograf” is insufficient, “Grafana” is used.

Grafana is an open source platform for analysis and monitoring created by Grafana labs. It can be connected to multiple databases, and it displays the information in dashboards that include a multitude of plugins and different applications that can connect to the system.

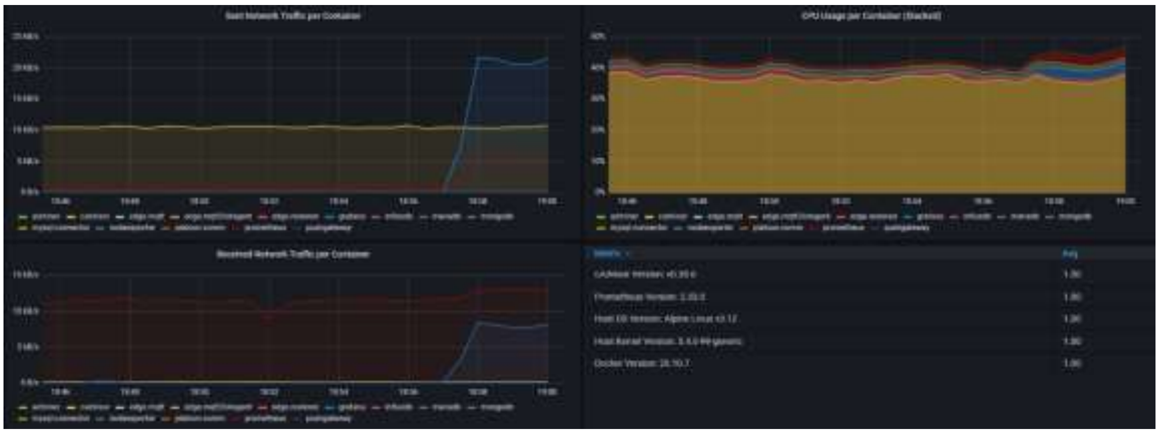


Figure 14: Traffic and CPU General Usage Dashboard

Grafana supports different data sources out of the box. Among them: Graphite[xii], Prometheus[xiii], ElasticSearch[xiv], InfluxDB, PostgreSQL[xv], MySQL[xvi], CloudWatch[xvii], Azure Monitor[xviii] and others.

It can collect data from the system it runs on, such as disk usage, RAM, CPU, system load, connections, and many more, and also includes a growing list of input plugins, such as apache, consul[xix], couchDB[xx], Docker[xxi], Elasticsearch [xxii], Fluentd[xxiii], HAproxy[xxiv], http POST, among others. Its output is usually sent to a database such as InfluxDB.

Apart from data Grafana can be used as a dashboard to visualize the main metrics of the deployed nodes. The following figures show the metrics related to the use of the node CPU, the CPU use of each of the images downloaded and in operation, as well as the memory in use of each of the images.

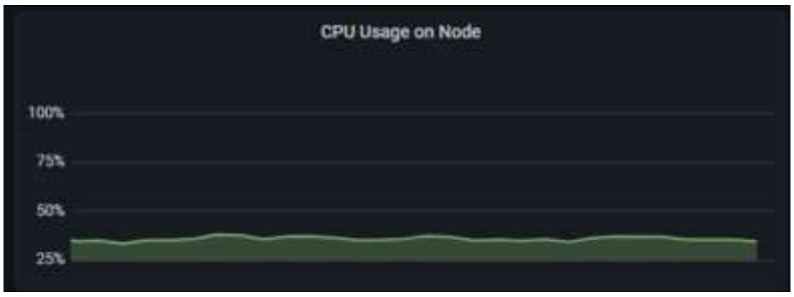


Figure 15: CPU Usage Dashboard

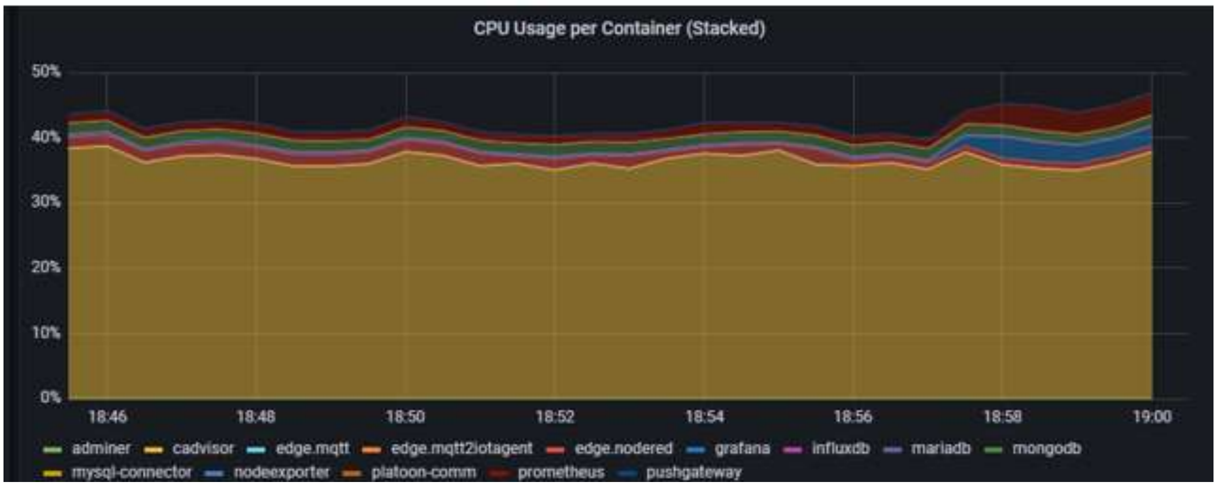
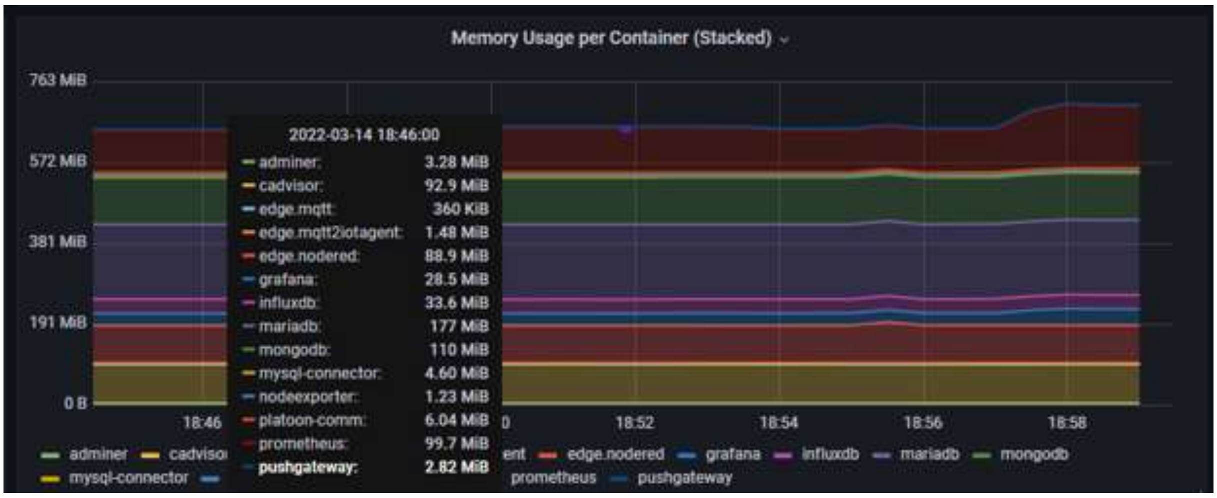


Figure 16: CPU usage per Container



```
2022-03-14 18:46:00
adminer:            3.28 MiB
cadvisor:           92.9 MiB
edge.mqtt:          360 KiB
edge.mqtt2iotagent: 1.48 MiB
edge.nodered:       88.9 MiB
grafana:            28.5 MiB
influxdb:           33.6 MiB
mariadb:            177 MiB
mongodb:            110 MiB
mysql-connector:    4.60 MiB
nodeexporter:       1.23 MiB
platoon-comm:       6.04 MiB
prometheus:         99.7 MiB
pushgateway:        2.82 MiB
```

Figure 17: Memory usage per Container

Node-RED[xxv], initially developed by IBM and currently part of the JS Foundation, is a stream-based programming tool. It is a free and open source tool. In Node-RED, applications are described as a set of black boxes (nodes) connected to each other (creating a flow). Each node receives data, works with it, and then passes the result to the next node. One of the main advantages of Node-RED is that it can run on low-cost hardware (eg Raspberry Pi) and in the cloud, and is designed to be used by a wider range of users (not just professionals). However, the functions must be written in JavaScript, which can be a barrier for less advanced users.Node-RED is then used to check the correct functioning of the application and to store the results in the InfluxDB database.

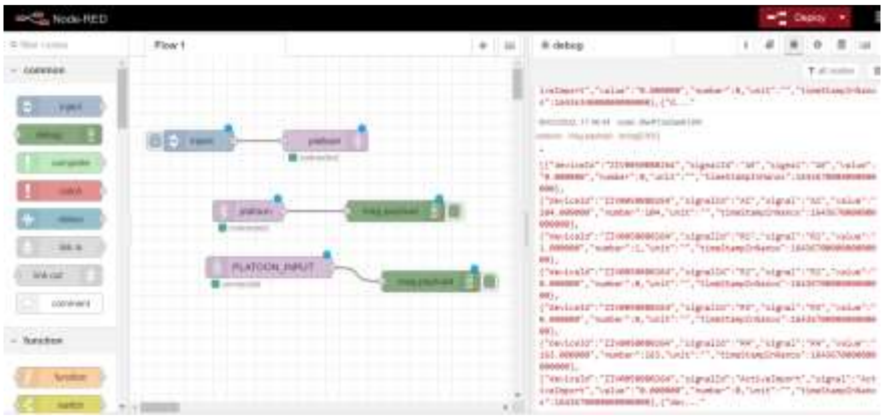


Figure 18: Node Red Test Dashboard

Access to these tools is done through the different enabled ports, which are the following:

Port 5300 : Chronograf (display of the results of the different containers)

Port 3000: Grafana to visualize the different metrics (CPU, Memory) in the edge node.

Port 1880 : Node-Red (for testing purposes)

All these “open source” tools are downloaded as containers and stored in the docker-registry (image registry) of the edge node management platform.

In the case of Pilot 2b, the platform “IoTHub02.Onesaitplatform” is being used, Onesait Platform is a web application running on a server (local or in the cloud) that allows remote management of all edge devices deployed in the field. The goal is to have two-way communication with edge devices without the need for physical presence. It also aims to make the edge device provisioning, commissioning, and maintenance processes easier and faster. Last but not least, it allows the user to manage devices as a group or individually, having control over the content of the devices, their updates and the environments / projects to which they belong.

Once these images are uploaded to the platform's registry, they can be easily downloaded to the edge-node as detailed below.

4.2 Configuration

Normally the open-source docker images are extracted from a docker hub (public image repository). For this project, in order to facilitate their use in the different pilots, a docker-registry service has been deployed where images can be downloaded in an restricted environment, this is the case of IoTHub02, where the platform provides access to images through a docker-registry that serves as a repository for images to be downloaded.

The IoTHub02 platform deployed for the project provides a Docker Registry repository, where all Docker images used by edge devices are stored, enabling secure, fast and flexible local deployment of Docker Containers.The following subsections contain the details regarding the software/hardware prerequisites and configuration instructions for the developed dashboard.

4.2.1 Software and Hardware Prerequisites

Downloading containers from a registry is the normal mechanism used by all container architectures. It is not necessary to connect to the node to download the image, this task is carried out automatically by the agent. Image registries are systems designed to scale massively.

The hardware prerequisites that a machine must have are mainly the following:

- 64-bit architecture (preferably Intel for integration with cybersecurity).
- Linux operating system (preferably from the Red Hat or Ubuntu family).
- The machine must have a TPM for local signature and certificate storage

To proceed with the deployment of the different containers of a given application, the only condition is to install an edge agent on the node. The Agent is in charge of validating the presence or not of a container orchestrator (docker-compose or podman) or pods (k8s versions) and managing the commands deployed by the management IoT platform (establishing and maintaining communication with that IoT platform).

4.2.2 Configuration Instructions

Image settings are defined in docker-compose.yml file, Compose is a tool for defining and running multi-container Docker applications, with Compose, a YAML file is used to configure the different services of your application, then, with a single command, all services are created and started from that configuration.

Using Compose is basically a three-step process:

1. Define the app's environment with a Dockerfile so it can be reproduced anywhere. To do this, the Chronograf image needed for the implementation of the dashboards is downloaded from its official repository to a Git repository, where it is built. a new image according to the environment variables and necessary configurations according to the project where they are going to be executed.
2. Once the new image is built, it is deployed in a Project docker registry defining the associated docker-compose.yml file defining also the services that make up the application so they can be run together in an isolated environment.
3. Lastly run docker compose up and the Docker compose command starts and runs your entire app.

The procedure followed to have the different open-sources in the project and in particular Chronograf (InfluxDB) as a dashboard tool has been as follows:

1.- Download the image from its original repository (e.g https://hub.docker.com/_/chronograf)

2.- To build the customized application, through the use of a Dockerfile (Dockerfile is simply a text-based script of instructions that is used to create a container image). Deploy that new image in a GIT registry.

3.- Deploy the image in the node, this action can be done in two different ways:

a. Directly via a download action from the Git Registry
b. Through the use of an IoT management platform, as previously detailed in the IoTHub02 platform (https://iothub02.onesaitplatform.com/) project. The use of this platform also allows the management of the basic operation agent (mentioned before), the definition of the docker-compose,yml file for the docker operation together with the rest of the services, as well as the visualization of the entire operation environment.

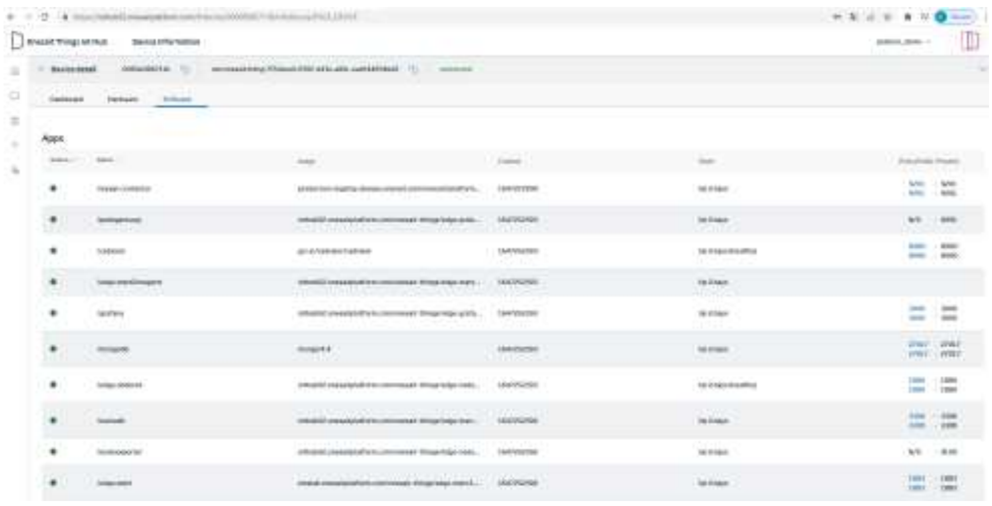


Figure 19: Docker Registry in the IoT platform

Download the image from the platform to the edge-node through the commands:

- docker login iothub02.onesaitplatform.com/ -u platoon_demo -p
  and
- docker pull iothub02.onesaitplatform.com/onesait-things/edge-influxdb:1.0.0.RELEASE

The image is now available on the node for execution.

4.-To execute this project as a whole we will launch a docker-compose file with the following instruction:

docker-compose -f dev-ops/docker-compose.yml up -d –build

Once this instruction is executed, the docker will be loaded in the following route, accessing it the different dashboards that are required can be defined.:

- Chronograf: http://NodeIP:5300/

The docker compose we have used to download the image is the following (partially only for InfluxDB (Chronograf)):

```
influxdb:
container_name: influxdb
image: iothub02.onesaitplatform.com/onesait-things/edge-influxdb:1.0.0.RELEASE
environment:
        - INFLUXDB_DB=demo
        - INFLUXDB_ADMIN_USER=admin
        - INFLUXDB_ADMIN_PASSWORD=password
        volumes:
        - ./influxdb:/var/lib/influxdb
        ports:
        - 8086:8086
        networks:
        - edgenet
        driver: bridge
```

4.3 User guide

As detailed above, Chronograf offers a complete dashboard solution for visualizing all the data stored in the InfluxDB data base, the basic instructions for building a visualization dashboard are as follows:

1. Create a new dashboard
   Click Dashboards in the navigation bar and then click the Create Dashboard button. A new dashboard is created and ready to begin adding cells.

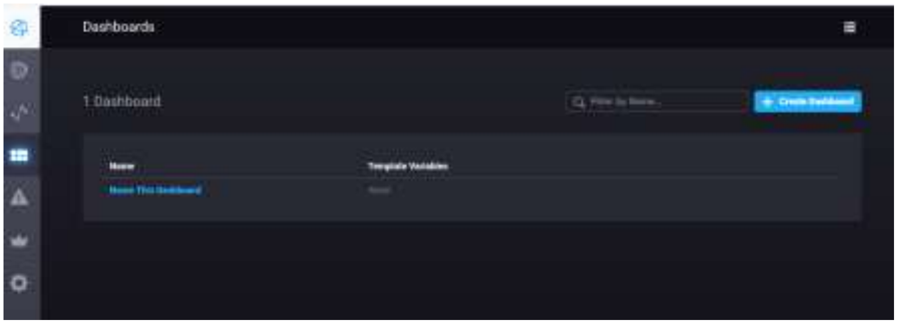
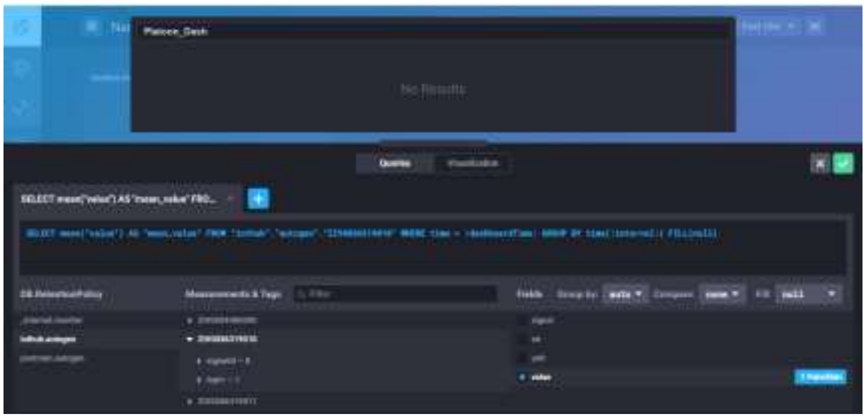


Figure 20: Create a Dashboard

2. Name your dashboard
   Click Name This Dashboard and type a new name. For example, "PlatoonDash".

3. Enter cell editor mode
   In the first cell, titled "Untitled Cell", click Add Data to open the cell editor mode



```
SELECT mean("value") AS "mean_value" FROM "[illegible]"."autogen"."[illegible]" WHERE time > :dashboardTime: GROUP BY time(:interval:) FILL(null)
```

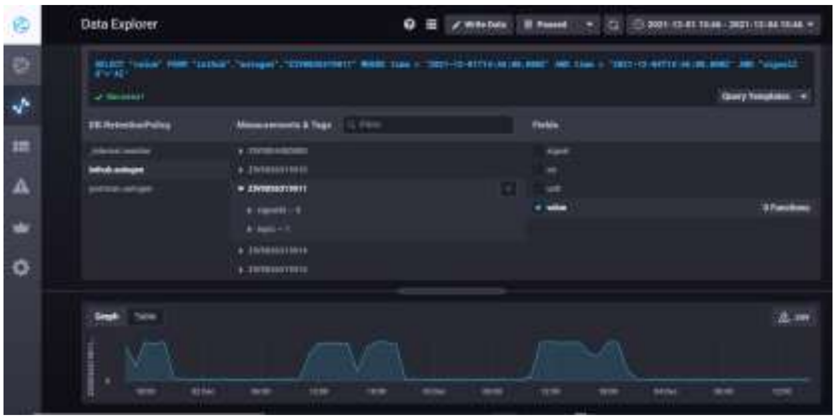
Figure 21: Select data for the Dashboard

4. Create your query

   Click the Add a Query button to create an InfluxQL query. In query editor mode, use the builder to select from your existing data and allow Chronograf to format the query for you. Alternatively, manually enter and edit a query. Chronograf allows you to move seamlessly between using the builder and manually editing the query; when possible, the interface automatically populates the builder with the information from your raw query.

   For our example, the query builder is used to generate a query that shows the average idle CPU usage grouped by host (in this case, there are three hosts). By default, Chronograf applies the MEAN() function to the data, groups averages into auto-generated time intervals (:interval:), and shows data for the past hour (:dashboardTime:). Those defaults are configurable using the query builder or by manually editing the query.

   In addition, the time range (:dashboardTime: and :upperDashboardTime:) are configurable on the dashboard.



```
[illegible]
```

Figure 22: Dashboard result

5 Visual Analytics Pipelines and Visualisation Templates

This section contains a description of the different visual analytics pipelines and visualisation templates implemented for each of the scenarios defined in WP5, namely, non-edge (batch) scenario and edge (real-time) scenario.

5.1 Non-Edge Scenario

Figure 23 shows the visual analytics pipeline for the non-edge (batch) scenario. In this case, historical batch data is ingested from a static csv file. Then, this data is semantified using the semantic adapter and then integrated with data from several sources using the federated query engine (explained in D5.2 and D5.3). Afterwards, SANSA retrieves the data from the federated query engine and processes the data (explained in D5.2). Finally, the results generated by SANSA are visualized in the Batch-data Visualisation Dashboard explained in section 3 of this document.

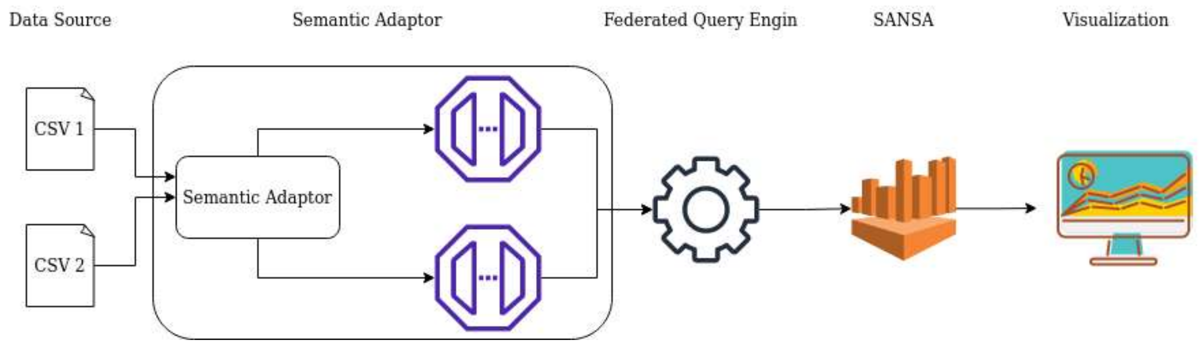


Figure 23: Visual Analytics Pipeline - Non-edge (batch) scenario

For the integration of the data analytics tool (SANSA) and the visualization dashboard we have used an API which SANSA provides to retrieve the analytics result (/api/getRegressionResult).

Regarding the visualization templates in this scenario we used the TimeLine plot. The following figure shows the visualization dashboard.

Figure 24: Visualisation Template - Non-edge (batch) scenario

For more information about the integration and test of all the components please visit D5.1 and D5.5.

5.2 Edge Scenario

Figure 24 shows the visual analytics pipeline for the edge (real-time) scenario (elaborated in Pilot 2a). In this case, dynamic real-time data is ingested from edge computing devices. Then, this data is processed at the edge in a containarised (dockerized) data analytics tool implemented in the edge (edge service). The processed data is saved in a database (Influx DB) developed particularly for storing and retrieving time-series data. Finally, the processed data is visualized in the real-time dashboard based on Grafana explained in section 4.

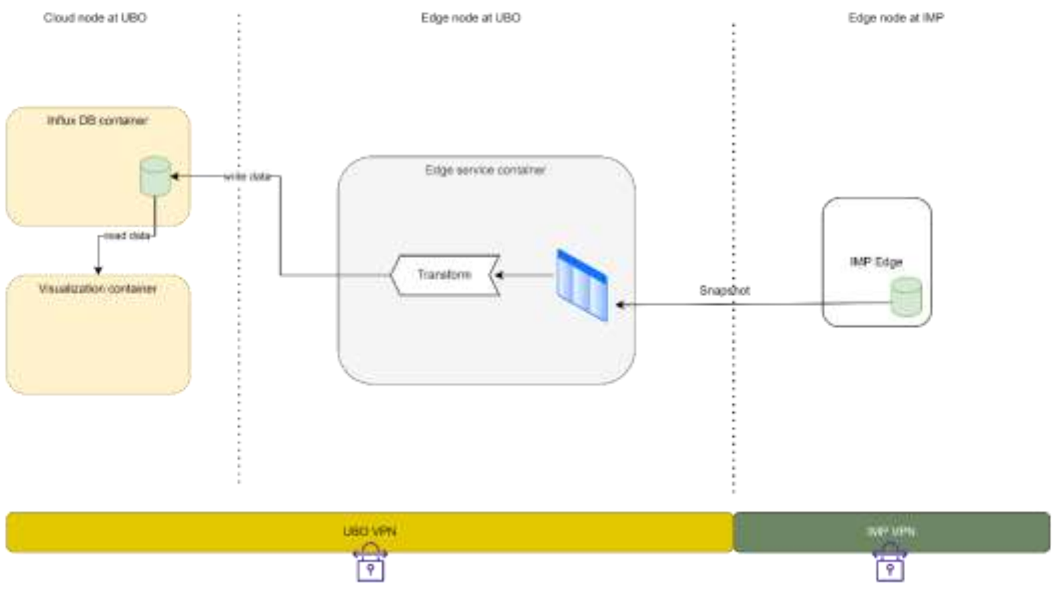
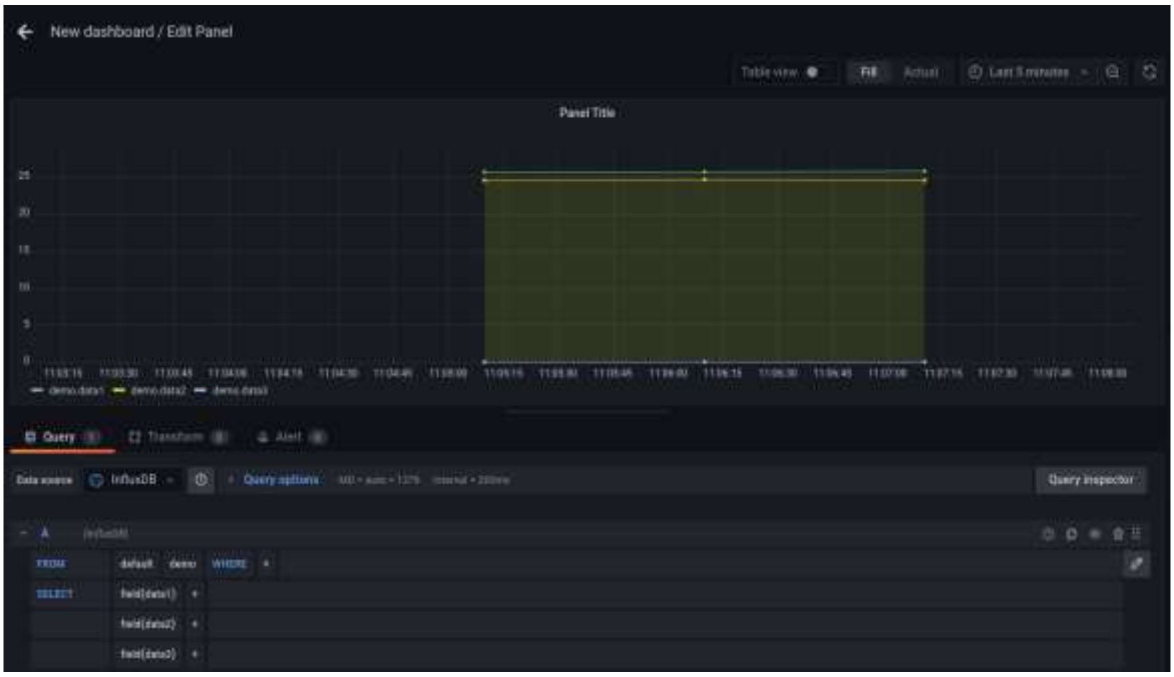


Figure 25: Visual Analytics Pipeline - Non-edge (batch) scenario

For the integration of the Influx-DB with Grafana an already existing connector available in SwarmPit was used (see D5.1 for more details).

Regarding the visualization templates in this scenario we used a time series plot. The following figure shows the visualization dashboard.



```
A  (InfluxDB)
FROM    default  demo  WHERE  +
SELECT  field(data1)  +
        field(data2)  +
        field(data3)  +
```

Figure 26: Visualisation Template - Edge (real-time) scenario

For more information about the integration and test of all the components please visit D5.1 and D5.5.

6 Conclusions

In this task two open source alternatives of data analytics dashboards have been developed and implemented: 1) Batch-data Visualisation Dashboard. 2) Real-time Visualisation Dashboard. Each of the alternatives have their own advantages and disadvantages and are suited to each of the two scenarios defined in deliverable D5.1.

On the one hand, the Batch-data Visualisation Dashboard has been implemented and tested for the non-edge scenario. The Batch visualisation dashboard is bult on Python and Angular based on the Generic Visualisation Toolbox developed in task T4.6. The target user of the tool are energy experts with high domain knowledge but low coding skills. Therefore, the main focus has been to define an intuitive and simple Graphical Energy Interface that almost anyone can easily use. Also, all the underlying source code is publicly available in the PLATOON GitHub Repository. Nevertheless, due this simplicity cannot be used to deal with real-time data and the number of type of charts and functionalities are limited.

On the other hand, the Real-time Visualisation Dashboard has been implemented and validated for the edge (real-time) scenario based on existing open source solutions such as Grafana and Chronograf. Unlike the batch dashboard, this solution allows users to view graphics on the screen, representations of real time data, events and process prompts, as well as historical data, offering a set of standard display panels, screens, and charts. Nevertheless, some of these advanced functionalities are more difficult to use energy experts with but low coding skills.

References

[i] The pandas development team, "pandas," 2021. [Online]. Available: https://pandas.pydata.org. [Accessed 17 December 2021].

[ii] Bokeh contributors, "The Bokeh Visualization Library," 2021. [Online]. Available: https://bokeh.org. [Accessed 17 December 2021].

[iii] https://fastapi.tiangolo.com/

[iv] https://angular.io/

[v] https://www.typescriptlang.org/

[vi] https://angular.io/guide/browser-support

[vii] https://www.influxdata.com/time-series-platform/

[viii] https://grafana.com/

[ix] https://www.influxdata.com/time-series-platform/telegraf/

[x] https://www.influxdata.com/time-series-platform/chronograf/

[xi] https://www.influxdata.com/time-series-platform/kapacitor/

[xii] https://graphiteapp.org/

[xiii] https://prometheus.io/

[xiv] https://www.elastic.co/es/what-is/elasticsearch

[xv] https://www.postgresql.org/

[xvi] https://www.mysql.com/

[xvii] https://aws.amazon.com/es/cloudwatch/

[xviii] https://azure.microsoft.com/es-es/services/monitor/

[xix] https://camel.apache.org/components/3.15.x/consul-component.html

[xx] https://couchdb.apache.org/

[xxi] https://www.docker.com/

[xxii] https://www.elastic.co/es/what-is/elasticsearch

[xxiii] https://www.fluentd.org/

[xxiv] https://www.haproxy.com/

[xxv] https://nodered.org/