Grant Agreement N° 872592

PLATOON

Digital platform and analytic tools for energy

Deliverable D5.1
Reference Platform Instantiation

Contractual delivery date:

M27

Actual delivery date:

31/03/2022

Responsible partner:

P03: UBO, GERMANY

Project Title	PLATOON – Digital platform and analytic tools for energy
Deliverable number	D5.1
Deliverable title	Reference Platform Instantiation
Author(s):	UBO, ENGIE, TECN, IAIS, ENG, IMP, GIR, SAM, TIB, PDM, PI, MINSAIT
Responsible Partner:	UBO (Partner nº3)
Date:	18/03/2022
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:	The aim of this task is to create a reference platform instantiation of the PLATOON reference architecture defined in D2.1 (V1) and D2.5 (V2) using the Open-Source Components developed in WP2, WP3 and WP4. The PLATOON reference architecture specifies the main building blocks for the interoperability between heterogeneous digital platforms and analytics tools for energy that are based on different technologies. The reference instantiation conducted in this task serves as a proof-of-concept for the PLATOON reference architecture and also as a testbed for doing an integration test of the different open-source components explained in D5.5. Software solutions such as Docker and Swarmpit are discussed and how they assist system administrators with the deployment and support of the different components. In addition, implementation details of the platform instantiation are discussed in detail.
Keyword List:	Reference Platform, Open-Source Instantiation, Implementation, Docker, Swarmpit
Reviewer(s):	Martino Maggio (ENG) Erik Maqueda (TECN) Philippe Calvez (ENGIE)
Approved by:	
Recommended/mandatory readers:	All WP5 partners. WP4 and WP6 partners for reference for implementation and testing. WP8 and WP9 partners for exploitation and dissemination.

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	28/01/2022	1st version of TOC	UBO
v0.2	11/03/2022	Semantic Adapters and FQP	TIB
v0.3	18/03/2022	First finished version for internal review	All
v0.4	19/03/2022	Final version for internal Review	ENG, UBO
v0.5	25/03/2022	Correction based on internal review	UBO

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
FQP	Federated Query Processing
GA	Grant Agreement
GAM	General Assembly Meeting
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

This task provides a reference platform instantiation of the PLATOON reference architecture defined in D2.1 (V1) and D2.5 (V2) using the Open-Source Components developed in WP2, WP3 and WP4. The reference platform instantiation serves as a proof-of-concept for the PLATOON reference architecture and also as a testbed for doing an integration test of the different open-source components explained in D5.5.

This deliverable will briefly discuss the different components of the PLATOON reference architecture. The deliverables D5.2, D5.3, and D5.4 will focus on different components in a more detail manner.

This deliverable briefly introduces the 3 different testing scenarios that will be conducted in deliverable D5.5. Also, it discusses the server hardware used to run the different components of the reference architecture.

On a software level, we introduce and discuss Docker, Docker Compose, Docker Swarm, and Swarmpit, which are integral parts of the deployment and support of the different PLATOON software components.

The deliverable provides a step-by-step installation guide to enable system administrators to setup and run the PLATOON reference architecture. For this, we provide detailed instructions on how to install and configure Swarmpit. As an example, the installation and configuration instructions of the simplified version of the Edge scenario are described. More details about the deployment of individual components for the corresponding testing scenario can be found in D5.5.

The resulting reference instantiation is open source. Hence, using the information provided in this deliverable, interested companies can create a base setup of the PLATOON reference architecture, ready for deployment of different building blocks and data analytics tools for different use-cases. The reference platform instantiation discussed in this deliverable serves as an essential first step for a successful integration with the PLATOON ecosystem.

1 Introduction

The goal of this task is to provide a reference platform instantiation of the PLATOON reference architecture. The PLATOON reference architecture, as defined in deliverable D2.1(V1) and D2.5(V2), specifies the main building blocks, interfaces, interactions, and capabilities to enable the interoperability between heterogeneous digital platforms and analytic tools in the energy domain. The reference platform instantiation serves as a proof-of-concept for the PLATOON reference architecture as well as a testbed for integration test of the different open-source components explained in D5.5.

The PLATOON reference architecture consists of multiple components which interact together to allow a seamless integration amongst heterogenous digital platforms that are implemented using specific technologies to meet the requirements of the different large-scale pilots. The different components will be discussed briefly in the next section, where an overview of the reference architecture will be given. In this deliverable a high-level description of some open-source components are provided that can be used for an open-source reference instantiation. The deliverables D5.2, D5.3, and D5.4 will focus on the respective components in more detail. Finally, D5.5 includes the installation and configuration instructions of individual components for the corresponding integration testing scenarios inspired by the energy domain. These integration-tested components will form the basis for the reference platform instantiation in this deliverable.

For the reference platform instantiation, we use modern, enterprise-level server hardware to run the different components of the PLATOON reference architecture. The chosen hardware ensures that the reference platform instantiation and the tests performed on it are representative of typical hardware scenarios present in the energy domain. In this deliverable, the details of the hardware setup used throughout Working Package 5 will be discussed.

Due to the complexity of the PLATOON reference architecture and the many different components involved, management and surveillance of the components can be a daunting task. Therefore, we utilize the software solutions Docker and Swarmpit to assist system administrators with the deployment and support of PLATOON software components. Docker is a virtualization engine which allows developers to run software in so called containers. Each component of the PLATOON reference architecture may consist of one or several such docker containers. Swarmpit operates on top of Docker to facilitate the management of all the different docker containers from the different components in the PLATOON reference architecture. For this, Swarmpit provides an interface that visualizes the different containers and helps in managing them. In this deliverable, we will discuss both Docker and Swarmpit and show their important role in instantiating the PLATOON reference architecture.

In the next section, we start discussing the reference architecture and its components. In Section 4, the hardware specification of the server used for the reference architecture instantiation are presented. Next, in Section 5, we go over the implementation details and discuss Docker and Swarmpit in detail. We conclude the deliverable in Section 6.

2 Reference Architecture

In Work Package 5 (WP5) the PLATOON Reference Architecture is instantiated using the open-source components developed in WP2, WP3 and WP4. This makes it possible to test the interaction of the versatile and technically complex components with each other shown in the figure below. The pipelines described in T.5.1, T.5.2, T.5.3, and T.5.4 are possible stacked solutions that use the developed technologies.

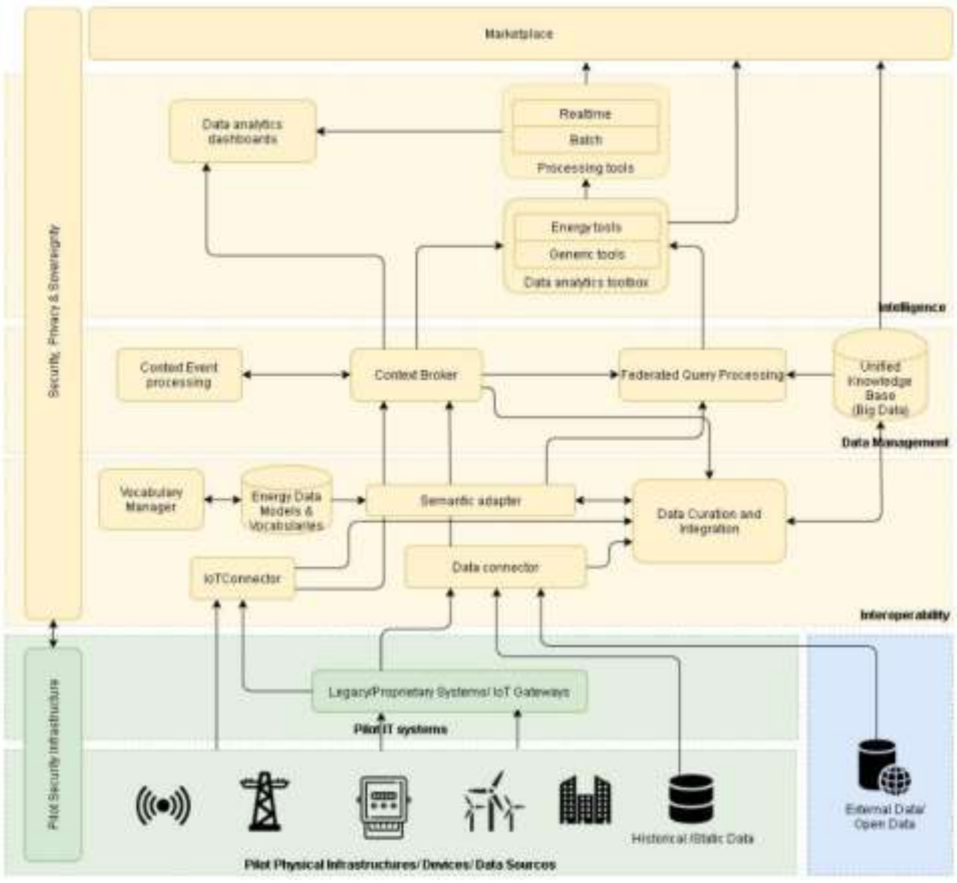

Marketplace
Security, Privacy & Sovereignty
Data analytics dashboards
Realtime
Batch
Processing tools
Energy tools
Generic tools
Data analytics toolbox
Intelligence
Context Event processing
Context Broker
Federated Query Processing
Unified Knowledge Base (Big Data)
Data Management
Vocabulary Manager
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

Apart from the Reference Architecture and the developed open-source components, the reference platform developed in this task makes use of the common APIs, data models and dockers specifications defined in deliverables D2.2/D2.6, D2.3/D2.7 and D4.1, respectively. In addition, the reference instantiation complies with the data security, privacy and sovereignty layer defined in WP3.

In the reference instantiation explained in this task, the different components are containerized and communicated with each other through REST API.

Regarding the semantic pipeline, the data is converted into Knowledge Graph data by the Semantic Adapter. These Knowledge Graphs can be used with the help of Federated Query Processing Engines. Relevant data can be retrieved through common SPARQL queries. This Knowledge Graph data can also be used by data analytics components, which in turn can be visualized by the data analytics dashboard. The semantic annotations are defined using the Semantic Data model from task T.2.3. The semantic pipeline is explained in more detail in deliverables D5.2 and D5.3.

Regarding the data security, privacy and sovereignty layer, the open-source IDS connector, Metadata registry and IDS vocabulary provider are used to ensure that confidential data exchange takes place. More details of the data exchange approaches can be found in the deliverables of WP3.

Regarding the intelligence layer, open-source data analytics tools and visualisation dashboards are implemented. Both are implemented for real-time (edge) and batch (non-edge) scenarios. More details about the Data Analytic Toolbox and the Analytic Dashboards are provided in deliverables D5.2 and D5.4.

2.1 IOT Connectors

As detailed in the deliverable D2.2 - PLATOON-WP2-D2.2-Open API Specifications - the IoT Connector is in charge of transmitting the raw data coming from the device to the virtual entity representation at the Data Management Layer; sending commands or actions request to an actuator device; mapping of device and its features to a virtual entity with related attributes and metadata. IoT Connectors should also be able to deal with security aspects (authentication and authorization of the channel) and data sovereignty (IDS) aspects.

In particular the IoT Connector module is responsible of some main tasks:

- The IoT Connector must know the state of each device managed by it. It is also responsible of verifying the health of the connection to the device. If the connection with some device is lost, the IoT Connector should be able to generate alarms for other components in the system.
- IoT Connector must store and manage metadata about the device that could be used to enrich the data obtained from the device, for example, localization and device characteristics as measure precision.
- Connector must manage the security of the communication with the devices. This mainly focused on the authentication of devices and authorization to manage which devices can connect with which further data processors in the system. But this security has to be also managed from the device perspective, that is, the device needs to know that it is sending data to the right IoT Connector within a secure channel, for example, avoiding man in the middle attacks.
- IoT Connectors must be able to send commands to devices with such a capacity. That is, the IoT Connector will act as a gateway for the rest of the component in the architecture avoiding the complexity of knowing low level protocols. It will translate data from the semantic adapters to the specific protocols used by the supported devices.
- IoT Connector must be able to collect data from the devices, transform that data to the PLATOON data model defined in D2.3. In this way, data will be able to be used in other modules of the architecture without the need to know the internal data model of each device. Actually, this transformation rules should be easy to develop and deploy. New devices or changes in the firmware of existing devices will add the need of deploy new data transformation rules with the minimum impact for the whole system or other device connections. The collection of data must include de capacity to define data verification rules that ensure that the data received have a minimum of quality.

2.2 Semantic Adapters and Federated Querying

The knowledge base creation pipeline presented in Figure 2 integrates a semantic adapter and a federated query processing; it follows the PLATOON reference architecture presented in Figure 1. This pipeline is agonistic of the query engine used to implement the semantic connector and the federated query processing engine. The semantic adapter converts data sources in different formats, e.g., CSV, JSON or relational databases, into RDF; this conversion is guided by mapping rules specified in a mapping language, e.g., SPARQL-Generate or RDF

Mapping Language (RML). These rules define declaratively concepts (i.e., classes and properties) in the PLATOON semantic data models (presented in D2.3) in terms of data collected from the PLATOON data sources. As a result, during the execution of these rules, the semantic adapter populates classes and properties in a unified knowledge base. Then, the pipeline uploads this unified knowledge base into a federation of SPARQL endpoints (e.g., in Virtuoso or Fuseki). The federated query processing engine provides an interface to these endpoints; it enables users to explore the knowledge graph and retrieve the answers to a SPARQL query. Two implementations of the pipeline are available; one resorts to SDM-RDFizer and the other one to SPARQL-Generate. SDM-RDFizer alternative is further explained in the following subsection and the corresponding integration tests are reported as part of D5.5. SPARQL-Generate alternative is covered as part of D5.2.

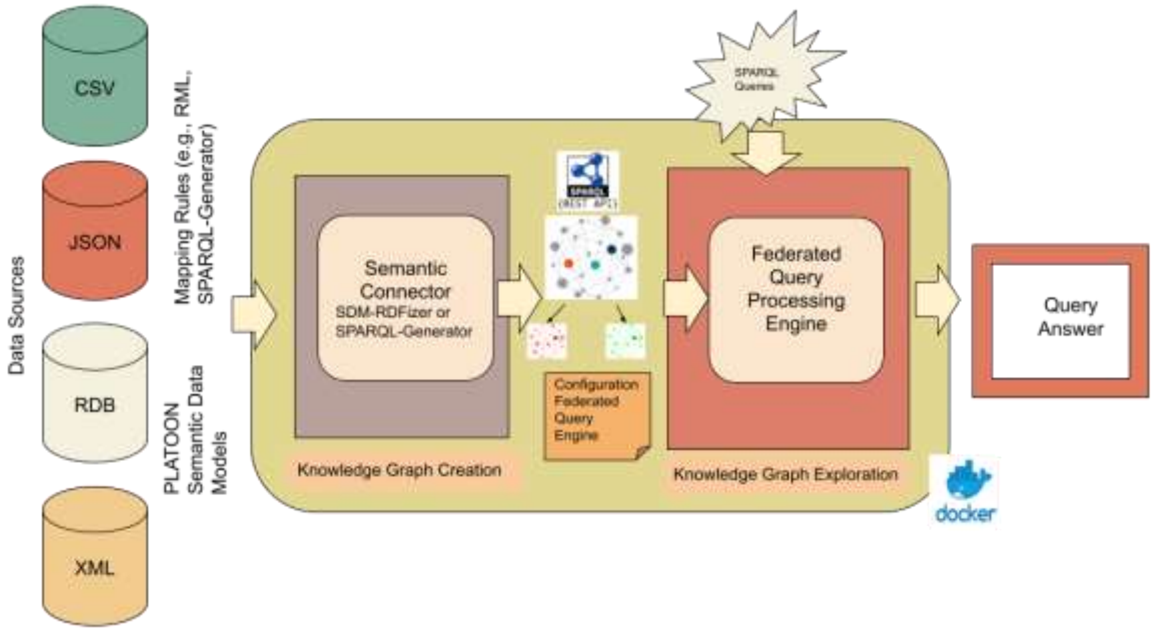
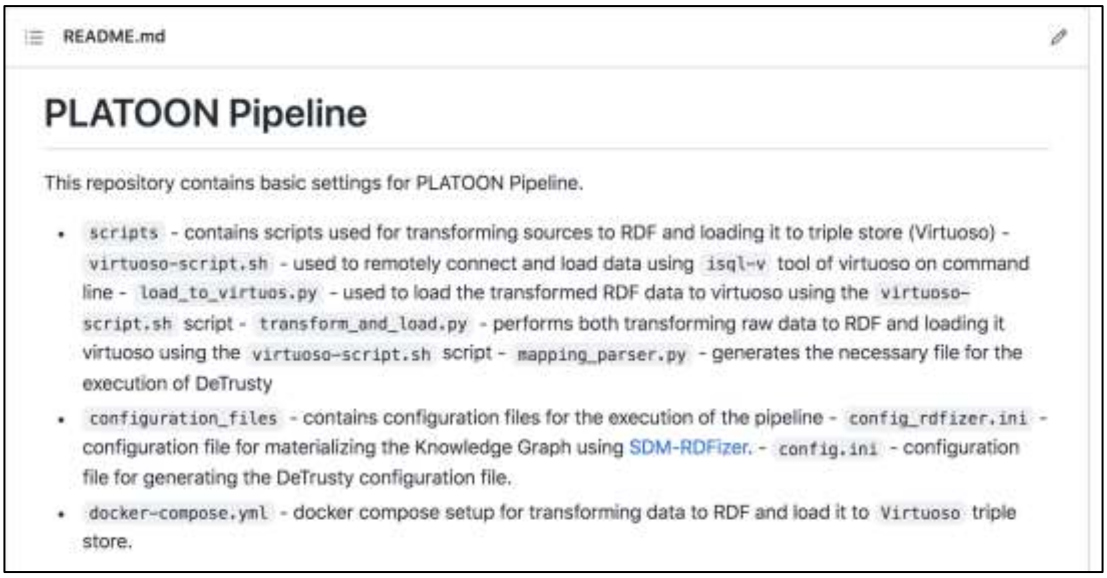

Data Sources
CSV
JSON
RDB
XML
Mapping Rules (e.g., RML, SPARQL-Generator)
PLATOON Semantic Data Models
Semantic Connector
SDM-RDFizer or SPARQL-Generator
Knowledge Graph Creation
SPARQL (REST API)
Configuration Federated Query Engine
SPARQL Queries
Federated Query Processing Engine
Knowledge Graph Exploration
docker
Query Answer

Figure 2: Knowledge Graph Creation Pipeline

2.2.1 A Semantic Adapter based on SDM-RDFizer

The pipeline is implemented as a bash script that executes a series of Docker images, each one implements a component of the pipeline. As a result, the pipeline is comprised of three Docker images: a) the SDM-RDFizer image; b) the Virtuoso images; and c) the FQP image. This pipeline is available in a GitHub repository[1] and uses dockerized components.

[1] https://github.com/SDM-TIB/PLATOONPipeline


README.md

PLATOON Pipeline

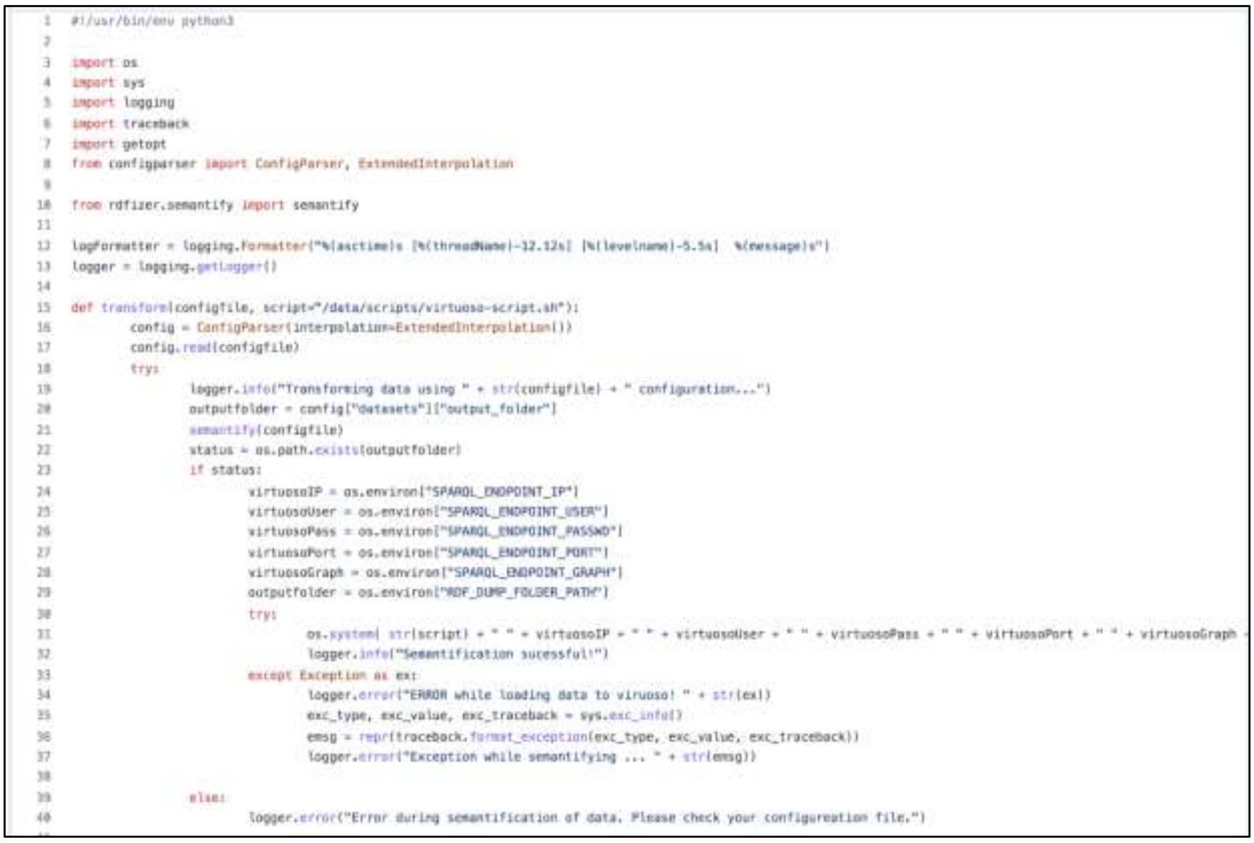
This repository contains basic settings for PLATOON Pipeline.

- `scripts` - contains scripts used for transforming sources to RDF and loading it to triple store (Virtuoso) - `virtuoso-script.sh` - used to remotely connect and load data using `isql-v` tool of virtuoso on command line - `load_to_virtuos.py` - used to load the transformed RDF data to virtuoso using the `virtuoso-script.sh` script - `transform_and_load.py` - performs both transforming raw data to RDF and loading it virtuoso using the `virtuoso-script.sh` script - `mapping_parser.py` - generates the necessary file for the execution of DeTrusty
- `configuration_files` - contains configuration files for the execution of the pipeline - `config_rdfizer.ini` - configuration file for materializing the Knowledge Graph using SDM-RDFizer. - `config.ini` - configuration file for generating the DeTrusty configuration file.
- `docker-compose.yml` - docker compose setup for transforming data to RDF and load it to `Virtuoso` triple store.

Figure 3: The PLATOON Pipeline

The aforementioned GitHub repository contains required files and scripts for the execution of the PLATOON pipeline. These components are:

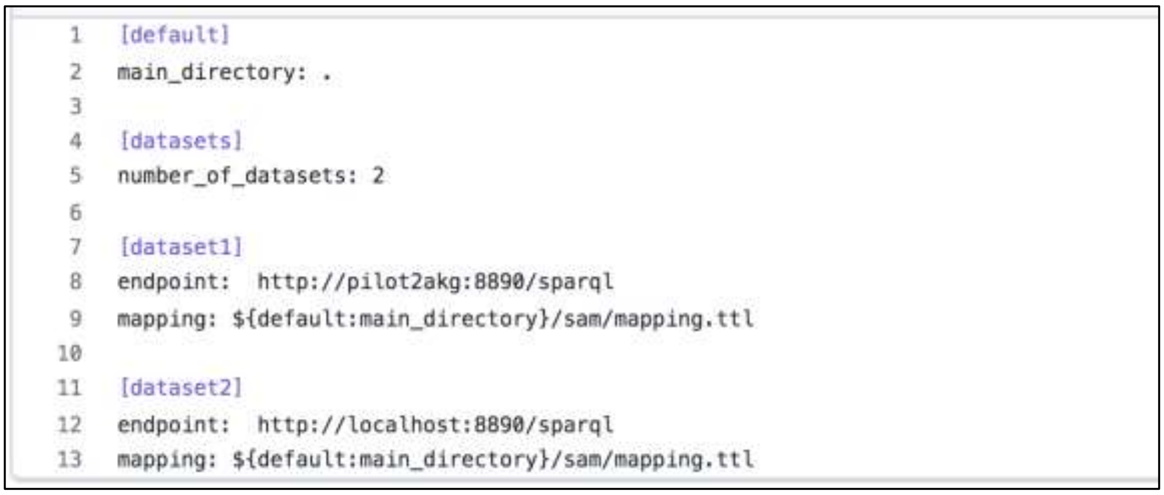
- ***Scripts:*** is a folder containing the scripts that transform mapping files and their corresponding data sources into RDF data, which is then loaded into the triple store (Virtuoso). These scripts are:
  - *Virtuoso-script.sh:* used to remotely connect and load data using the *isql-v* tool of Virtuoso.
  - *Load_to_virtuoso.py: executes* the *virtuoso-script.sh* to upload transformed RDF data to the Virtuoso triple store.
  - *Transform_and_load.py:* performs both the transformation of raw data into RDF data by invoking the SDM-RDFizer and uploads the data into a triple store by using *virtuoso-script.sh.*
  - *Mapping_parser.py:* generates the input file necessary for the execution of FQP by associating the classes from the mapping files with its corresponding predicates and endpoint that contains the RDF data.



```python
#!/usr/bin/env python3

import os
import sys
import logging
import traceback
import getopt
from configparser import ConfigParser, ExtendedInterpolation

from rdfizer.semantify import semantify

logFormatter = logging.Formatter("%(asctime)s [%(threadName)-12.12s] [%(levelname)-5.5s]  %(message)s")
logger = logging.getLogger()

def transform(configfile, script="/data/scripts/virtuoso-script.sh"):
        config = ConfigParser(interpolation=ExtendedInterpolation())
        config.read(configfile)
        try:
                logger.info("Transforming data using " + str(configfile) + " configuration...")
                outputfolder = config["datasets"]["output_folder"]
                semantify(configfile)
                status = os.path.exists(outputfolder)
                if status:
                        virtuosoIP = os.environ["SPARQL_ENDPOINT_IP"]
                        virtuosoUser = os.environ["SPARQL_ENDPOINT_USER"]
                        virtuosoPass = os.environ["SPARQL_ENDPOINT_PASSWD"]
                        virtuosoPort = os.environ["SPARQL_ENDPOINT_PORT"]
                        virtuosoGraph = os.environ["SPARQL_ENDPOINT_GRAPH"]
                        outputfolder = os.environ["RDF_DUMP_FOLDER_PATH"]
                        try:
                                os.system( str(script) + " " + virtuosoIP + " " + virtuosoUser + " " + virtuosoPass + " " + virtuosoPort + " " + virtuosoGraph
                                logger.info("Semantification sucessful!")
                        except Exception as ex:
                                logger.error("ERROR while loading data to viruoso! " + str(ex))
                                exc_type, exc_value, exc_traceback = sys.exc_info()
                                emsg = repr(traceback.format_exception(exc_type, exc_value, exc_traceback))
                                logger.error("Exception while semantifying ... " + str(emsg))

                else:
                        logger.error("Error during semantification of data. Please check your configureation file.")
```

Figure 4: Portion of transform_and_load.py illustrating the uploading of RDF data in the triples store.

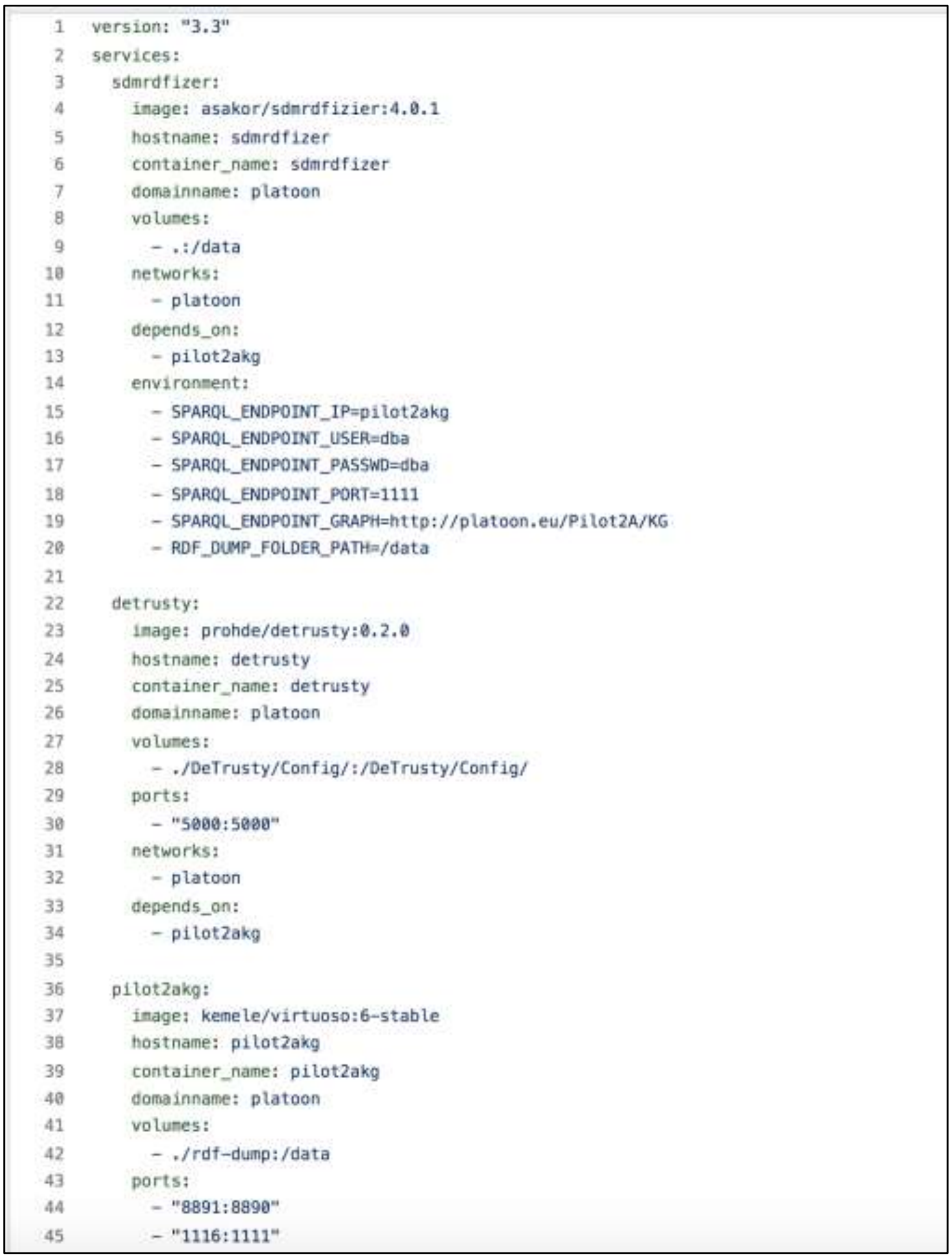
- ***Configuration_files:*** is a folder that contains the configuration files that are necessary for the execution of the SDM-RDFizer and mapping_parser.py script.



```ini
[default]
main_directory: .

[datasets]
number_of_datasets: 2

[dataset1]
endpoint:  http://pilot2akg:8890/sparql
mapping: ${default:main_directory}/sam/mapping.ttl

[dataset2]
endpoint:  http://localhost:8890/sparql
mapping: ${default:main_directory}/sam/mapping.ttl
```

Figure 5: Example of Configuration File illustrating the execution of the SDM-RDFizer.

- ***Docker-compose.yml:*** docker compose creates the docker images of the SDM-RDFizer [1], DeTrusty [2], and Virtuoso [3].



```
version: "3.3"
services:
  sdmrdfizer:
    image: asakor/sdmrdfizier:4.0.1
    hostname: sdmrdfizer
    container_name: sdmrdfizer
    domainname: platoon
    volumes:
      - .:/data
    networks:
      - platoon
    depends_on:
      - pilot2akg
    environment:
      - SPARQL_ENDPOINT_IP=pilot2akg
      - SPARQL_ENDPOINT_USER=dba
      - SPARQL_ENDPOINT_PASSWD=dba
      - SPARQL_ENDPOINT_PORT=1111
      - SPARQL_ENDPOINT_GRAPH=http://platoon.eu/Pilot2A/KG
      - RDF_DUMP_FOLDER_PATH=/data

  detrusty:
    image: prohde/detrusty:0.2.0
    hostname: detrusty
    container_name: detrusty
    domainname: platoon
    volumes:
      - ./DeTrusty/Config/:/DeTrusty/Config/
    ports:
      - "5000:5000"
    networks:
      - platoon
    depends_on:
      - pilot2akg

  pilot2akg:
    image: kemele/virtuoso:6-stable
    hostname: pilot2akg
    container_name: pilot2akg
    domainname: platoon
    volumes:
      - ./rdf-dump:/data
    ports:
      - "8891:8890"
      - "1116:1111"
```

Figure 6: Example of docker-compose.yml file illustrating the docker compose file used for the generation of the docker images that are required for the execution of the Pipeline.

DeTrusty (a.k.a. FQP) is a query engine that implements federated query processing against SPARQL endpoints. It decomposes the input query into star-shaped sub-queries, i.e., all triple patterns in a sub-query share the same subject. If an RDF type statement is present in a sub-query, it will be used to identify the sources that contribute to the sub-query in question. If no such statement is present, DeTrusty selects all sources that contribute to RDF classes that contain all predicates of the sub-query. This allows DeTrusty to minimize the number of contacted SPARQL endpoints. Each sub-query is executed over the previously selected sources for the sub-query. The partial results retrieved from the endpoints are combined at the query engine level using non-blocking physical operators. DeTrusty creates bushy plans in order to speed up the query execution. These features enable DeTrusty to continuously generate complete and sound query results while minimizing the number of contacted endpoints and query execution time. The current version of DeTrusty is capable of executing SPARQL SELECT queries. The SERVICE clause from SPARQL 1.1 is also implemented. Some SPARQL 1.1 features are not yet implemented, e.g., GROUP BY and aggregate functions. FQP can be run in a Docker container. After providing DeTrusty with the semantic source description of the set of SPARQL endpoints, it can be used via its HTTP API as if it was a regular SPARQL endpoint. Deliverable D2.8 provides a detailed description of the federated query engines integrated into the PLATOON framework; it also reports on the results of the performance empirical evaluation of DeTrusty on Pilot 2a knowledge base.

2.3 IDS Connectors

In PLATOON the IDS connector is the component allowing the different pilot to exchange, share and process digital content. At the same time, the Connector ensures that the data sovereignty of the data owner is always guaranteed. The reference implementation of the IDS connector adopted in PLATOON is the TRUE (TRUsted Engineering) connector: it is an open-source implementation, developed by Engineering, that has been reused and extended in order to fit the specific project needs.

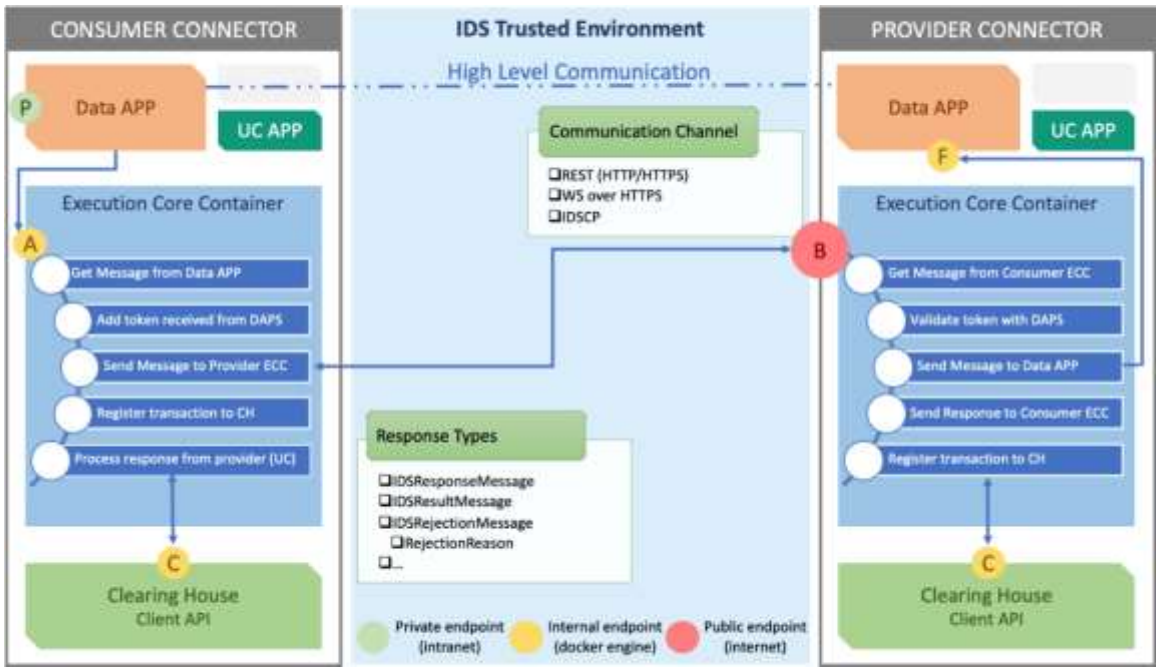


Figure 7 - TRUE connector architecture and interaction

The Figure 7 shows the main components of the TRUE connector and the interactions among two connectors that exchange data. Following a description of these components:

- **Execution Core Container (ECC)** is in charge of (1) the data exchange using HTTP/HTTPS, WS over HTTPS or IDSCP2, taking advantage of the IDS Information Model to represent the data, (2) the interactions with an external Identity Provider to require and validate token, (3) the communication with an IDS Broker for managing the information and with the Clearing House for registering the transactions.

- **Data APP**: this component represents a trivial data application build to generate and/or consume data on top of the Execution Core Container.
- **Usage-Control Application (UC APP)**: a component in charge of applying usage control policies to the data.

As described in the above figure the basic communication flow between two connectors (used by a Data Provider and a Data Consumer) can be resumed in the following steps.

1. the Consumer receive a data request to its P endpoint, it forwards the request to its internal Execution Core Container (A endpoint)
2. The Consumer's ECC receives the message from its Data APP,
3. The Consumer's ECC interacts with an external Identity Provider to retrieve the token of the Consumer and
4. The Consumer's ECC sends the appropriate IDS message to the Provider's ECC using one of the provided communication channels (B endpoint).
5. The Provider's ECC receives the message and validates the token against the Identity provider,
6. ECC retrieves the actual data from its Data APP (F endpoint)
7. Provider ECC returns data to the Consumer's ECC
8. Consumer's ECC processes the response, applies the usage control policies and forward the data to the original requester.

It is important to underline that every transaction is logged into the Clearing House (C endpoints).

The TRUE connector in PLATOON has been extended to be integrated with CaPe suite an ICT suite for a consent-based user-centric personal data management that acts as an intermediary and as a tool of communication between data subjects and controllers/processors, supporting the generation and management of dynamic consents. Also, it has been developed a new, open-source, Data Usage Control module as an evolution of the open-source IDS Dataspace Connector that supports usage policies written in the IDS Usage Control Language based on ODRL. More information about these developments can be found in deliverable D3.4

The implementation of the TRUE connector is available in open source in the PLATOON Github repository (https://github.com/PLATOONProject/true-connector).

2.4 Metadata Registry

The Metadata Registry is a registry for datasets and apps/data analytics tools and is the main component of PLATOON Marketplace. It can be used to register, update, or unregister the Connector or resource (Resource or AppResource) metadata. Self-description of the Metadata Registry is also available, and one can query for any information provided in it. The following list shows the main functions of the Metadata Registry:

- **Description Request**: Gets the self-description of the Metadata Registry.
- **Register/Update Connector**: Registers the Connector if it doesn't exist in the Metadata Registry or update the Connector if it exists. Users can also include Resources (metadata of the data) while registering a Connector.
- **Unregister Connector**: Unregisters the Connector from the Metadata Registry.
- **Update Resource**: Updates the information of a particular Resource of a Connector.
- **Unregister Resource**: Removes a particular Resource from a Connector.
- **Register App**: Registers metadata of an App in the Resource Catalog of the Connector.

- **Unregister App**: Unregisters metadata of an App from the Resource Catalog of the Connector.
- **Query**: Queries the SPARQL triples in Fuseki triple store.

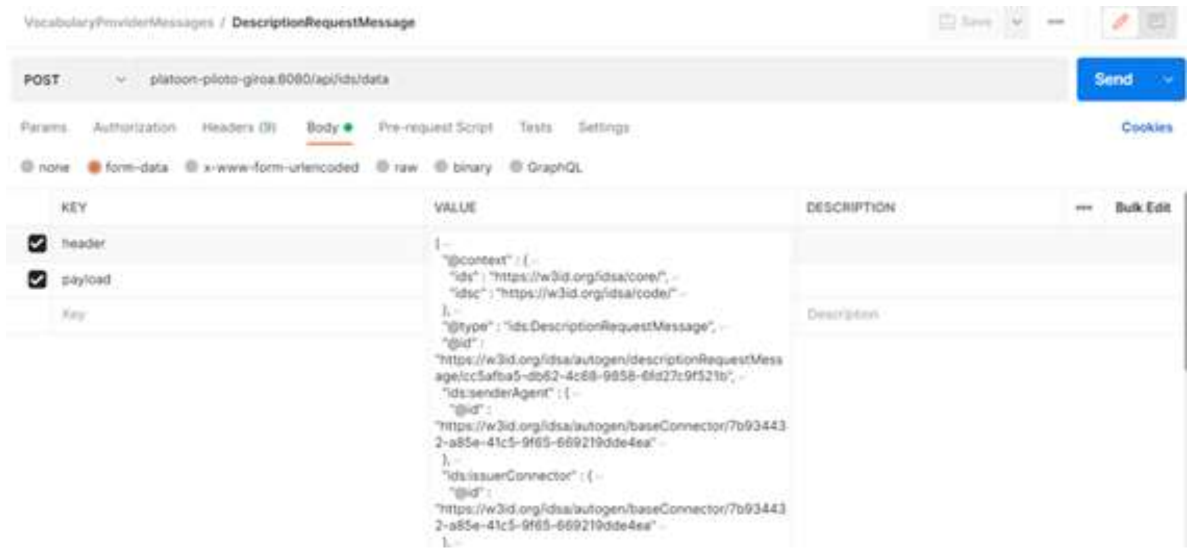
Note that all data in the Metadata Registry is metadata only. The register, update and unregister of connectors, Resource, and AppResource mentioned above are also related to metadata.

The implementation of the meta data registry is available in open source in the PLATOON Github repository (https://github.com/PLATOONProject/Metadata-Registry).

2.5 IDS Vocabulary Provider

In WP3 an IDS Vocabulary Provider was developed. This is a central component that manages all the vocabularies (ontologies, reference data models, metadata elements), which can annotate and describe datasets and data analytics tools.

The IDS Vocabulary Provider plays an essential role within the PLATOON reference architecture as it is the link between the Data Governance, Security, Privacy and Sovereignty layer (based on IDS reference architecture) and the Interoperability layer formed. The vocabulary provider provides direct Machine to Machine communication allowing to query the different vocabularies and exchange metadata through the IDS connectors. Several IDS messages have been implemented to allow to register to the vocabulary provider and send queries. The figure below shows an IDS query message:

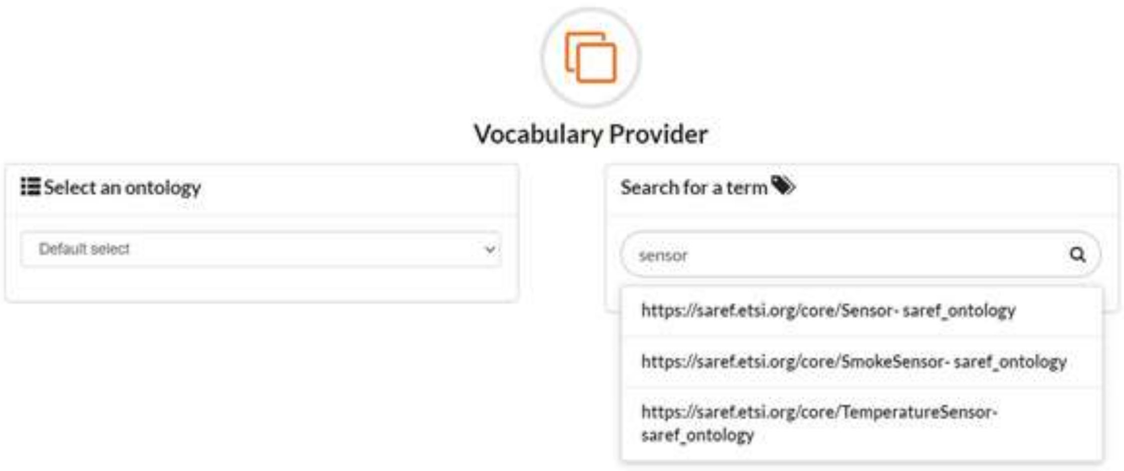


POST platoon-piloto-giroa:8080/api/ids/data

	KEY	VALUE	DESCRIPTION
✓	header	{ "@context" : { "ids" : "https://w3id.org/idsa/core/", "idsc" : "https://w3id.org/idsa/code/" }, "@type" : "ids:DescriptionRequestMessage", "@id" : "https://w3id.org/idsa/autogen/descriptionRequestMessage/cc5afba5-db62-4c68-9858-6fd27c9f521b", "ids:senderAgent" : { "@id" : "https://w3id.org/idsa/autogen/baseConnector/7b934432-a85e-41c5-9f65-669219dde4ea" }, "ids:issuerConnector" : { "@id" : "https://w3id.org/idsa/autogen/baseConnector/7b934432-a85e-41c5-9f65-669219dde4ea" },	
✓	payload		

Figure 8: PLATOON IDS Vocabulary Provider - IDS Messages - Description Request Message

In addition, the Vocabulary Provider has a Graphical User Interface (GUI), where users can manage vocabularies (upload/upgrade/delete), search for specific terms, visualize the vocabularies in a network graph and execute SPARQL queries.



https://saref.etsi.org/core/Sensor- saref_ontology

https://saref.etsi.org/core/SmokeSensor- saref_ontology

https://saref.etsi.org/core/TemperatureSensor- saref_ontology

Figure 9: PLATOON IDS Vocabulary Provider GUI - Home Screen

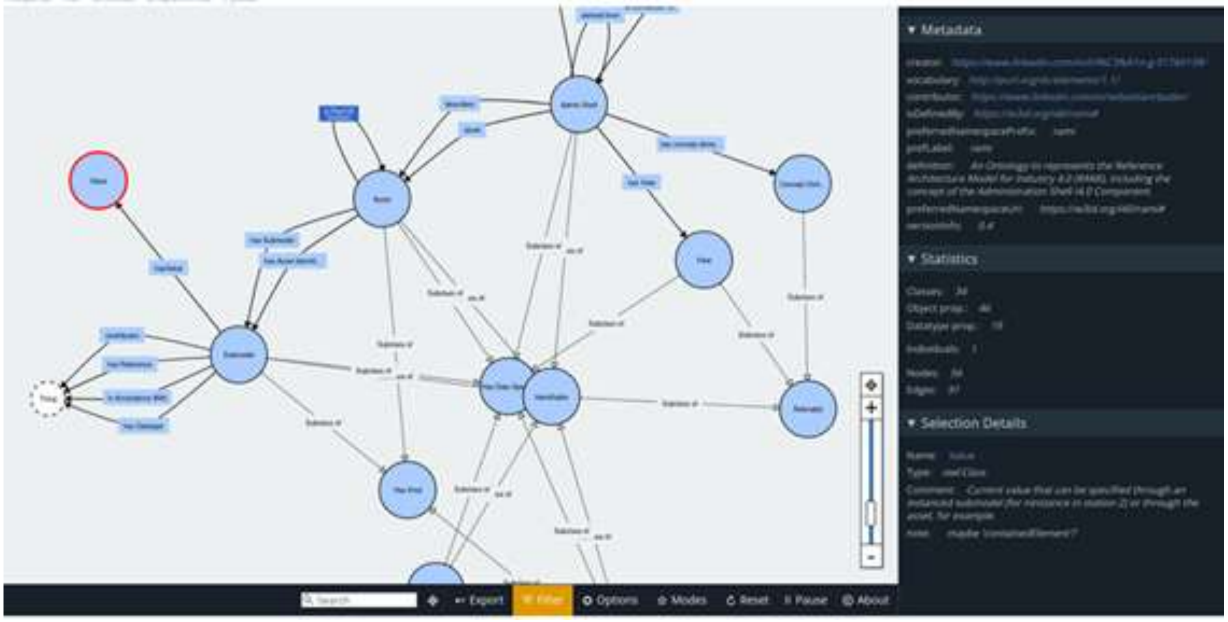


Figure 10: PLATOON IDS Vocabulary Provider – Visualization – Network Diagram

The developed solution is open source and it is available in: https://github.com/PLATOONProject/PLATOON_IDS-Vocabulary-Provider

More detailed explanation about the available functionalities and configuration instructions are defined in deliverable D3.5.

2.6 Data Analytics Tools

For data analytics, there are two different tools available. The first tool is a containerized demo tool following the specification defined in D4.1 which implements REST APIs as defined in D2.2. Thanks to the use of these REST APIs, the tool can be replaced by any other data analytic tool without the need to implement a completely new pipeline. In addition to this first demo tool, a second tool is available which focuses on the requirements of pilot 3A. This second tool is further explained in deliverable D5.2. In both tools, SANSA is an integral part which will be discussed in the following.

In WP4 and WP5, the Scalable Big Semantic Analytics Stack (SANSA) was further developed. The enhancements include generic tools that can be used for classification, clustering, and anomaly detection. In the context of the Platoon use cases, the pipelines described in the proposal were also extended by the option of regression in order to implement regression-based baseline forecasting. The following tools were developed for this purpose:

- DistSim [4]
  - Similarity Estimation as backbone for clustering and unsupervised classification [5]
  - Machine Learning pipeline feature extraction on Knowledge Graphs [6]
  - Generic downstream machine learning pipelines for RDF Knowledge Graphs
- DistAD [7]
  - Numeric Outlier detection within RDF KG
- Semantic Web Analysis with Flavour of Micro-Services [8]
  - Rest Interface for SANSA and Spark based Pipelines

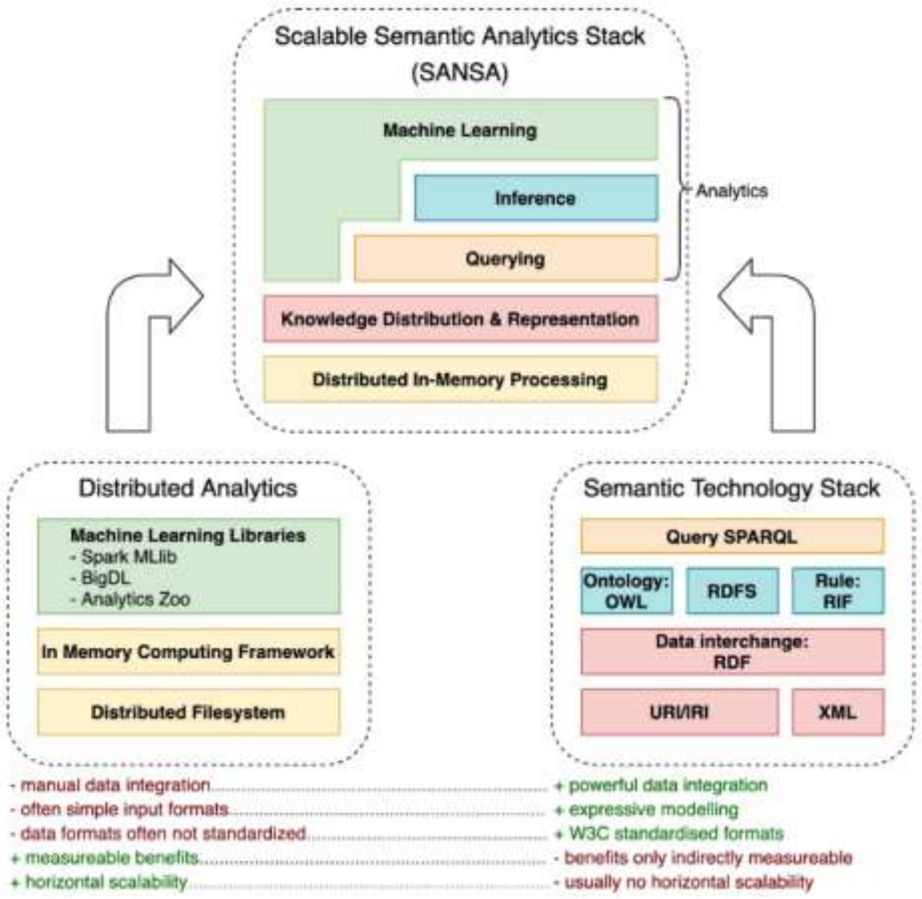

Scalable Semantic Analytics Stack (SANSA)
Machine Learning
Inference
Querying
Analytics
Knowledge Distribution & Representation
Distributed In-Memory Processing
Distributed Analytics
Machine Learning Libraries
- Spark MLlib
- BigDL
- Analytics Zoo
In Memory Computing Framework
Distributed Filesystem
Semantic Technology Stack
Query SPARQL
Ontology: OWL
RDFS
Rule: RIF
Data interchange: RDF
URI/IRI
XML
- manual data integration
- often simple input formats
- data formats often not standardized
+ measureable benefits
+ horizontal scalability
+ powerful data integration
+ expressive modelling
+ W3C standardised formats
- benefits only indirectly measureable
- usually no horizontal scalability

Figure 11: The SANSA ecosystem

As Part of WP5, especially T.5.1 and T.5.5, we evaluated within a Non-Edge Scenario these PLATOON related generic analytic modules of SANSA. As SANSA builds on top of Apache Spark for distributed Processing and HDFS for Data Management and Storage, the respective containers include these modules. In addition, we build on top of Apache Jena 4.x and Java 11. For the REST Management, we use Apache Livy to propose a rest interface for handling Rest interactions with the analytic pipelines. The Details of Rest Exposure can be found in [8] while the respective Regression pipeline can be found in [6]. The results are stored either in Tabular data which suits the requirements of PyNalia or can be stored as native Semantic data.

List of the modules which are evaluated in the context of SANSA are:

- Apache Livy
- Apache Spark (incl. MLlib)
- Apache Jena
- Hadoop File System (HDFS)
- HUE
- Apache Zeppelin

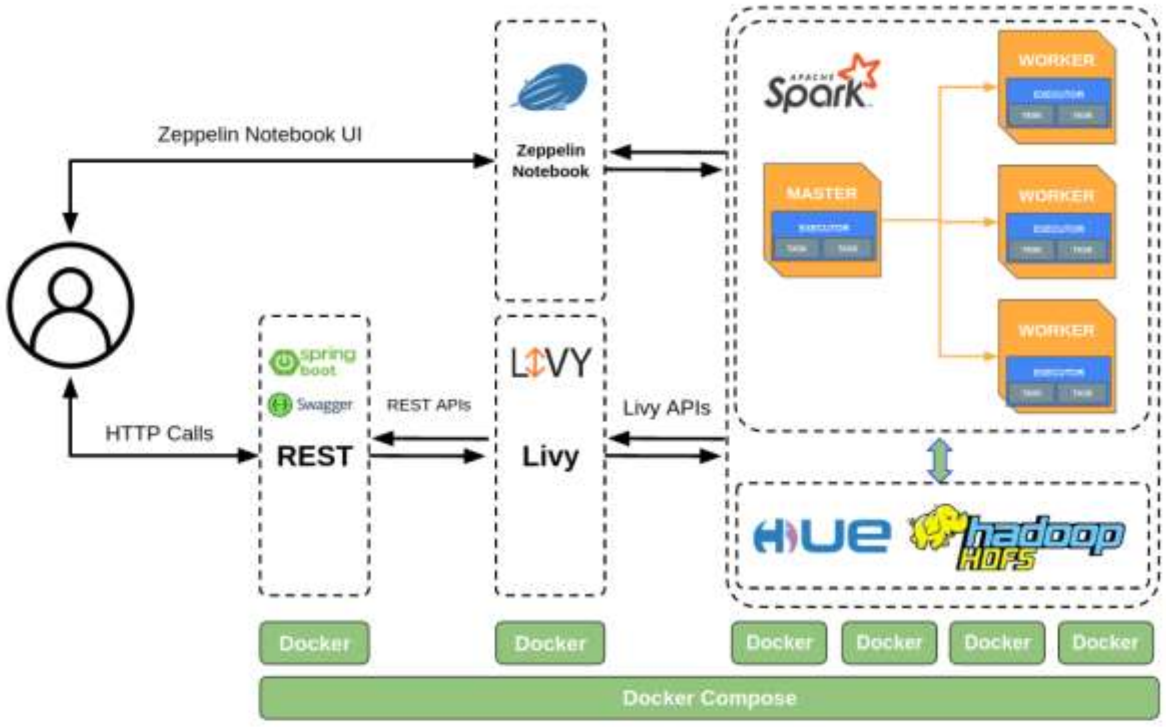


Figure 12: The different modules of SANSA

2.7 Data Analytics Dashboard

In task T5.4 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.

The target user of the tool are energy experts with high domain knowledge but low coding skills. Thus, the Dashboard has been defined to be intuitive and user-friendly providing a collection of reusable visualization templates/chart that can be easily integrated into customer-oriented cloud-based dashboard for predictive analytics and insights analysis.

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 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.

On the other hand, the Real-time Dashboard 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.

More details about the development, functionalities and configuration of the dashboard are provided in T5.4.

3 Testing Scenarios

We conceived different scenarios for the purposes of integration testing which is documented in D5.5. The first scenario for our tests is the so-called Edge-Scenario. The focus of the scenario is to stream data originating from IoT devices or edge computing devices and transform and store them in dedicated databases. The data is then processed and visualized. The second scenario, the so-called Non-Edge-Scenario is focused on batch data using static sources. The data is transformed into RDF and can be queried using a federated query engine. Again, the data is processed and finally visualized. The final scenario, the so-called data-sharing scenario, focuses on data exchange using IDS components.

3.1 Edge-Scenario

The Edge-Scenario consumes data coming from edge computing devices which is then stored into a database for further processing. The data in this scenario is time-series data. Once processed, the outcome of the processing can be visualized by an appropriate component. The following figure illustrates the architecture of the Edge-Scenario.

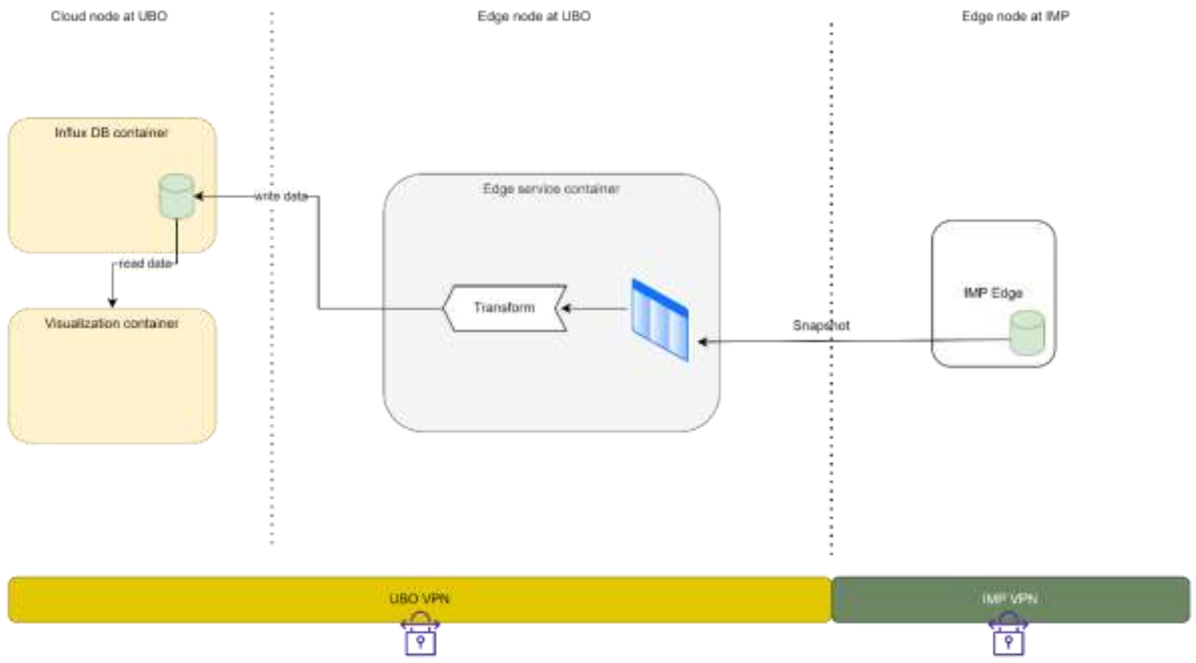


Figure 13: Edge Scenario architecture

3.2 Non-Edge-Scenario

This scenario focuses on data sources containing static data. In this scenario, we use RDFizer [1] as semantic adaptor to transform the data into RDF. This makes the data available for querying over a federated query engine. The analytics part is done by SANSA as discussed in Section 2.6. Finally, the result of SANSA is visualized. The following figure shows the architecture of the Non-Edge-Scenario.

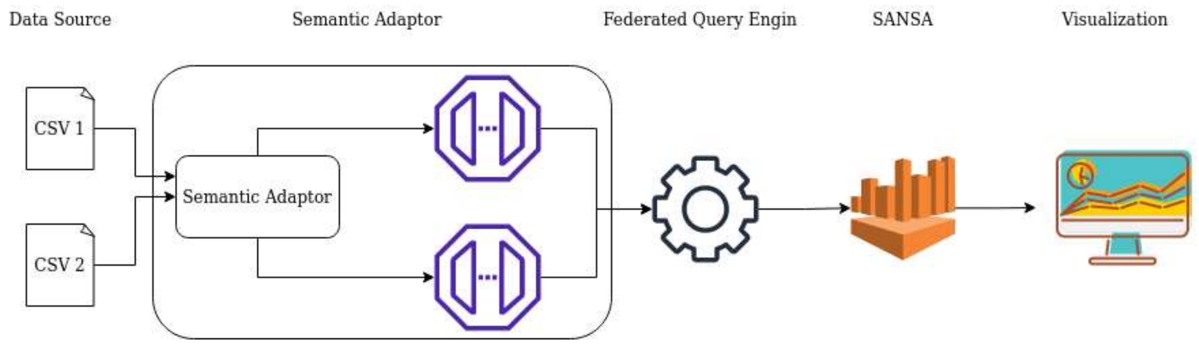


Figure 14: Non-Edge Scenario architecture

3.3 Data-Sharing Scenario

In this scenario, the interaction between the Metadata Registry, two TRUE connectors and the Vocabulary Provider is tested. The scenario tests the sharing of data between the two connectors using JSON. The following figure gives an overview of the architecture.

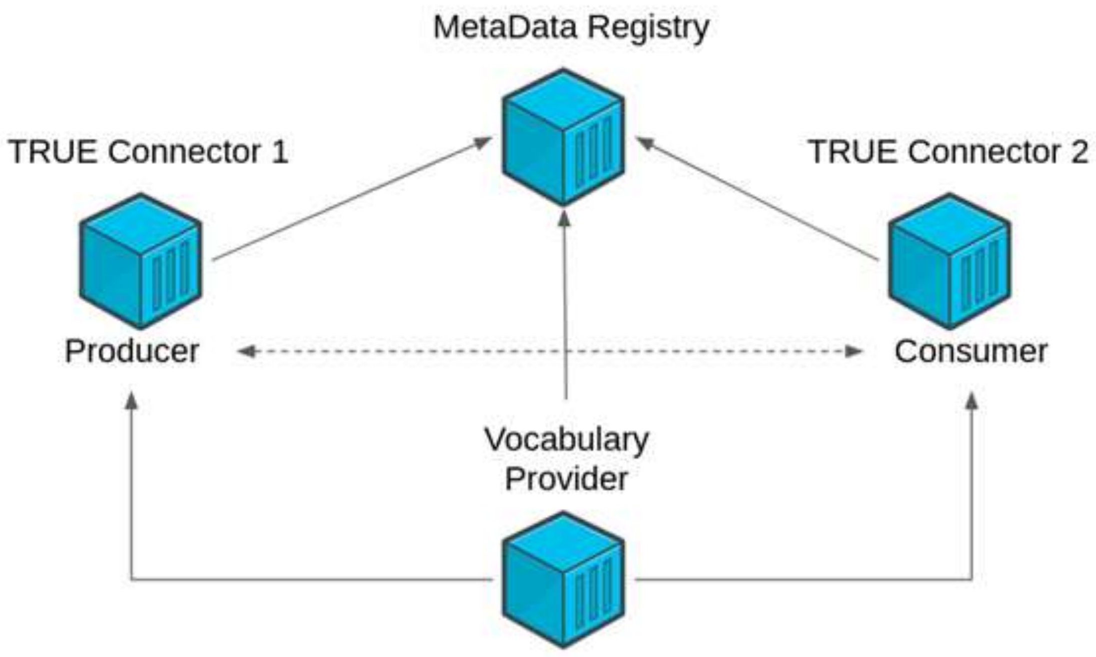


Figure 15: Datasharing scenario overview

Deliverable D5.5 contains dedicated configuration and setup details for implementing the specific testing scenarios and the corresponding integrations test results.

4 Hardware Specification

In this section, we give an overview of the hardware specifications used for the evaluation of the reference architecture. Specific considerations have been given to the fact that a lot of these containers have to be instantiated simultaneously to support the different scenarios.

Mainboard	Supermicro X11DPG-QT-CPU
CPU	2x: Xeon® Scalable Processor "Cascade Lake" Silver 4214R, 2.40 GHz, 12-Core, Socket 3647, 16,5 MB L3 Cache 2400 MHz, 3x UPI, Tcase 87°C
RAM	192GB (12x 16GB; 3 Channel with each 4 16GB DIMMs) DDR4 / PC2933
SAS-Controller	SAS2/SAT3 RAID-Controller Card LSI MEGARAID MEGARAID SAS 9361-8I SGL, 8 Port LSI00417 (05-25420-08)
Hard Drives	6x: 4TB HD, Seagate Exos E 7E8 4TB, 512n, SAS 12Gb/s (ST4000NM0025)
GPU	4x: PCIe x16 Graphics Card NVIDIA® GeForce RTX 2080 Super 8GB GDDR6 NVIDIA® Turing GPU Architecture: 2944 CUDA Cores (single precision), 57 Tensor Cores Formfactor: 2 PCI slots Video Memory: 8 GB GDDR6 Reference: ASUS TURBO-RTX2080S-8G-EVO
Network Interface	2x 10 GBASE-T onboard (RJ45), IPMI 2.0 Management port onboard (RJ45) 1x Dual port Supermicro AOC-STG-I2T, 2x 10GB-LAN, RJ45

5 Implementation Details

As explained in WP2 deliverables and in D4.1, PLATOON components should be interoperable and deployable in any environment that adheres to the PLATOON reference architecture, APIs, and data models. This criterion can be met by packaging the software into containers. A container is a standardized executable software unit that contains the source code as well as all its dependencies or the associated packages. The containerized program is separated from its surroundings. As a result, a container is simple to ship and deploy into diverse environments, even if they differ from those used during development. Docker [9] is the container technology used throughout the PLATOON project.

5.1 Docker

Docker is a collection of platform as a service (PaaS) solutions that provide software in containers via OS-level virtualization. Among the several container technologies, Docker is extensively used and interoperable with all major providers of local operating systems and cloud computing platforms, and it has established a number of industry standards. Docker technology is not open source; however, it is available for free (under the Apache License 2.0). Furthermore, Docker has a large user base and an active community, and it has been embraced by many businesses. There is now an enormous database of Docker images accessible, as well as rich documentation to help developers throughout various Docker tasks. A Docker image is created by following the instructions in a text file called a Dockerfile. The Dockerfile also includes the container's primary command of execution or the path to a shell script containing the execution instructions. The image is composed of numerous layers, including a basic operating system layer as well as levels holding run times and configuration files. Containers can execute as standalone executables from the docker image. Figure 16 shows the difference between Docker and Virtualization Technology.

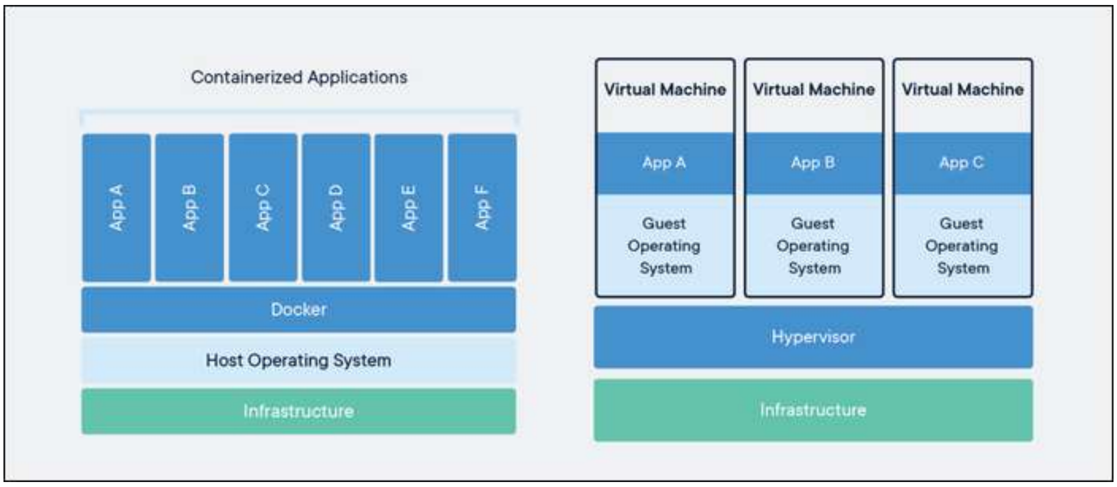
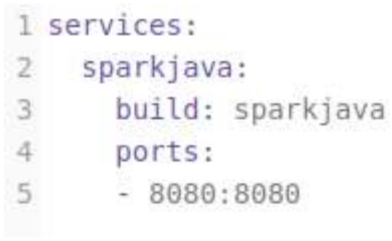


Figure 16: Difference between Docker and Virtualization Technology [10]

5.2 Docker Compose

Docker Compose is a tool that allows you to define and operate Docker applications consisting of multiple Docker containers. It configures the application's services using YAML files and executes the construction and startup of all containers with a single command. The docker-compose CLI program allows users to conduct tasks on numerous containers at the same time, such as generating images, scaling containers, restarting stopped containers, and more. Image manipulation commands and user-interactive options are irrelevant in Docker Compose since they only address one container. The docker-compose.yml file is used to specify the services of an application and contains numerous configuration settings. The build option, for example, specifies configuration settings such as the Dockerfile location, while the command option allows one to change default Docker commands, among other things. A simple docker-compose file might look then one in Figure 17.



```
services:
  sparkjava:
    build: sparkjava
    ports:
     - 8080:8080
```

Figure 17: A simple Docker-Compose file

5.3 Docker Swarm

Docker Swarm provides native Docker container clustering capability, converting a set of Docker engines into a single virtual Docker engine. Swarm mode is incorporated with Docker Engine in Docker 1.12 and later. Users may use the docker swarm CLI program to start Swarm containers, create discovery tokens, list nodes in the cluster, and more. The docker node CLI utility lets users perform commands to manage nodes in a swarm, such as listing nodes, updating nodes, and deleting nodes from the swarm. Docker uses the Raft consensus mechanism to control swarms.

5.4 Swarmpit

Swarmpit [11] is a user-friendly interface for Docker Swarm clusters. Using Swarmpit, we can manage docker stacks, services, secrets, volumes, networks, and docker nodes. Also, we can manage swarm clusters using the UI. The following image depicts the Swarmpit UI.

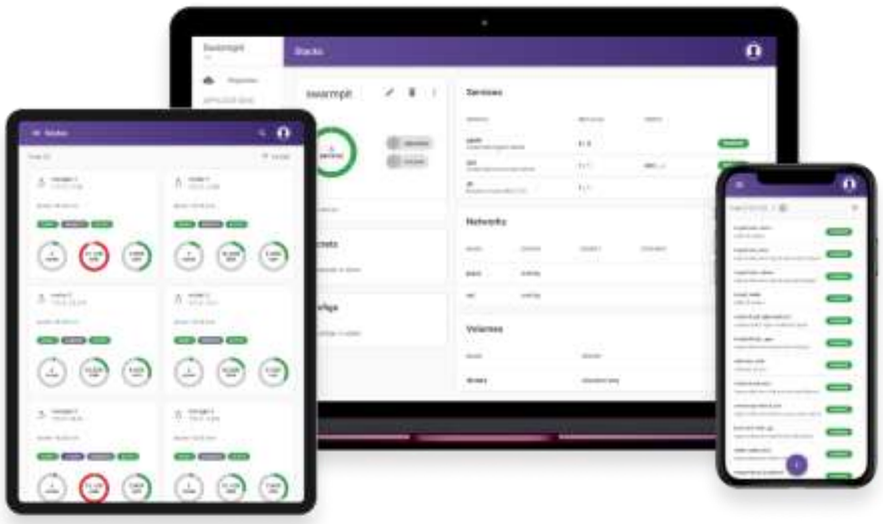


Figure 18: The Swarmpit UI [12]

4.4.1 Installing Swarmpit

To install Swarmpit, the following steps need to be done:

1. Install docker on the server
2. Start the docker service

For installing docker the following steps need to be done:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

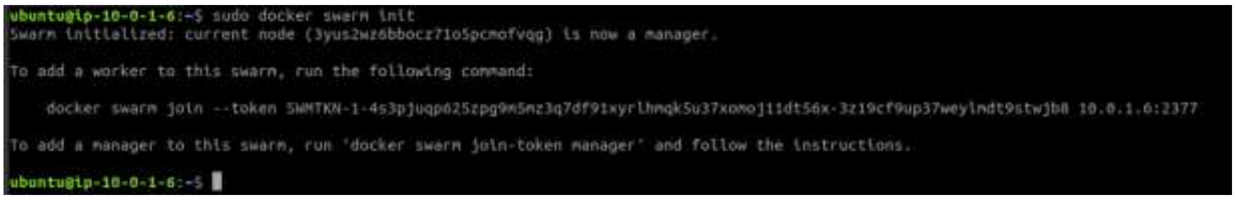
After installing docker, one can test if Docker is working via following command:

```
$ sudo docker run hello-world
```

This command downloads and executes a test image in a container. When the container is executed, it prints a message and then exits.

After installing docker, we need to enable swarm mode in docker via:

```
$ sudo docker swarm init
```



```
ubuntu@ip-10-0-1-6:~$ sudo docker swarm init
Swarm initialized: current node (3yus2wz6bbocz71o5pcmofvqg) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-4s3pjuqp625zpg9m5mz3q7df91xyrlhmqk5u37xomoj11dt56x-3z19cf9up37weylndt9stwjb8 10.0.1.6:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

ubuntu@ip-10-0-1-6:~$
```

Figure 19: Swarmpit initialization

The final step is to run Swarmpit itself as a docker container:

```
$ docker run -it --rm  --name swarmpit-installer –volume
      /var/run/docker.sock:/var/run/docker.sock
swarmpit/install:1.9
```

Now Swarmpit runs on port 888 and you can access Swarmpit under http://localhost:888

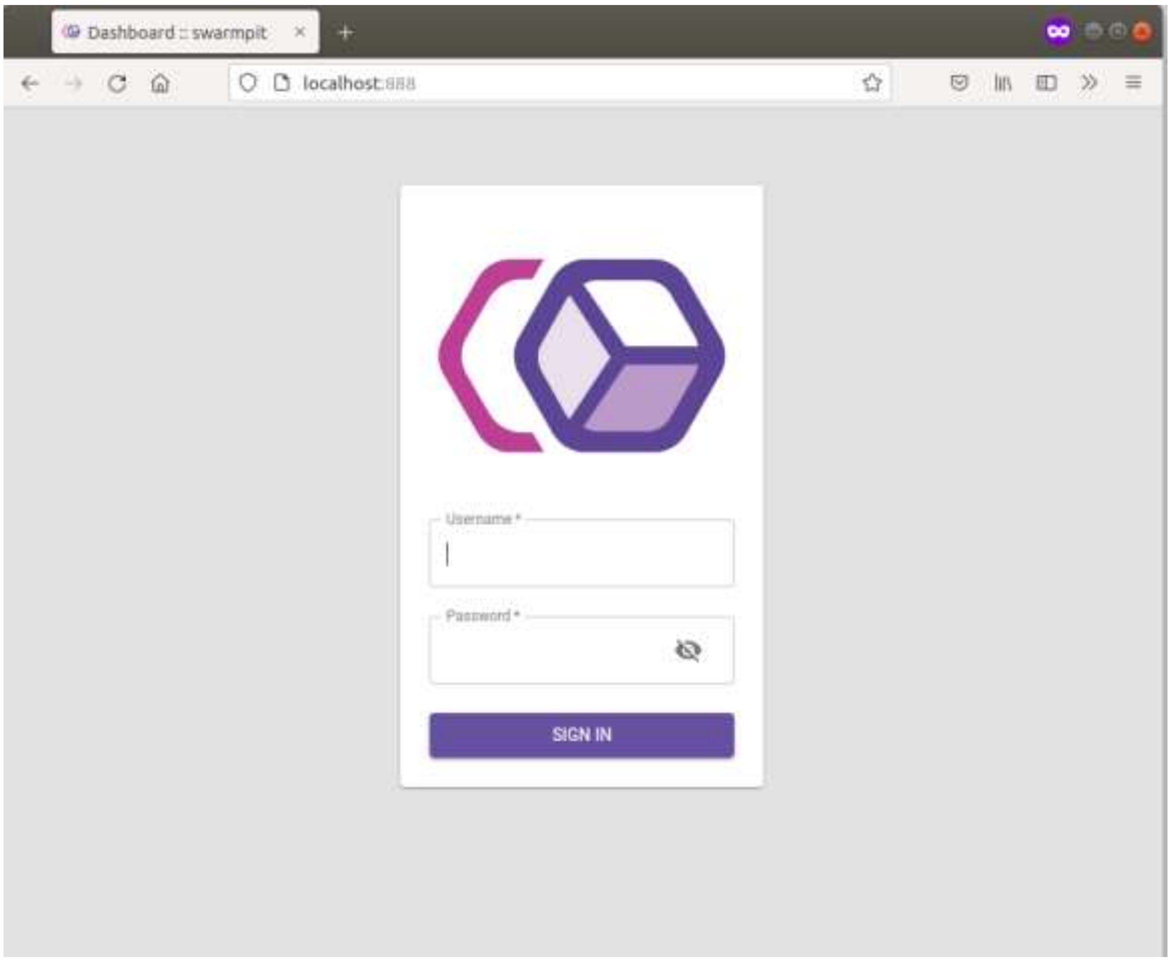


Figure 20: Swarmpit after startup

Swampit consists of mainly 4 components:

- app: Swarmpit
- agent: Swarmpit agent
- db: CouchDB (Application data)
- influxdb: InfluxDB (Cluster statistics)

Couch DB contains users and stack files, configs, and secrets. When we want to restore Swarmpit, we need to make a backup of this database, which is inside a docker volume located at /var/lib/docker/volumes.

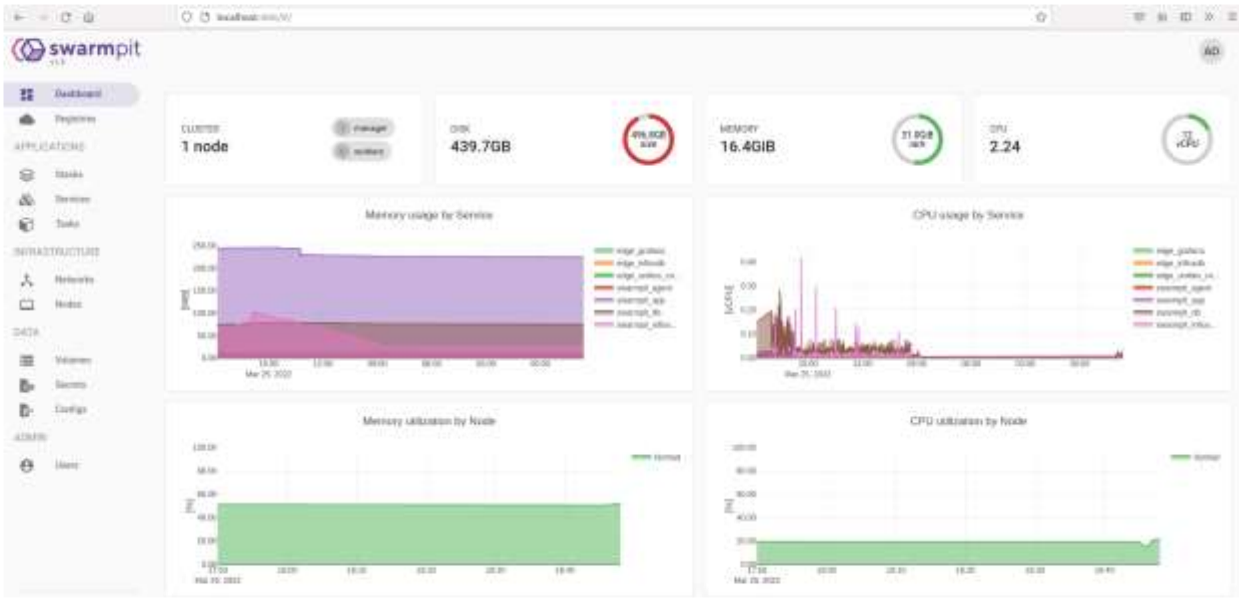


Figure 21: Overview of the Swarmpit UI

To deploy a stack in Swarmpit, go to UI → Applications → Stacks → New Stack

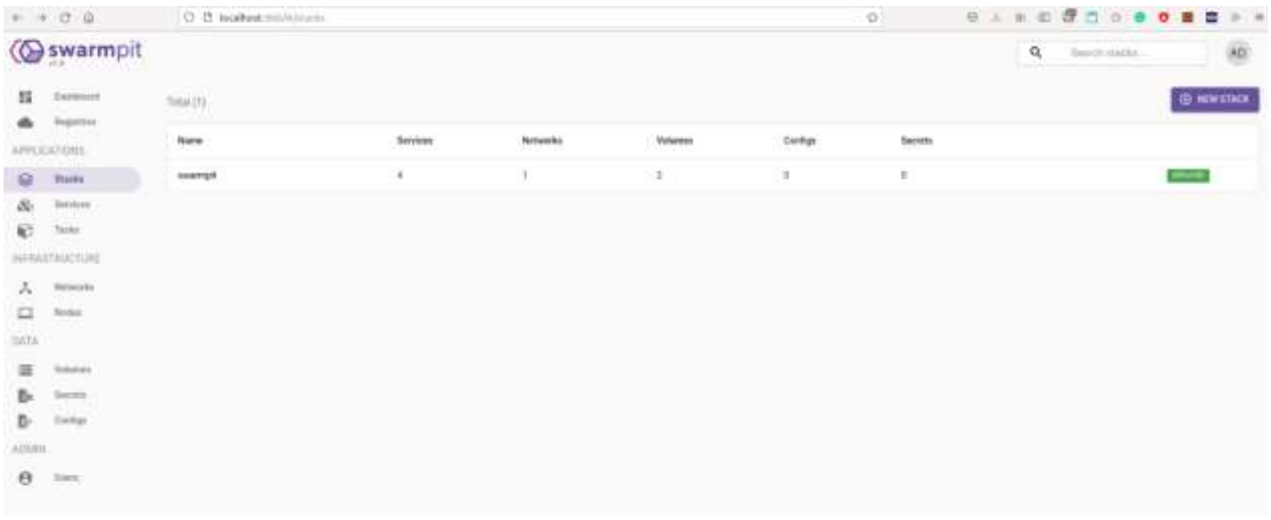
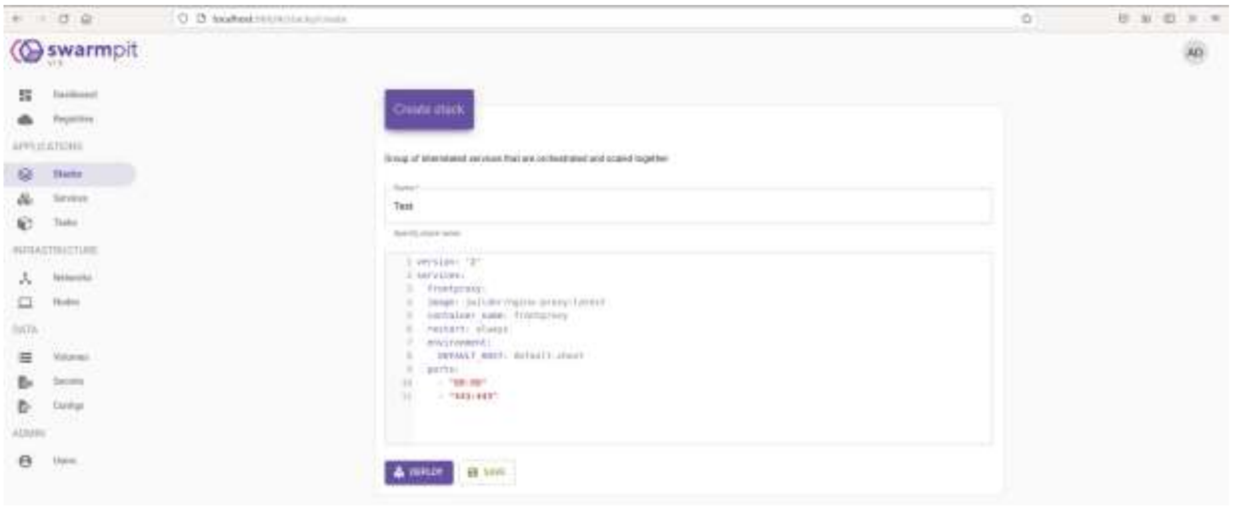


Figure 22: Deploying a stack in Swarmpit

In this stage the user can copy-paste the content of the docker-compose.yml file. For the mentioned scenarios in Section 3, the docker-compose files have been defined in D5.5.



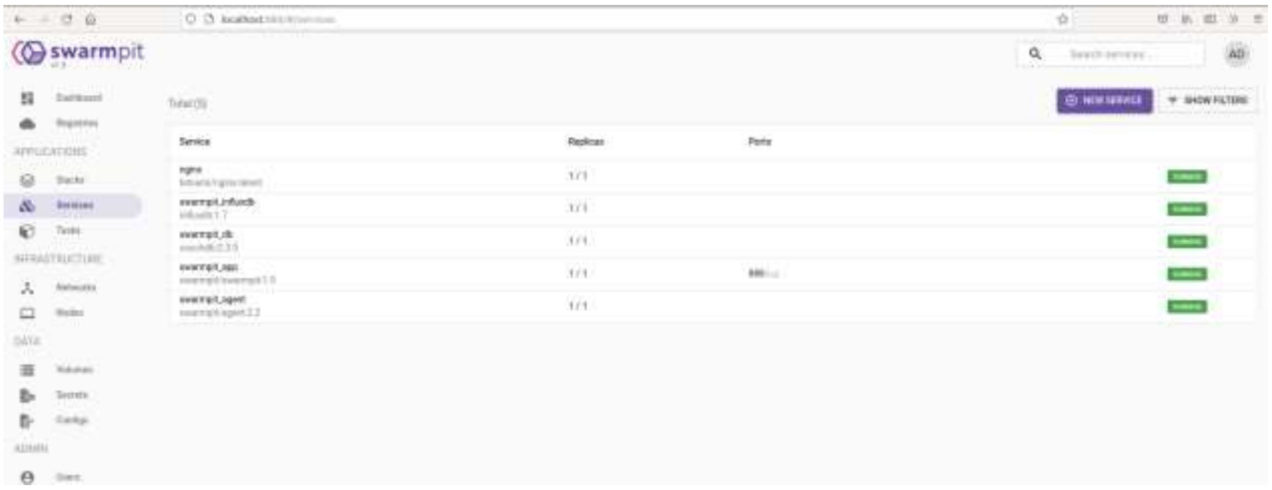
Group of interrelated services that are orchestrated and scaled together

Name*: Test

```
version: '3'
services:
  frontproxy:
  image: jwilder/nginx-proxy:latest
  container_name: frontproxy
  restart: always
  environment:
    DEFAULT_HOST: default.vhost
  ports:
    - "80:80"
    - "443:443"
```

Figure 23: Creating a stack file in Swarmpit UI

Finally, press deploy button:



Total (5)

Service	Replicas	Ports	
nginx bitnami/nginx:latest	1 / 1		[illegible]
swarmpit_influxdb influxdb:1.7	1 / 1		[illegible]
swarmpit_db couchdb:2.3.0	1 / 1		[illegible]
swarmpit_app swarmpit/swarmpit:1.9	1 / 1	888 [illegible]	[illegible]
swarmpit_agent swarmpit/agent:2.2	1 / 1		[illegible]

Figure 24: After deployment the service is listed in the UI

As an example, the simplified version of the Edge scenario will be deployed in Swarmpit. More details about the deployment and individual components can be found in D5.5.
The Edge scenario contains three components; edge data provider, database, and visualizer. Following docker-compose.yml file chains the Edge scenario components:

```
version: '3.6'
services:
  influxdb:
    image: influxdb:1.8.10
    container_name: influxdb
    restart: always
    environment:
      - INFLUXDB_DB=influx
      - INFLUXDB_ADMIN_USER=admin
      - AUTH_ENABLED=false
    ports:
      - '8086:8086'


  grafana:
    image: grafana/grafana
    container_name: grafana-server
    restart: always
    depends_on:
      - influxdb
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
      - GF_INSTALL_PLUGINS=
    links:
      - influxdb
    ports:
      - '3000:3000'
    user: "root"
  unittes_container:
    image: andrejcampa/unittest_edge
    environment:
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_DB=influx
      - INFLUX_DB_IP=influxdb
      - INFLUXDB_ADMIN_PASSWORD=password
    ports:
      - "8087:8086"
    depends_on:
      - influxdb
```

To deploy this, as explained, the user can visit the Swarmpit UI and select Applications → Stacks → New Stack. By copy pasting the above content and selecting a name for the stack, the user can press the deploy button. After pressing deploy button, Swarmpit will deploy the mentioned components and will show the status of each component as follows:

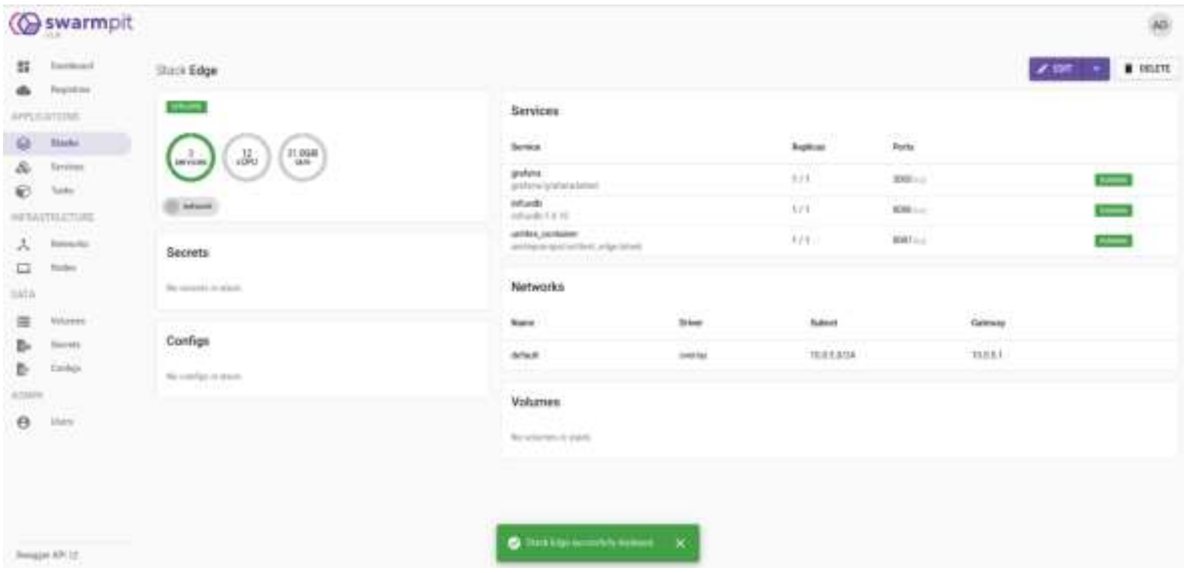


Figure 25: After deployment the service is listed in the UI

To validate that the deployment was successful, the user can visit http://localhost:3000 which shows the visualization. The user needs to configure visualization tool to see the data (all the details have been explained in D5.5. For more information, please check D5.5). The following figure shows the data which are being delivered from edge node every minute.

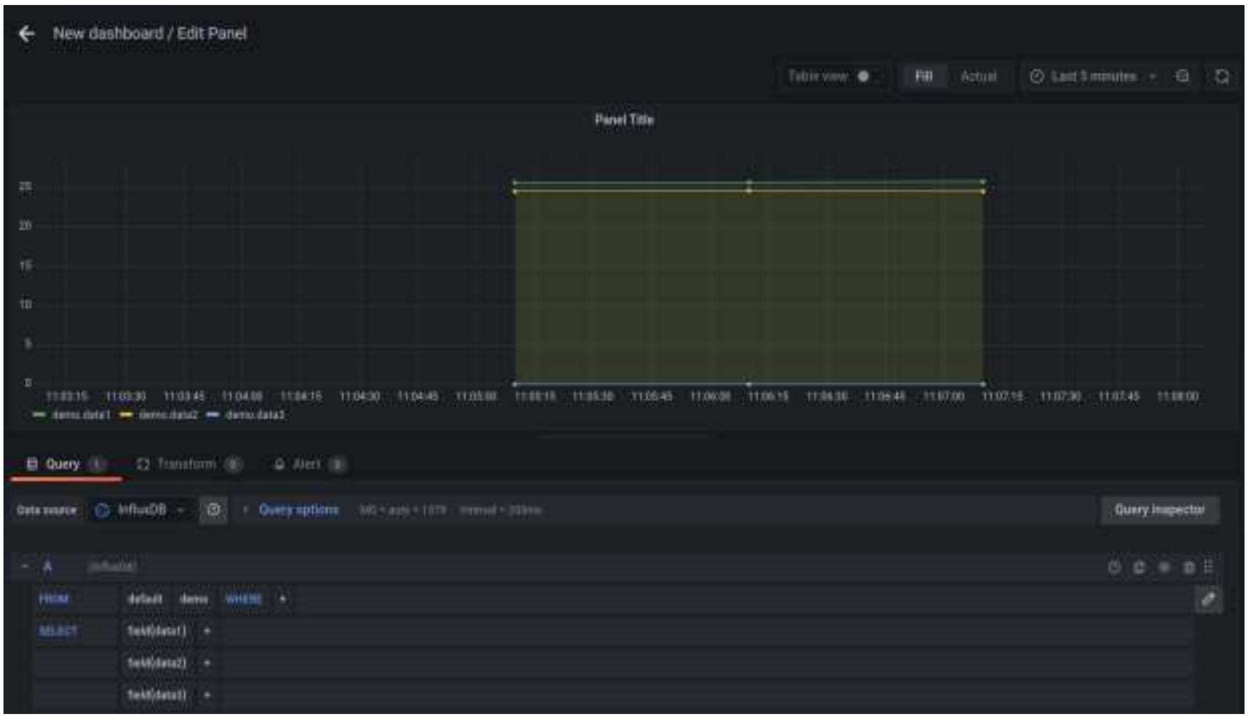


Figure 26: Visualization tool after configuration

The three mentioned test scenarios in Section 3 will be deployed and tested in D5.5 where it is explained how to deploy them via Swarmpit.

6 Conclusions

In this deliverable, we provided a reference platform instantiation of the PLATOON reference architecture which serves as a proof-of-concept for the PLATOON reference architecture.

We introduced the components and hardware used for the platform instantiation. The provided hardware specifications serve as a reference point for choosing appropriate hardware components for a PLATOON setup on an enterprise infrastructure.

In order to guarantee the interoperability and flexibility of the PLATOON reference architecture instantiation, we developed the reference platform instantiation based on Swarmpit. To enable system administrators to setup a PLATOON instantiation on their own infrastructure, we provided step-by-step installation details on how to setup Swarmpit and the required dependencies. Once installed and running, the corresponding docker-compose.yml files can be used to deploy the respective components required.

As a result of this task two major outcomes are obtained. On the one hand, the developed reference platform instantiation serves as a testbed for doing an integration test in D5.5 of the different open-source components developed in WP2, WP3 and WP4 before their final implementation and validation in large scale pilots in WP6. On the other hand, with the information provided in this deliverable, companies interested in joining the ecosystem can setup a base environment for future deployment of different components. Thanks to the modular nature of the different components, the PLATOON reference architecture can be implemented in different ways adapting to the circumstances of the respective company. As such, the reference platform instantiation serves as an entry point and main building block for a successful start in using PLATOON.

7 References

[1] Scientific Data Management Group at TIB, “SDM-TIB/SDM-RDFizer,” 2022. [Online]. Available: https://github.com/SDM-TIB/SDM-RDFizer. [Accessed 27 February 2022].

[2] Scientific Data Management Group at TIB, “SDM-TIB/DeTrusty,” 2022. [Online]. Available: https://github.com/SDM-TIB/DeTrusty. [Accessed 27 February 2022].

[3] OpenLink Software, “OpenLink Software: Virtuoso Homepage,” [Online]. Available: https://virtuoso.openlinksw.com/. [Accessed 25 March 2022].

[4] C. F. Draschner, J. Lehmann and H. Jabeen, “DistSim - Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs,” in *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, 2021.

[5] F. Bakhshandegan Moghaddam, C. Draschner, J. Lehmann and H. Jabeen, “Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor,” in *Proceedings of the 17th International Conference on Semantic Systems, SEMANTICS 2021*, 2021.

[6] C. F. Draschner, C. Stadler, F. Bakhshandegan Moghaddam, J. Lehmann and H. Jabeen, “DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs,” in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

[7] F. Bakhshandegan Moghaddam, J. Lehmann and H. Jabeen, “DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs,” 2022. [Online]. Available: https://hajirajabeen.github.io/publications/ICSC_2022_DistAnomalies_CR.pdf. [Accessed 18 March 2022].

[8] F. Bakhshandegan Moghaddam, C. F. Draschner, J. Lehmann and H. Jabeen, “Semantic Web Analysis with Flavor of Micro-Services,” in *LAMBDA Big Data Analytics Summer School. Doctoral Workshop*, 2021 forthcoming.

[9] Docker, Inc., “docker,” 2021. [Online]. Available: https://www.docker.com. [Accessed 17 December 2021].

[10] A. Heddings, “What Does Docker Do, and When Should You Use It?,” 24 November 2020. [Online]. Available: https://www.cloudsavvyit.com/490/what-does-docker-do-and-when-should-you-use-it/. [Accessed 27 February 2022].

[11] Topmonks, “Swarmpit. Operate your docker infrastructure like a champ.,” 2020. [Online]. Available: https://swarmpit.io. [Accessed 27 February 2022].

[12] The Swarmpit contributors, “Swarmpit GitHub Page,” 2021. [Online]. Available: https://github.com/swarmpit/swarmpit. [Accessed 27 February 2022].