This project has received funding from the European's Union Horizon 2020 research innovation program under grant agreement No. 769355

D6.4 PIXEL data acquisition, information hub and data representation v2

Deliverable No.	D.6.4	**Due Date**	30/06/2020
Type	Other	**Dissemination Level**	Public
Version	1.0	**Status**	Final
Description	Deliverable D6.4 is the second version of the "PIXEL data acquisition, information hub and data representation" deliverable. Both deliverables are the main asset of software documentation for this project, including data sources, collecting mechanisms, technologies, protocols, the operational analytics engine, operational tools and the visualization and notification module. This version contains the design of integrated information system, requirements of the platform, installation, and user guide.		
Work Package	WP6		

Authors

Name	Partner	e-mail
José A. Clemente Pérez	P02 PRO	jclemente@prodevelop.es
Julian Martinez Montes	P02 PRO	jmartinez@prodevelop.es
Miguel Ángel Llorente	P02 PRO	mllorente@prodevelop.es
Ismael Torres	P02 PRO	itorres@prodevelop.es
Benjamin Molina	P01 UPV	benmomo@upvnet.upv.es
Flavio Fuart	P03 XLAB	Flavio.fuart@xlab.si
Damjan Murn	P03 XLAB	Damjan.murn@xlab.si
Dejan Štepec	P03 XLAB	dejan.stepec@xlab.si
Tomaž Martinčič	P03 XLAB	Tomaz.martincic@xlab.si
Marc Despland	P06 ORANGE	marc.despland@orange.com

History

Date	Version	Change
10-May-2020	0.1	ToC and task assignments
15-May-2020	0.2	ToC update
15-June-2020	0.3	Merge contributions
17-June-2020	0.4	Software contributions
19-June-2020	0.9	Internal review
25-June-2020	0.9.1	Update Security section
29-June-2020	1.0	Ready for submit

Key Data

Keywords	ICT framework, data acquisition, cybersecurity, operational tools, visualisation, dashboards, User Manual, PIXEL Platform
Lead Editor	Jose A. Clemente, P02 PRO
Internal Reviewer(s)	P12 ASPM, P07 CREOCEAN

Abstract

PIXEL Enabling ICT Infrastructure framework is one of the key outcomes of PIXEL activities. The main goal is to compose a complete data-centric port solution, allowing data-level interoperability of different systems, including legacy industrial and port operations systems.

This framework provides solid technological foundations for efficient and cost effective execution of models, simulations and predictions that are part of the PIXEL environmental impacts assessment model, to be used by ports of the future for efficient management and tackling environmental issues.

The most important asset of this deliverable is the provision of software, installation and user guide for the data acquisition layer, information hub, operational tools, dashboard & notifications and security & privacy modules. **PIXEL Data acquisition** provides software mechanisms to enable appropriate data acquisition in different port areas, from logistic agents and public data sources. Data acquisition is based on FIWARE, a curated framework of open source platform components for smart solutions, financed through several EU research programmes. **PIXEL Information Hub** is the primary information source in all port related activities and is being designed to strengthen the capacity and accuracy of port of the future logistics processes and to maintain a high level of service and offer a system which will is in line with the needs and expectations of users. **PIXEL Operational tools** enable model-based simulations and analysis of data gathered and fused in the PIXEL Information Hub to provide more flexible operations and create decision-making tools, resulting from other PIXEL activities. **PIXEL Integrated Dashboard and Notifications** provide the visual environment to show in a single dashboard the different KPIs, user interfaces for the operational tools and the configuration and management tools needed to control other PIXEL framework components. **PIXEL Security and Privacy** is a transversal activity that provides end-to-end security for the PIXEL platform by deploying basic cybersecurity mechanisms for all other ICT components.

PIXEL Enabling ICT Infrastructure framework has been designed to support generalization to other ports or terminals with similar needs.

Statement of originality

This document contains material, which is the copyright of certain PIXEL consortium parties, and may not be reproduced or copied without permission. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.
The information contained in this document is the proprietary confidential information of the PIXEL consortium (including the Commission Services) and may not be disclosed except in accordance with the consortium agreement.
The commercial use of any information contained in this document may require a license from the proprietor of that information.
Neither the project consortium as a whole nor a certain party of the consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information.
The information in this document is subject to change without notice.

The content of this report reflects only the authors' view. The Innovation and Networks Executive Agency (INEA) is not responsible for any use that may be made of the information it contains.

Table of contents

List of tables

List of figures

List of acronyms

Acronym	Explanation
AIS	Automatic Identification System
API	Application Programming Interface
ARPA	Agenzia regionale per la protezione ambientale
CPU	Central Processing Unit
CRUD	Create, Read, Update and Delete
CSV	Comma-Separated Values
CURL	Client URL
DAL	Data Acquisition Layer
DOA	Description of Action
ETD	Estimated Time of Departure
FAIR	Facility for Antiproton and Ion Research
GPMB	Grand port maritime de Bordeaux
Gogs	Go Git Service
GUI	Graphic User Interface
HTTP	Hypertext Transfer Protocol
ICT	Information and communications technology
ID	Identity
IDM	Identity Manager (FIWARE)
IH	Information Hub
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LTS	Long Term Storage
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
NGSI	Next Generation Sensors Initiative
NMEA	National Marine Electronics Association
OS	Operating System
OT	Operational Tools
PCS	Port Community System
PA	Predictive Algorithm
PAP	Policy Administration Point
PAS	Port Activity Scenario
PDP	Policy Decission Point
PEI	Port Environmental Index
PEP	Policy Execution Point
PIXEL	Port IoT for Environmental Leverage
PMS	Port Management System
PMIS	Port Management Information System
REST	Representational state transfer
SILI	Sistema Informativo Logistico Integrato
SotA	State-of-the-Art
TCP	Transmission Control Protocol
UI	User Interface
UUID	Universally Unique Identifier
WP	Work Package
WSDL	Web Services Description Language

1. About this document

This deliverable is the second version of the “PIXEL data acquisition, information hub and data representation”, it describes and provides the work that has been done in the technical tasks of WP6, Enabling ICT (Information and communications technology) infrastructure framework. The deliverable consists of:

- The reporting part (this document), where it is described the components developed including the installation and user guide.
- The final version of the source code for software components.

This deliverable, together with 6.5, is intended to be useful guide when installing, starting up and using the PIXEL platform in the different pilots, as well as to be able to expand it if needed

1.1. Deliverable context

Table 1: Deliverable context

Keywords	Lead Editor
Objectives	*Objective 1:* Enable the IoT-based connection of port resources, transport agents and city sensor networks. This deliverable provided the final software for IoT enablement and interconnection of different data providers including Port information systems, sensors and third parties’ providers like Marine traffic.
	Objective 2: Achieve an automatic aggregation, homogenization and semantic annotation of multi-source heterogeneous data from different internal and external actors. This deliverable provides the implementation of the reference model provided in D6.1. A lot of effort has been done in the Data acquisition and Information Hub Components to provide a generic data aggregation mechanism and homogenization.
	Objective 3: Develop an operational management dashboard to enable a quicker, more accurate and in-depth knowledge of port operations. This deliverable provides the dashboard component that allows the users to easily interact with the PIXEL platform, resulting from T6.4.
	Objective 4: Model and simulate port operations processes for automated optimisation. D6.4 provides the operational tools component resulting from T6.5. This component gives high-level technological support for the configuration and execution of predictive algorithms models developed in WP4.
	Objective 5: Develop predictive algorithms. Similar as for Objective 4, D6.4 provides the operational tools component resulting from T6.5. The Operational tools provide a generic framework for the setup and execution of predictive algorithms.
Work plan	This deliverable is the result of work performed from M7 to M26 on tasks T6.2 – PIXEL Data Acquisition, T6.3 - PIXEL Information Hub, Task 6.4 - PIXEL Operational Tools, T6.5 - PIXEL Integrated Dashboard and Notification, Task 6.6 – PIXEL Security and Privacy.

Milestones	This is the final deliverable for verification of the MS7 ICT solution developed.
Deliverables	This deliverable is the second version of the deliverable 6.3, and provides the final version of the PIXEL Platform. In addition to the D6.3, the following deliverables have been taken into account: • Requirements, scenarios and use cases defined in D3.2, D3.3 and D3.4 have been used to identify relevant ICT-related tasks. • Models (D4.2) and Predictive algorithms (D4.4) This deliverable, together with the D6.5 will be used for the development of the pilots, whose description will be made in the deliverables D7.2 and D7.3
Risks	**WT5#6 Technical activities are not completed on time, are not aligned with the main objective, are not accurate or present a lack of consistency.** This deliverable shows that technical activities related to T6.2 – T6.6 have been executed in a timely fashion in accordance with the architecture proposed in D6.1. **WT5#14 Due to harshly divergences between formats of output/input data of ICT systems to integrate, the development can be delayed or paralyzed, and some extra effort will be needed to carry out the project.** Particular attention is being devoted to the analysis and definition of data models in WP6. While the generic principles have been provided in D6.1, this deliverable provides a more detailed list of data entities identified in PIXEL. **WT5#15 IoT components have security vulnerabilities.** A lot of effort has been dedicated defining the security component of the PIXEL platform to mitigate this risk.

1.2. The rationale behind the structure

This report describes the work performed in T6.2 – T6.6 of PIXEL. Except the introduction and conclusion, each section provides specific content defined in the DoA deliverable description. Topics that are covered in each PIXEL component are further split in sub-sections for each of those components:

1. PIXEL Data Acquisition
2. PIXEL Information Hub
3. PIXEL Operational Tools
4. PIXEL Integrated Dashboard and Notification
5. PIXEL Security and Privacy

This report consists of the following sections:

1. About this document: Deliverable context in relation to the PIXEL DoA, work packages, tasks and other deliverables.
2. Introduction: Relation with PIXEL objectives, uses cases and requirements.
3. PIXEL Platform Software. Description of the software release that is part of this deliverable. The section provides the description of files provided for each PIXEL component.

4. PIXEL Platform: Provides the description, the installation and user’s guide of the components.
5. Conclusions: Closing remarks.

1.3. Version-specific notes

This is the second version of the “PIXEL data acquisition, information hub and data representation”. This report and software release provides the results of WP6 performed until M26 of the project.

2. Introduction

Developments of the PIXEL ICT Infrastructure Framework are driven by real needs of the ports involved in the project. Those needs and requirements are the key results of WP3. In order to keep development in line with overall PIXEL objectives, each technical deliverable provides an introductory section where the relation with objectives, use cases and requirements is defined. The following sections in this chapter provide that overview for software modules developed in WP6.

2.1. Relation with PIXEL objectives and use cases

All work described in D6.4 and performed as part of tasks T6.2-T6.6 aims to fulfil objectives 1, 2, 3, 4 and 5, which are listed in section 1.1 of this document.

This deliverable presents the final software version of the different main architectural modules. The development of DAL (Data Acquisition Layer) has the potential to connect (Obj1) different data sources to a common broker in a standardized way. Here the concept of data source is wide and applies not only to sensors in ports but also to open data and some existing port applications (e.g. vessel calls) providing data. The DAL represents the first level of data integration and support, by means of NGSI agents, automatic aggregation (Obj2) of connected data sources.

The second level of data integration is represented by the IH (Information Hub), where heterogeneous port data is brought to a common space (hub) and exposed in a harmonized (Obj2) way. To achieve this objective, IH exposes and API (Application Programming Interface) for discovering, querying and storing data. Such port data has also been converted into more common and useful semantic data formats (Obj2) that will later be integrated in higher level applications: some examples will be provided in PIXEL with the Dashboard and Operational Tools (Obj4, Obj5), but in fact the objective is general for future applications developed by ports and/or port agents.

The IH (and in fact also the DAL) is designed not only to treat data but also to provide an interoperable framework for actors, such as port agents and city networks (Obj1) to exchange data and optimize resources and common policies.

To appropriately manage (Obj3) and exploit all available data in the IH, a dashboard has been developed, which provides an operational UI (User Interface). Such a dashboard can represent data in different ways according to various port profiles (e.g. environmental manager, gate manager, etc.) and their access rights. Moreover, it encompasses additional features such as notifications, and Operational Tools, which can use the data to manage models and predictive algorithms with the aim of providing better insights of port operations (Obj3, Obj4, and Obj5).

To see real examples of input data used in the pilots under development, you can review the section 2.1 of the deliverable 6.3. Such inputs should be acquired from multiple data sources (port data sources, open data sources, sensors, …), stored and represented properly in the IH for further processing and visualization.

2.2. Relation to requirements

Requirements, gathered in WP3, that are related to the ICT framework are one of the main drivers of development of the PIXEL platform. A full list of requirements directly or indirectly related to the implementation of PIXEL software components can be reviewed in the D6.3 - Table 5.

3. PIXEL Platform Source Code

All software components produce under PIXEL platform are available through online code repository.

3.1. PIXEL Git repository

The version control system being used at PIXEL is Gogs (Go Git Service). This implementation is based on Git. Gogs, is written in Go language and is open source (MIT license). It is multiplatform software that has an interface very similar to GitHub.

3.2. Software release and documentation

The different software components in PIXEL are being developed under two main objectives:

- *Open source approach*. most of the developed modules will be released as open source, while the source code is available in an internal Git repository, so that any partner within the Consortium can get an easy access and can potentially contribute (as a tester or developer). In the future, once the final releases of each software module are available and have been tested in the pilots, the aim is to port the source code to a public repository (e.g. GitHub[1]).
- *Containerization*. Though services may be implemented in different languages, one important goal is to encapsulate the different modules into (Docker) containers, so that they facilitate the deployment within the PIXEL platform. In fact, this is a best practice for continuous integration, delivery and deployment. Similar to the previous step, the aim is to port the implemented containers into a public repository (e.g. Docker Hub[2]).

Table 2: Software release overview

Component	Description	Repository URL	D6.4 Section
PIXEL Data Acquisition Layer	This repository contains the files needed to install the PIXEL DAL	https://gitpixel.satrdlab.upv.es/marc.despland/DataAcquisitionLayer	PIXEL Data acquisition
PIXEL Information Hub	Repository that provides Docker deployment script for deploying Information Hub	https://gitpixel.satrdlab.upv.es/xlab/information-hub-aggregator	PIXEL Information Hub
PIXEL Operational Tools	Files needed to deploy Operational Tools Component	https://gitpixel.satrdlab.upv.es/benmomo/otpixel-v2/src/master	PIXEL Operational Tools
PIXEL Dashboard & Notifications	This repository contains all the files needed to install the PIXEL Dashboard & Notifications Layer	https://gitpixel.satrdlab.upv.es/jclemente/dashboardPIXEL_v2/src/master	PIXEL Integrated Dashboard and Notifications
PIXEL Security & Privacy	This repository contains the documentation and files needed to install the PIXEL Security Layer	https://gitpixel.satrdlab.upv.es/marc.despland/Security	PIXEL Security
Installation of the pilot	Files necessary for the installation of GPMB pilot	https://gitpixel.satrdlab.upv.es/marc.despland/Installation	Pilots Installation

[1] https://github.com/

[2] https://hub.docker.com/

4. PIXEL platform description

4.1. Introduction

The objective of PIXEL is to provide a software solution / platform where making use of IoT devices and IT data sources exploit the data in the ports of the future.

In the end PIXEL is the first smart, flexible and scalable solution for reducing environmental impacts while enabling the optimization of operations in port ecosystems through IoT.

Next figure depicts the global architecture of PIXEL including the interaction with the different data sources and the output to the devices that will work with PIXEL.

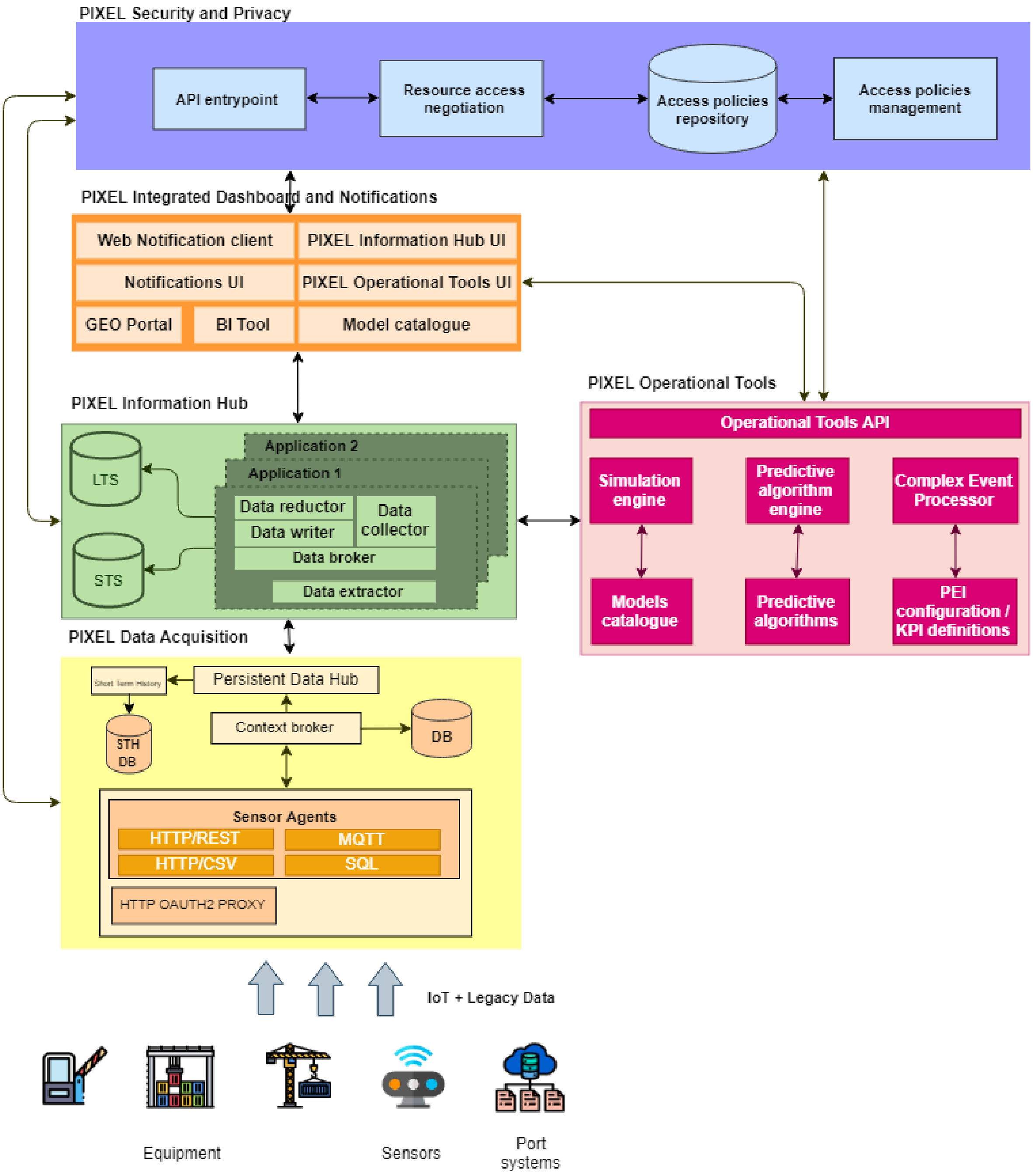


Figure 1: Global architecture of PIXEL

4.2. Installation Guide

This section will illustrate the installation process of PIXEL. It will start with the necessary prerequisites. To continue talking about the different modules, where a brief summary of each component will be made, it will mention the technologies used in each component, its installation process, configuration (in case something needs to be configured). For each component, an issues & solution section is provided.

4.2.1. Pre-requirements

4.2.1.1. Installation of docker and docker-compose

Table 3: Installation of Docker and docker-compose

Installation of docker
sudo apt-get update → Update the apt package index *sudo apt-get install apt-transport-https* → Install packages to allow apt to use a repository over HTTPS *sudo apt-get install ca-certificates* → Allows the system (and web-browser) to check security certificates *sudo apt-get install curl* → Install curl (tool for transferring data) *sudo apt-get install software-properties-common* → Install scripts needed for managing software *curl -fsSL get.docker.com -o docker.sh* → Download from get.docker.com the docker.sh script to install the latest version of Docker Engine *sudo sh get-docker.sh* → Execute script get-docker.sh
Test the installation of Docker
docker version →Check docker version. To verify the installed Docker version number *sudo usermod -aG docker ${USER}* →Add user to the docker group. Allow to execute docker without sudo
Test the execution of Docker
docker run hello-world → To check if you can access and download images from Docker Hub. The result will tell you that Docker is working properly
Installation of docker-compose
sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose →Download the docker-compose version 1.22 *sudo chmod +x /usr/local/bin/docker-compose* → Apply executable permissions to the binary
Test the installation of Docker-compose
docker-compose --version → Check that the version has been installed correctly

4.2.2. PIXEL Data acquisition

4.2.2.1. Summary

The main purpose of the Data Acquisition Layer is to interface the external data sources to the PIXEL Information Hub and to convert the original and heterogeneous data format to PIXEL Data Models.

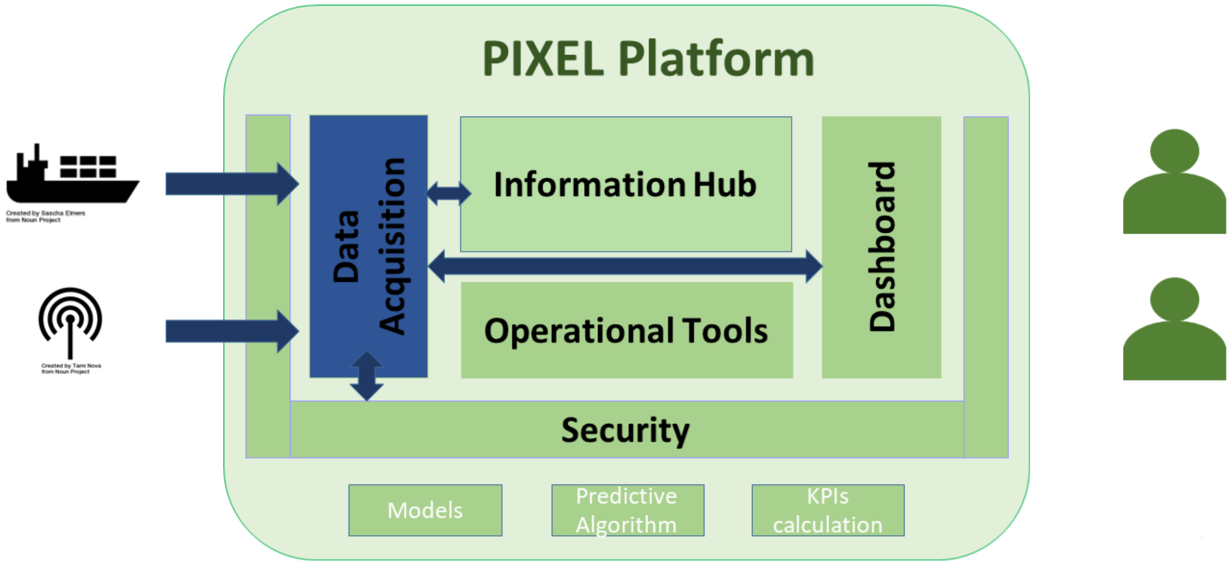


Figure 2: Purpose of Data Acquisition Layer

Data Acquisition exposed API to Dashboard to allow admin to manage the different NGSI Agents, and it interacts with PIXEL Security to protect the NGSI Agents that exposed API.

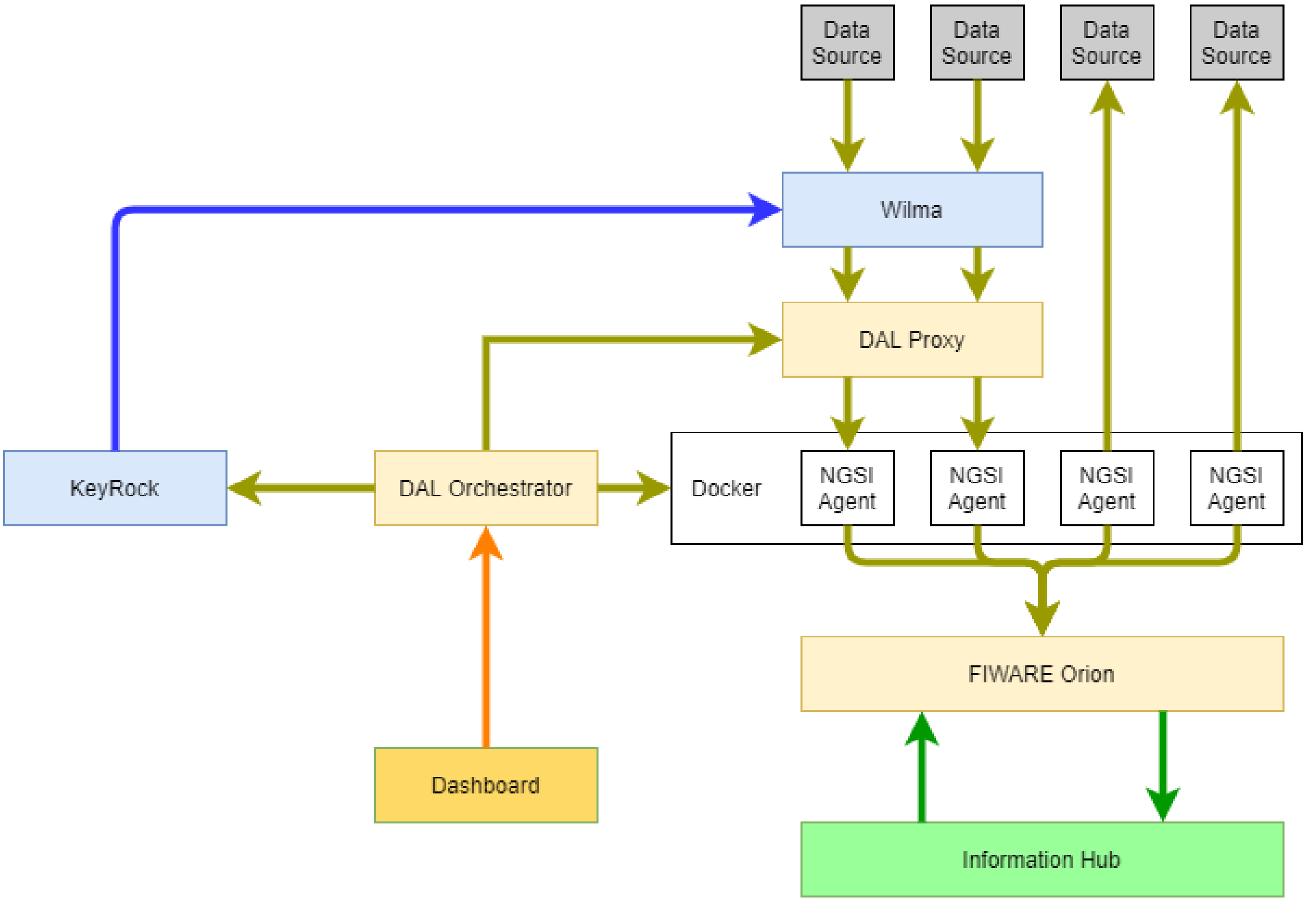


Figure 3: Interaction of DAL with the rest of PIXEL components

Main activity of DAL is NGSI Agents pushing Data to Orion. Then Information Hub subscribe to new data using the subscription API of FIWARE Orion. It is then notify using a call-back when new data arrive.

The API of FIWARE Orion is well documented by FIWARE. The documentation on the NGSIv2 API is available here, and documentation on how to use it is available here.

To facilitate NGSI Agents management, Data Acquisition provides an orchestrator that pilot the creation of the NGIS Agents using a Docker Interface. DAL Orchestrator also communicates with DAL Proxy to automatically exposed new daemon NGSI Agents behind WILMA. DAL Orchestrator exposes its API to the Dashboard to allow creation of admin UI.

DAL Orchestrator also communicates with Keyrock to manage the permissions on WILMA for each NGSI Agents that exposed an API through the DAL Proxy.

4.2.2.2. How to install

The full installation process relies on docker-compose.

The process of the installation is split in three different steps:

1. Configuration : You must adapt the docker-compose-*.yaml file and feed the different secrets value
2. Build : it is the docker-compose build process, a helper script ./build.sh is provided
3. Installation : it is the docker-compose up process, a helper script is provided

Different installation flavour is provided:

- full : it will install all the component
- orion : it will install only orion component
- dal : it will install dal-orchestrator and dal-proxy

By default, the full option is selected

Table 4 Installation of DAL

Installation of full DAL
cd DataAcquisitionLayer *./build.sh* *./install.sh*

4.2.2.3. Configuration

Platform specifics configuration are done using the docker-compose-*.yaml file and with the configuration of the secrets files for each component

Table 5: Environment variables per each service in docker-compose file

Environment variables	
Orion Database	
MONGO_INITDB_ROOT_USERNAME=mongo	The admin user of the mongo database
MONGO_INITDB_DATABASE=admin	The name of the admin database
MONGO_INITDB_ROOT_PASSWORD_FILE	The path to the admin password secret file (usually do not change this value)
ORIONDB_PASSWORD_FILE	The internal path to the secret file containing the database password (usually do not change this value)
Orion	
DB_HOST=dal-orion-db	The database host

DB=orion	The name of the main database
DB_USER=orion	The user to use to connect
DB_PASSWORD_FILE	The internal path to the secret file containing the database password (usually do not change this value)
DAL-Proxy	
API_LISTEN_PORT=8080	The port use by the proxy to listen for management API
API_LISTEN_IP	The IP address use by the proxy to listen for management API
PROXY_LISTEN_PORT	The port use to listen for proxy request
PROXY_LISTEN_IP	The IP use to listen for proxy request
ORCHESTRATOR_API_URL	The URL of the DAL -Orchestrator
ORCHESTRATOR_TOKEN_FILE	The secret file containing the token to access DAL-Orchestrator
PROXY_API_TOKEN_FILE	The secret file containing the token to access DAL-Proxy management API
DAL-Orchestrator	
SCHEMA_REPOSITORY_URL	The URL of the public repository of data models schema
SCHEMA_REPOSITORY	The container internal folder where the Data Models repository is mounted on (do not change it)
NGSIAGENT_NETWORK	The docker network to use to create new NGSI Agent
NGSIAGENT_KEY=pixel	The key to identified NGSI Agent image (do not change it)
PROXY_API_URL	The URL of the Proxy management API
ORCHESTRATOR_LISTEN_PORT	The port the orchestrator listens to
ORCHESTRATOR_LISTEN_IP	The IP address the orchestrator listens to
ORION_API	The URL of the ORION API
ORCHESTRATOR_TOKEN_FILE	The secret file containing the token to access DAL-Orchestrator
PROXY_API_TOKEN_FILE	The secret file containing the token to access DAL-Proxy management API
Secrets	
Orion Database	
orion.db.password	The password for the user orion (random)
orion.db.root.password	The password for the admin user (random)
Orion	

orion.db.password	The password for the user orion (random)
DAL-Proxy	
dal.proxy.api.token	The token to secure Proxy API access (random)
dal.orchestrator.api.token	The token to secure Orchestrator API access (random)
DAL-Orchestrator	
dal.proxy.api.token	The token to secure Proxy API access (random)
dal.orchestrator.api.token	The token to secure Orchestrator API access (random)

4.2.2.4. Component status

Table 6: How to check the status of each service once has been deployed

How do you verify the service has been correctly deployed?
Orion
Executing "docker-compose ps" to check that the service is in "Up" state. Check the version API : curl http://<ip>:<port>/version
Mongo - Database
With the command "docker-compose ps" you check that the service is in "Up" status, and with the command "telnet IP 27001" that the TCP port is listening
DAL-Proxy
With the command "docker-compose ps" you check that the service is in "Up" status. Use check status API request : curl http://<ip management>:<port management>/api/status
DAL-Orchestrator
With the command "docker-compose ps" you check that the service is in "Up" status. Use check status API request : curl http://<ip management>:<port management>/api/status

4.2.2.5. Issues & Solution

DAL-Proxy and DAL-Orchestrator synchronize them self at start-up. We do not care in which order they start, so you can restart them if you have any issue with them.

For Orion, rely on the official documentation: https://fiware-orion.readthedocs.io/en/master/

4.2.3. PIXEL Information Hub

4.2.3.1. Summary

As stated in D6.2, the Information Hub (IH) is a functional block in charge of centralising all the data retrieved from DAL, homogenising and storing in a database capable to support big queries and scale horizontally. Unlike the DAL, the Information Hub is designed to be high performant and scalable, and the data is stored to support long-term queries. This is considered the central storage point of the IoT solution in PIXEL and is the block that replies the queries from other functional blocks (such as Operational Tools or

Dashboards) and externals (API). The IH's main components are a high-performance data broker and a NoSQL database, although it contains accessory components that support its correct functioning.

The following diagram depicts the architecture of Information Hub (source: D6.2):

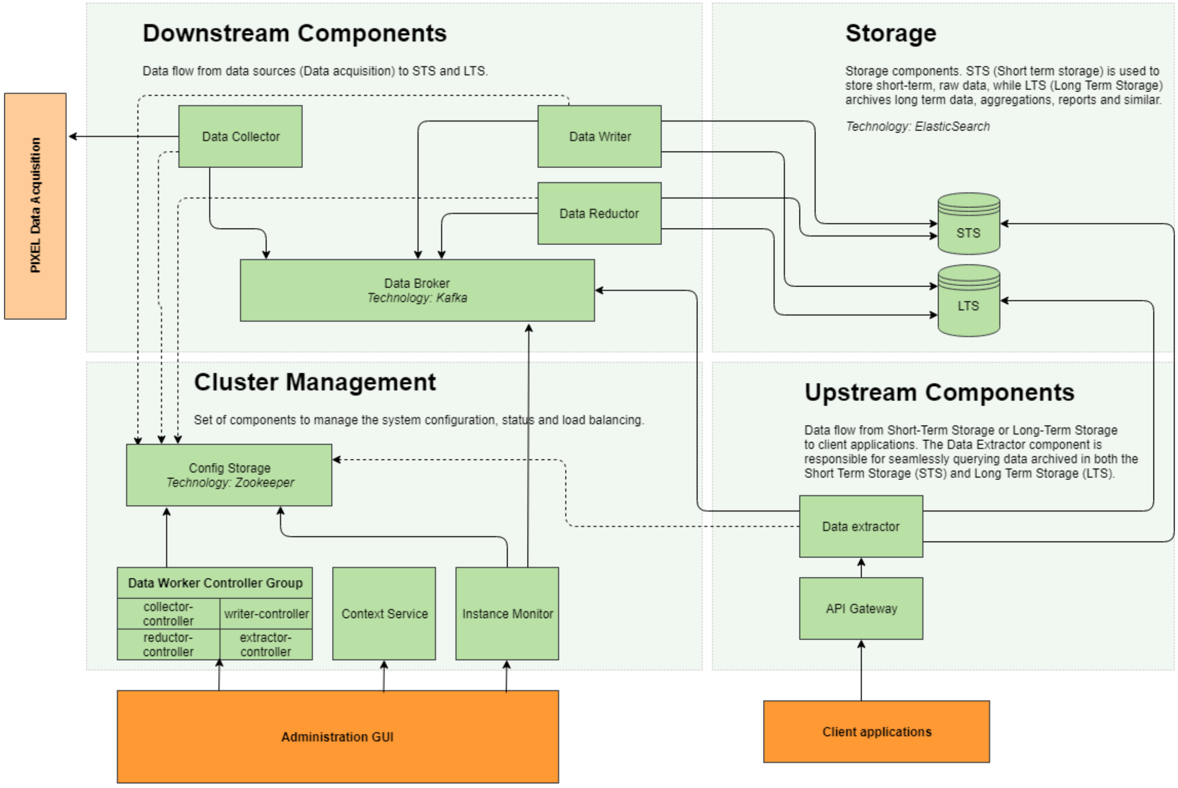
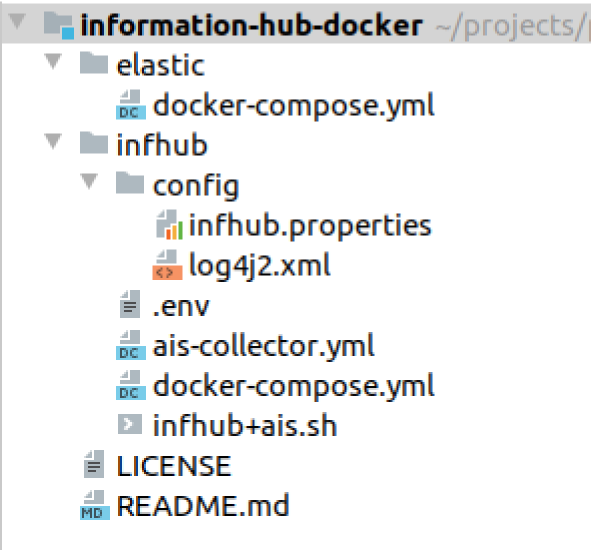

Downstream Components
Data flow from data sources (Data acquisition) to STS and LTS.
PIXEL Data Acquisition
Data Collector
Data Writer
Data Reductor
Data Broker
Technology: Kafka
Storage
Storage components. STS (Short term storage) is used to store short-term, raw data, while LTS (Long Term Storage) archives long term data, aggregations, reports and similar.
Technology: ElasticSearch
STS
LTS
Cluster Management
Set of components to manage the system configuration, status and load balancing.
Config Storage
Technology: Zookeeper
Upstream Components
Data flow from Short-Term Storage or Long-Term Storage to client applications. The Data Extractor component is responsible for seamlessly querying data archived in both the Short Term Storage (STS) and Long Term Storage (LTS).
Data extractor
API Gateway
Data Worker Controller Group
collector-controller
writer-controller
reductor-controller
extractor-controller
Context Service
Instance Monitor
Administration GUI
Client applications

Figure 4: PIXEL Information Hub architecture

4.2.3.2. How to install

Information Hub is distributed as a set of Docker images and can be deployed using Docker Compose tool. The information-hub-docker repository provides Docker Compose projects for installing Information Hub and Elasticsearch. The installation is split into two parts (two Docker Compose projects) - Information Hub and Elasticsearch which are located in the *infhub* and *elastic* folder respectively. The *infhub* Docker Compose project installs Information Hub together with its prerequisites Apache Kafka and ZooKeeper. The *elastic* Docker Compose project installs Elasticsearch and Kibana version 7.2.0. Alternatively, a custom installation of Elasticsearch can be used (considering that the supported version of Elasticsearch is 7.2.x). All services of Information Hub are installed to the single machine and likewise Elastic services are installed to a single machine which can be the same or different than for Information Hub.



```
information-hub-docker ~/projects/
    elastic
        docker-compose.yml
    infhub
        config
            infhub.properties
            log4j2.xml
        .env
        ais-collector.yml
        docker-compose.yml
        infhub+ais.sh
    LICENSE
    README.md
```

Figure 5: Information-hub-docker

The Information Hub Docker Compose project also includes the AIS Data Collector service which collects AIS data from AISHub data sharing service. The installation of this service is optional and requires an AISHub membership.

The Docker Compose deployment of Information Hub consists of the following services:

Table 7: Docker Compose deployment of Information Hub

Service	Docker Image
Apache ZooKeeper	wurstmeister/zookeeper
Apache Kafka	wurstmeister/kafka
Orion Data Collector	docker.pixel-ports.eu/information-hub/orion-data-collector
Controller	docker.pixel-ports.eu/information-hub/information-hub-controller
Data Writer	docker.pixel-ports.eu/information-hub/srv-data-writer
Data Monitor	docker.pixel-ports.eu/information-hub/srv-data-monitor
Data Extractor	docker.pixel-ports.eu/information-hub/srv-data-extractor
Elasticsearch Proxy	docker.pixel-ports.eu/information-hub/elasticsearch-proxy
AIS Data Collector (optional)	docker.pixel-ports.eu/information-hub/ais-data-collector

Additionally, the provided Docker Compose deployment of Elasticsearch & Kibana consists of the following services:

Table 8: Docker Compose for Elasticsearch & Kibana

Service	Docker Image
Elasticsearch	docker.elastic.co/elasticsearch/elasticsearch:7.2.0
Kibana	docker.elastic.co/kibana/kibana:7.2.0

4.2.3.2.1. Requirements

Requirements for the Information Hub installation are:

- Docker
- Docker Compose
- Orion Context Broker for Orion Data Collector

The Information Hub installation has been tested on Ubuntu Linux 18.04 LTS and CentOS Linux 7.2 with Elasticsearch version 7.2.0.

4.2.3.2.2. Installing Elasticsearch

Clone or download the information-hub-docker repository to your Linux server then go into the *elastic* folder inside the repository. The provided Docker Compose file will deploy Elasticsearch and Kibana version 7.2.0. Elasticsearch can be installed on the same or different machine than Information Hub.

Prerequisites

The default operating system limits on virtual memory (*mmap counts*) is likely to be too low for Elasticsearch, which may result in out of memory exceptions. See Elasticsearch documentation for details.

To check *vm.max_map_count* value, run:

```
sysctl vm.max_map_count
```

In case the `vm.max_map_count` value is too low (recommended value is 262144), you have to increase the limit by running following command:

```
sysctl -w vm.max_map_count=262144
```

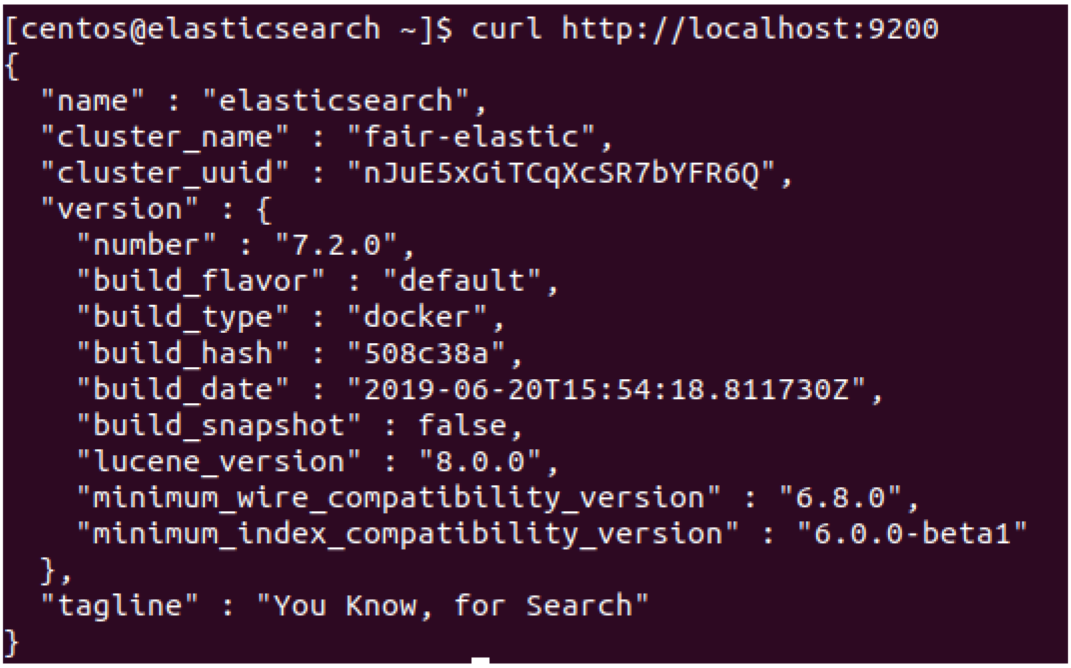
Start Elasticsearch

To start the Elasticsearch together with Kibana, run the following command:

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

Check if Elasticsearch is running by making following request:

```
curl http://localhost:9200
```



```
[centos@elasticsearch ~]$ curl http://localhost:9200
{
  "name" : "elasticsearch",
  "cluster_name" : "fair-elastic",
  "cluster_uuid" : "nJuE5xGiTCqXcSR7bYFR6Q",
  "version" : {
    "number" : "7.2.0",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "508c38a",
    "build_date" : "2019-06-20T15:54:18.811730Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 6: Checking elasticsearch is running

You can check Elasticsearch logs for errors by running:

```
docker-compose logs elasticsearch
```

In case you find following error in Elasticsearch logs:

```
elasticsearch    | ERROR: [1] bootstrap checks failed
elasticsearch    | [1]: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144]
```

you have to increase the *vm.max_map_count* limit.

Kibana dashboard is available at the following address: http://HOST:5601/app/kibana

4.2.3.2.3. Installing Information Hub

Clone or download the information-hub-docker repository to your Linux server, go into the *infhub* folder inside the repository and follow the steps below. The provided Docker Compose file deploys Information Hub together with its prerequisites Apache Kafka and ZooKeeper to a single machine.

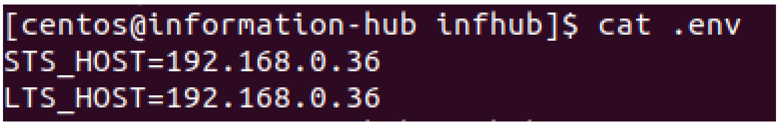
Edit Configuration Files

Before starting the deployment, check following two configuration files and adjust them to your environment if needed:

- *.env*
- *config/infhub.properties*
- *config/log4j2.xml*

***.env* configuration file**

The *.env* file contains environment variables referenced in the Docker Compose file:



```
[centos@information-hub infhub]$ cat .env
STS_HOST=192.168.0.36
LTS_HOST=192.168.0.36
```

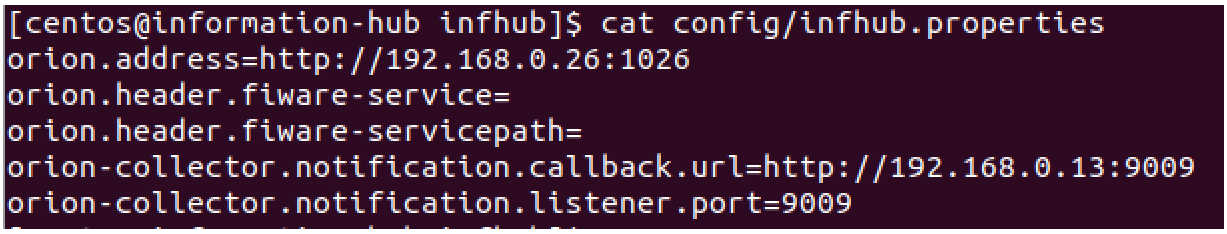
Figure 7: Environmental variables

Configuration settings are:

- *STS_HOST*: IP address or hostname of Elasticsearch in the role of Short-Term Storage
- *LTS_HOST*: IP address or hostname of Elasticsearch in the role of Long-Term Storage

infhub.properties configuration file

The *infhub.properties* configuration file contains configuration settings for Information Hub:



```
[centos@information-hub infhub]$ cat config/infhub.properties
orion.address=http://192.168.0.26:1026
orion.header.fiware-service=
orion.header.fiware-servicepath=
orion-collector.notification.callback.url=http://192.168.0.13:9009
orion-collector.notification.listener.port=9009
```

Figure 8: Properties of infhub.properties configuration file

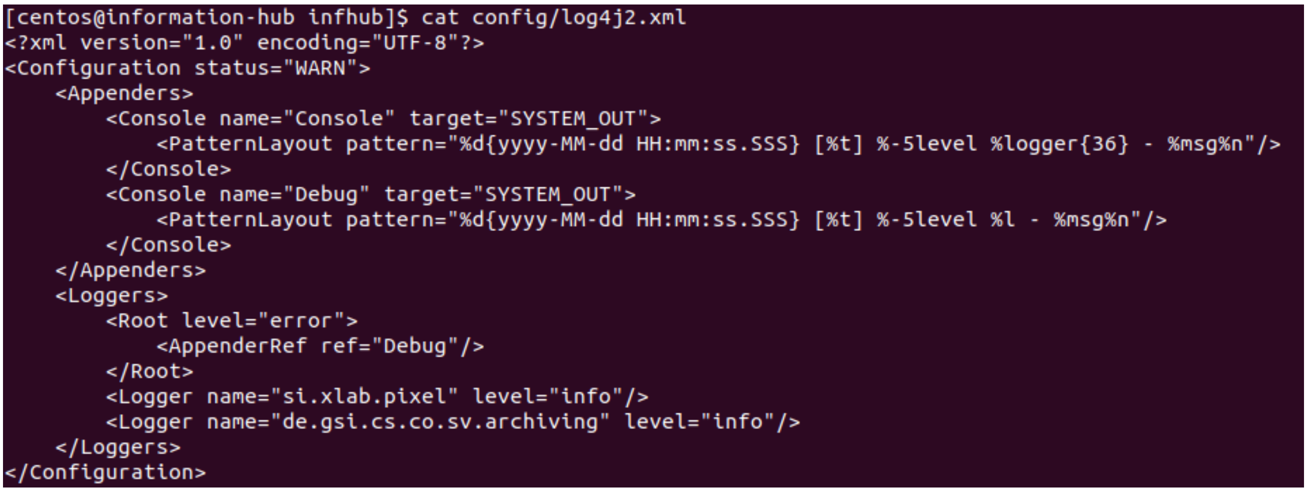
Configuration settings are:

- *orion.address*: Orion Context Broker endpoint address.
- *orion.header.fiware-service*: value of the `Fiware-Service` HTTP header to use when sending requests to Orion Context Broker. This header is used by Orion Context Broker in multi-tenant / multi-service deployment to identify the service / tenant.

- *orion.header.fiware-servicepath*: value of the `Fiware-ServicePath` HTTP header to use when sending requests to Orion Context Broker. This header is used by Orion Context Broker to define the scope of an entity.
- *orion-coll.notification.callback.url*: address of Information Hub endpoint accepting notification messages from Orion Context Broker. Default value `*http://172.17.0.1:9009*` can be used if Orion is deployed to the same machine as Information Hub.
- *orion-coll.notification.listener.port*: callback address of Orion Data Collector notification listener. The Collector subscribes to notifications from Orion Context Broker providing this address of an endpoint where Orion Context Broker should send notification messages to.
- *orion-coll.notification.listener.port*: local port on which Orion Data Collector notification listener should listen.

log4j2.xml configuration file

The *log4j2.xml* file contains Log4j logging configuration for Information Hub services:



```
[centos@information-hub infhub]$ cat config/log4j2.xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
        <Console name="Debug" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %l - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Debug"/>
        </Root>
        <Logger name="si.xlab.pixel" level="info"/>
        <Logger name="de.gsi.cs.co.sv.archiving" level="info"/>
    </Loggers>
</Configuration>
```

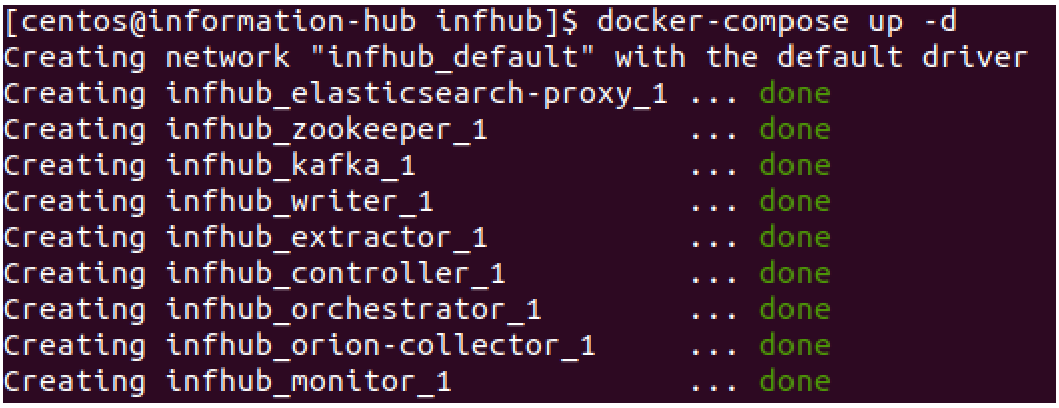
Figure 9: Logging configuration for Information Hub

The default logging level is *info* for Information Hub classes. Change to *debug* or *trace* level for more detailed logging information.

Start Information Hub

Start the Information Hub by running following command from the *infhub* folder inside the information-hub-docker repository:

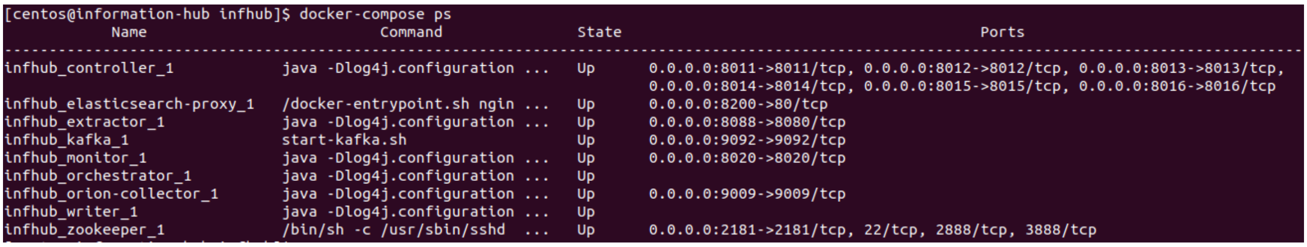
```
docker-compose up -d
```



```
[centos@information-hub infhub]$ docker-compose up -d
Creating network "infhub_default" with the default driver
Creating infhub_elasticsearch-proxy_1 ... done
Creating infhub_zookeeper_1           ... done
Creating infhub_kafka_1               ... done
Creating infhub_writer_1              ... done
Creating infhub_extractor_1           ... done
Creating infhub_controller_1          ... done
Creating infhub_orchestrator_1        ... done
Creating infhub_orion-collector_1     ... done
Creating infhub_monitor_1             ... done
```

Figure 10: Starting Information Hub

Using the ‘`docker-compose ps`’ command you can check the state of Information Hub services and verify that Information Hub has been correctly deployed:



```
[centos@information-hub infhub]$ docker-compose ps
           Name                            Command               State                                   Ports
-------------------------------------------------------------------------------------------------------------------------------------------
infhub_controller_1           java -Dlog4j.configuration ...   Up      0.0.0.0:8011->8011/tcp, 0.0.0.0:8012->8012/tcp, 0.0.0.0:8013->8013/tcp,
                                                                       0.0.0.0:8014->8014/tcp, 0.0.0.0:8015->8015/tcp, 0.0.0.0:8016->8016/tcp
infhub_elasticsearch-proxy_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8200->80/tcp
infhub_extractor_1            java -Dlog4j.configuration ...   Up      0.0.0.0:8088->8080/tcp
infhub_kafka_1                start-kafka.sh                   Up      0.0.0.0:9092->9092/tcp
infhub_monitor_1              java -Dlog4j.configuration ...   Up      0.0.0.0:8020->8020/tcp
infhub_orchestrator_1         java -Dlog4j.configuration ...   Up
infhub_orion-collector_1      java -Dlog4j.configuration ...   Up      0.0.0.0:9009->9009/tcp
infhub_writer_1               java -Dlog4j.configuration ...   Up
infhub_zookeeper_1            /bin/sh -c /usr/sbin/sshd  ...   Up      0.0.0.0:2181->2181/tcp, 22/tcp, 2888/tcp, 3888/tcp
```

Figure 11: Checking state of Information Hub

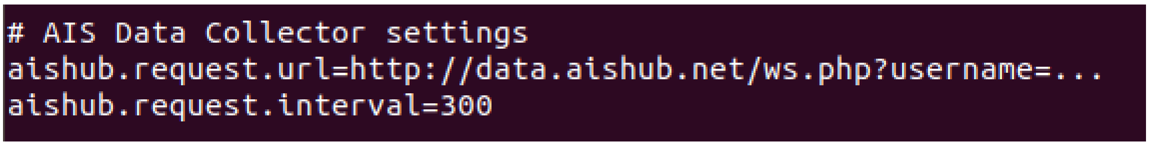
After the Information Hub has started, it will subscribe to the data sources registry at Orion Context Broker (managed by the *DAL Inquisitor*), register data sources and import initial data records.

4.2.3.2.4. Installing AIS Data Collector (Optional)

Note: AISHub membership is required. Only AISHub members are allowed to access AISHub web service and retrieve AISHub data.

Go to the *infhub* folder inside the *information-hub-docker* repository from where Information Hub has been started.

Before starting the AIS Data Collector, make sure that following configuration properties are added to the `config/infhub.properties` configuration file:



```
# AIS Data Collector settings
aishub.request.url=http://data.aishub.net/ws.php?username=...
aishub.request.interval=300
```

Figure 12: AIS Data Collector settings

The meaning of configuration properties above is:

- *aishub.request.url*: AISHub endpoint (request URL) from where AIS data can be retrieved. This URL is provided by AISHub after obtaining a membership.
- *aishub.request.interval*: interval in seconds for retrieving AIS data

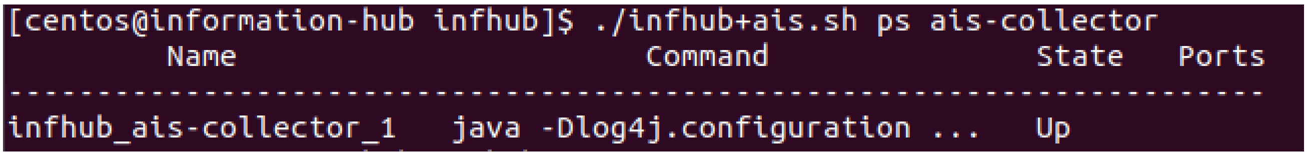
If Information Hub is already running, start AIS Data Collector by running following command:

```
./infhub+ais.sh up -d ais-collector
```

infhub+ais.sh is a convenience script which runs:

```
docker-compose -f docker-compose.yml -f docker-compose-ais.yml \
  <parameters>
```

You can check whether AIS Data Collector is up and running using following command:



```
[centos@information-hub infhub]$ ./infhub+ais.sh ps ais-collector
          Name                          Command               State    Ports
---------------------------------------------------------------------------
infhub_ais-collector_1    java -Dlog4j.configuration ...    Up
```

Figure 13: Checking state of AIS Data Collector

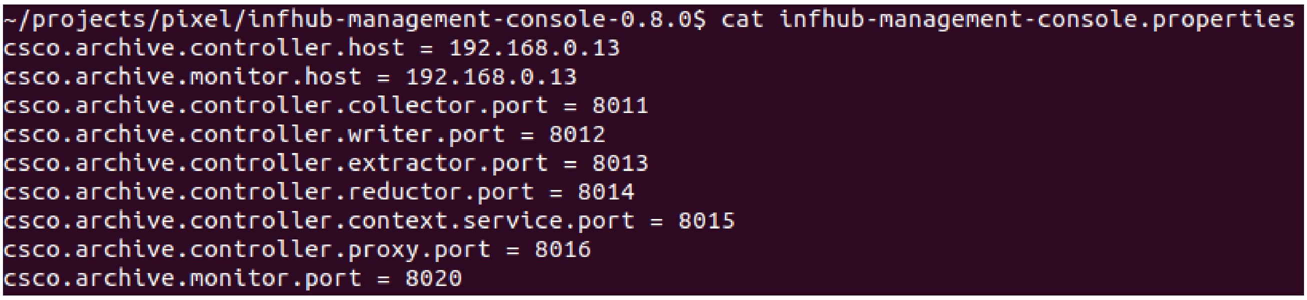
Alternatively, you can start Information Hub together with AIS Data Collector by running:

```
./infhub+ais.sh up -d
```

4.2.3.2.5. Installing Information Hub Management Console

Information Hub Management Console is a Java desktop application and is distributed as a ZIP archive containing an executable JAR package with dependencies and a configuration file. Java 8 is required to run it.

Download the ZIP archive, extract it and navigate to the *infhub-management-console* directory. Open the *infhub-management-console.properties* file in an editor and modify configuration settings as needed for your environment. *csco.archive.controller.host* and *csco.archive.monitor.host* settings specify IP address or hostname of Information Hub Controller and Monitor components respectively. Since all Information Hub services are installed to the same machine using the provided Docker Compose project, both values have the same value.

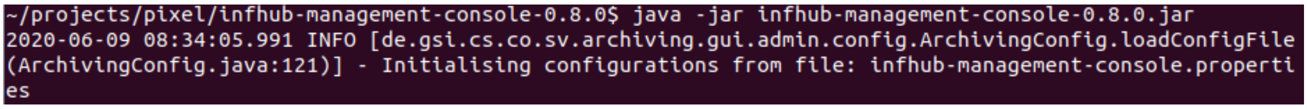


```
~/projects/pixel/infhub-management-console-0.8.0$ cat infhub-management-console.properties
csco.archive.controller.host = 192.168.0.13
csco.archive.monitor.host = 192.168.0.13
csco.archive.controller.collector.port = 8011
csco.archive.controller.writer.port = 8012
csco.archive.controller.extractor.port = 8013
csco.archive.controller.reductor.port = 8014
csco.archive.controller.context.service.port = 8015
csco.archive.controller.proxy.port = 8016
csco.archive.monitor.port = 8020
```

Figure 14: Information Hub Management Console

To run the Information Hub Management Console, use the following command:

```
java -jar infhub-management-console-<version>.jar
```



```
~/projects/pixel/infhub-management-console-0.8.0$ java -jar infhub-management-console-0.8.0.jar
2020-06-09 08:34:05.991 INFO [de.gsi.cs.co.sv.archiving.gui.admin.config.ArchivingConfig.loadConfigFile
(ArchivingConfig.java:121)] - Initialising configurations from file: infhub-management-console.properti
es
```

Path to the *infhub-management-console.properties* configuration file can be provided as a command line parameter otherwise the current directory is used.

4.2.3.3. Configuration

Information Hub configuration is described in the Edit Configuration Files chapter and is carried out before Information Hub is started. After starting up no further action is needed.

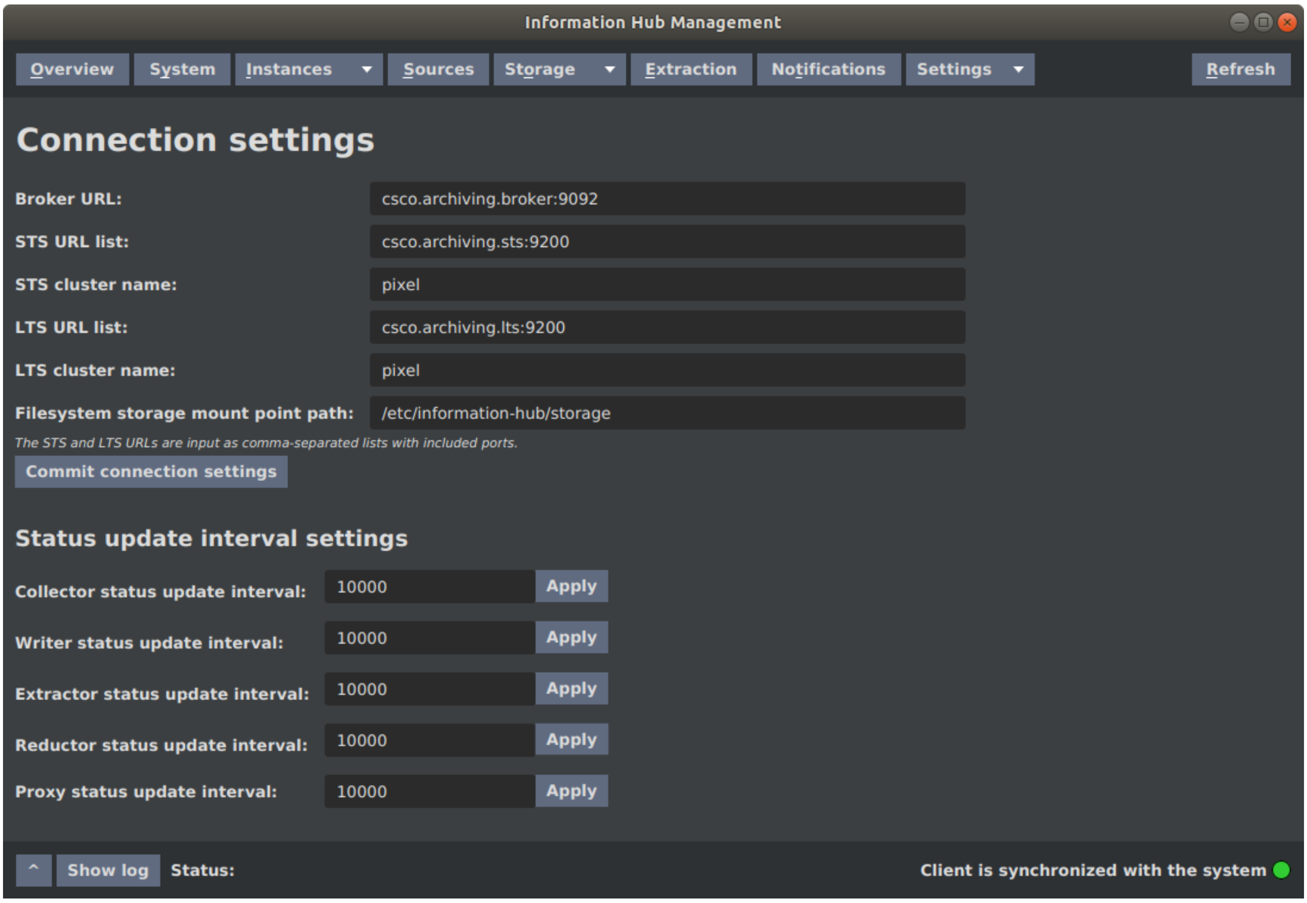
If you edit configuration files at runtime, Information Hub or just the affected services have to be restarted. To restart Information Hub, run the following command from *infhub* folder inside *information-hub-docker* repository:

```
docker-compose restart
```

To restart just specific services, run:

```
docker-compose restart <list of services>
```

In addition to that, some configuration of Information Hub can be made via the Information Hub management console as shown in the figure below. See Settings View chapter of the Information Hub user’s guide.



Connection settings

Broker URL: csco.archiving.broker:9092

STS URL list: csco.archiving.sts:9200

STS cluster name: pixel

LTS URL list: csco.archiving.lts:9200

LTS cluster name: pixel

Filesystem storage mount point path: /etc/information-hub/storage

The STS and LTS URLs are input as comma-separated lists with included ports.

Commit connection settings

Status update interval settings

Collector status update interval: 10000 Apply

Writer status update interval: 10000 Apply

Extractor status update interval: 10000 Apply

Reductor status update interval: 10000 Apply

Proxy status update interval: 10000 Apply

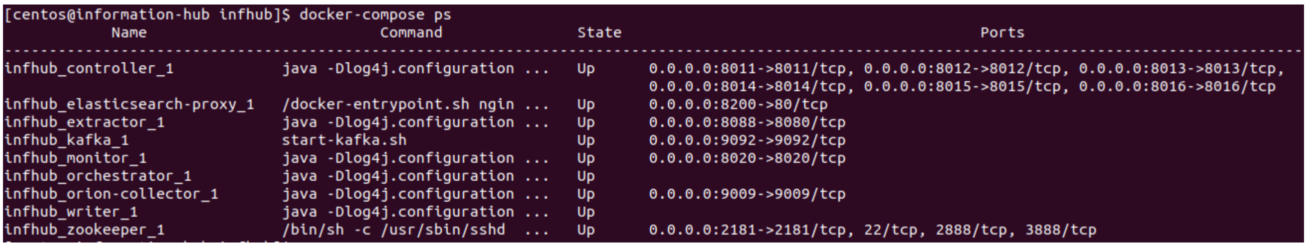
Status:

Client is synchronized with the system

Figure 15: Information Hub settings

4.2.3.4. Monitoring

To verify that Information Hub services are up and running, execute the ‘*docker-compose ps*’ command in the *inhub* folder inside the *information-hub-docker* repository from where Information Hub was started:



```
[centos@information-hub infhub]$ docker-compose ps
            Name                          Command               State                                         Ports
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
infhub_controller_1            java -Dlog4j.configuration ...   Up      0.0.0.0:8011->8011/tcp, 0.0.0.0:8012->8012/tcp, 0.0.0.0:8013->8013/tcp,
                                                                        0.0.0.0:8014->8014/tcp, 0.0.0.0:8015->8015/tcp, 0.0.0.0:8016->8016/tcp
infhub_elasticsearch-proxy_1   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:8200->80/tcp
infhub_extractor_1             java -Dlog4j.configuration ...   Up      0.0.0.0:8088->8080/tcp
infhub_kafka_1                 start-kafka.sh                   Up      0.0.0.0:9092->9092/tcp
infhub_monitor_1               java -Dlog4j.configuration ...   Up      0.0.0.0:8020->8020/tcp
infhub_orchestrator_1          java -Dlog4j.configuration ...   Up
infhub_orion-collector_1       java -Dlog4j.configuration ...   Up      0.0.0.0:9009->9009/tcp
infhub_writer_1                java -Dlog4j.configuration ...   Up
infhub_zookeeper_1             /bin/sh -c /usr/sbin/sshd  ...   Up      0.0.0.0:2181->2181/tcp, 22/tcp, 2888/tcp, 3888/tcp
```

Figure 16: Checking state of Information Hub

Furthermore, to monitor Information Hub services using some monitoring software (e.g. Nagios) following endpoints can be used for monitoring rules configuration to make sure that each component is working correctly:

Table 9: Testing Orion Data Collector

Component: Orion Data Collector
Endpoint: http://IH_HOST:8011/archivingSystem/collector/v1/admin/instance
Expected response: `HTTP/1.1 200 OK`

```
{
   "instances": [
      "9bf4b17684ba"
   ],
   "instanceDescriptions": {
      "9bf4b17684ba": {
         "hostname": "9bf4b17684ba",
         "enabled": true,
         "active": true,
         "status": "OK"
      }
   }
}
```

Table 10: Testing Data Writer

Component: Data Writer

Endpoint: http://IH_HOST:8012/archivingSystem/writer/v1/admin/instance

Expected response:

```
HTTP/1.1 200 OK
{
   "instances": [
      "f4b81709c4be"
   ],
   "instanceDescriptions": {
      "f4b81709c4be": {
         "hostname": "f4b81709c4be",
         "enabled": true,
         "active": true,
         "status": "OK"
      }
   }
}
```

Table 11: Testing Data Extractor

Component: Data Extractor

Endpoint: http://192.168.0.13:8013/archivingSystem/extractor/v1/admin/instance

Expected response:

```
HTTP/1.1 200 OK
{
   "instances": [
      "cc5726394f71"
   ],
   "instanceDescriptions": {
      "cc5726394f71": {
         "hostname": "cc5726394f71",
         "enabled": true,
         "active": true,
         "status": "OK"
      }
   }
}
```

Table 12: Testing Controller

Component: Controller

Endpoint:
http://192.168.0.13:8011/archivingSystem/collector/v1/admin
http://192.168.0.13:8012/archivingSystem/writer/v1/admin
http://192.168.0.13:8013/archivingSystem/extractor/v1/admin
http://192.168.0.13:8015/archivingSystem/context/v1

Expected response:

```
HTTP/1.1 200 OK
```

Table 13: Testing Data Monitor

Component: Data Monitor

Endpoint: http://IH_HOST:8020/archivingSystem/monitor/v1/notification

Expected response:

```
HTTP/1.1 200 OK
{
   "page": 0,
   "pageCount": ...,
   "notifications": [...]
}
```

Table 14: Testing Orchestrator

Component: Orchestrator

Endpoint: http://IH_HOST:8015/archivingSystem/context/v1/components/Orchestrator

Expected response:

```
HTTP/1.1 200 OK
{
   "instances": [
      "Orchestrator/cd22b2d308e4"
   ],
   "instanceDescriptions": {
      "Orchestrator/cd22b2d308e4": {
         "componentId": "Orchestrator",
         "hostname": "cd22b2d308e4",
         "enabled": true,
         "active": true,
         "status": "OK"
      }
   }
}
```

Table 15: Testing Elasticsearch Proxy

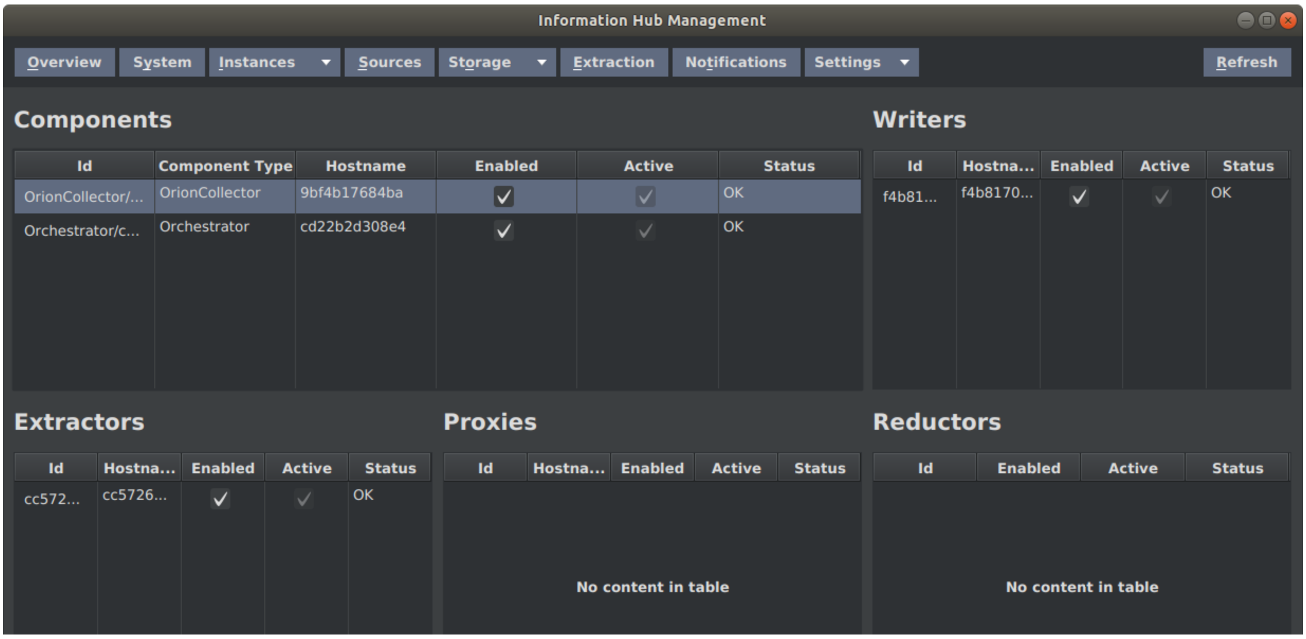
Component: Elasticsearch Proxy

Endpoint: http://IH_HOST:8200/

Expected response:

```
HTTP/1.1 200 OK
{
  "name" : "elasticsearch",
  "cluster_name" : "fair-elastic",
  "cluster_uuid" : "nJuE5xGiTCqXcSR7bYFR6Q",
  "version" : {
    "number" : "7.2.0",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "508c38a",
    "build_date" : "2019-06-20T15:54:18.811730Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Information Hub components can be also monitored using Information Hub Management console:



Components

Id	Component Type	Hostname	Enabled	Active	Status
OrionCollector/...	OrionCollector	9bf4b17684ba	☑	☑	OK
Orchestrator/c...	Orchestrator	cd22b2d308e4	☑	☑	OK

Writers

Id	Hostna...	Enabled	Active	Status
f4b81...	f4b8170...	☑	☑	OK

Extractors

Id	Hostna...	Enabled	Active	Status
cc572...	cc5726...	☑	☑	OK

Proxies

Id	Hostna...	Enabled	Active	Status

No content in table

Reductors

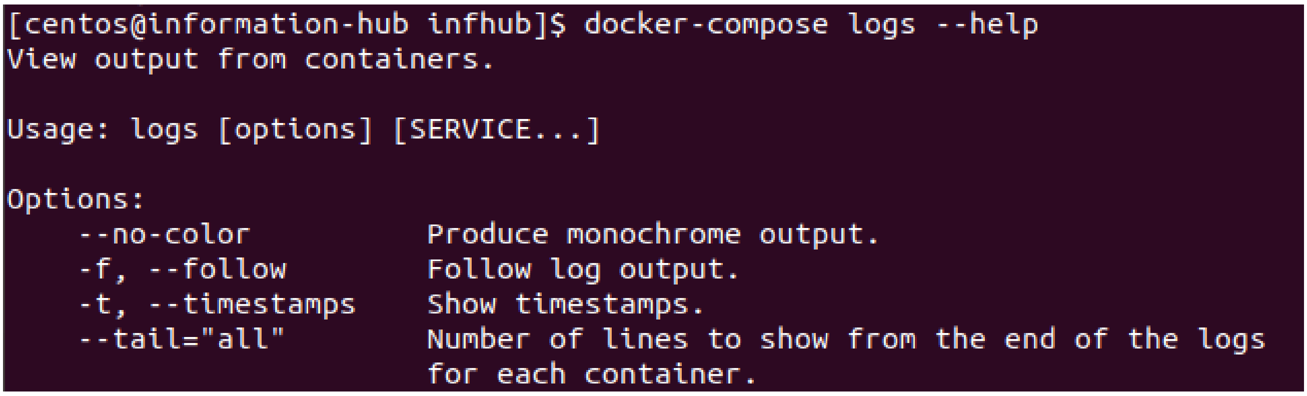
Id	Enabled	Active	Status

No content in table

Figure 17: Information Hub Management Console

4.2.3.5. Issues & Solution

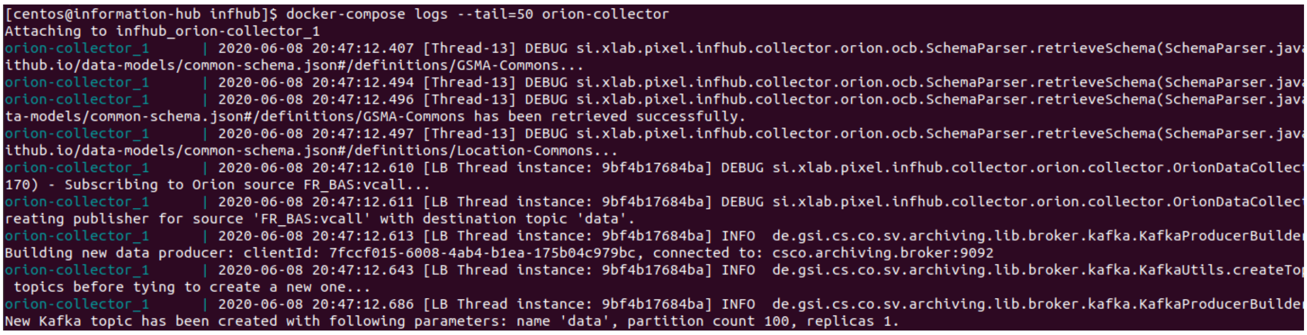
In case any issues arise, checking the Information Hub log files is the first step for determining the nature of the problem. Go to the *information-hub-docker/infhub* folder from where Information Hub has been started and use *docker-compose logs* command to view the application logs:



```
[centos@information-hub infhub]$ docker-compose logs --help
View output from containers.

Usage: logs [options] [SERVICE...]

Options:
    --no-color          Produce monochrome output.
    -f, --follow        Follow log output.
    -t, --timestamps    Show timestamps.
    --tail="all"        Number of lines to show from the end of the logs
                        for each container.
```

Figure 18: Checking the logs

For example, to view the last lines of Orion Collector logs, use the command below:



```
[centos@information-hub infhub]$ docker-compose logs --tail=50 orion-collector
Attaching to infhub_orion-collector_1
orion-collector_1      | 2020-06-08 20:47:12.407 [Thread-13] DEBUG si.xlab.pixel.infhub.collector.orion.ocb.SchemaParser.retrieveSchema(SchemaParser.java
ithub.io/data-models/common-schema.json#/definitions/GSMA-Commons...
orion-collector_1      | 2020-06-08 20:47:12.494 [Thread-13] DEBUG si.xlab.pixel.infhub.collector.orion.ocb.SchemaParser.retrieveSchema(SchemaParser.java
orion-collector_1      | 2020-06-08 20:47:12.496 [Thread-13] DEBUG si.xlab.pixel.infhub.collector.orion.ocb.SchemaParser.retrieveSchema(SchemaParser.java
ta-models/common-schema.json#/definitions/GSMA-Commons has been retrieved successfully.
orion-collector_1      | 2020-06-08 20:47:12.497 [Thread-13] DEBUG si.xlab.pixel.infhub.collector.orion.ocb.SchemaParser.retrieveSchema(SchemaParser.java
ithub.io/data-models/common-schema.json#/definitions/Location-Commons...
orion-collector_1      | 2020-06-08 20:47:12.610 [LB Thread instance: 9bf4b17684ba] DEBUG si.xlab.pixel.infhub.collector.orion.collector.OrionDataCollec
170) - Subscribing to Orion source FR_BAS:vcall...
orion-collector_1      | 2020-06-08 20:47:12.611 [LB Thread instance: 9bf4b17684ba] DEBUG si.xlab.pixel.infhub.collector.orion.collector.OrionDataCollec
reating publisher for source 'FR_BAS:vcall' with destination topic 'data'.
orion-collector_1      | 2020-06-08 20:47:12.613 [LB Thread instance: 9bf4b17684ba] INFO  de.gsi.cs.co.sv.archiving.lib.broker.kafka.KafkaProducerBuilde
Building new data producer: clientId: 7fccf015-6008-4ab4-b1ea-175b04c979bc, connected to: csco.archiving.broker:9092
orion-collector_1      | 2020-06-08 20:47:12.643 [LB Thread instance: 9bf4b17684ba] INFO  de.gsi.cs.co.sv.archiving.lib.broker.kafka.KafkaUtils.createTop
 topics before tying to create a new one...
orion-collector_1      | 2020-06-08 20:47:12.686 [LB Thread instance: 9bf4b17684ba] INFO  de.gsi.cs.co.sv.archiving.lib.broker.kafka.KafkaProducerBuilde
New Kafka topic has been created with following parameters: name 'data', partition count 100, replicas 1.
```

Figure 19: Viewing the last lines of Orion Collector logs

4.2.4. PIXEL Operational Tools

4.2.4.1. Summary

4.2.4.1.1. Main concepts and architecture

The Operational Tools (OT) are mainly in charge of bringing closer to the user both the models and predictive algorithms developed within the PIXEL project. By user here we mean administrators and managers analysing port operations by means of simulation models and predictive algorithms. In order to reach that goal, a set of high-level tasks are defined:

- Publish models and/or predictive algorithms.
- Edit and configure the models and/or predictive algorithms.
- Execute models and/or predictive algorithms.
- Schedule models and/or predictive algorithms to be executed at a specific time once or periodically.
- Define different operational and environmental Key Performance Indicators (KPIs), based on specific data available in the information hub for tracking and monitoring purposes.
- Establish some pattern detection mechanism. The most basic one is the use of triggers.
- Get the trends of a model and/or predictive algorithm (e.g. historical data).

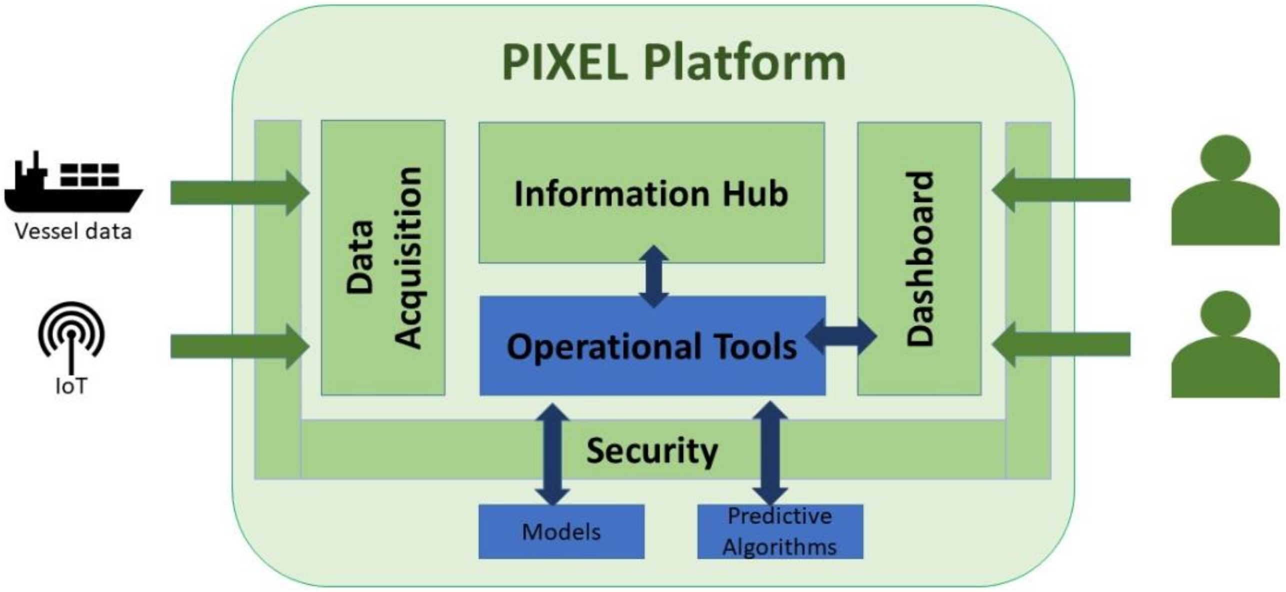


Figure 20: Operational Tools - Architecture overview

The functional overview of the Operational Tools is depicted in the figure next page. Several internal components can be identified:

- OT UI: this is the graphical interface to access (most of) the underlying functionalities. This component provides independence and autonomy, but it can be later integrated as part of the PIXEL dashboard to provide a single-entry point for administrators.
- OT API: backend API implementing the functionalities needed. This component is aligned with the PIXEL security framework in order to fulfil all required security policies (e.g. authentication, authorization, etc.).
- Publication component: it allows publishing both models and predictive algorithms. By publishing it may be necessary to deploy the models as Docker containers. Besides, the models ‘and predictive algorithms’ configurations can also be edited.
- Engine: this component is responsible for executing the different models and predictive algorithms. The execution can be invoked in real time or scheduled.
- Data processing: it is responsible for managing trends from specific data (KPIs) and also for some internal data adaptations required.
- Event processing: this component is responsible for real-time monitoring of indicators and trigger specific actions depending on previously configured rules. It includes a connector (bridge) to be integrated with an external notification system.
- Database: the database includes description of the models and predictive algorithms that can be used, KPI description, rules as well as other configuration and output related parameters necessary for the correct behaviour of the internal building blocks.

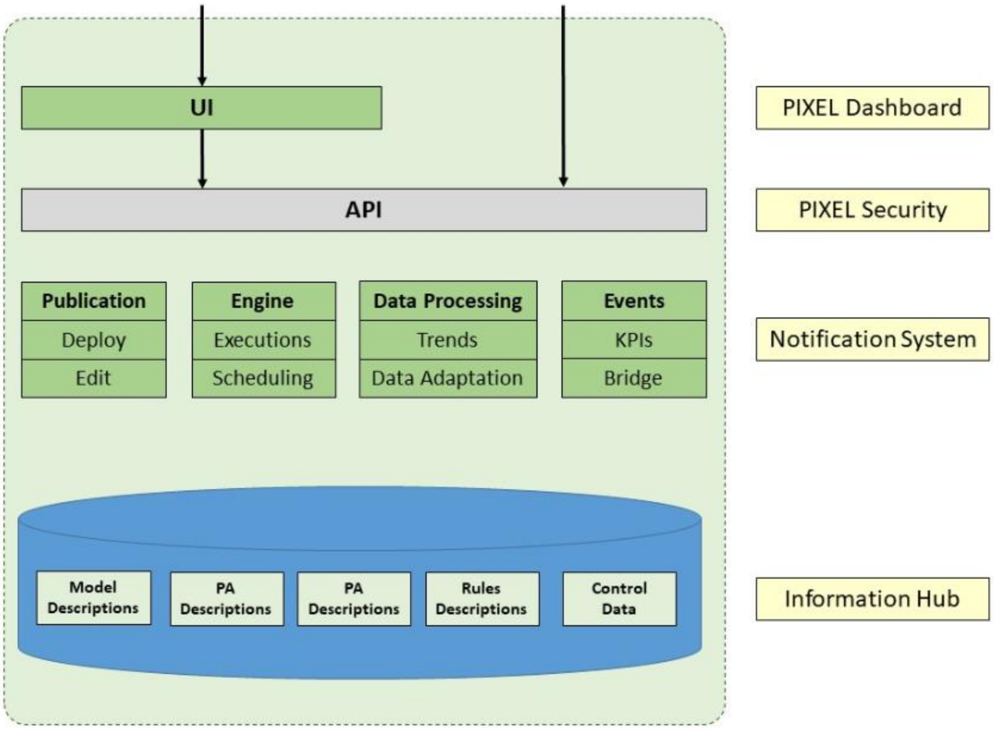


Figure 21: Operational Tools - Functional overview

4.2.4.1.2. Models

Models are entities in the PIXEL platform than will be used by port administrators to run and simulate models and predictive algorithms with different input parameters. As every model and predictive algorithm is different from each other and has its own internals, there is a need to homogenize a common abstract model entity to be the internal representation in the PIXEL platform. It encompasses two different types of developments that have been done within the PIXEL project:

- Models: models relate to energy, traffic and environment. A specific model is the Port Environmental Index (PEI). For more information about the models, please check the PIXEL main documentation repository by clicking here.
- Predictive algorithms: predictive algorithms relate to estimating time of arrival in ports, traffic at gates and use of AIS data. For more information about the models, please check the PIXEL main documentation repository by clicking here.

The figure next page shows the process experienced by any model or predictive algorithm that is going to be used inside the PIXEL platform:

- The model or predictive algorithm is first drafted as algorithm and then implemented as program.
- The model is encapsulated into a Docker container to convert it into a portable component. Additionally, an OT adaptor is attached to his Docker container in order to be integrated into the PIXEL platform.
- Through the publication process the model or predictive algorithm becomes aware into the PIXEL platform. The Docker image is pulled from the (open) GitHub repository and can be used internally.
- After published, the model or predictive algorithm can be executed by passing the appropriate arguments (parameters) as JSON file. The description of this JSON file will be described in future sections. The execution can run immediately (real time) or it can be scheduled to be performed periodically (e.g. every day or week).
- The results of the model are stored into the PIXEL Information Hub, which can be queried by the PIXEL dashboard to visualize them in form of particular graphs depending on the model or predictive algorithm.

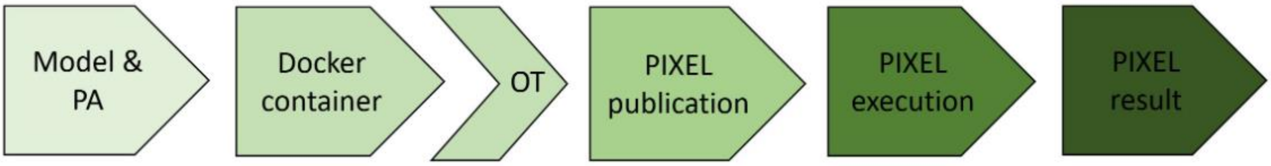


Figure 22: Link between models and the Operational Tools

4.2.4.1.3. Key Performance Indicators

According to Wikipedia a Key Performance Indicator (KPI) is a type of performance measurement. KPIs evaluate the success of an organization or of a particular activity in which it engages. For the PIXEL project, we envision that basic KPIs will mostly refer to:

- Sensors: the PIXEL platform encompasses an IoT network and can therefore monitor any integrated sensor. Some of the sensors may represent an important impact on the decision made from port authorities (e.g. depending on the tide level or the wind speed some cargo type is not recommended to be loaded/unloaded).
- Models and Predictive algorithms: models and predictive algorithms are typically complex and provide various outputs; however, some specific items of the output can be considered of crucial importance and be characterized as KPIs.

More complex KPIs can be potentially defined by combining previous ones, but there is a need to define a common format for them as data entity. PIXEL has followed the FIWARE Data model, which specification can be accessed here. Some extensions have been added, when needed, to particularize it to port and model needs (e.g. environmental KPIs for the PEI calculation). You can find more information on the main documentation repository of PIXEL, clicking here, as there is a section dedicated to Data Models.

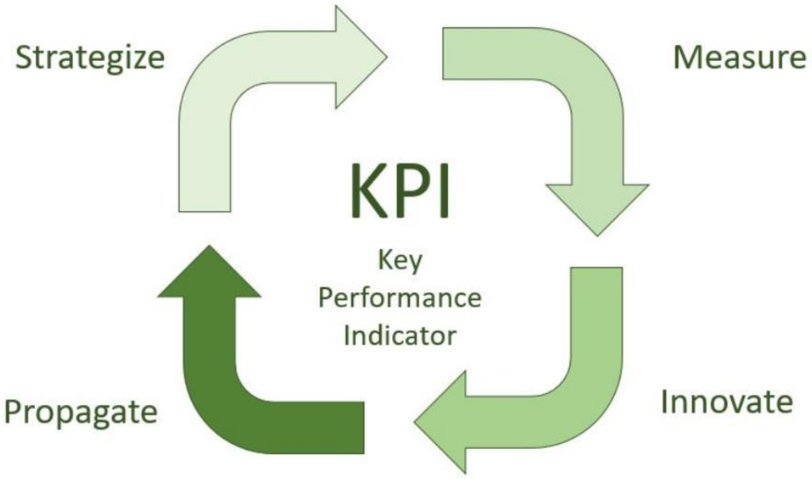


Figure 23: Key Performance Indicators

4.2.4.1.4. Event processing

According to Wikipedia an Event Processing is a method of tracking and analysing (processing) streams of information (data) about things that happen (events) and deriving a conclusion from them. Complex event processing, or CEP, consists of a set of concepts and techniques developed in the early 1990s for processing real-time events and extracting information from event streams as they arrive. The goal of complex event processing is to identify meaningful events (such as opportunities or threats) in real-time situations and respond to them as quickly as possible.

Considering that the PIXEL platform uses as main database Elasticsearch, the selected and natural choice as CEP engine refers to ElastAlert. You can find detailed information about ElastAlert by clicking here. Some of its main features are reliability, modularity and easiness to set up and configure.

From the perspective of the Operational Tools, and considering the current needs of the target ports, this will mainly be related to monitored KPIs where some thresholds are reached. For these situations, rules and alerts are 'templatized' to facilitate the configuration to port operators and define proper actions. More complex actions are possible and supported through ElastAlert; this will be commented in the Developer's Guide subsection, explaining possible extensions.

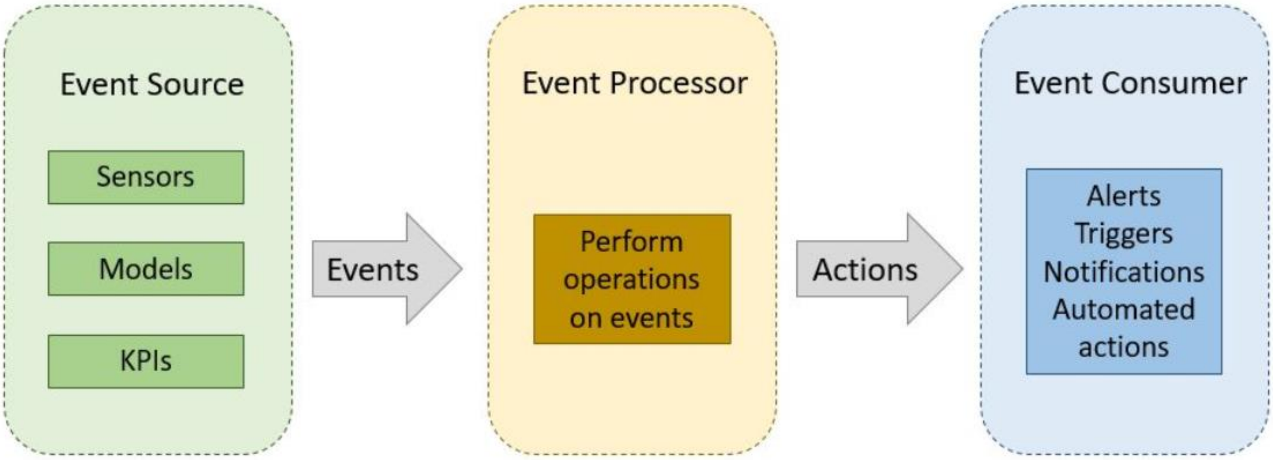


Figure 24: Operational Tools- Event Processing overview

4.2.4.2. How to install

4.2.4.2.1. Note about Docker

Though it was intended to provide a Docker image for every component of the architecture in order to generate a common 'docker-compose' approach, the Operational Tools have special requirements that imposed additional barriers:

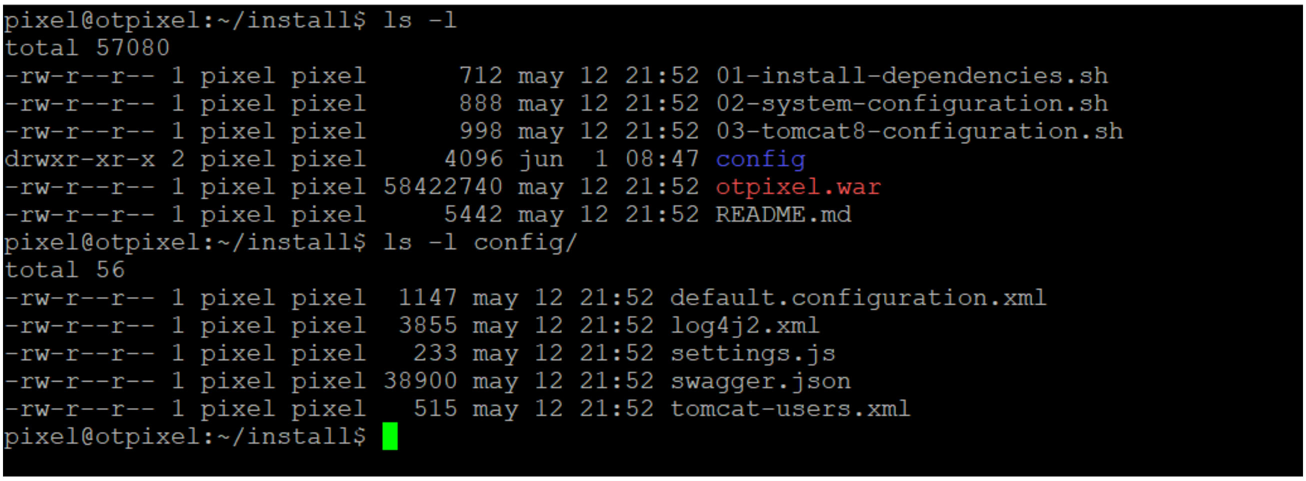
- **Docker execution**: Running a Docker within a Docker (DiD, Docker in Docker) is difficult, tricky and not recommended in various cases. See this article (https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/) for more information about it.
- **Docker client libraries**: OT is developed in Java and there exist the possibility to use a docker-java client to build a Docker image. However, current implementations of the library in Maven are giving strong library dependency problems and are not well documented. It is envisioned to make a port of the current implementation to support a Java docker client, but it will take time and is envisioned for the next version

4.2.4.2.2. Requirements

The Operational Tools have been tested on Linux Ubuntu Server 18.04 LTS. Basically, you will need to install JDK 8, Tomcat and some additional libraries. Everything is done via shell scripts and configuration files, that you can edit. The Operational Tools have been developed as a Tomcat application (WAR file), but this manual will not impose compiling from the sources; instead, it will provide a default precompiled WAR file serving as template to be configured.

4.2.4.2.3. Installation

Download the files under the ‘install’ folder of the OT GitHub repository to your Linux server (3 shell script files, 1 WAR file, and a ‘config’ folder with 5 files). Then follow the different steps below.



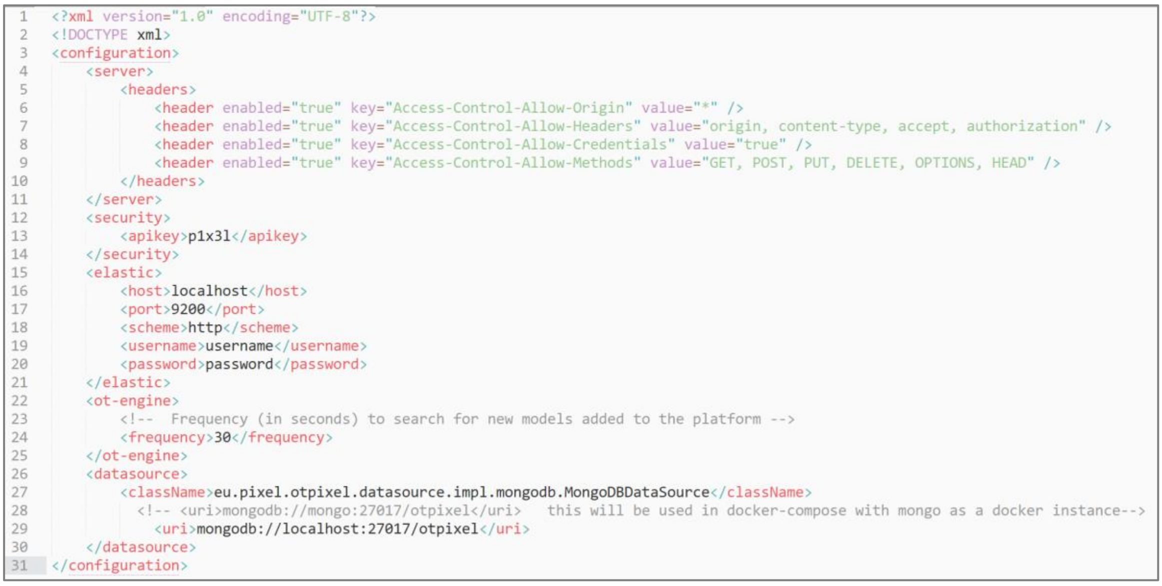
```
pixel@otpixel:~/install$ ls -l
total 57080
-rw-r--r-- 1 pixel pixel      712 may 12 21:52 01-install-dependencies.sh
-rw-r--r-- 1 pixel pixel      888 may 12 21:52 02-system-configuration.sh
-rw-r--r-- 1 pixel pixel      998 may 12 21:52 03-tomcat8-configuration.sh
drwxr-xr-x 2 pixel pixel     4096 jun  1 08:47 config
-rw-r--r-- 1 pixel pixel 58422740 may 12 21:52 otpixel.war
-rw-r--r-- 1 pixel pixel     5442 may 12 21:52 README.md
pixel@otpixel:~/install$ ls -l config/
total 56
-rw-r--r-- 1 pixel pixel  1147 may 12 21:52 default.configuration.xml
-rw-r--r-- 1 pixel pixel  3855 may 12 21:52 log4j2.xml
-rw-r--r-- 1 pixel pixel   233 may 12 21:52 settings.js
-rw-r--r-- 1 pixel pixel 38900 may 12 21:52 swagger.json
-rw-r--r-- 1 pixel pixel   515 may 12 21:52 tomcat-users.xml
pixel@otpixel:~/install$
```

Figure 25: Installation and configuration files for the Operational Tools

1. STEP 1: Edit the configuration

Under the 'conf' directory, you will find 5 different files to edit:

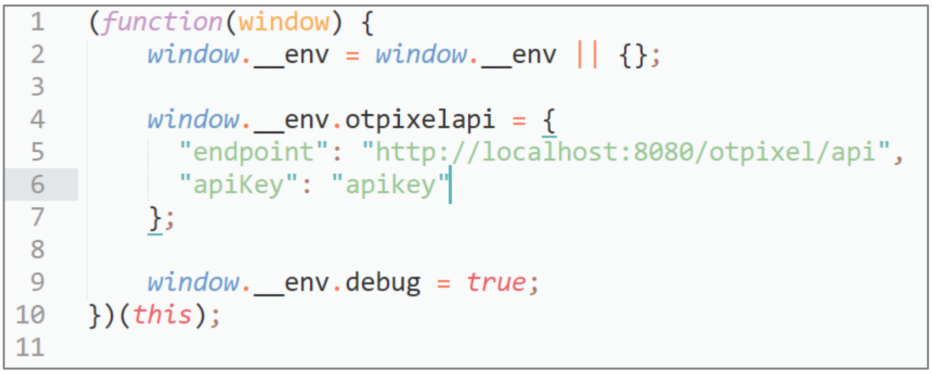
- **default.configuration.xml**: Here you will have to edit/change some parameters, such as the location of the Elasticsearch server and the location of the MongoDB server (data source element). You can leave the other parameters as they are.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml>
<configuration>
    <server>
        <headers>
            <header enabled="true" key="Access-Control-Allow-Origin" value="*" />
            <header enabled="true" key="Access-Control-Allow-Headers" value="origin, content-type, accept, authorization" />
            <header enabled="true" key="Access-Control-Allow-Credentials" value="true" />
            <header enabled="true" key="Access-Control-Allow-Methods" value="GET, POST, PUT, DELETE, OPTIONS, HEAD" />
        </headers>
    </server>
    <security>
        <apikey>p1x3l</apikey>
    </security>
    <elastic>
        <host>localhost</host>
        <port>9200</port>
        <scheme>http</scheme>
        <username>username</username>
        <password>password</password>
    </elastic>
    <ot-engine>
        <!--  Frequency (in seconds) to search for new models added to the platform -->
        <frequency>30</frequency>
    </ot-engine>
    <datasource>
        <className>eu.pixel.otpixel.datasource.impl.mongodb.MongoDBDataSource</className>
          <!-- <uri>mongodb://mongo:27017/otpixel</uri>   this will be used in docker-compose with mongo as a docker instance-->
            <uri>mongodb://localhost:27017/otpixel</uri>
    </datasource>
</configuration>
```

Figure 26: OT- Default configuration file

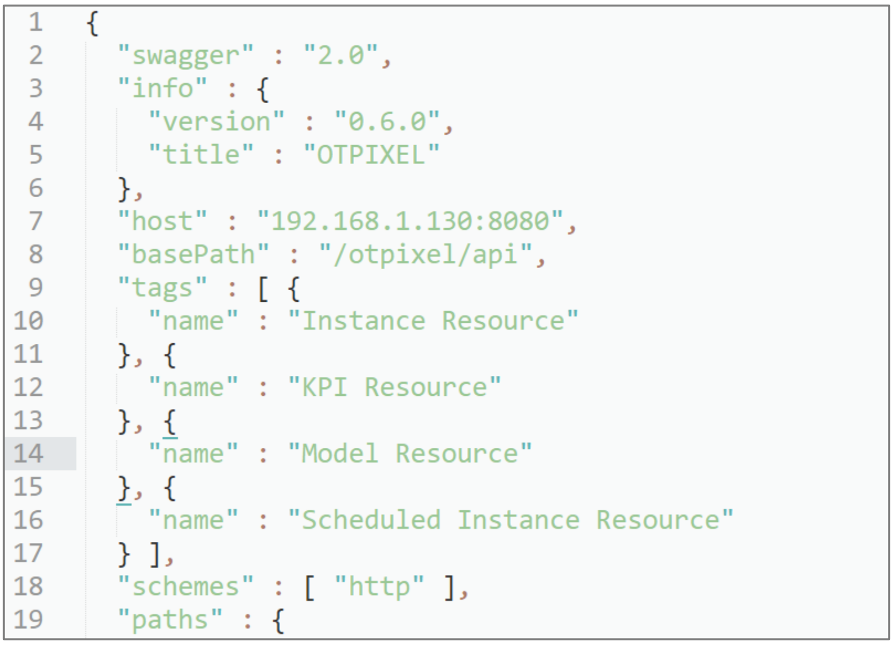
- **log4j.xml**: this is the Log4J configuration file. Probably you do not need to configure it at all. All logs are set by default under /var/log/tomcat with various logging files to track different activities of the engine.
- **settings.js**: Just edit and change here the current IP of the server where you are deploying the OT application, as well as the apiKey you want to use.



```
(function(window) {
    window.__env = window.__env || {};

    window.__env.otpixelapi = {
      "endpoint": "http://localhost:8080/otpixel/api",
      "apiKey": "apikey"
    };

    window.__env.debug = true;
})(this);

```

Figure 27: OT- Settings UI configuration file

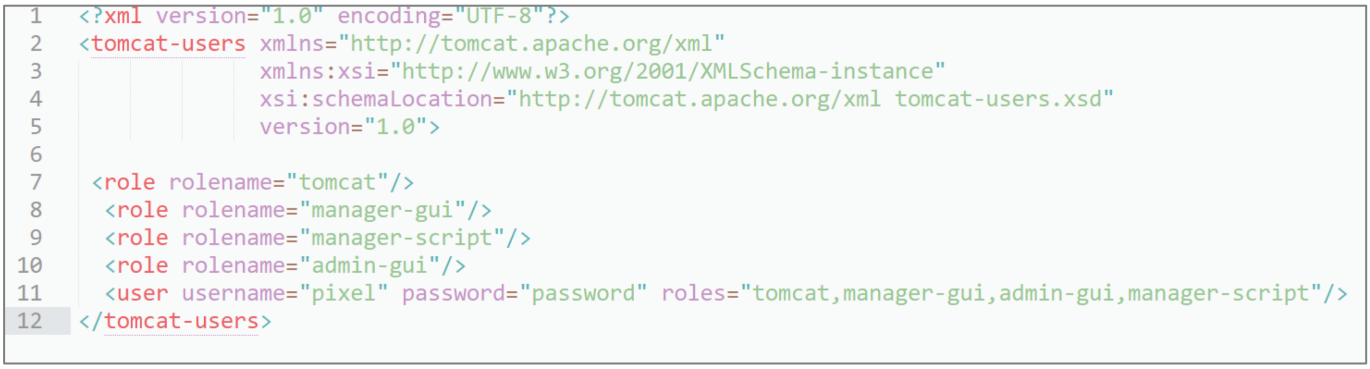
- **swagger.json**: Just edit and change here the current IP of the server where you are deploying the OT application (host element).



```
{
  "swagger" : "2.0",
  "info" : {
    "version" : "0.6.0",
    "title" : "OTPIXEL"
  },
  "host" : "192.168.1.130:8080",
  "basePath" : "/otpixel/api",
  "tags" : [ {
    "name" : "Instance Resource"
  }, {
    "name" : "KPI Resource"
  }, {
    "name" : "Model Resource"
  }, {
    "name" : "Scheduled Instance Resource"
  } ],
  "schemes" : [ "http" ],
  "paths" : {
```

Figure 28: OT –Swagger configuration file

- **tomcat-users.xml**: Just change and insert here the password you want to use for later updates (redeployments). This is in fact optional but allows doing updates without reinstalling again everything.



```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">

 <role rolename="tomcat"/>
  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <role rolename="admin-gui"/>
  <user username="pixel" password="password" roles="tomcat,manager-gui,admin-gui,manager-script"/>
</tomcat-users>
```

Figure 29: OT- Tomcat configuration for user deployment

Note: In Linux it is difficult to estimate the current IP of a server, as it may have various IPs (localhost, docker interfaces, bridged interfaces, etc.). Therefore, we have opted for inputting the IP in the files 'settings.json' and 'swagger.json'.

2. **STEP 2: Run the scripts**

After configuring the files, return to the previous ‘install’ folder, and start running the scripts one by one as administrator.

```
$ sudo sh 01-install-dependencies
```

This will update and upgrade the system, and install all required libraries (e.g. JDK 8, Tomcat 8, etc.)

```
sudo sh 02-system-configuration
```

This will make some system configuration to allow the *tomcat8* user manage docker instances. As the script includes the tomcat8 user into the 'docker' group (/etc/group file), it typically requires a restart so that the changes become effective.

```
sudo sh 03-tomcat8-configuration
```

This will rebuild the WAR considering the configuration files under the 'conf' directory and deploy it in Tomcat8.

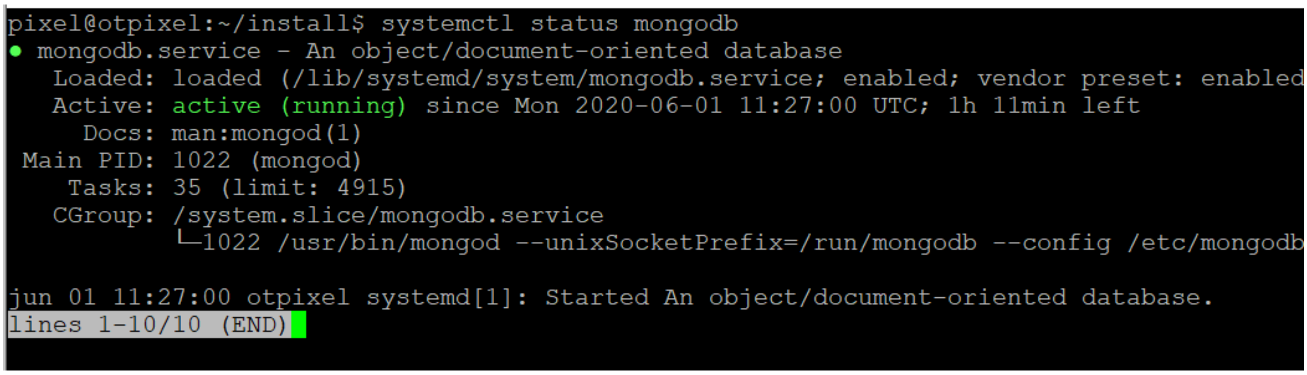
4.2.4.3. Configuration

The configuration has already been provided at installation time (see previous step). No further action is necessary. All services should be up and running (mongo, tomcat8 server and tomcat8 application).

How do you verify the service has been correctly deployed?

Mongo - Database

Supposing Mongo is running as service within the host server, just type in the command line.



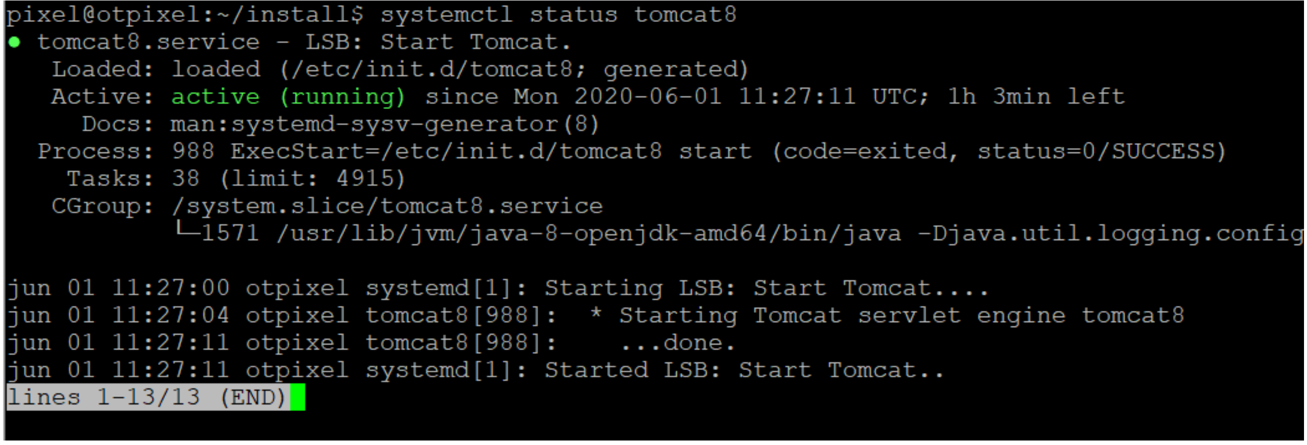
```
pixel@otpixel:~/install$ systemctl status mongodb
● mongodb.service - An object/document-oriented database
   Loaded: loaded (/lib/systemd/system/mongodb.service; enabled; vendor preset: enabled
   Active: active (running) since Mon 2020-06-01 11:27:00 UTC; 1h 11min left
     Docs: man:mongod(1)
 Main PID: 1022 (mongod)
    Tasks: 35 (limit: 4915)
   CGroup: /system.slice/mongodb.service
           └─1022 /usr/bin/mongod --unixSocketPrefix=/run/mongodb --config /etc/mongodb

jun 01 11:27:00 otpixel systemd[1]: Started An object/document-oriented database.
lines 1-10/10 (END)
```

Figure 30: Testing if Mongo is running

You should see (in green) if the server is active and running properly; otherwise, you will see an error.
If Mongo has been installed elsewhere (not localhost) or as a docker instance, you can use the command "***docker-compose ps***" to check that the service is in "***Up***" status, and with the command "***telnet IP 27001***" that the TCP port is listening.
Note: Remember to configure Mongo server to support non-localhost requests, if necessary

Tomcat8 server

Similar as for Mongo, just type in the command line



```
pixel@otpixel:~/install$ systemctl status tomcat8
● tomcat8.service - LSB: Start Tomcat.
   Loaded: loaded (/etc/init.d/tomcat8; generated)
   Active: active (running) since Mon 2020-06-01 11:27:11 UTC; 1h 3min left
     Docs: man:systemd-sysv-generator(8)
  Process: 988 ExecStart=/etc/init.d/tomcat8 start (code=exited, status=0/SUCCESS)
    Tasks: 38 (limit: 4915)
   CGroup: /system.slice/tomcat8.service
           └─1571 /usr/lib/jvm/java-8-openjdk-amd64/bin/java -Djava.util.logging.config

jun 01 11:27:00 otpixel systemd[1]: Starting LSB: Start Tomcat....
jun 01 11:27:04 otpixel tomcat8[988]:  * Starting Tomcat servlet engine tomcat8
jun 01 11:27:11 otpixel tomcat8[988]:    ...done.
jun 01 11:27:11 otpixel systemd[1]: Started LSB: Start Tomcat..
lines 1-13/13 (END)
```

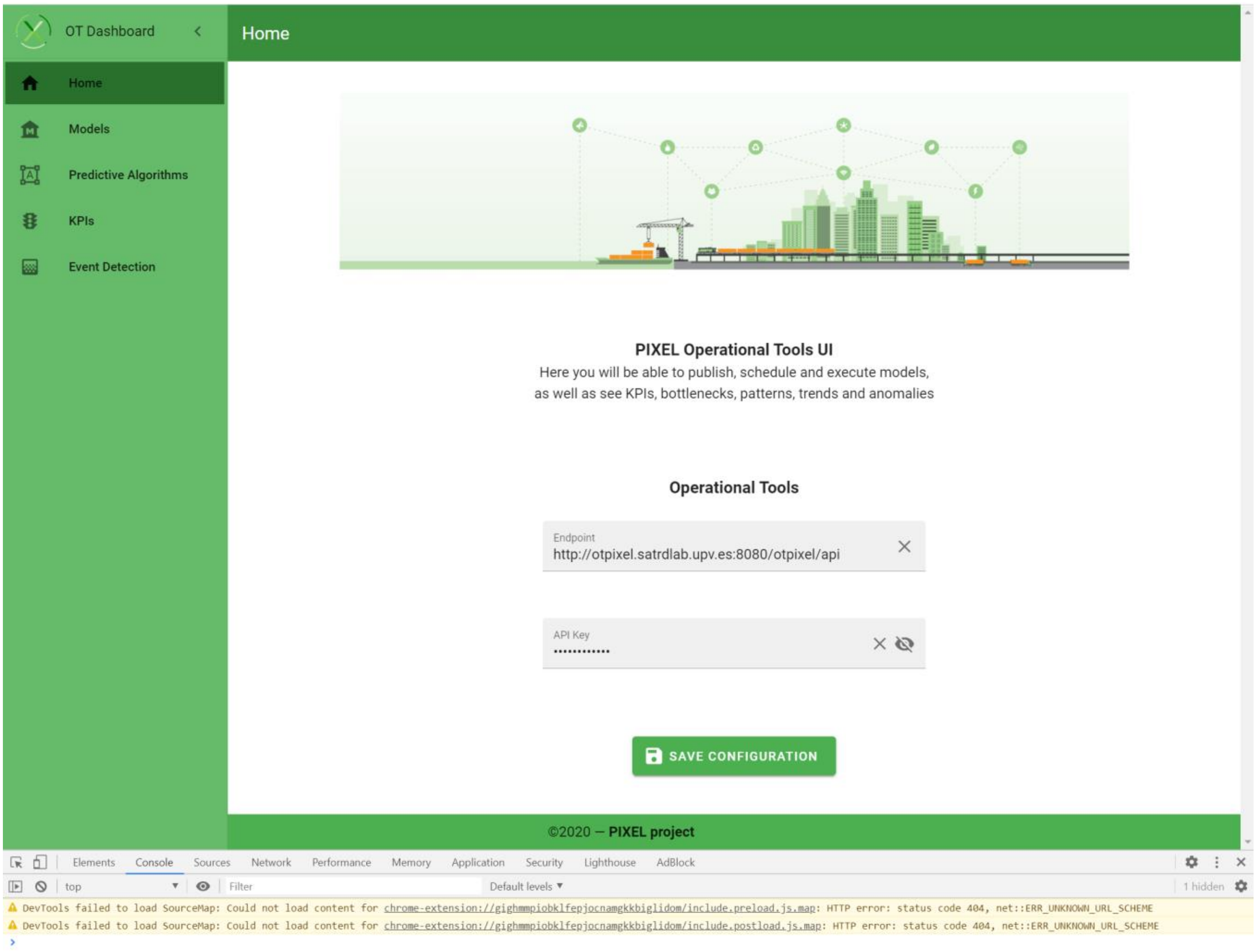
Figure 31: Testing if Tomcat8 is running

You should see (in green) if the server is active and running properly; otherwise, you will see an error.

Tomcat OT application - UI

Open a web browser and go to http://<your-server-ip>:8080/otpixel/ui.

You should be able to see the UI of the application. Even if you cannot see neither models nor predictive algorithms (not yet deployed), you should not see any error in the 'Developer's panel' of the browser



PIXEL Operational Tools UI

Here you will be able to publish, schedule and execute models, as well as see KPIs, bottlenecks, patterns, trends and anomalies

Operational Tools

Endpoint
http://otpixel.satrdlab.upv.es:8080/otpixel/api

API Key
•••••••••••••

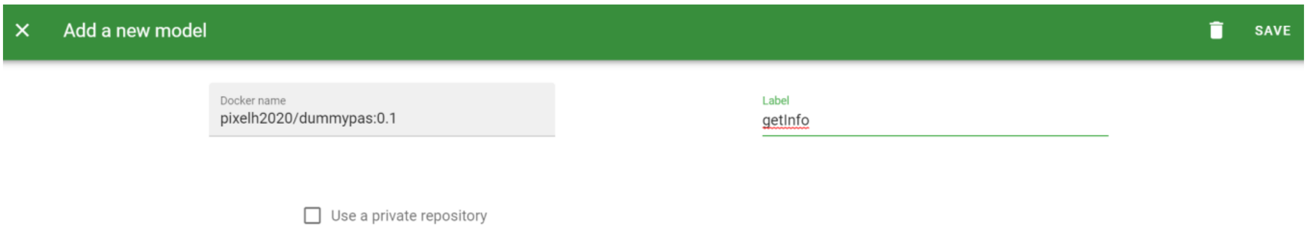
©2020 — **PIXEL project**

```
⚠ DevTools failed to load SourceMap: Could not load content for chrome-extension://gighmmpiobklfepjocnamgkkbiglidom/include.preload.js.map: HTTP error: status code 404, net::ERR_UNKNOWN_URL_SCHEME
⚠ DevTools failed to load SourceMap: Could not load content for chrome-extension://gighmmpiobklfepjocnamgkkbiglidom/include.postload.js.map: HTTP error: status code 404, net::ERR_UNKNOWN_URL_SCHEME
>
```

Figure 32: Testing if the UI is running

If you want to perform a more advanced test, then go to '***Models'*** on the Left Menu and click on the '***Add a new Model'*** button. Just enter the following:

- Docker name: pixelh2020/dummypas:0.1
- Label: getInfo



Docker name: pixelh2020/dummypas:0.1

Label: getInfo

□ Use a private repository

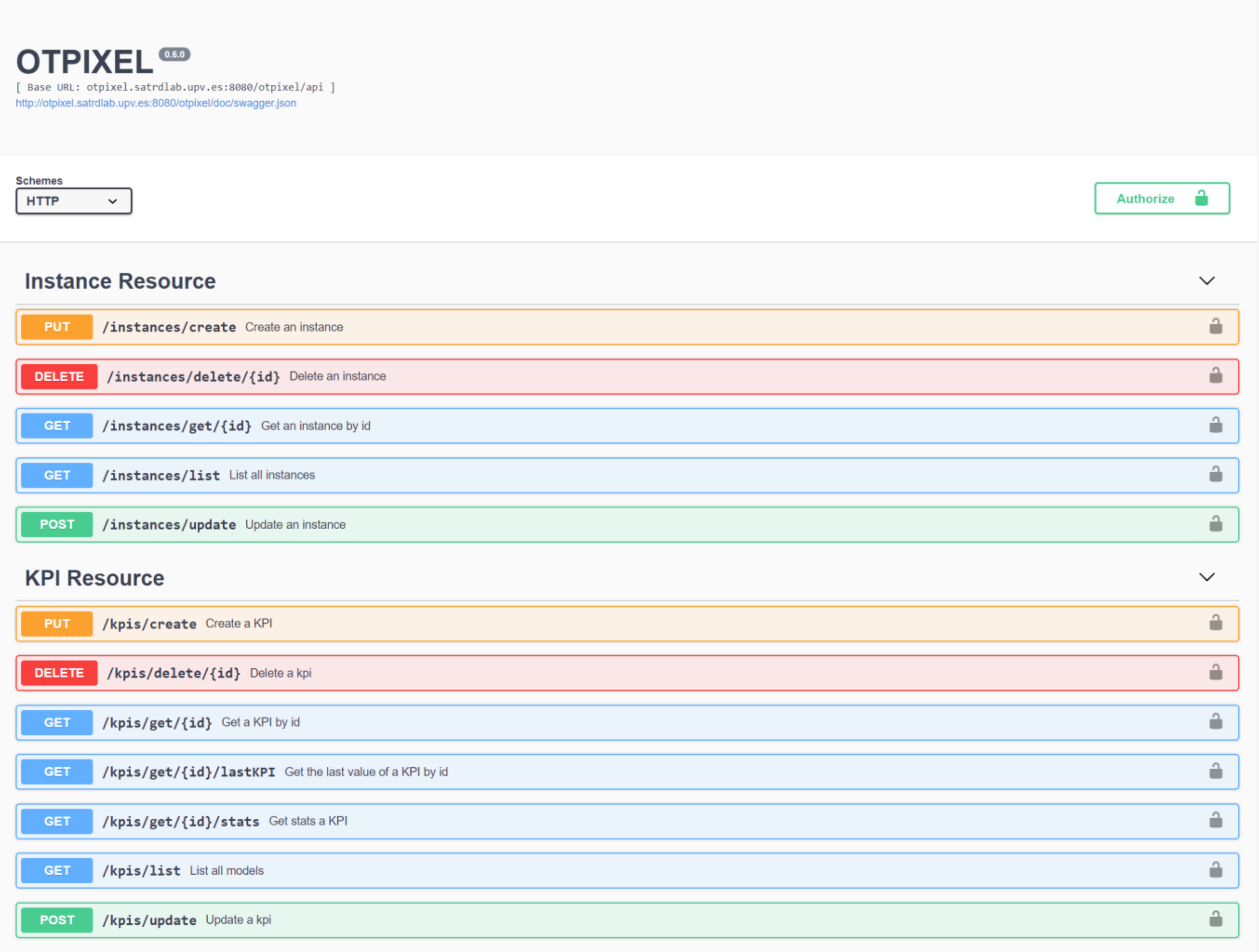
Figure 33: Testing if a specific functionality is working

You will see that the new model should have been entered in the list of models, with a status of ***'created'***. Just wait a couple of minutes (the Docker image needs to be pulled from the *Dockerhub* repository and this could take a while) and refresh the screen. Now the ***'status***' should have change to one of:

- **deployed**: this means that everything went properly. By clicking on the 'Edit' icon of this model, you may see the details.
- **error**: there has been an error. More information may be obtained by checking the log file (otpixelEngineCreateModel.log); this is commented in the next section

Tomcat OT application – Swagger

Open a web browser and go to http://<your-server-ip>:8080/otpixel/doc
You should be able to see the Swagger UI of the application. You can click on 'Authorize', enter your ***apiKey*** and start testing the API. As there are no models or predictive algorithms, you should get an empty array.



OTPIXEL

[Base URL: otpixel.satrdlab.upv.es:8080/otpixel/api]

http://otpixel.satrdlab.upv.es:8080/otpixel/doc/swagger.json

Instance Resource

PUT `/instances/create` Create an instance

DELETE `/instances/delete/{id}` Delete an instance

GET `/instances/get/{id}` Get an instance by id

GET `/instances/list` List all instances

POST `/instances/update` Update an instance

KPI Resource

PUT `/kpis/create` Create a KPI

DELETE `/kpis/delete/{id}` Delete a kpi

GET `/kpis/get/{id}` Get a KPI by id

GET `/kpis/get/{id}/lastKPI` Get the last value of a KPI by id

GET `/kpis/get/{id}/stats` Get stats a KPI

GET `/kpis/list` List all models

POST `/kpis/update` Update a kpi

Figure 34: Testing the Swagger

4.2.4.4. Issues & Solution

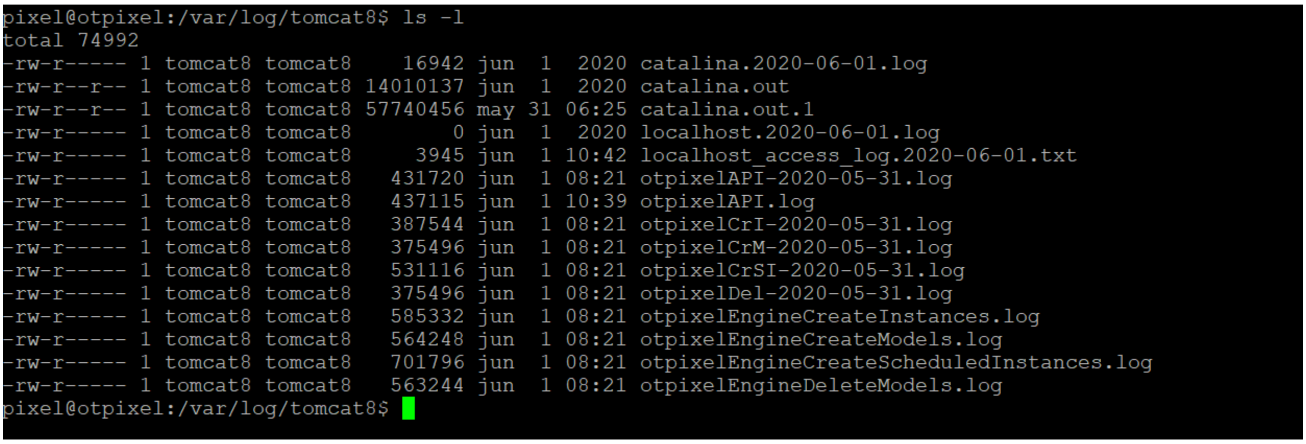
4.2.4.4.1. Deploy/update a new OT version

As commented before, new updates are released a tomcat application (WAR files), therefore the only action to perform consists in redeploying the WAR file in the Tomcat8 server. However, it is recommendable to undeploy the previous OT application first, as it uses several threads to manage different tasks internally. Redeploying on top of a running application does not prevent the previous threads to stop running.

4.2.4.4.2. Check for logs

The Operational Tools includes a series of different log files to monitor the activity of different tasks independently:

- **otpixelAPI.log**: general log file for OT.
- **otpixelEngineCreateModels.log**: management thread of the OT Engine to manage the creation of models and predictive algorithms.
- **otpixelEngineDeleteModels.log**: management thread of the OT Engine to manage the deletion of models and predictive algorithms.
- **otpixelEngineCreateInstances.log**: management thread of the OT Engine to manage the creation of instances.
- **otpixelEngineCreateScheduledInstances.log**: management thread of the OT Engine to manage the creation of scheduled instances.



```
pixel@otpixel:/var/log/tomcat8$ ls -l
total 74992
-rw-r----- 1 tomcat8 tomcat8    16942 jun  1  2020 catalina.2020-06-01.log
-rw-r--r-- 1 tomcat8 tomcat8 14010137 jun  1  2020 catalina.out
-rw-r--r-- 1 tomcat8 tomcat8 57740456 may 31 06:25 catalina.out.1
-rw-r----- 1 tomcat8 tomcat8        0 jun  1  2020 localhost.2020-06-01.log
-rw-r----- 1 tomcat8 tomcat8     3945 jun  1 10:42 localhost_access_log.2020-06-01.txt
-rw-r----- 1 tomcat8 tomcat8   431720 jun  1 08:21 otpixelAPI-2020-05-31.log
-rw-r----- 1 tomcat8 tomcat8   437115 jun  1 10:39 otpixelAPI.log
-rw-r----- 1 tomcat8 tomcat8   387544 jun  1 08:21 otpixelCrI-2020-05-31.log
-rw-r----- 1 tomcat8 tomcat8   375496 jun  1 08:21 otpixelCrM-2020-05-31.log
-rw-r----- 1 tomcat8 tomcat8   531116 jun  1 08:21 otpixelCrSI-2020-05-31.log
-rw-r----- 1 tomcat8 tomcat8   375496 jun  1 08:21 otpixelDel-2020-05-31.log
-rw-r----- 1 tomcat8 tomcat8   585332 jun  1 08:21 otpixelEngineCreateInstances.log
-rw-r----- 1 tomcat8 tomcat8   564248 jun  1 08:21 otpixelEngineCreateModels.log
-rw-r----- 1 tomcat8 tomcat8   701796 jun  1 08:21 otpixelEngineCreateScheduledInstances.log
-rw-r----- 1 tomcat8 tomcat8   563244 jun  1 08:21 otpixelEngineDeleteModels.log
pixel@otpixel:/var/log/tomcat8$ 
```

Figure 35: OT log files

4.2.5. PIXEL Integrated Dashboard and Notifications

4.2.5.1. Summary

PIXEL Integrated Dashboard and Notifications is the component that has the capability of representing data stored in the IH meaningful **combined visualizations in real time**. Also, it provides the capability to send notifications based on the status of the data received from the sensors. Finally, this module **provides (aggregates and homogenises) all the UI for the different functional blocks** (Operational Tools for example).

Which are the components in which the dashboard is divided and how do they interact with other components of PIXEL?

- *Dashboard* component is divided in two subcomponents:
    - *Frontend*: Offers a web application based on the VueJS Framework, this component exposes the UI with which the user interacts.
    - *Backend*: Exposes all the services needed for the dashboard. Moreover, it connects to the IDM service to ensure users are authorized. It has a non-relational database and communicates with PIXEL Operational Tools for the management of the containers. Backend also has a component responsible of alerts.
        - *Backend of alerts*: It has a service that exposes a REST API to create different types of alerts. Once launched they are sent directly to the backend. It requires connection to PIXEL Information Hub.
- *Proxy*. To maintain a single point of entry. There are redirections for all the PIXEL components. All the services must be exposed through this component.

Next figure depicts the components involved in the component.

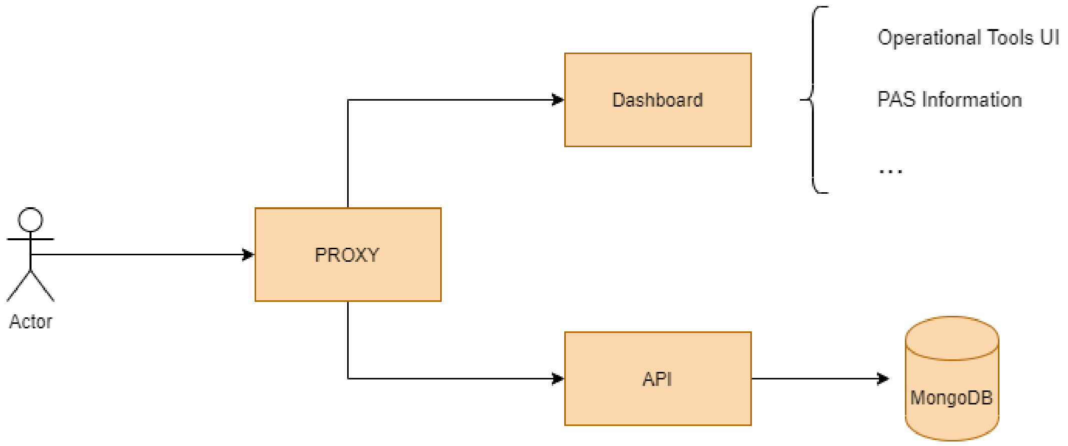


Figure 1: PIXEL Dashboard components

The functional overview of the different options that has the Dashboard are:

- *Overview*. View where the visualizations created by the end-user and published are shown. In this way, they are accessible as soon as the user accesses to the platform.
- *Views*. Component responsible for creating the different types of visualizations (**Gantt diagram, Table, etc.**) of the data coming from sensors.
- *Dashboard*. UI responsible for creating dashboards using visualizations created in the previous section.
- *Permission*. Component aligned with the **PIXEL Security & Privacy** component in order to fulfil all required security policies (e.g. authentication, authorization, roles, permission, etc.).
- *PAS Information*. User interface to fill in the different entities (**resources, rules and supplier chain**) needed as input for the PAS Model (**Port Activity Scenario**).
- *Map*. Component that will show **geolocated data** (sensors, devices, etc.) from the different ports.
- *Alerts*. Component responsible for **real-time monitoring of data** and trigger alerts depending on their value.
- *Operational Tools*. User interface to access the functionalities for the **Operational Tools** component.

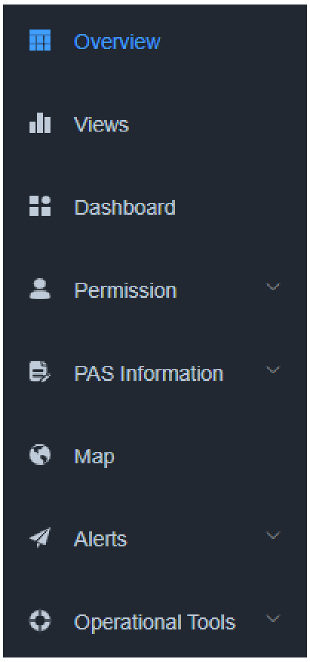


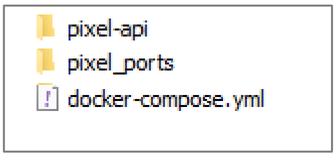
Figure 3: Dashboard menu entries

4.2.5.2. How to install

The installation of the dashboard has as a prerequisite the installation of docker and docker-compose (see Installation of docker and docker-compose).

For the installation it will be necessary to have a directory with the following elements:

- *docker-compose file*: File where the necessary services to raise an instance of the dashboard will be described.
- *pixel_ports folder*: It will contain the source code of the client solution. The code does not need to be compiled. When the docker-compose instruction is launched, the build of this solution is performed.
- *pixel_api folder*: It contains the code of the server solution. This solution is dashboard's API. It is in charge of interacting with MongoDB to store / list: alerts, notifications, resources, rules, etc. The entities with which the dashboard component interacts.



pixel-api
pixel_ports
docker-compose.yml

Figure 2: Services included in docker-compose file

The services included in the docker-compose file are:

- *kibana*: It raises an instance of Kibana on which ElastAlert plugin is installed.
- *mongo*: It raises an instance of MongoDB.
- *API*: This service is responsible for compiling the server solution. Once compiled it exposes our customer services for their interaction.
- *dashboard*: This service compiles the client solution. Once compiled, it takes care of building a server with the client solution.
- *elastalert*: Plugin installed on Kibana that acts as a notification engine. The service provided by this component is in charge of seeing if the rules are met in order to launch the notifications.
- *webapp*: Service that raises the UI of the component responsible for the definition of rules / alerts.

4.2.5.3. Configuration

The installation is done by executing the following instruction in the directory where the docker-compose is located:

```
sudo docker-compose up –d
```

This instruction is in charge of raising all the services of the file.

Table 16: How to verify the Dashboard is correctly deployed

How do you verify the service has been correctly deployed?
Kibana – UI DEV
Executing "docker-compose ps" to check that the service is in "Up" state, on the other hand, it is also recommended to check that the TCP port is listening with the command "telnet IP 5601"
Mongo - Database
With the command "docker-compose ps" you check that the service is in "Up" status, and with the command "telnet IP 27001" that the TCP port is listening
Dashboard – Api
With the command "docker-compose ps" you check that the service is in "Up" status, and with the command "telnet IP 3000" that the TCP port is listening
Dashboard - client
With the command "docker-compose ps" you check that the service is in "Up" status, and with the command "telnet IP 8080" that the TCP port is listening
Elastalert – server alerts
With the command "docker-compose ps" you check that the service is in "Up" status, and with the command "telnet IP 3030" that the TCP port is listening
Praeco – UI Alerts
With the command "docker-compose ps" you check that the service is in "Up" status, and with the command "telnet IP 8085" that the TCP port is listening

4.2.5.4. Issues & Solution

- **How to deploy / update the dashboard version**: Once the dashboard component is initially deployed, it is possible to update the version of the server solution (*api*) or the client solution (*dashboard*). To do so, it will be necessary to update the source in the corresponding folder: *pixel_ports* (dashboard), *pixel_api* (api). Both services can now be deployed again. To do this you have to raise the docker-compose instance but only of these components not of the whole file. The instruction to be executed in each case is:
    - **dashboard:** *sudo docker-compose up --build -d dashboard*
    - **api:** *sudo docker-compose up --build -d api*

4.2.6. PIXEL Security

4.2.6.1. Summary

The main function of the security layer is to secure the access to the API of the other components from outside the platform and to provide a solution for identity management.

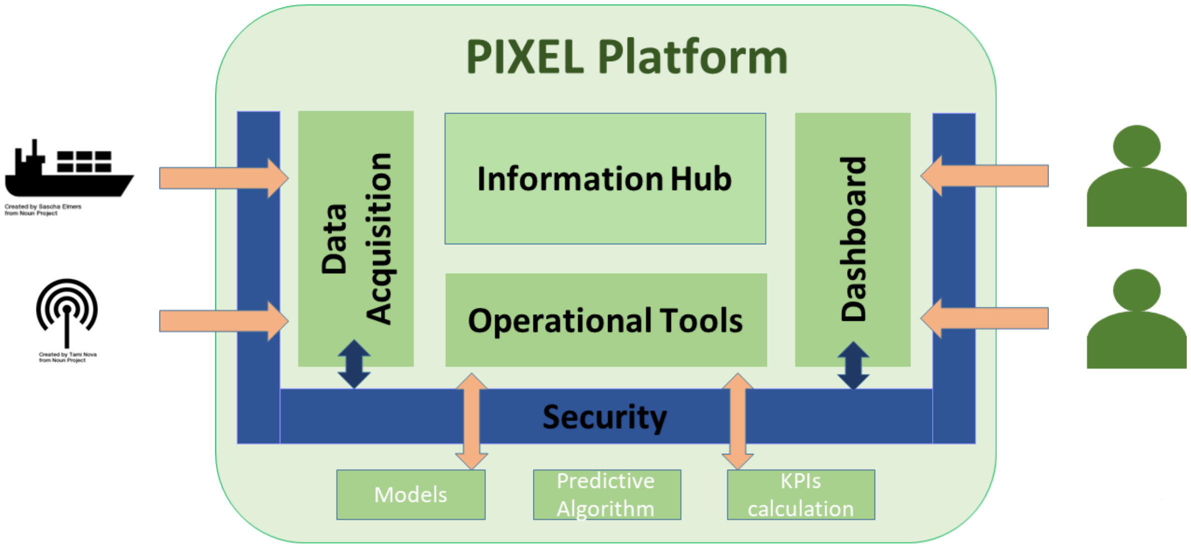


Figure 36: Purpose of PIXEL Security Layer

The security layer secures the access to the NGSI Agents that exposed an API in the Data Acquisition Layer, but it also provides security to the dashboard UI to access the PIXEL’s API (Dashboard, Information Hub and Operational Tools).

We rely on the FIWARE architecture and solution to implement those features in PIXEL, using the FIWARE Generic Enablers:

- KeyRock : The Identity Manager
- Wilma (PEP Proxy) : The OAuth2 proxy that check the access
- AuthzForce : An XACML authorization solution

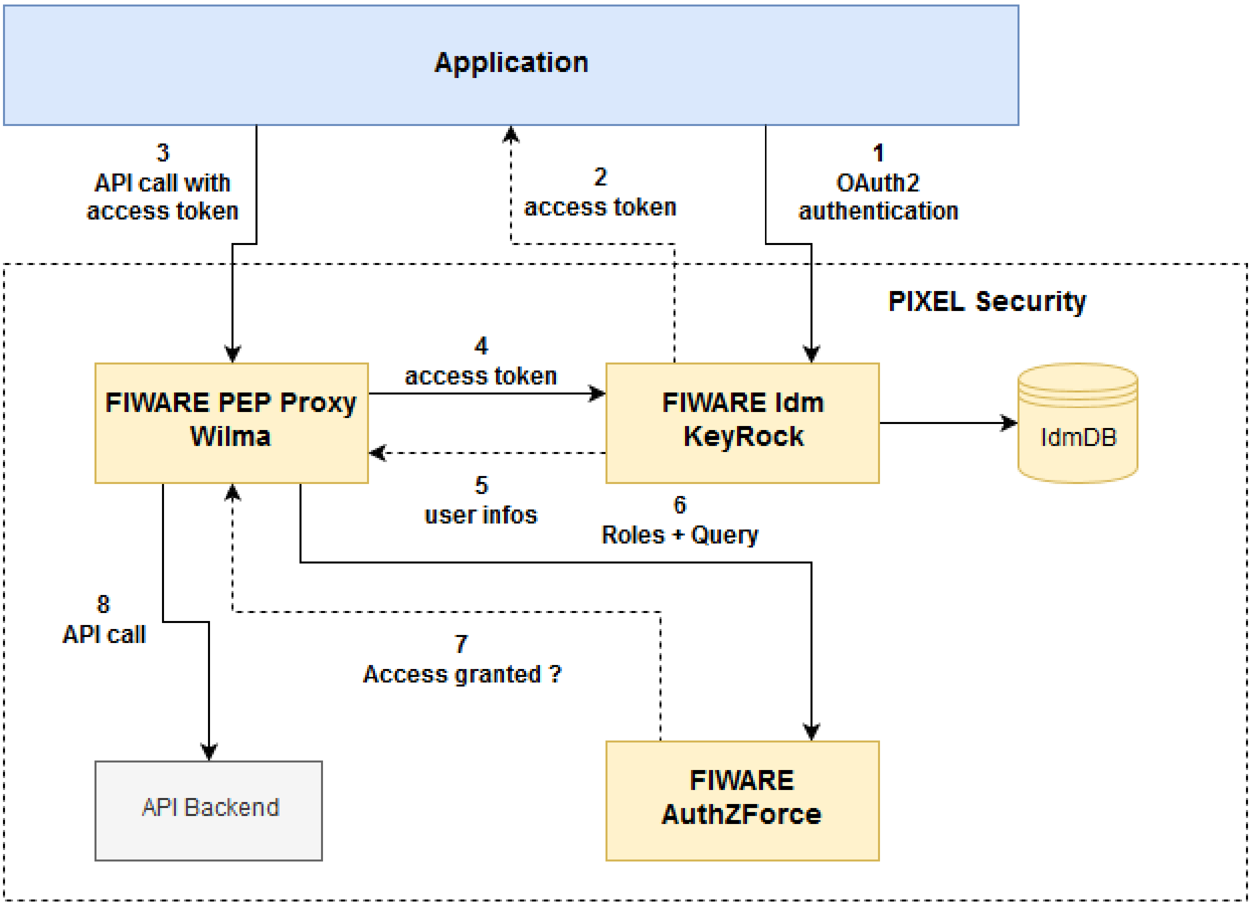


Figure 37: Diagram of PIXEL Security Layer

4.2.6.2. How to install

The full installation process relies on docker-compose.

The process of the installation is split in three different steps:

- Configuration : You must adapt the docker-compose-*.yaml file and feed the different secrets value
- Build : it is the docker-compose build process, a helper script ./build.sh is provided
- Installation : it is the docker-compose up process, a helper script is provided

Table 17: Installation of Security Layer

Installation of Security Layer
cd Security *./build.sh* *./install.sh*

4.2.6.3. Configuration

Platform specifics configurations are done using the docker-compose-*.yaml file and with the configuration of the secret files for each component:

Table 18: Environment variables for security layer

Environment variables

Wilma	
PEP_PROXY_AZF_PORT	The port use to contact AuthZForce
PEP_PROXY_AZF_HOST	The host that exposed AuthZForce
PEP_PROXY_APP_PORT	The port use to contact the backend protected by Wilma (DAL-Proxy for example)
PEP_PROXY_APP_HOST	The host that exposed the backend protected by Wilma (DAL-Proxy for example)
PEP_PROXY_APP_ID	The APP ID created in Keyrock for this PEP Proxy
PEP_PROXY_IDM_PORT	The port use to contact KeyRock
PEP_PROXY_IDM_HOST	The host that exposed KeyRock
PEP_PROXY_PORT	The port Wilma listen to
KeyRock	
IDM_DB_HOST	The address of the MySQL Database
IDM_DB_USER	The MySQL user to connect the database
IDM_HOST	The URL to contact KeyRock
IDM_PORT	The port KeyRock listen to
IDM_AUTHZFORCE_HOST	The host to contact AuthZForce
AuthZForce	
Secrets	
Wilma	
sec_wilma_pub.token.secret	A token use for encryption (random)
sec_wilma_pub.password	The password for the PEP Proxy created with the application in KeyRock
sec_wilma_pub.proxy.username	The id of the PEP Proxy created with the application in KeyRock
MySQL	
idm.db.pass	The MySQL root password (random)
KeyRock	
idm.db.pass	The MySQL root password (random)
idm.admin.pass	The password for the admin user of KeyRock (random)
idm.session.secret	A token use for encryption (random)

4.2.6.4. Component status

Table 19: How to check the status of the different services in the docker-compose file

How do you verify the service has been correctly deployed?

MySQL
Executing "docker-compose ps" to check that the service is in "Up" state , and with the command "telnet IP 3306" that the TCP port is listening
KeyRock
With the command "docker-compose ps" you check that the service is in "Up" status, and check you can connect to KeyRock with the admin/password
Wilma
With the command "docker-compose ps" you check that the service is in "Up" status.
AuthZForce
With the command "docker-compose ps" you check that the service is in "Up" status.

4.2.6.5. PIXEL specifics deployment

When you have deployed KeyRock and AuthzForce, you can deploy as many Wilma that you need. Wilma is just a simple HTTP Proxy that you install in the middle of the API flow to check the request provide a valid token and is allow to access the resources.

By default we use Wilma to protect NGSI Agents that is configured as daemon, and we will also use it to protect the access to private API from the outside of the platform.

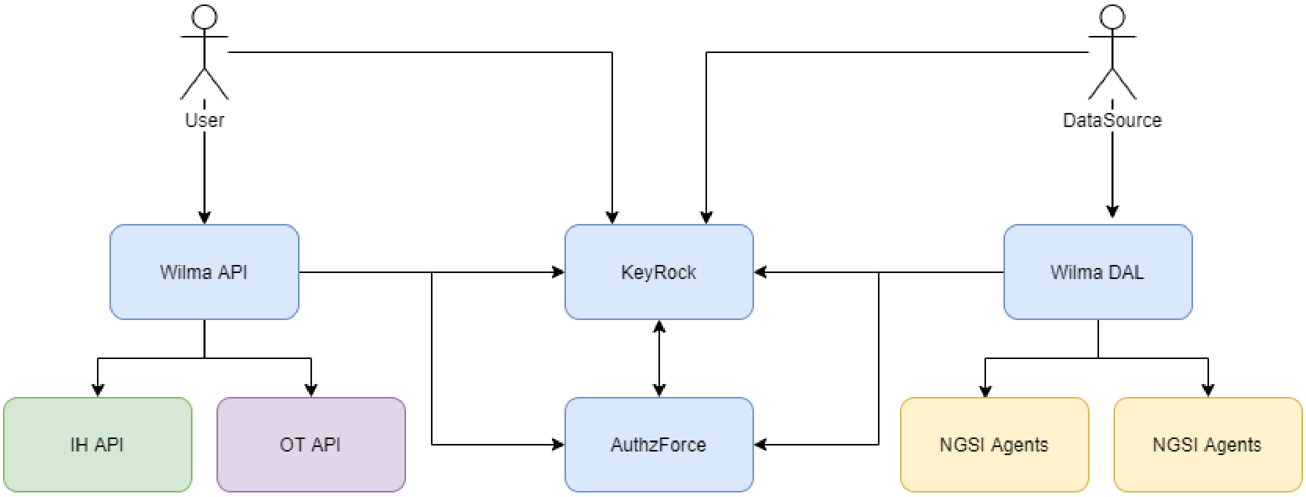
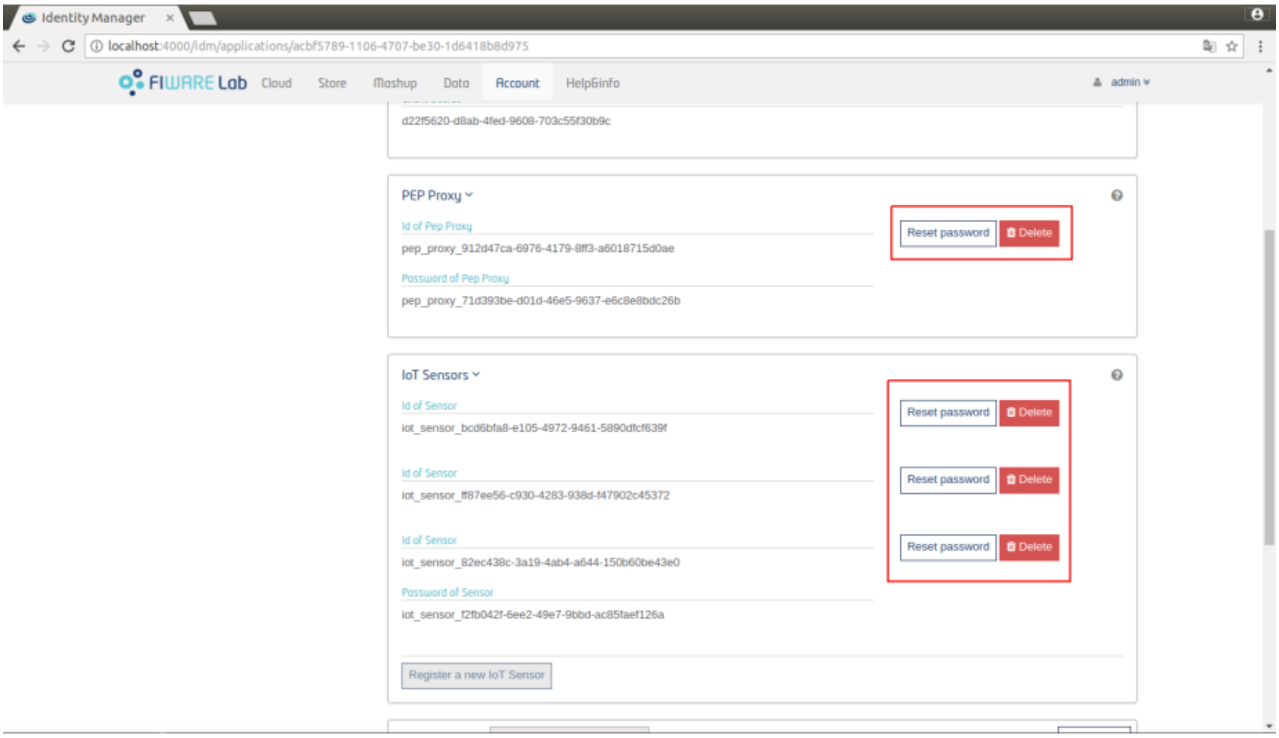


Figure 38: Wilma diagram

To install other Wilma, you have to create the corresponding Application in KeyRock using the UI or the API.

Then retrieve the application id, pep_proxy id and password and then set those parameters in the docker compose file.


Identity Manager
localhost:4000/idm/applications/acbf5789-1106-4707-be30-1d6418b8d975
FIWARE Lab
Cloud
Store
Mashup
Data
Account
Help&info
admin

d22f5620-d8ab-4fed-9608-703c55f30b9c

PEP Proxy

Id of Pep Proxy
pep_proxy_912d47ca-6976-4179-8ff3-a6018715d0ae

Password of Pep Proxy
pep_proxy_71d393be-d01d-46e5-9637-e6c8e8bdc26b

IoT Sensors

Id of Sensor
iot_sensor_bcd6bfa8-e105-4972-9461-5890dfcf639f

Id of Sensor
iot_sensor_ff87ee56-c930-4283-938d-f47902c45372

Id of Sensor
iot_sensor_82ec438c-3a19-4ab4-a644-150b60be43e0

Password of Sensor
iot_sensor_f2fb042f-6ee2-49e7-9bbd-ac85faef126a

Figure 39: Parameters needed to install more than one Wilma

4.2.6.6. Issues & Solution

Most of the problems with FIWARE Security modules are the provisioning of the application parameters on Wilma. A simple test could allow to control everything works as expected.

Table 20: Issues related with Wilma

Authorization Basic

We authenticate against an application, we need 2 information for that application

- *Client id*
- *Client secret*

Then we can combine them to create the *Authorization Basic* token:

- *base64(client_id:client_secret)*

Access Token Request

```
POST /oauth2/token HTTP/1.1
Host: id.<pilot>.port-pixel.eu
Authorization: Basic <authorization basic token>
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=<user email>&password=<user password>
```

user email and password have to be URL Encoded, the token is valid 1 hour. To refresh it you can authenticate again, or use the refresh token.

Access Token Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
```

```
{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"bearer",
    "expires_in":3600,
    "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
}
```

Token Verification

```
curl https://id.<pilot>.port-pixel.eu/user?access_token=<access_token>
    {
      "organizations": [
        {
          "id": "13e88767-7473-472d-9c33-110c5bed2a57",
          "name": "test_org",
          "description": "my org",
          "website": null,
          "roles": [
            {
              "id": "9c4e8db4-a56b-4731-bfc6-7dd8fb2fbea3",
              "name": "test"
            }
          ]
        }
      ],
      "displayName": "My User",
      "roles": [
        {
          "id": "9c4e8db4-a56b-4731-bfc6-7dd8fb2fbea3",
          "name": "test"
        }
      ],
      "app_id": "ff03921a-a772-4220-9854-e2d499ae474a",
      "isGravatarEnabled": false,
      "email": "myuser@test.com",
      "id": "myuser",
      "authorization_decision": "",
      "app_azf_domain": "",
      "username": "myuser"
    }
```

Request A Resource Access

For example to access https://dal.<pilot>.pixel-ports.eu/orion/version

```
curl -H “X-Auth-Token: <access_token>” https://dal.<pilot>.pixel-ports.eu/orion/version
```

An HTTP 200 OK should be received, otherwise check the components logs to investigate

Refer to the official documentation for KeyRock, Wilma and AuthZForce.

4.3. Pilots Installation

To allow simple pilots installation, PIXEL proposes to install the full platform on only two host running docker.

The full docker-compose are fully configured and only some small parameters have to be modified.

4.3.1. Architecture

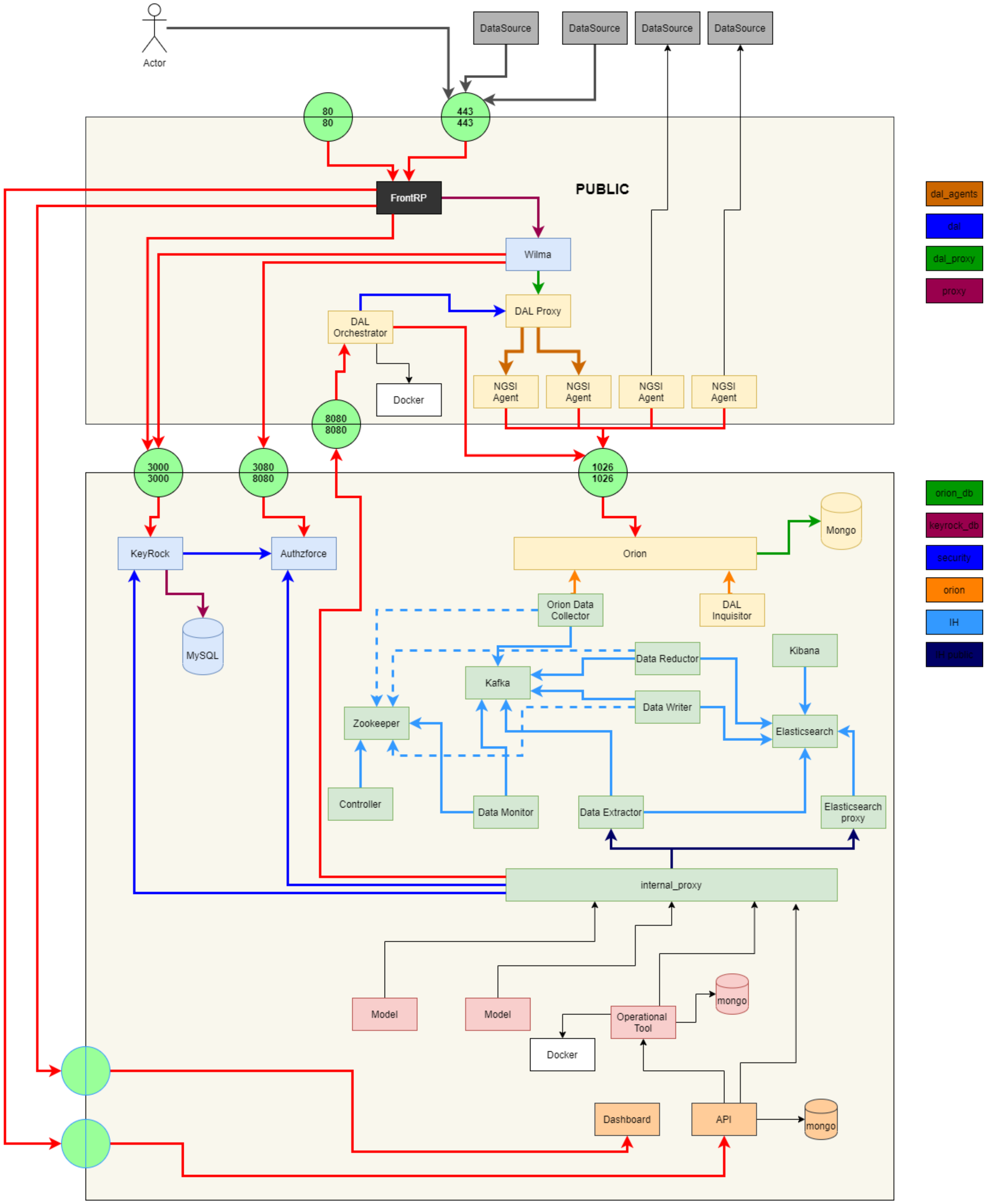


Figure 40: PIXEL Architecture diagram

4.3.2. How to install

4.3.2.1. Core Host

Table 21: Installation of the CORE Host

Installation of the CORE Host

Install docker and requirements

```
apt update
apt upgrade -y
apt-get install -y apt-transport-https ca-certificates curl gnupg-agent soft-
ware-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
apt-get install -y docker-ce docker-ce-cli containerd.io
curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
apt install -y git
echo "vm.max_map_count=262144" >> /etc/sysctl.conf
sysctl -w vm.max_map_count=262144
```

Retrieve the Core Archive

```
mkdir /opt/pixel
cd /opt/pixel/
GIT_SSL_NO_VERIFY=false git clone
https://gitpixel.satrdlab.upv.es/marc.despland/Installation.git
cd /opt/pixel/Installation/docker/core
```

Configure the scripts

```
vi .env
```

```
PUBLIC_HOST_IP=10.66.16.137
CORE_HOST_IP=10.12.182.193
PIXEL_DOMAIN=frbod.pixel-ports.eu
PIXEL_INTERNAL_DOMAIN=pixel.internal
```

Configure the secrets

```
Set all the secret files with random value.
A quick way to do it:
docker run -it --rm -v ${PWD}/secrets:/app/secrets pixelh2020/secrets:1.0.0
```

Network Security

```
./pixel-rules.install.sh
cp pixel-rules /etc/init.d
/etc/init.d/pixel-rules start
<check the rules before running the last commands>
update-rc.d pixel-rules defaults
```

Build the images

```
./build.sh
```

Install the containers

```
./install.sh
```

Initial configuration

```
./dal-provisionning.sh
```

```
> provisioning@1.0.0 start /app
> node index.js

Token                          : 99f4bf97-e915-417b-9285-09023905a491
Organization PIXEL             : 96a7da6e-1bbc-4ee3-aee8-dacab079d485
Appli DAL NGSIAGENTS PROXY     : 5ff34b1c-4e41-4b2e-9085-0f52b0b1c810
PEP Proxy password             : pep_proxy_6c7b2771-1704-42c1-ab04-d8753401f3a2
PEP Proxy oauth_client_id      : pep_proxy_5fe89b1c-4e41-4b2e-9085-0f52b0b1c810
Keyrock          .............   Done
Subscription created           : 5ed36ccd502bffe0fedc6847
Inquisitor       .............   Done
```

You can run the build and install process as often you need. It is also used for updating the platform.

4.3.2.2. Public Host

Table 22: Installation of the PUBLIC Host

Installation of the PUBLIC Host

Install docker and requirements

```
apt update
apt upgrade -y
apt-get install -y apt-transport-https ca-certificates curl gnupg-agent soft-
ware-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
apt-get install -y docker-ce docker-ce-cli containerd.io
curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
apt install -y git
```

Retrieve the Core Archive

```
mkdir /opt/pixel
cd /opt/pixel/
GIT_SSL_NO_VERIFY=false git clone
https://gitpixel.satrdlab.upv.es/marc.despland/Installation.git
cd /opt/pixel/Installation/docker/public
```

Configure the scripts

```
vi .env

PUBLIC_HOST_IP=10.66.16.137
CORE_HOST_IP=10.12.182.193
PIXEL_DOMAIN=frbod.pixel-ports.eu
PIXEL_INTERNAL_DOMAIN=pixel.internal
```

Configure the secrets

```
Set all the secret files with random value.
A quick way to do it:
docker run -it --rm -v ${PWD}/secrets:/app/secrets pixelh2020/secrets:1.0.0
Be careful with those secrets, use the result of dal-provisionning.sh on CORE
```

- `sec_wilma_pub.password : PEP Proxy password`
- `sec_wilma_pub.proxy.username: PEP Proxy oauth_client_id`

Network Security

```
./pixel-rules.install.sh
cp pixel-rules /etc/init.d
/etc/init.d/pixel-rules start
<check the rules before running the last commands>
update-rc.d pixel-rules defaults
```

Build the images

```
./build.sh
```

Generate the certificate

We need a wildcard certificate for the chosen domain.

Here we propose to generate it using Let's Encrypt

We choose to use `*.<un/locode>.pixel-ports.eu`

In order to generate it, you will need to contact UPV to create a DNS TXT Entry Here is the process for `*.frbod.pixel-ports.eu`

```
cd /opt/pixel
mkdir LetsEncrypt
cd LetsEncrypt
docker run -it --rm $(PwD):/etc/letsencrypt --entrypoint certbot pixelh2020/certbot certonly --manual -m infos@pixel-ports.eu -d *.frbod.pixel-ports.eu
cp live/frbod.pixel-ports.eu/fullchain.pem Installation/docker/public/frontrp/
cp live/frbod.pixel-ports.eu/privkey.pem Installation/docker/public/frontrp/
```

Install the containers

```
./install.sh
```

4.4. User’s Guide

4.4.1. PIXEL Data acquisition

DAL-Orchestrator and DAL-Proxy present a swagger-UI with their API documentation:

- http://<ip>:<port>/api-docs

For Orion, refer to the official documentation: https://fiware-orion.readthedocs.io/en/master/

4.4.1.1. NGSI Agents

NGSI Agents are the small software use to import data from external data sources into PIXEL through the Data Acquisition Layer. We have 3 kinds of NGSI Agents:

- daemon: running as a server to received data
- scheduled: starts automatically at the given period
- manual: running only when asked

In order to run as an NGSI Agent your Docker container need some special configurations. Those configurations are done using Docker LABEL that could be overwrite when deploying an agent on the destination platform

In order to be identified the docker image of an agent has to contains specifics labels.

- **Labels for all agents**
  - **ngsiagent="pixel"**: this is the key label to be identified as a NGSI Agent
  - **ngsiagent.type="daemon"**: define the type of NGSI Agent daemon, scheduled or manual
  - **ngsiagent.datasources="[\"urn:pixel:DataSource:dummies\"]"**: this label provide the name of the data source manage by this agent
  - **ngsiagent.datamodels="[\"/Dummies/minimal-schema.json\"]"**: this label provides the path to each JSON Schema generate by the agent

The Data Models Path is the relative path to the specs folder of the Data Models repository.

For example, for the data model TideSensorObserved the label should set like this: ngsiagent.datamodels="[\"/Pixel/TideSensorObserved/schema.json\"]"

- **Labels for daemon agents**
  - **ngsiagent.internal.port**: the port exposing the API, it has also to be specified with ÈXPOSE
  - **ngsiagent.internal.path**: the base path of the API configured in the agent
  - **ngsiagent.external.path**: the base path of the API configured in the proxy to expose the agent
- **Labels for scheduled agents**
  - **ngsiagent.scheduled:** the frequency to run the agent (CRON format)

```
* * * * * *
| | | | | |
| | | | | +-- Year              (range: 1900-3000)
| | | | +---- Day of the Week   (range: 1-7, 1 standing for Monday)
| | | +------ Month of the Year (range: 1-12)
| | +-------- Day of the Month  (range: 1-31)
| +---------- Hour              (range: 0-23)
+------------ Minute            (range: 0-59)
```

- **Examples**
- **Daemon**

```
FROM nginx
LABEL ngsiagent="pixel"
LABEL ngsiagent.type="daemon"
LABEL ngsiagent.internal.port="80"
LABEL ngsiagent.internal.path="/api"
LABEL ngsiagent.external.path="/empire"
LABEL ngsiagent.datasources="[\"urn:pixel:DataSource:dummies\"]"
LABEL ngsiagent.datamodels="[\"/Dummies/minimal-schema.json\"]"
EXPOSE 80
ENV PIXEL=test
ENV MYTEST=pixel
RUN mkdir /usr/share/nginx/html/api
RUN echo "Execute order 66" > /usr/share/nginx/html/api/order
ENTRYPOINT ["nginx"]
CMD ["-g", "daemon off;"]
```

- **Scheduled**

```
FROM ubuntu
LABEL ngsiagent="pixel"
LABEL ngsiagent.type="scheduled"
LABEL ngsiagent.scheduled="* * * * *"
LABEL ngsiagent.datasources="[\"urn:pixel:DataSource:dummies\"]"
LABEL ngsiagent.datamodels="[\"/Dummies/minimal-schema.json\"]"
ENV PIXEL=test
ENV MYTEST=pixel
ENV SCHEDULED_DELAY=0
COPY docker_entrypoint.sh /docker_entrypoint.sh
RUN chmod u+rx /docker_entrypoint.sh
ENTRYPOINT ["/docker_entrypoint.sh"]
```

- **Manual**

```
FROM ubuntu
LABEL ngsiagent="pixel"
LABEL ngsiagent.type="manual"
LABEL ngsiagent.datasources="[\"urn:pixel:DataSource:dummies\"]"
LABEL ngsiagent.datamodels="[\"/Dummies/minimal-schema.json\"]"
ENTRYPOINT ["/bin/bash"]
CMD ["date"]
```

4.4.1.2. Quick start guide

NGSI Image management

For security purpose, right now you have to **docker pull** the NGSI Agents images directly on the host. A next version will propose to manage that using the API.

You can request the list of available NGSI Agents images already available on the host with an API call:

```
curl  -H "X-Auth-Token: default"  http://172.17.0.1:8888/api/images
[
    {
        "id": "sha256:620877b976447800bc7ce8672d6b688369b429ad77afba0968f20088c8daf8fd",
        "tag": "pixelh2020/frbodtidesensor:1.0.0"
    }
]
```

Get a template

When you have chosen the image of your NGSI Agents, you can generate a template to create it

```
curl  -H "X-Auth-Token: default"
http://172.17.0.1:8888/api/images/sha256:620877b976447800bc7ce8672d6b688369b429ad77afba096
8f20088c8daf8fd/template
{
    "name": "/?[a-zA-Z0-9_-]+",
    "image": "pixelh2020/frbodtidesensor:1.0.0",
    "type": "scheduled",
    "scheduled": "22 * * * *",
    "datasources": [
        "urn:pixel:DataSource:frbod:TideSensorObserved"
    ],
    "datamodels": [
        "/Pixel/TideSensorObserved/schema.json"
    ],
    "environment": [
        {
            "key": "PATH",
            "value": "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
        },
        {
            "key": "NODE_VERSION",
            "value": "13.6.0"
        },
        {
            "key": "YARN_VERSION",
            "value": "1.21.1"
        },
        {
            "key": "NODE_TLS_REJECT_UNAUTHORIZED",
            "value": "0"
        },
        {
            "key": "ORION_URL",
            "value": "changeit"
        },
        {
            "key": "NAMI_AUTH_URL",
            "value": "https://nami.bordeaux-port.fr/?q=accueil"
        },
        {
            "key": "NAMI_URL",
            "value": "https://nami.bordeaux-port.fr/hauteurs"
        },
        {
            "key": "NAMI_LOGIN",
            "value": "changeit"
        },
        {
            "key": "NAMI_PASSWORD",
            "value": "changeit"
        },
        {
            "key": "FIWARE_SERVICE="
        },
        {
            "key": "FIWARE_SERVICE_PATH="
        }
    ]
}
```

Create the NGSI Agent

Change the name of the agent (it will be the name of the container) and adjust the parameters or let their default values.
Be sure your name matches the given pattern.

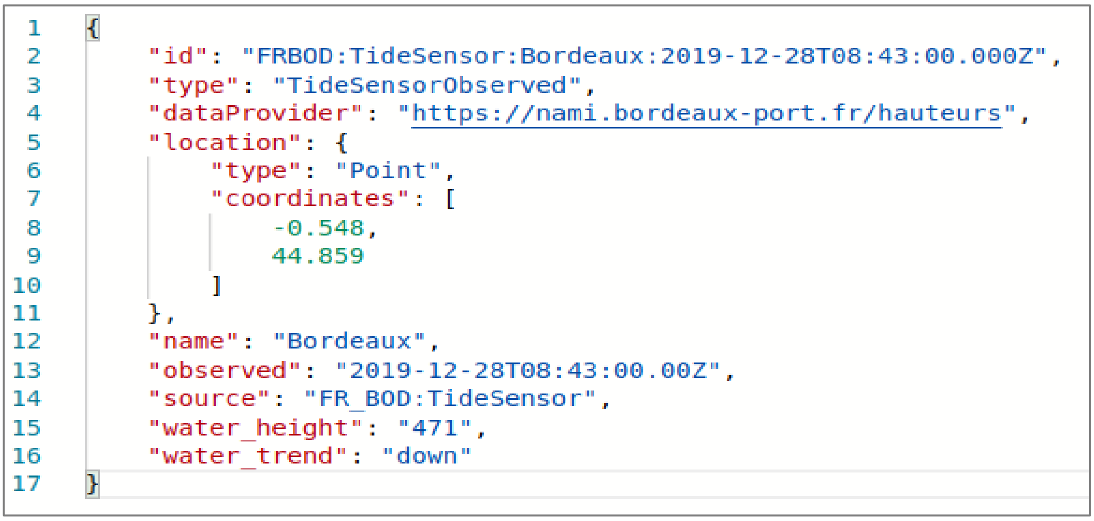
```
curl  -X POST -H "X-Auth-Token: default"  http://172.17.0.1:8888/api/ngsiagent -d @- <<EOF
{
    "name": "/my-agent",
    "image": "pixelh2020/frbodtidesensor:1.0.0",
    "type": "scheduled",
    "scheduled": "22 * * * *",
    "datasources": [
        "urn:pixel:DataSource:frbod:TideSensorObserved"
    ],
    "datamodels": [
        "/Pixel/TideSensorObserved/schema.json"
    ],
    "environment": [
        {
            "key": "ORION_URL",
            "value": "http://172.17.0.1:1026"
        },
        {
            "key": "NAMI_LOGIN",
            "value": "mylogin"
        },
        {
            "key": "NAMI_PASSWORD",
            "value": "mypassword"
        }
    ]
}
EOF
```

4.4.2. PIXEL Information Hub

4.4.2.1. Importing Data Sources from DAL to Information Hub

When started, Information Hub (specifically, Orion Data Collector) subscribes to the *DataSource* entity type in Orion Context Broker. The *DataSource* entities are managed by DAL Inquisitor and represent a registry of data sources available in Orion Context Broker. When a new data entity is created in Orion, DAL Inquisitor checks if the data source is already registered and creates a corresponding *DataSource* entity if needed. Because Information Hub is subscribed to the *DataSource* entity changes, Orion sends a notification message to the Information Hub's listener and thus notifies the Information Hub that a new data source has been created. The notification message contains the new *DataSource* entity.

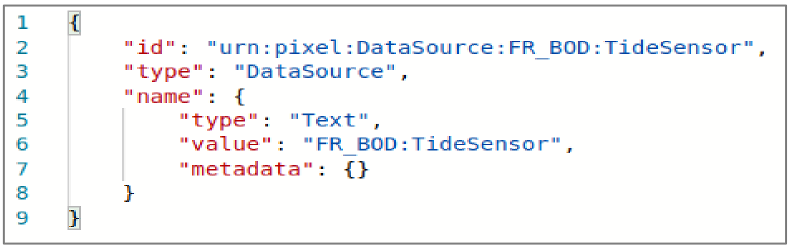
Let us take as an example tide sensor data source. The type of Orion entity is *TideSensorObserved* (specified by the *type* attribute) and the source ID is '*FR_BOD:TideSensor*' (specified by the *source* attribute). A sample entity is depicted below in *keyValues* (compact) representation:



```
{
    "id": "FRBOD:TideSensor:Bordeaux:2019-12-28T08:43:00.000Z",
    "type": "TideSensorObserved",
    "dataProvider": "https://nami.bordeaux-port.fr/hauteurs",
    "location": {
        "type": "Point",
        "coordinates": [
            -0.548,
            44.859
        ]
    },
    "name": "Bordeaux",
    "observed": "2019-12-28T08:43:00.00Z",
    "source": "FR_BOD:TideSensor",
    "water_height": "471",
    "water_trend": "down"
}
```

Figure 41: Example of TideSensorObserved

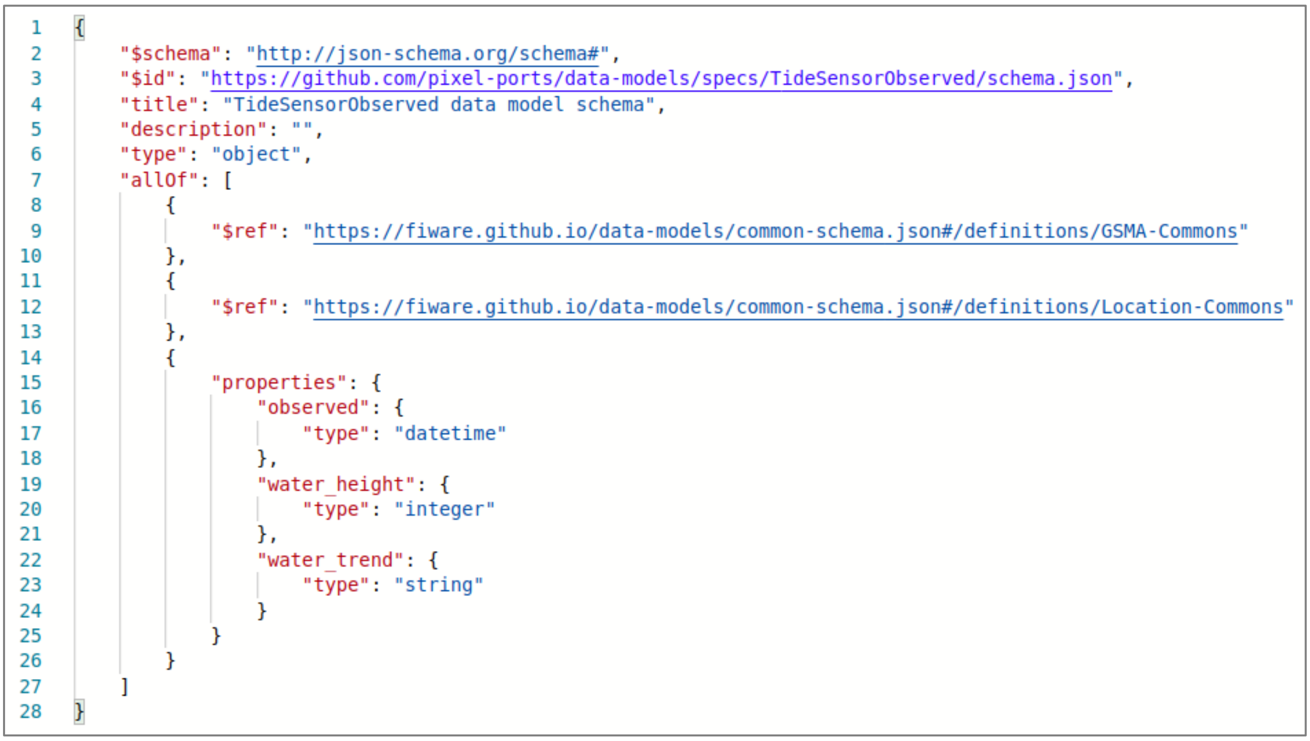
The corresponding *DataSource* entity created by DAL Inquisitor looks as follows:



```
{
    "id": "urn:pixel:DataSource:FR_BOD:TideSensor",
    "type": "DataSource",
    "name": {
        "type": "Text",
        "value": "FR_BOD:TideSensor",
        "metadata": {}
    }
}
```

Figure 42: Entity created by Data Acquisition Layer

Each data source has a corresponding data model which is stored in Orion as a *DataModel* entity. The *DataModel* entity specifies the schema of the data model. The schema is mandatory, a data source cannot be imported to the Information Hub without it. A schema for the *TideSensorObserved* data model is depicted below:



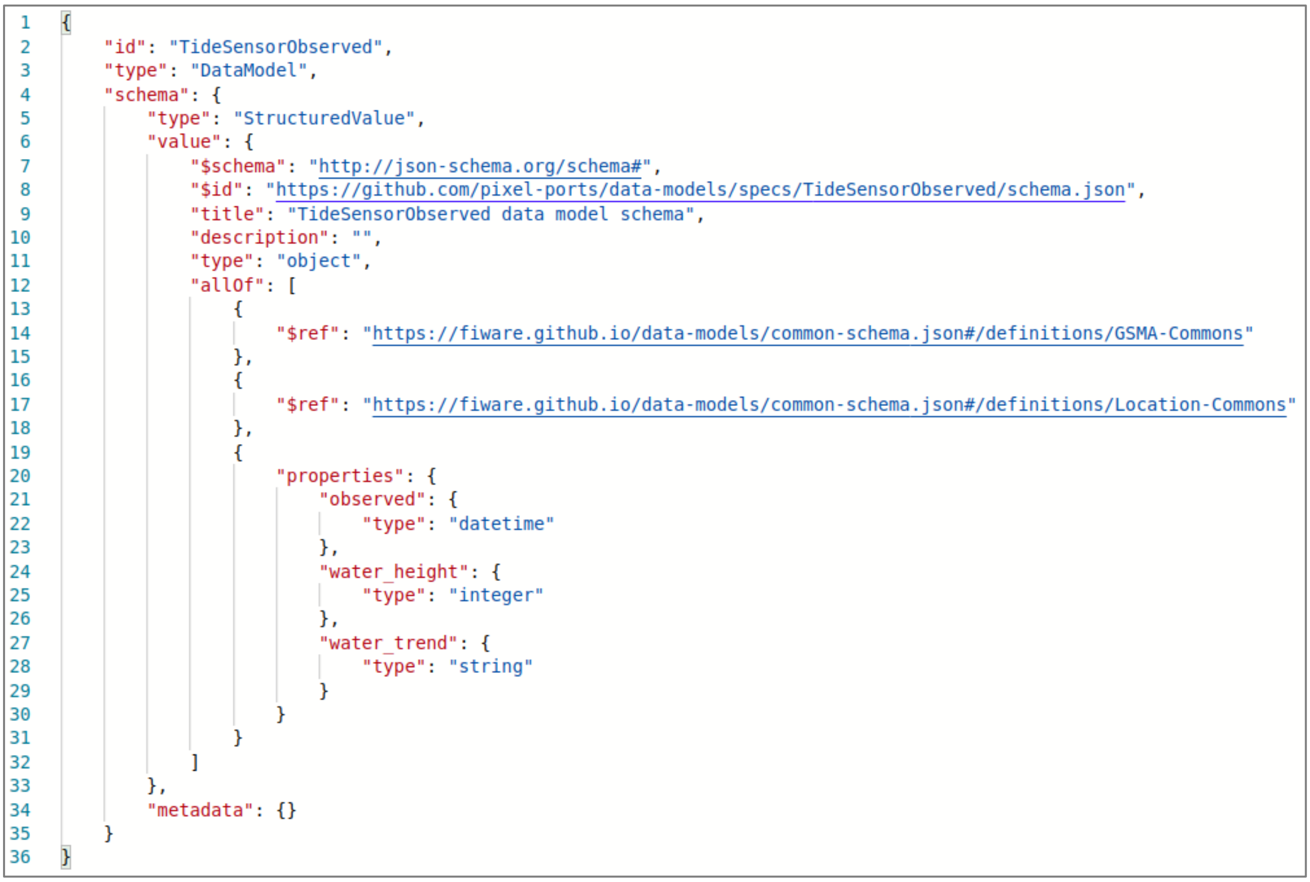
```
{
    "$schema": "http://json-schema.org/schema#",
    "$id": "https://github.com/pixel-ports/data-models/specs/TideSensorObserved/schema.json",
    "title": "TideSensorObserved data model schema",
    "description": "",
    "type": "object",
    "allOf": [
        {
            "$ref": "https://fiware.github.io/data-models/common-schema.json#/definitions/GSMA-Commons"
        },
        {
            "$ref": "https://fiware.github.io/data-models/common-schema.json#/definitions/Location-Commons"
        },
        {
            "properties": {
                "observed": {
                    "type": "datetime"
                },
                "water_height": {
                    "type": "integer"
                },
                "water_trend": {
                    "type": "string"
                }
            }
        }
    ]
}
```

Figure 43: Schema for the TideSensorObserved Data Model

The schema is stored in Orion as a *value* attribute of a *DataModel* entity with an ID corresponding to the Orion entity type (e.g. *TideSensorObserved*).

```
{
   "id": "<Orion type>",
   "type": "DataModel",
   "schema": {
      "type": "StructuredValue",
      "value": {<schema content>},
      "metadata": {}
   }
}
```

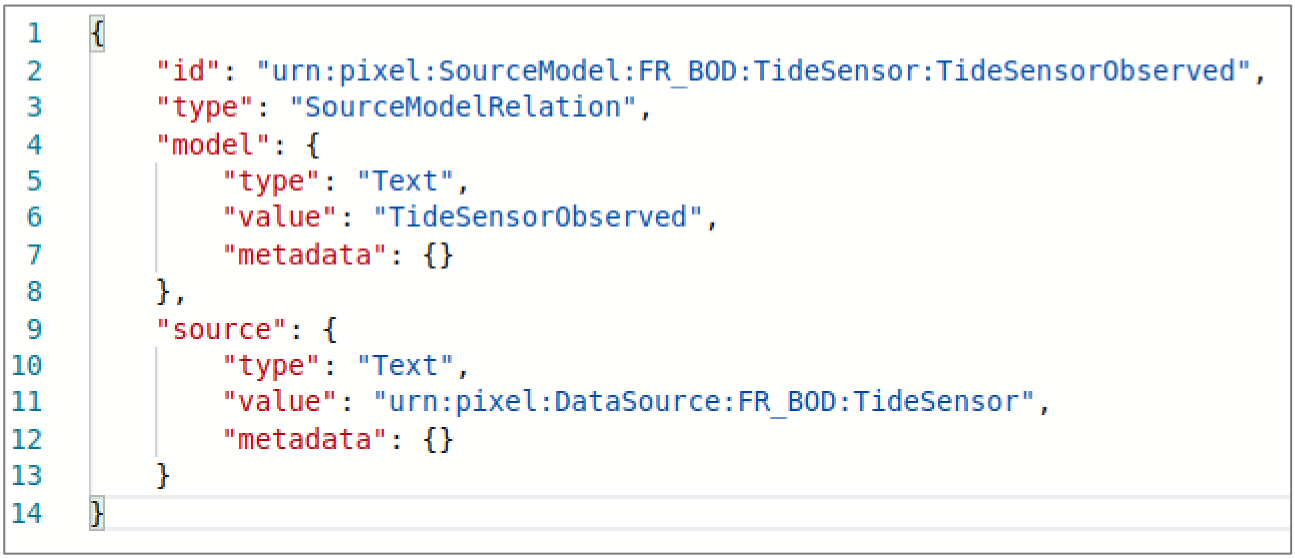
The *DataModel* entity for the *TideSensorObserved* Orion type is depicted below:



```json
{
    "id": "TideSensorObserved",
    "type": "DataModel",
    "schema": {
        "type": "StructuredValue",
        "value": {
            "$schema": "http://json-schema.org/schema#",
            "$id": "https://github.com/pixel-ports/data-models/specs/TideSensorObserved/schema.json",
            "title": "TideSensorObserved data model schema",
            "description": "",
            "type": "object",
            "allOf": [
                {
                    "$ref": "https://fiware.github.io/data-models/common-schema.json#/definitions/GSMA-Commons"
                },
                {
                    "$ref": "https://fiware.github.io/data-models/common-schema.json#/definitions/Location-Commons"
                },
                {
                    "properties": {
                        "observed": {
                            "type": "datetime"
                        },
                        "water_height": {
                            "type": "integer"
                        },
                        "water_trend": {
                            "type": "string"
                        }
                    }
                }
            ]
        },
        "metadata": {}
    }
}
```

Figure 44: Orion Type for the TideSensorObserved Data Model

When registering a new data source, DAL Inquisitor creates in addition to the *DataSource* entity also a *SourceModelRelation* entity which specifies the data model for the specified data source. *SourceModelRelation* entity for the '*FR_BOD:TideSensor*' data source is depicted below:



```json
{
    "id": "urn:pixel:SourceModel:FR_BOD:TideSensor:TideSensorObserved",
    "type": "SourceModelRelation",
    "model": {
        "type": "Text",
        "value": "TideSensorObserved",
        "metadata": {}
    },
    "source": {
        "type": "Text",
        "value": "urn:pixel:DataSource:FR_BOD:TideSensor",
        "metadata": {}
    }
}
```

Figure 45: SourceModelRelation entity for the Orion entity

When Information Hub receives a notification from Orion about new data source, following steps are taken:

- IH reads the *DataSource* entity contained in the notification message and extracts source URN.
- using the source URN, IH queries the Orion and retrieves corresponding *SourceModelRelation* entity.
- IH extracts data model ID from the *SourceModelRelation* entity.

- IH retrieves the *DataModel* entity with specified ID. If the *DataModel* entity is not available, import of the data source will fail.
- IH extracts schema from the *DataModel* entity, parses it and retrieves referenced external schemas if any.
- IH registers a data source type corresponding to the data model (Orion type) if not yet registered. Name of the data source type matches the model name (which matches Orion type) where forbidden character slash '/' is replaced with colon ':'.
- IH registers a data source corresponding to the Orion source. Name of the source matches the Orion source name (value of the *source* attribute of the Orion data entity) where forbidden character slash '/' is replaced with colon ':'.
- IH imports data source initial data from Orion (data entities which are already stored in Orion).
- IH subscribes to the Orion source to receive notifications when new entities are created, or existing ones modified.

To sum up, to import data source from DAL to Information Hub, following has to be done:

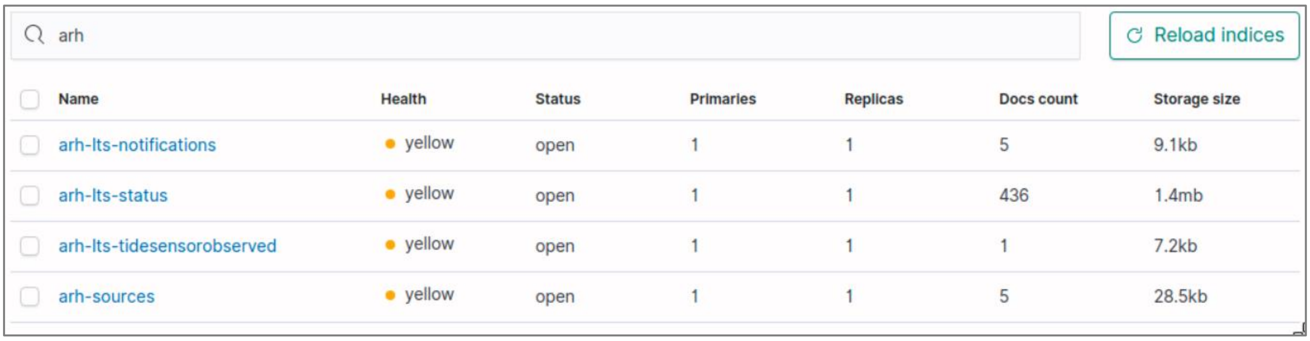
- prepare schema of your data model.
- create corresponding *DataModel* entity in Orion which contains the schema.
- insert data to Orion (i.e. create the first data entity).
- Information Hub will receive a notification that new data source has been created and automatically register the data source and import initial data.

Note: if data model is not available at the time when data source is created (when first data entity is created), importing data source to Information Hub will fail.

Data is stored to Elasticsearch index with the name obtained by concatenating following parts and separating them with '-' character:

- '*arh*' prefix (Information Hub prefix).
- storage type ('*lts*' for long-term storage or '*sts*' for short-term storage).
- source type ID in lower case.

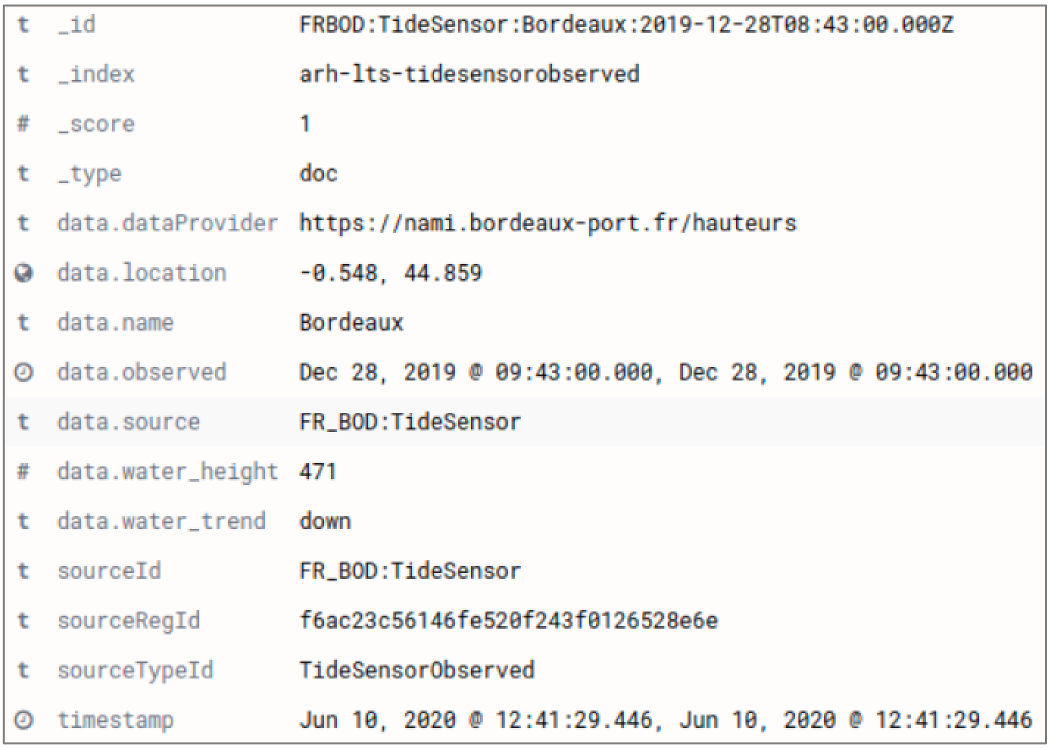
Data for all sources of the same source type is stored to the same index. For example, data of type *TideSensorObserved* (from all sources of this source type) is stored to the index with name '*arh-lts-tidesensorobserved*'. The figure below depicts the Elasticsearch indexes created by Information Hub after *TideSensorObserved* source has been imported:



□ Name	Health	Status	Primaries	Replicas	Docs count	Storage size
□ arh-lts-notifications	● yellow	open	1	1	5	9.1kb
□ arh-lts-status	● yellow	open	1	1	436	1.4mb
□ arh-lts-tidesensorobserved	● yellow	open	1	1	1	7.2kb
□ arh-sources	● yellow	open	1	1	5	28.5kb

Figure 46: Elasticsearch indexes

The figure below depicts the data record stored in Elasticsearch and presented in Kibana corresponding to the *TideSensorObserved* entity used in the example above:



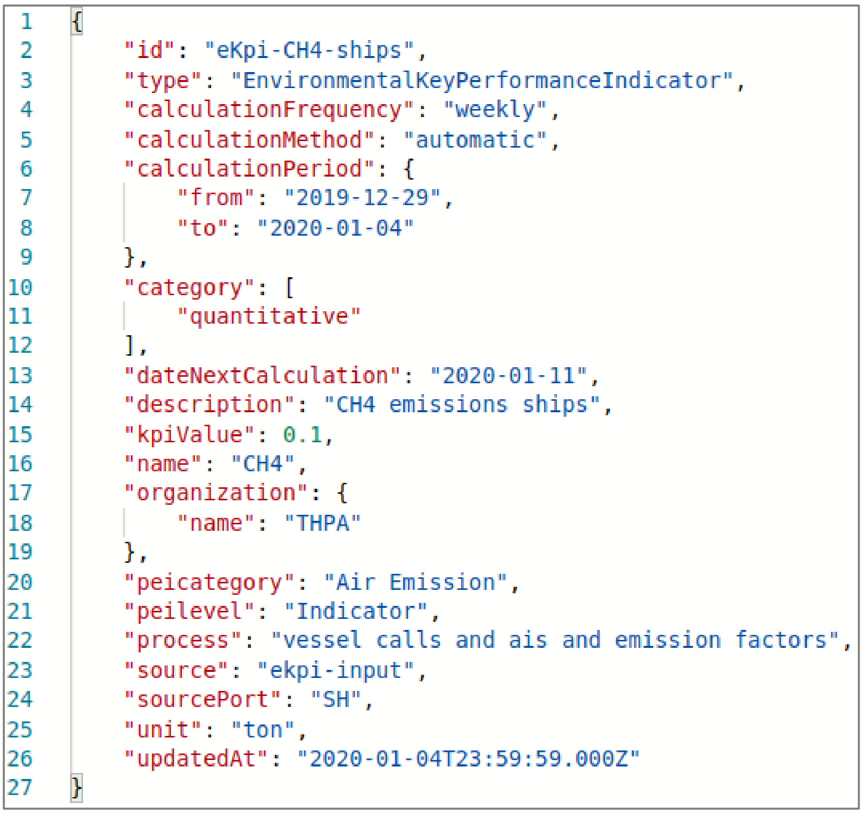
t	_id	FRBOD:TideSensor:Bordeaux:2019-12-28T08:43:00.000Z
t	_index	arh-lts-tidesensorobserved
#	_score	1
t	_type	doc
t	data.dataProvider	https://nami.bordeaux-port.fr/hauteurs
🌐	data.location	-0.548, 44.859
t	data.name	Bordeaux
🕘	data.observed	Dec 28, 2019 @ 09:43:00.000, Dec 28, 2019 @ 09:43:00.000
t	data.source	FR_BOD:TideSensor
#	data.water_height	471
t	data.water_trend	down
t	sourceId	FR_BOD:TideSensor
t	sourceRegId	f6ac23c56146fe520f243f0126528e6e
t	sourceTypeId	TideSensorObserved
🕘	timestamp	Jun 10, 2020 @ 12:41:29.446, Jun 10, 2020 @ 12:41:29.446

Figure 47: Register stored in elasticsearch

4.4.2.1.1. Data Flattening

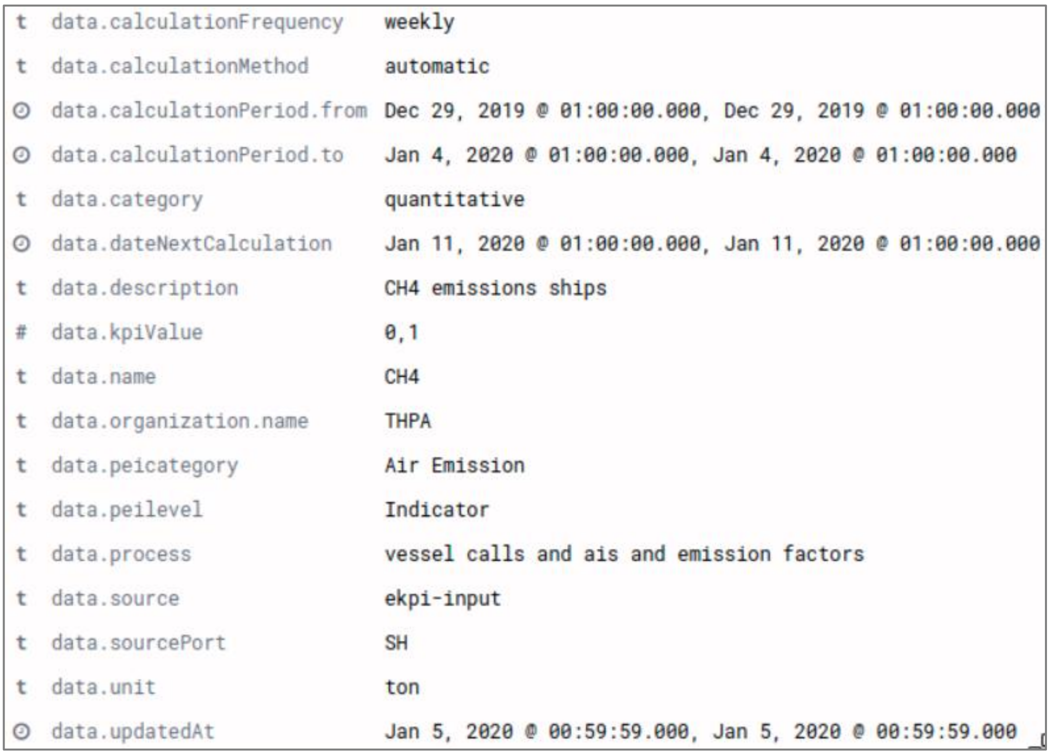
Structured data (data with nested objects) is not supported by Information Hub and has to be flattened to flat structure. The data flattening process is performed by the Orion Data Collector module of Information Hub after retrieving from Orion. Data is stored to Elasticsearch in flattened form. When retrieving the data from Information Hub using Data extractor API, flattened data is transformed back to the original form so the whole process is transparent to the user.

Nested attributes (single and multi-level) are flattened to a flat list of attributes using the dot separator. For example, the *EnvironmentalKeyPerformanceIndicator* Orion entity depicted in figure below as it is returned by Orion in *keyValues* format contains two nested attributes - *calculationPeriod* and *organization*. These two attributes are transformed to three attributes: ‘*calculationPeriod.from*’, ‘*calculationPeriod.from*’ and ‘*organization.name*’. The resulting data record is depicted in the figure below as it is shown in Kibana.



```
{
    "id": "eKpi-CH4-ships",
    "type": "EnvironmentalKeyPerformanceIndicator",
    "calculationFrequency": "weekly",
    "calculationMethod": "automatic",
    "calculationPeriod": {
        "from": "2019-12-29",
        "to": "2020-01-04"
    },
    "category": [
        "quantitative"
    ],
    "dateNextCalculation": "2020-01-11",
    "description": "CH4 emissions ships",
    "kpiValue": 0.1,
    "name": "CH4",
    "organization": {
        "name": "THPA"
    },
    "peicategory": "Air Emission",
    "peilevel": "Indicator",
    "process": "vessel calls and ais and emission factors",
    "source": "ekpi-input",
    "sourcePort": "SH",
    "unit": "ton",
    "updatedAt": "2020-01-04T23:59:59.000Z"
}
```

Figure 48: Resulting data stored



t	data.calculationFrequency	weekly
⊙	data.calculationMethod	automatic
⊙	data.calculationPeriod.from	Dec 29, 2019 @ 01:00:00.000, Dec 29, 2019 @ 01:00:00.000
⊙	data.calculationPeriod.to	Jan 4, 2020 @ 01:00:00.000, Jan 4, 2020 @ 01:00:00.000
t	data.category	quantitative
⊙	data.dateNextCalculation	Jan 11, 2020 @ 01:00:00.000, Jan 11, 2020 @ 01:00:00.000
t	data.description	CH4 emissions ships
#	data.kpiValue	0,1
t	data.name	CH4
t	data.organization.name	THPA
t	data.peicategory	Air Emission
t	data.peilevel	Indicator
t	data.process	vessel calls and ais and emission factors
t	data.source	ekpi-input
t	data.sourcePort	SH
t	data.unit	ton
⊙	data.updatedAt	Jan 5, 2020 @ 00:59:59.000, Jan 5, 2020 @ 00:59:59.000

Figure 49: Resulting data shown in Kibana

4.4.2.2. Retrieving Data from Information Hub

Data Extractor module of Information Hub provides a REST API for retrieving information about registered data sources and source types, retrieving time-series data from a selected data source using specified filters.

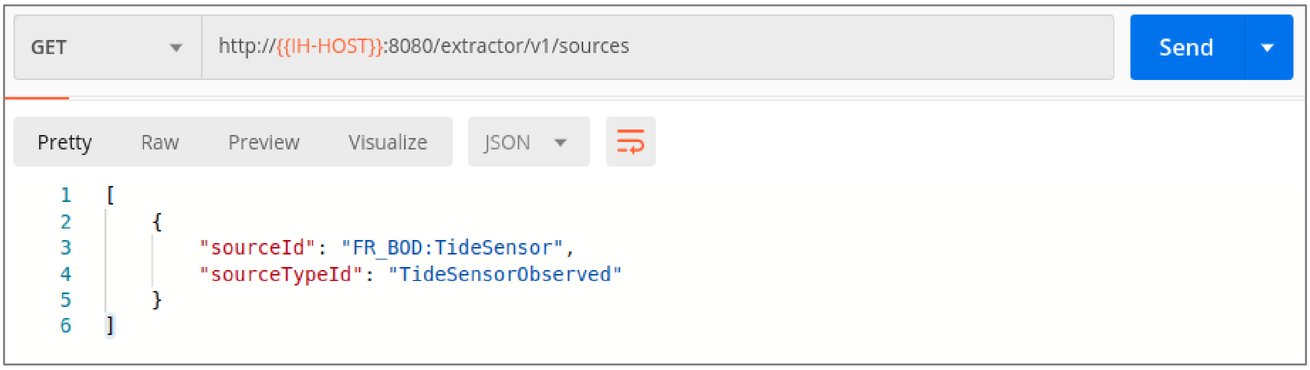
Data Extractor API is available by default at the following endpoint:

http://<IH_HOST>:8080/extractor/

4.4.2.2.1. Retrieving List of Registered Sources

To retrieve a list of all registered data sources in Information Hub, use the ‘GET /sources’ operation. The response contains list of data sources and for each source following attributes:

- *sourceId*: source ID.
- *sourceTypeId*: ID of corresponding source type in Information Hub.



http://{{IH-HOST}}:8080/extractor/v1/sources

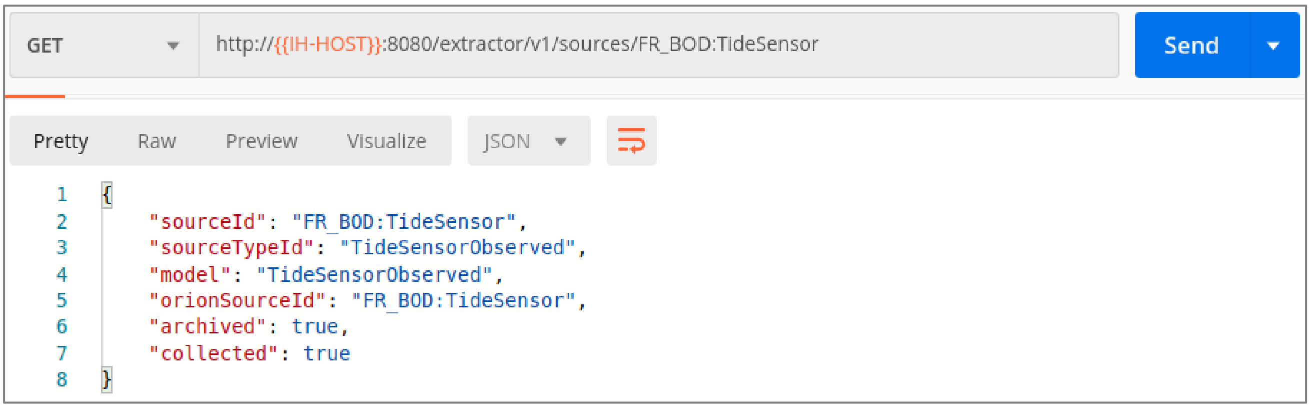
```
[
    {
        "sourceId": "FR_BOD:TideSensor",
        "sourceTypeId": "TideSensorObserved"
    }
]
```

Figure 50: List of sources with their attributes

4.4.2.2.2. Retrieving Info about Specific Source

To retrieve detailed information about a specific data source, use the ‘GET /sources/{sourceId}’ operation. The response contains following attributes:

- *sourceId*: source ID
- *sourceTypeId*: ID of corresponding source type in Information Hub
- *model*: data model name
- *orionSourceId*: originating Orion source
- *archived*: boolean value specifying whether source data is being archived (stored to Elasticsearch)
- *collected*: boolean value specifying whether source data is being collected by the Data Collector



http://{{IH-HOST}}:8080/extractor/v1/sources/FR_BOD:TideSensor

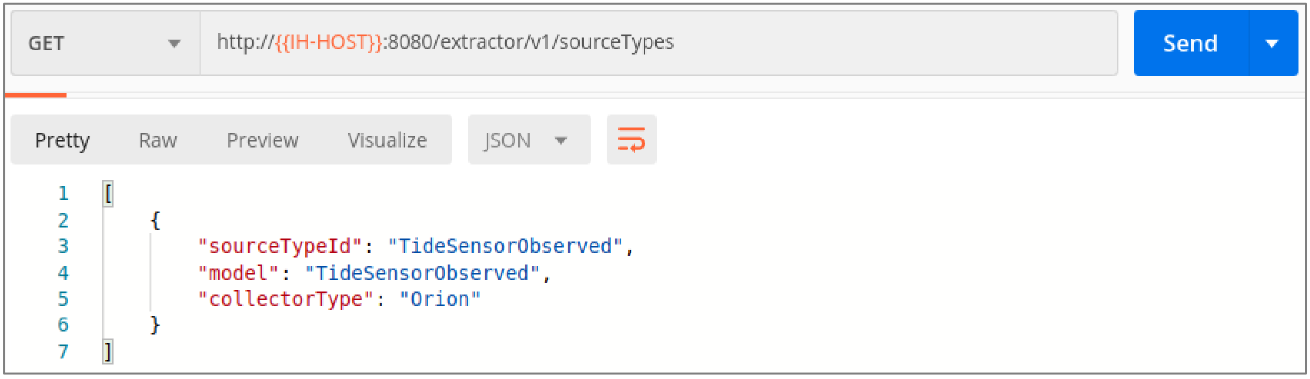
```
{
    "sourceId": "FR_BOD:TideSensor",
    "sourceTypeId": "TideSensorObserved",
    "model": "TideSensorObserved",
    "orionSourceId": "FR_BOD:TideSensor",
    "archived": true,
    "collected": true
}
```

Figure 51: Detailed information about a specific data source

4.4.2.2.3. Retrieving List of Registered Sources Types

To retrieve a list of all data source types in Information Hub, use the 'GET /sourceTypes' operation. The response contains list of data source types and for each source type following attributes:

- *sourceTypeId*: data source type ID.
- *model*: data model name.
- *collectorType*: type of Data Collector used by Information Hub to collect data from source of this source type (e.g. Orion Data Collector, AIS Data Collector).



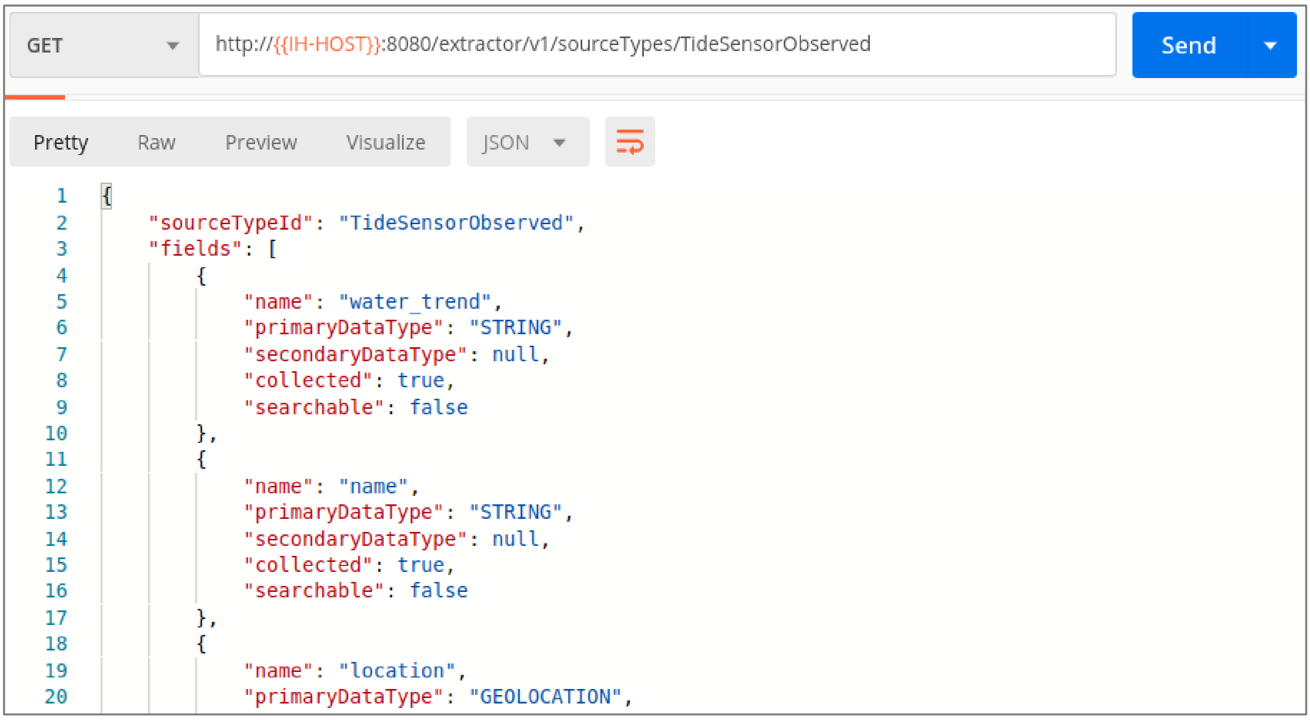
```
[
    {
        "sourceTypeId": "TideSensorObserved",
        "model": "TideSensorObserved",
        "collectorType": "Orion"
    }
]
```

Figure 52: List of all data source types in Information Hub

4.4.2.2.4. Retrieving Info about Specific Source Type

To retrieve detailed information about a specific data source type, use the 'GET /sourceTypes/{typeId}' operation. The response contains following attributes:

- *sourceTypeId*: data source type ID.
- *fields*: list of fields of this source type. For each field following attributes are provided:
    - *name*: name of the field.
    - *primaryDataType*: primary data type of the field.
    - *secondaryDataType*: secondary data type of the field. If a field value is an array, the primary data type is 'array' and the secondary data type specifies the type of array values.
    - *collected*: boolean value specifying whether this field is being collected by the Data Collector.
    - *searchable*: boolean value specifying whether this field is indexed for search by Elasticsearch.
- *model*: data model name.
- *schema*: data model schema content.
- *collectorType*: type of Data Collector used by Information Hub to collect data from source of this source type (e.g. Orion Data Collector, AIS Data Collector).



http://{{IH-HOST}}:8080/extractor/v1/sourceTypes/TideSensorObserved

```
{
    "sourceTypeId": "TideSensorObserved",
    "fields": [
        {
            "name": "water_trend",
            "primaryDataType": "STRING",
            "secondaryDataType": null,
            "collected": true,
            "searchable": false
        },
        {
            "name": "name",
            "primaryDataType": "STRING",
            "secondaryDataType": null,
            "collected": true,
            "searchable": false
        },
        {
            "name": "location",
            "primaryDataType": "GEOLOCATION",
```

Figure 53: Retrieving Info about a Specific Source Type

4.4.2.2.5. Retrieving Time-Series Data

To retrieve data for a specific data source in a specific time interval, use the ‘POST /data’ operation. The query is specified in the POST body and can contain following parameters:

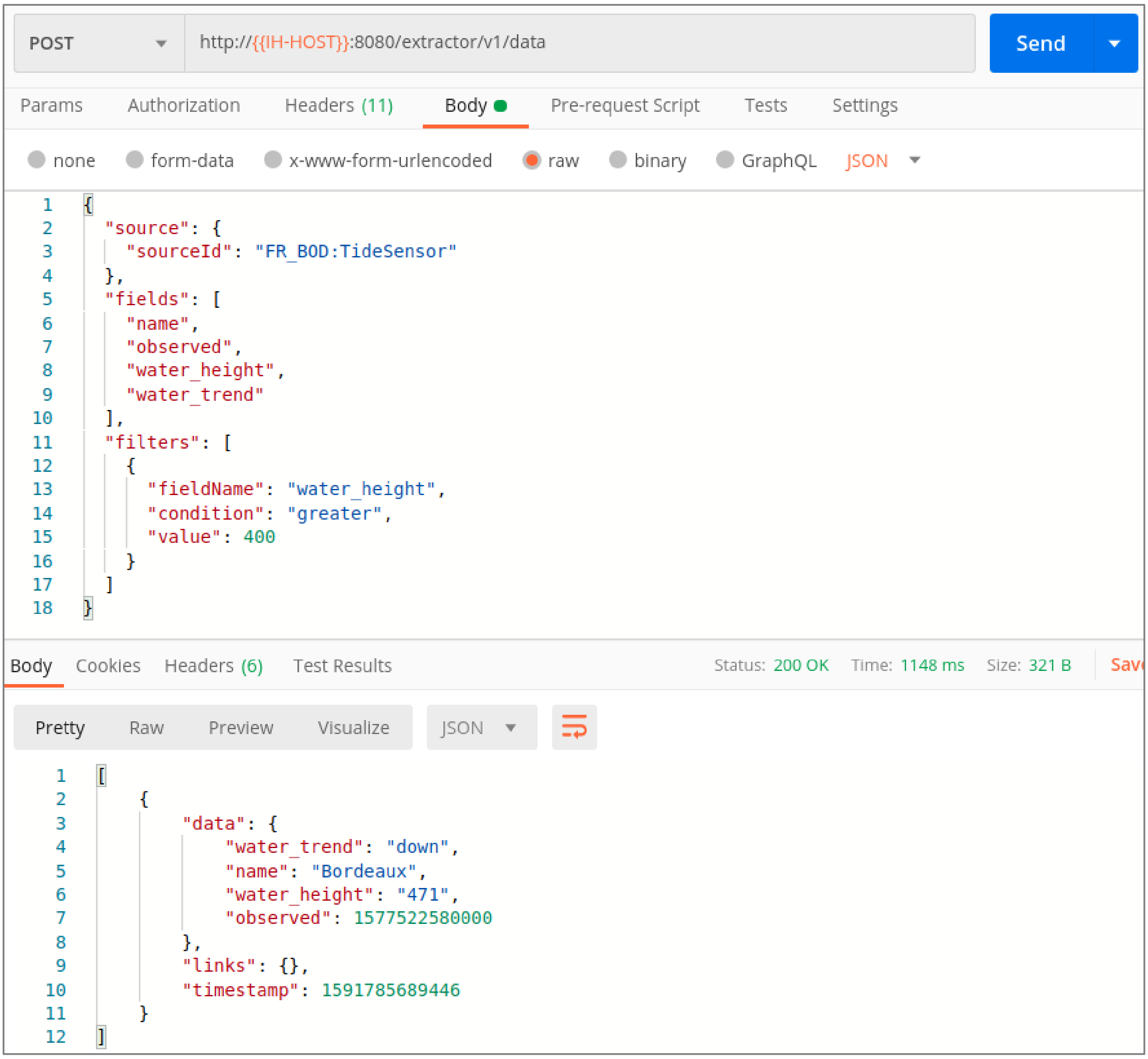
- *sourceId* (mandatory): ID of the source.
- *fields* (optional): list of fields to return.
- *filters* (optional): list of filters to apply. A filter is specified as an object with following three attributes:
    - *fieldname.*
    - *condition*: possible values are *equal*, *notEqual*, *equalOrGreater*, *equalOrLower*, *greater*, *lower*
    - *value.*
- *timeIntervals* (optional): array of time intervals for which to return data (applies to the timestamp attribute - time when record was stored to the IH).
- *storageTypes* (optional): storage types to include in the search. Possible values are *STS* (short-term storage) and *LTS* (long-term storage).

Data can be returned in JSON or CSV format. Requested format can be specified using *Accept* HTTP header and appropriate MIME type:

```
Accept: application/json | text/csv
```

If *Accept* header is not specified, data is returned in JSON format.

The figure below depicts a query for retrieving tide sensor measurements from ‘*FR_BOD:TideSensor*’ source with *name*, *observed*, *water height* and *water height* fields where *water height* value is greater than 400. Requested data format is JSON.



POST http://{{IH-HOST}}:8080/extractor/v1/data

```
{
  "source": {
    "sourceId": "FR_BOD:TideSensor"
  },
  "fields": [
    "name",
    "observed",
    "water_height",
    "water_trend"
  ],
  "filters": [
    {
      "fieldName": "water_height",
      "condition": "greater",
      "value": 400
    }
  ]
}
```

Status: 200 OK Time: 1148 ms Size: 321 B

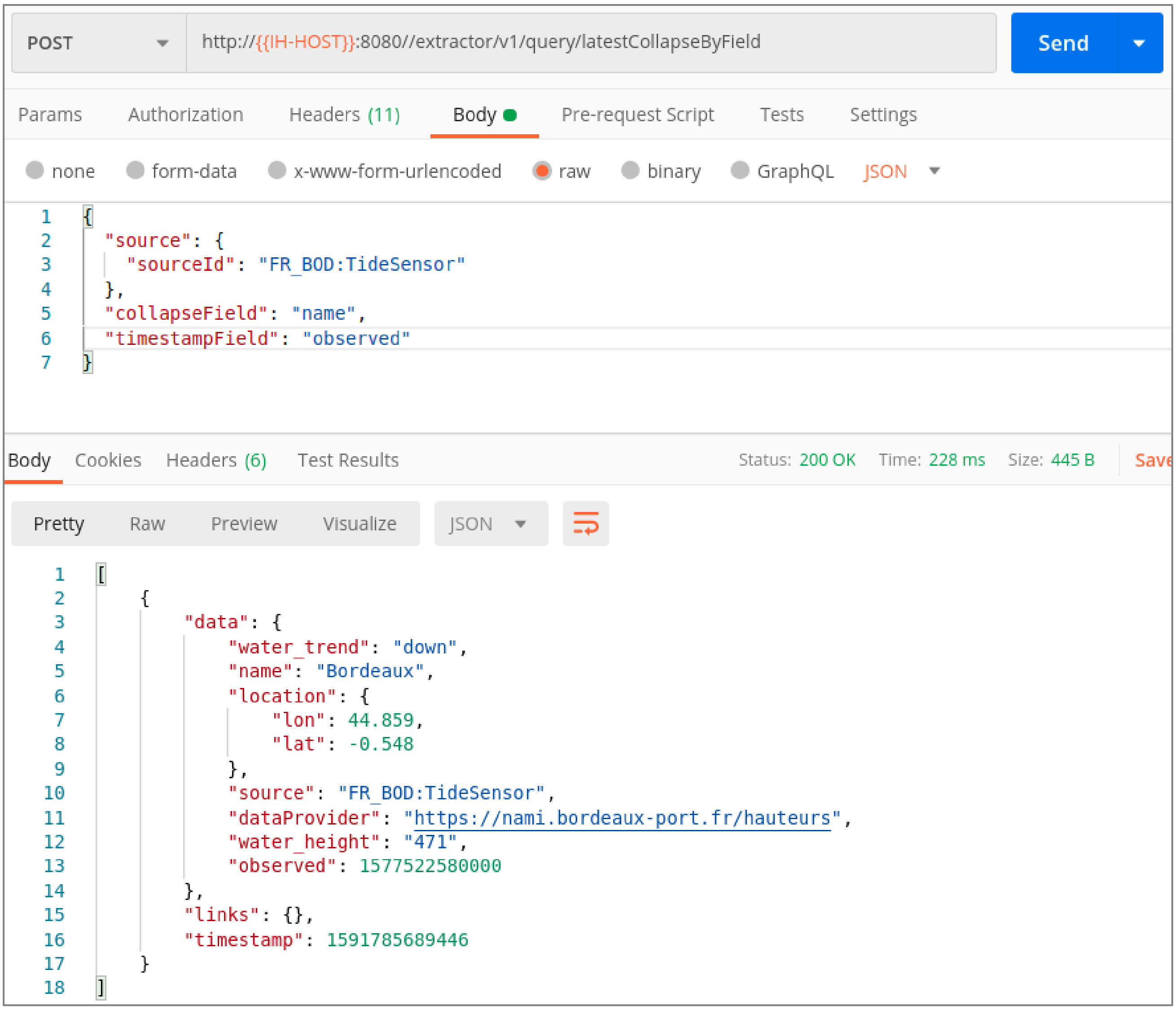
```
[
    {
        "data": {
            "water_trend": "down",
            "name": "Bordeaux",
            "water_height": "471",
            "observed": 1577522580000
        },
        "links": {},
        "timestamp": 1591785689446
    }
]
```

Figure 54: Retrieving Time-Series Data

4.4.2.2.6. Retrieving latest data record for each sensor

To retrieve the latest data record for each sensor of a specific data source, use the ‘POST /query/latestCollapseByField’ operation. The query is specified in the POST body and can contain following parameters:

- *sourceId* (mandatory): ID of the source.
- *collapseField* (mandatory): field name containing the sensor identifier.
- *timestampField* (mandatory): field name containing the timestamp of sensor values.



POST http://{{IH-HOST}}:8080//extractor/v1/query/latestCollapseByField

```
{
  "source": {
    "sourceId": "FR_BOD:TideSensor"
  },
  "collapseField": "name",
  "timestampField": "observed"
}
```

Status: 200 OK Time: 228 ms Size: 445 B

JSON

```
[
    {
        "data": {
            "water_trend": "down",
            "name": "Bordeaux",
            "location": {
                "lon": 44.859,
                "lat": -0.548
            },
            "source": "FR_BOD:TideSensor",
            "dataProvider": "https://nami.bordeaux-port.fr/hauteurs",
            "water_height": "471",
            "observed": 1577522580000
        },
        "links": {},
        "timestamp": 1591785689446
    }
]
```

Figure 55: Latest data record for TideSensor

4.4.2.3. Elasticsearch Proxy Service

Information Hub provides a proxy service directly to Elasticsearch REST API which is intended for dealing with data that does not originate from the DAL, i.e. to read and write results of models and predictive algorithms. Primarily it will be used by Operational Tools, PIXEL dashboard and models. Besides that, it enables users to make more advanced queries directly to Elasticsearch REST API.

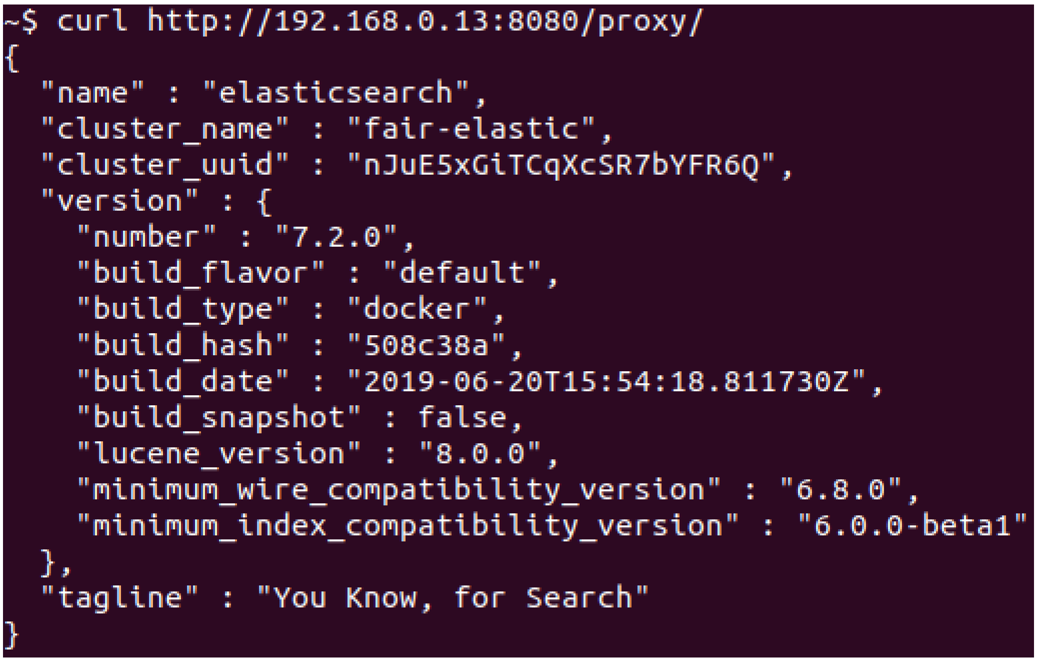
Access to Elasticsearch is restricted. Only read access is allowed to index created and managed by Information Hub (indexes with ‘arh’ prefix). Models are allowed to create their own indexes and have full access to them.

Note: nested data from the DAL is stored to Elasticsearch by Information Hub in flattened form. When retrieving it through Elasticsearch proxy service it is not unflattened - you get it in flattened form. On the other hand, if you use Data Extractor API, Data Extractor takes care for transforming data to the original form.

Elasticsearch is available by default at the following endpoint:

```
http://<IH_HOST>:8080/proxy/
```

The figure below depicts a simple call of Elasticsearch REST API using curl tool:



```
~$ curl http://192.168.0.13:8080/proxy/
{
  "name" : "elasticsearch",
  "cluster_name" : "fair-elastic",
  "cluster_uuid" : "nJuE5xGiTCqXcSR7bYFR6Q",
  "version" : {
    "number" : "7.2.0",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "508c38a",
    "build_date" : "2019-06-20T15:54:18.811730Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Figure 56: Call of Elasticsearch REST API using curl

4.4.2.4. Information Hub Management Console

4.4.2.4.1. Overview

The Information Hub Management Console is a desktop application developed in the Java programming language on top of the JavaFX platform. It is distributed in the form of a JAR file packed into a ZIP package together with the configuration file. For installation and configuration refer to the Installing Information Hub Management Console chapter. The application provides graphic components for controlling, configuring and monitoring the Information Hub. It is intended for system operators that configure and monitor the operation of Information Hub, including:

- configuring connections to storage services,
- selecting data sources for collection and storage,
- setting up data reduction and deletion,
- managing Data Worker Group components,
- monitoring storage capacities,
- monitoring the data flow and reduction,
- monitoring system notifications and errors,
- loading and storing the system configuration,
- triggering system maintenance.

The application can be graphically divided into three main areas: Menu area, Content area and Status area. The Menu area contains buttons for choosing the topmost displayed panel inside the main Content area, displaying GUI elements for controlling and monitoring the Information Hub. The 'Refresh' button reloads information in the currently opened panel by requesting it from relevant controllers.

The available content panels are:

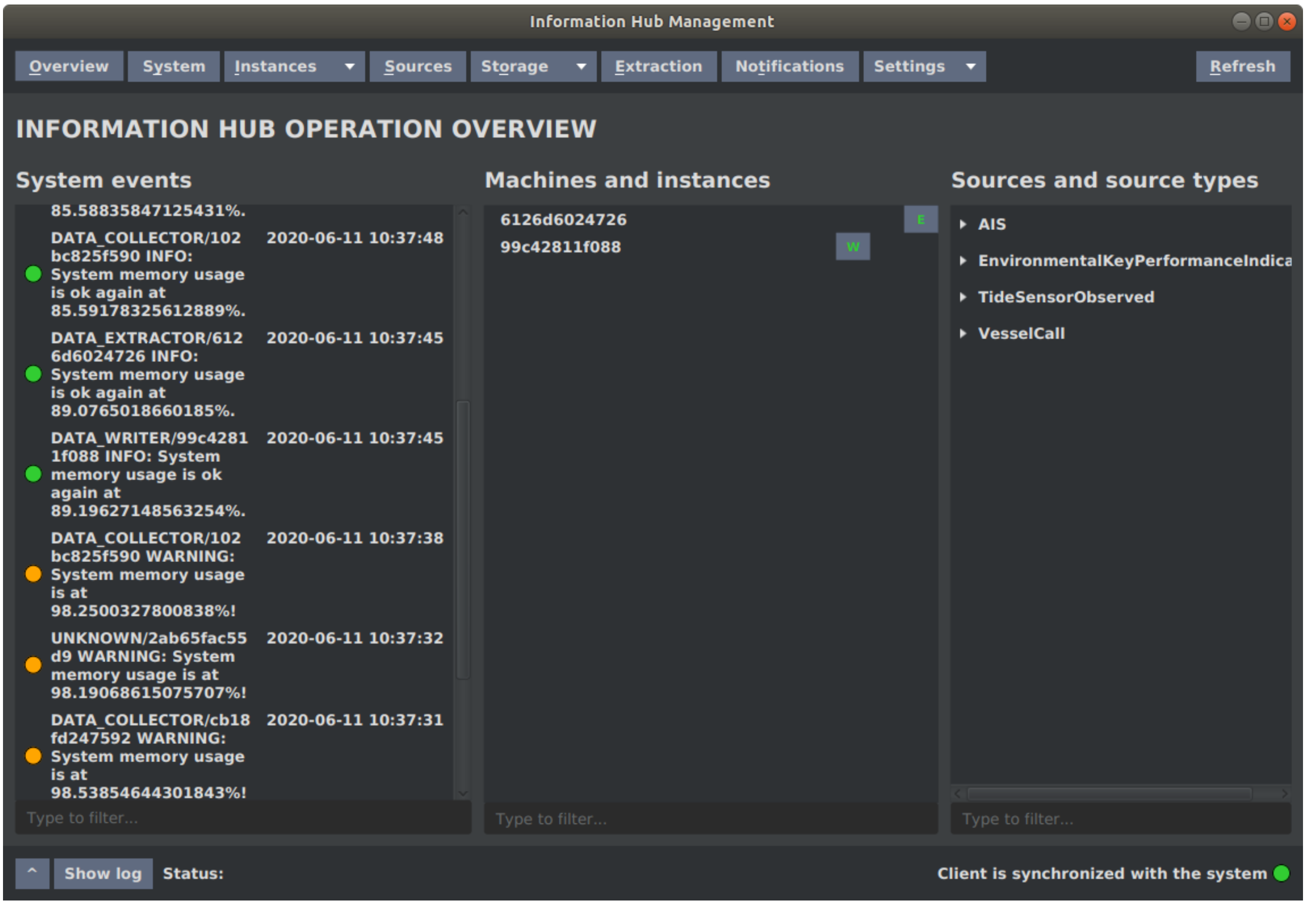
- **Overview**: also shown when the application starts, it serves as a general display for showing system notifications, machines, instances and their status.
- **System**: displays information about the machines in the Information Hub and their resources.
- **Instances**: displays tables listing instances of each type, together with their main status. By clicking on the drop-down arrow in the 'Instances' button, it is also possible to open more detailed panels specific to each instance type.
- **Sources**: displays information about the various data sources in the system and enables their configuration.

- **Storage**: using the dropdown arrow, it is possible to access panels detailing the Short-Term Storage, Long-Term Storage, Data Broker storage, reduction processes and algorithms. The default display (when clicking on the button) is the Short-Term Storage.
- **Extraction**: details about clients using the extractor instances, with the ability to block or limit their bandwidth.
- **Notifications**: displays lists of recent system and client notifications.
- **Settings**: access to GUI and connection settings, a panel for loading or storing the Information Hub configuration and a panel for toggling system maintenance mode, all accessible through the dropdown arrow. By default, the Connection settings panel opens with a click of the button.

The Status area located at the bottom of the application window contains an always visible display of the most recent error or warning message and a status marker, showing the severity (red for error, yellow for warning and green for normal) of the client application or the Information Hub. The 'Expand' button expands the status area to show several notification items instead of just one. The 'Show log' button opens the client notifications panel, showing the complete list of recent messages (available also through the 'Notifications' menu button). On the right-hand side of the Status area most recent client notification is displayed along with its status colour. If no relevant client notifications are available, the message displays "Client is synchronized with the system".

4.4.2.4.2. Overview View

The Overview menu button opens the **Overview** view as depicted in figure below, this panel is also the default view when management console is started:



INFORMATION HUB OPERATION OVERVIEW

System events

```
   85.58835847125431%.
●  DATA_COLLECTOR/102   2020-06-11 10:37:48
   bc825f590 INFO:
   System memory usage
   is ok again at
   85.59178325612889%.
●  DATA_EXTRACTOR/612   2020-06-11 10:37:45
   6d6024726 INFO:
   System memory usage
   is ok again at
   89.0765018660185%.
●  DATA_WRITER/99c4281  2020-06-11 10:37:45
   1f088 INFO: System
   memory usage is ok
   again at
   89.19627148563254%.
●  DATA_COLLECTOR/102   2020-06-11 10:37:38
   bc825f590 WARNING:
   System memory usage
   is at
   98.2500327800838%!
●  UNKNOWN/2ab65fac55   2020-06-11 10:37:32
   d9 WARNING: System
   memory usage is at
   98.19068615075707%!
●  DATA_COLLECTOR/cb18  2020-06-11 10:37:31
   fd247592 WARNING:
   System memory usage
   is at
   98.53854644301843%!
```

Type to filter...

Machines and instances

6126d6024726 E
99c42811f088 W

Type to filter...

Sources and source types

- AIS
- EnvironmentalKeyPerformanceIndica
- TideSensorObserved
- VesselCall

Type to filter...

^ Show log Status: Client is synchronized with the system ●

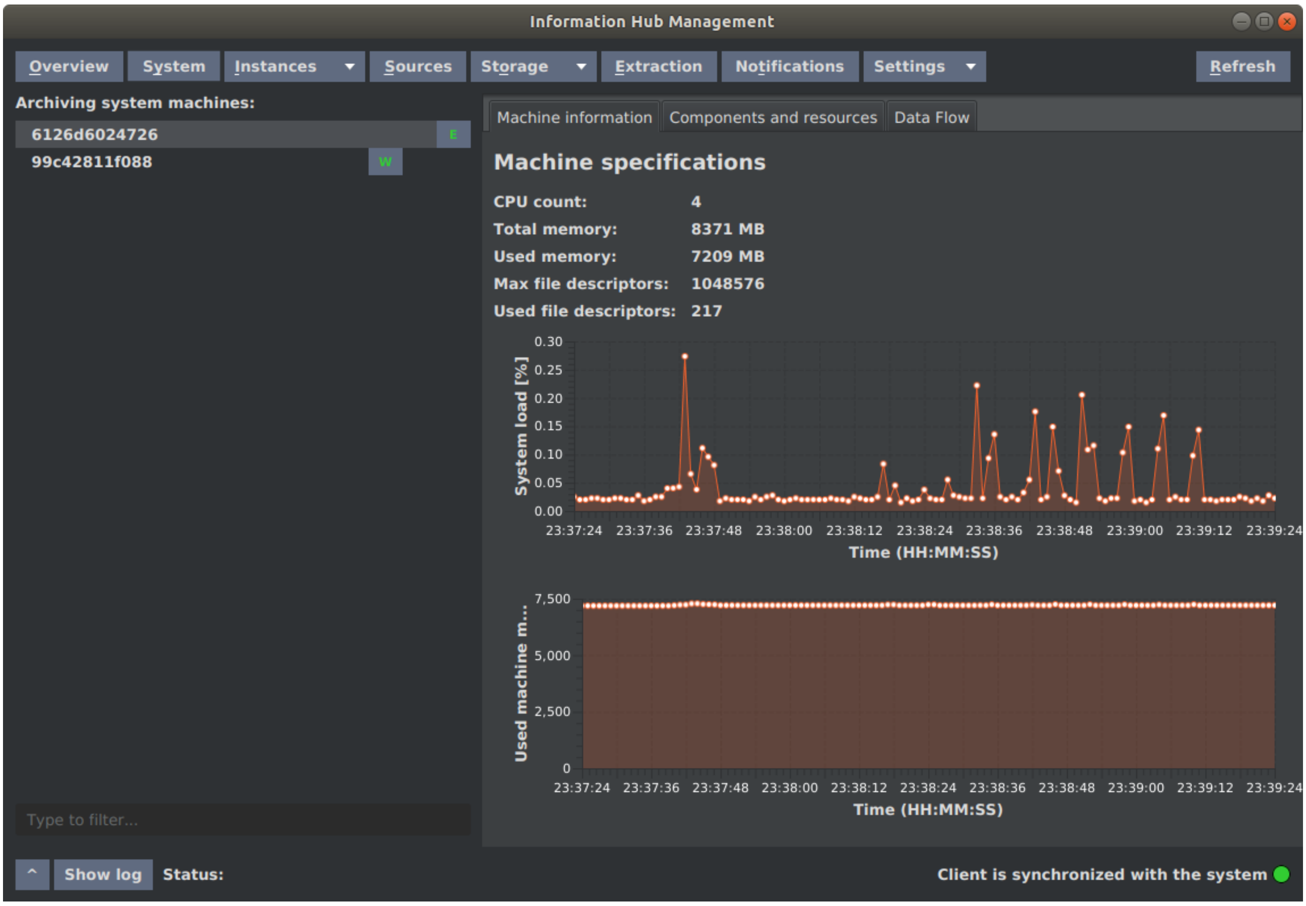
Figure 57: Information Hub Operation Overview

The panel contains following three sub-panels:

- **System events**: shows list of recent system notifications. The colour of the status light on the left indicates errors (red), warnings (yellow) or info (green) messages. The time on the right indicates when the message was generated in the system. By hovering over the messages, detailed information can be viewed. By clicking on them, the Notification panel will open.
- **Machines and instances**: show list of registered machines or Docker instances that are running services from the Data Worker Group.
- **Sources and source types** shows list of registered source types, a source type can be expanded to show all sources of that type.

4.4.2.4.3. System View

The 'System' menu button opens the **System** view as shown in the figure below:



Archiving system machines:

6126d6024726 E

99c42811f088 W

Machine specifications

CPU count: 4

Total memory: 8371 MB

Used memory: 7209 MB

Max file descriptors: 1048576

Used file descriptors: 217

Status:

Client is synchronized with the system ●

Figure 58: System View

The left-hand side sub panel displays a list of machines or Docker instances in Information Hub. Clicking on a specific machine displays its detailed information in the right-hand panel. For each machine, a button is displayed for each available service instance: collector (C), writer (W), reductor (R) or extractor (E). The colour of the letter indicates the instance status: green for normal operation, red for errors and other events that need intervention, yellow for services on standby, white for disabled and black for shut down instances. Clicking on the buttons opens a detailed configuration pane for the corresponding instance.

The right-hand side sub panel consists of following three tabbed panels:

- **Machine information**: displays machine system resources (number of CPUs, total and used RAM, the maximum and used file descriptors of the selected machine, as reported by the services running on the machine), graph of the machine CPU load (100% means full processing of all CPUs) in the last three minutes, graph of the machine memory usage in MB in the last three minutes, as reported by the Java processes of services running on the machine.

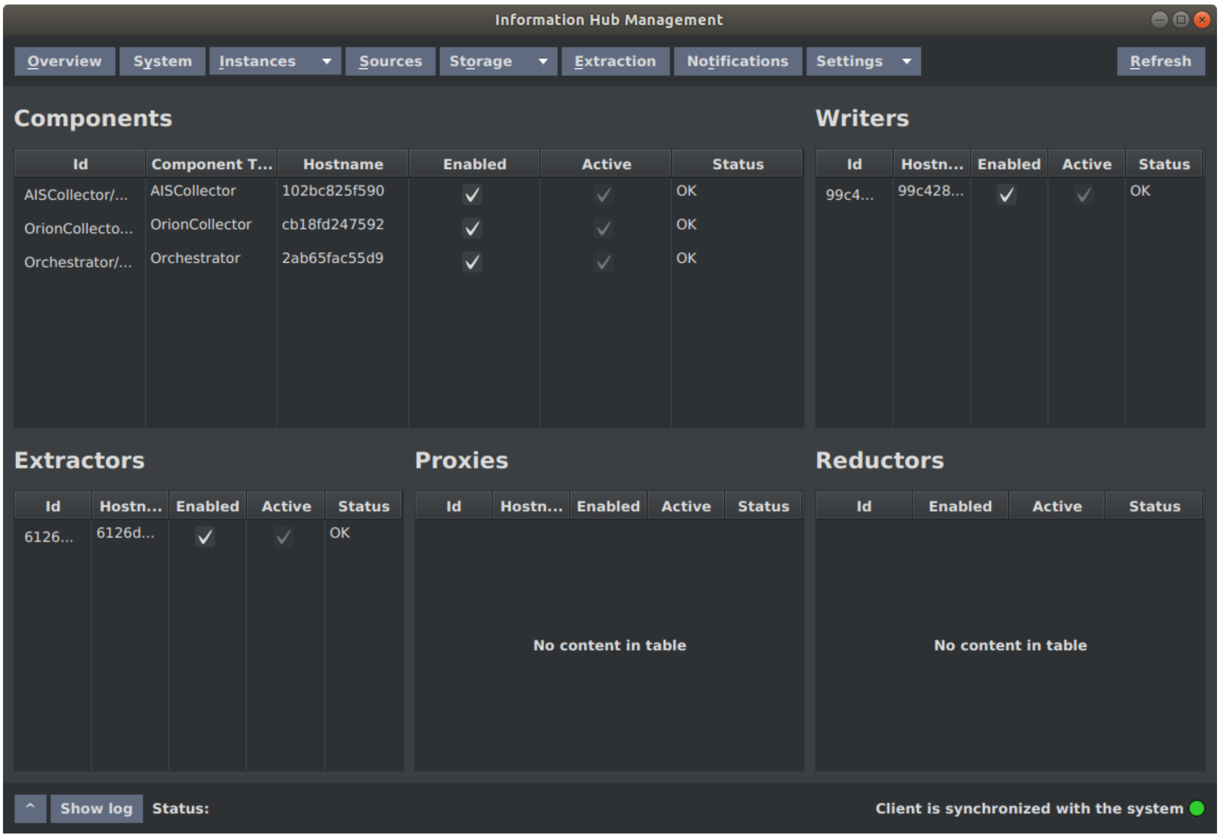
- **Components and resources**: display Process load and User memory graphs for the selected machine. The Process load graph is a combined graph of process loads of every service running on the machine (100% means full processing on all CPUs). The User memory graph is a combined graph of memory usage of every service running on the machine (collector, writer, reductor, extractor and proxy services). It displays the used process memory (in MB) as reported by the instance Java processes.
- **Data flow**: displays Data flow and Records flow graphs. The Data flow graph shows the cumulative source data being processed by the instance in KB/s. The Records flow graph shows the cumulative number of records being processed by the instance every second.

4.4.2.4.4. Instances View

The **Instances** view displays a list of worker instances categorized by the type of service (generic components, writers, extractors, proxies and reductors). Each table contains following columns:

- **Id**: the internal machine ID as stored in the system. Usually equal to the hostname, but it may be modified in settings.cfg files for running several instances of the same type on one machine.
- **Hostname**: the hostname of the machine as reported by the service Java process.
- **Enabled**: checkbox that allows quick toggling whether an individual instance is enabled or not (the Java process of a disabled instance is still running, but performs no processing and work is delegated to other instances of the same type).
- **Active**: a passive checkbox that shows if the Java process of this particular instance is active and the instance is connected to the Configuration Service.
-

Status: the current status of the instance. Possible values are: OK (service is enabled and active), DISABLED (service is disabled and not active), NOT_RUNNING (service is enabled but not active), STANDBY (service is active but not enabled), ERROR (service encountered an error), WARNING (service is running, but encountered an abnormal event).



Components

Id	Component T...	Hostname	Enabled	Active	Status
AISCollector/...	AISCollector	102bc825f590	✓	✓	OK
OrionCollecto...	OrionCollector	cb18fd247592	✓	✓	OK
Orchestrator/...	Orchestrator	2ab65fac55d9	✓	✓	OK

Writers

Id	Hostn...	Enabled	Active	Status
99c4...	99c428...	✓	✓	OK

Extractors

Id	Hostn...	Enabled	Active	Status
6126...	6126d...	✓	✓	OK

Proxies

Id	Hostn...	Enabled	Active	Status

No content in table

Reductors

Id	Enabled	Active	Status

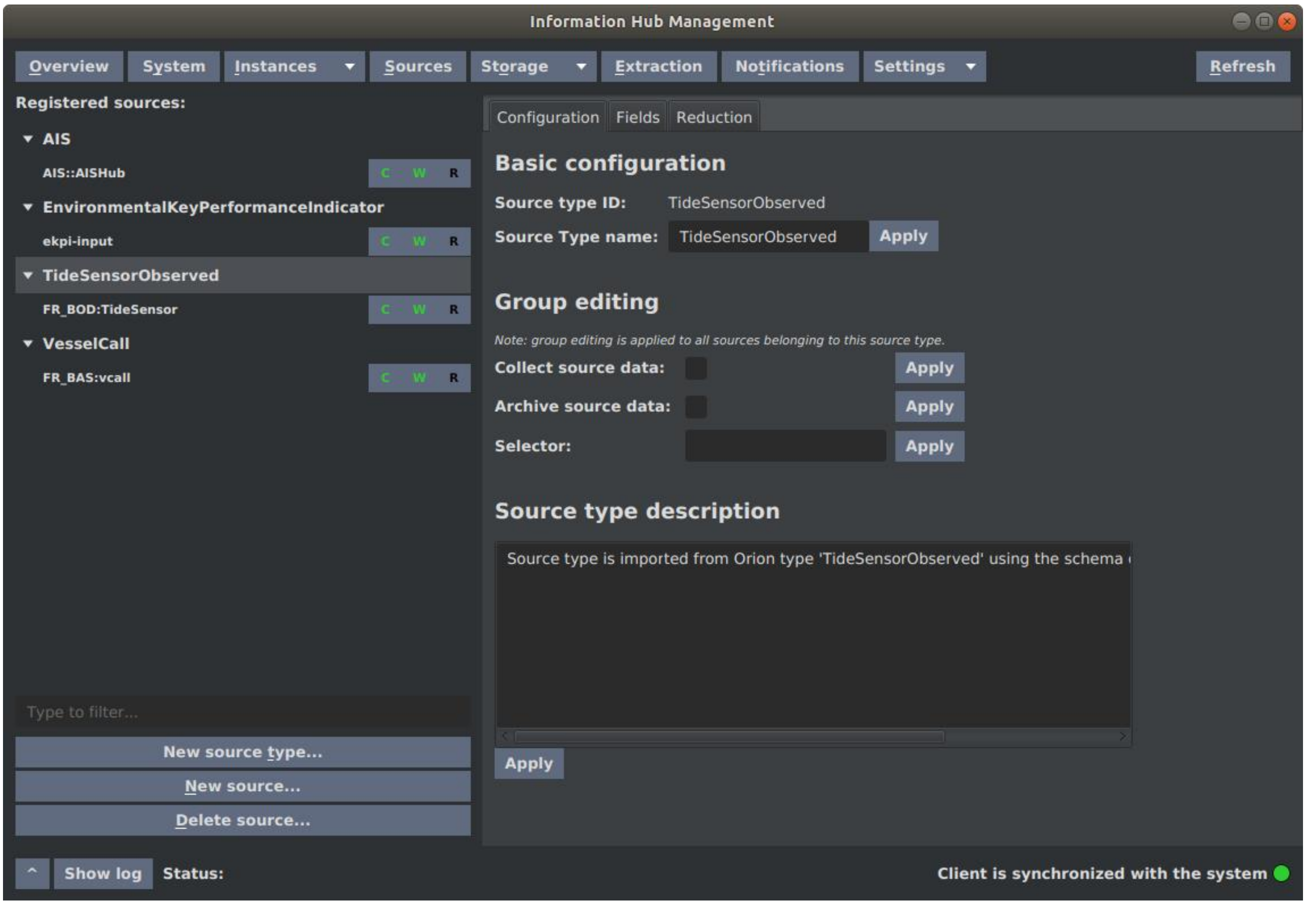
No content in table

Client is synchronized with the system ●

Figure 59: Instances View

4.4.2.4.5. Sources View

The **Sources** view displays a list of all registered sources and source types, details for selected source or source type and enables you to configure selected source or source type. The figure below shows the **Sources** view when a source type is selected:



Source type ID: TideSensorObserved

Source Type name: TideSensorObserved

Note: group editing is applied to all sources belonging to this source type.

Collect source data: □

Archive source data: □

Selector:

Source type is imported from Orion type 'TideSensorObserved' using the schema

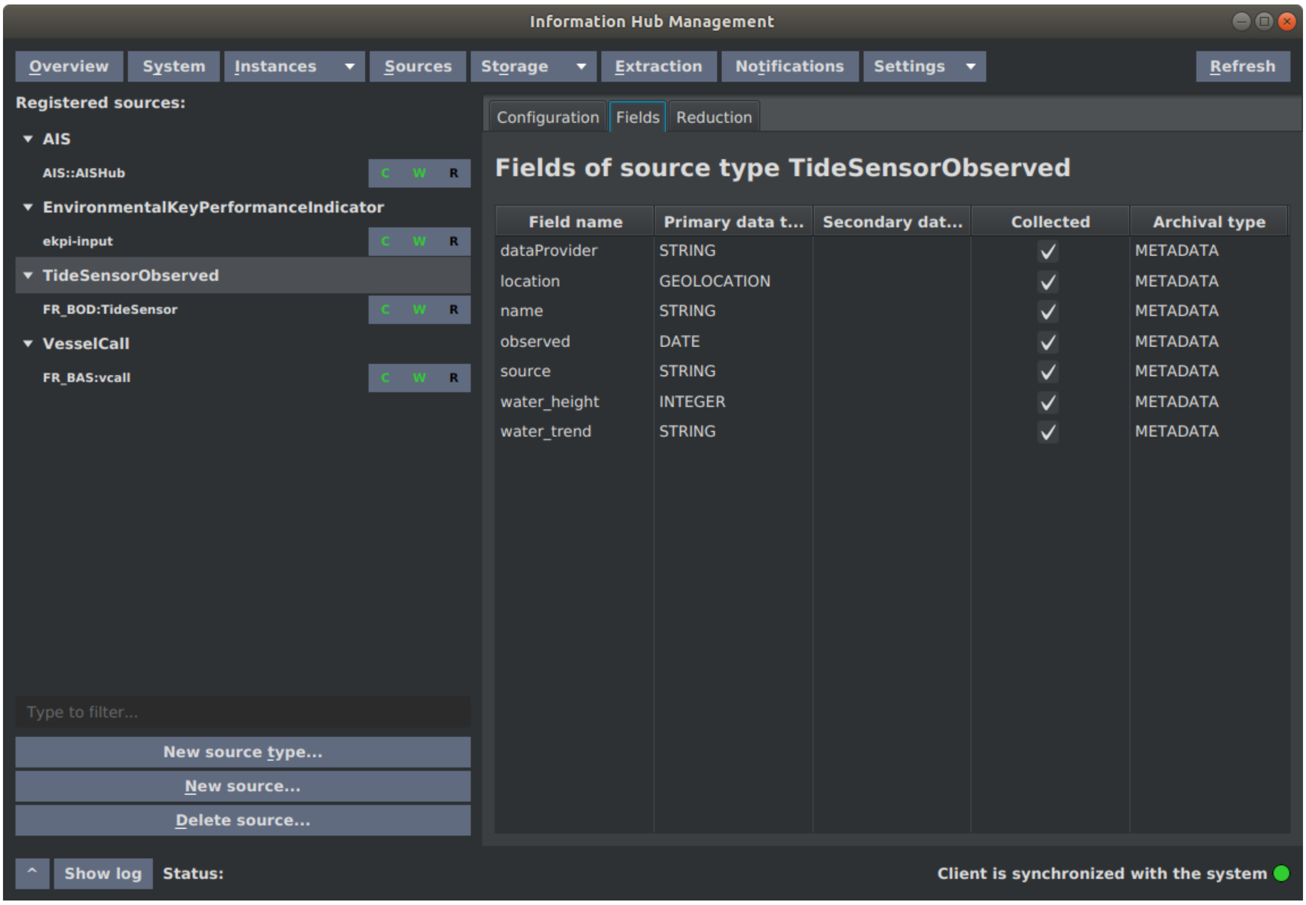
Figure 60: Sources View

The left-hand side panel shows a tree view of all registered source types. A source type can be expanded to show sources corresponding to that source type. Buttons on the right-hand side of source rows open the configuration of the corresponding instance. The colour of the letter indicates the status of the instance: green for normal operation, red for errors and other events that need intervention, yellow for services on standby, white for disabled and black for shut down instances.

The tree view can be filtered using the filter text box. Buttons for registering new source type and new source are not used in case of sources originating from Orion because registration is done automatically based on the notification from Orion. Delete button triggers deletion of selected items in the tree view. Source Types can be deleted with it as well, but its children need to be deleted first. Sources cannot be deleted if they are currently being collected.

The panel on the right-hand side displays details about the selected source type. Source type name can be modified using the provided text box. 'Collect source data' checkbox allows enabling or disabling of data collection for all child sources at once. The 'Apply' button needs to be clicked to commit the change. The 'Archive source data' button allows enabling or disabling of archival for all child sources at once. Archival means that source data is stored to Elasticsearch.

The figure below shows the management console when **Fields** tab is selected:



Fields of source type TideSensorObserved

Field name	Primary data t...	Secondary dat...	Collected	Archival type
dataProvider	STRING		✓	METADATA
location	GEOLOCATION		✓	METADATA
name	STRING		✓	METADATA
observed	DATE		✓	METADATA
source	STRING		✓	METADATA
water_height	INTEGER		✓	METADATA
water_trend	STRING		✓	METADATA

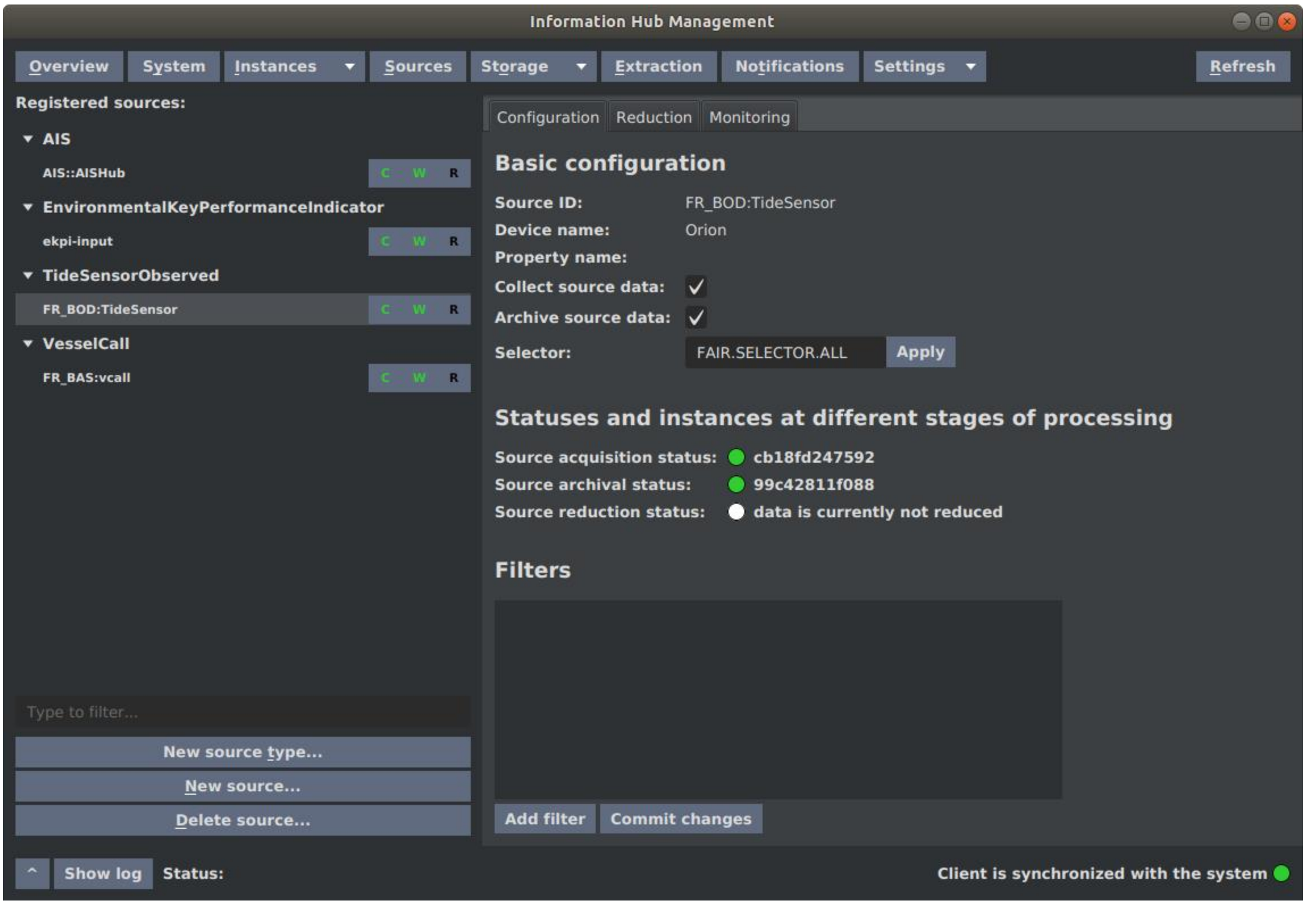
Client is synchronized with the system ●

Figure 61: Management Console when Fields tab is selected

The **Fields** tab shows all fields of the selected source type. The table has following columns:

- Field name.
- Primary data type.
- Secondary data type: if primary data type is ARRAY, this defines data type of the contained elements. If the primary data type is scalar, the secondary type is empty.
- Collected: the checkbox enables or disables the collection of the field. It will still be collected from the source, but will be excluded by the Data Collector from passing it on to subsequent stages of processing.
- Archival type: defines how the field will be treated by the Information Hub.

The figure below shows the **Sources** view when a source is selected in the left-hand sub panel:



Basic configuration

Source ID: FR_BOD:TideSensor
Device name: Orion
Property name:
Collect source data: ✓
Archive source data: ✓
Selector: FAIR.SELECTOR.ALL

Statuses and instances at different stages of processing

Source acquisition status: ● cb18fd247592
Source archival status: ● 99c42811f088
Source reduction status: ○ data is currently not reduced

Filters

Status:

Client is synchronized with the system ●

Figure 62: Management Console when Source is selected

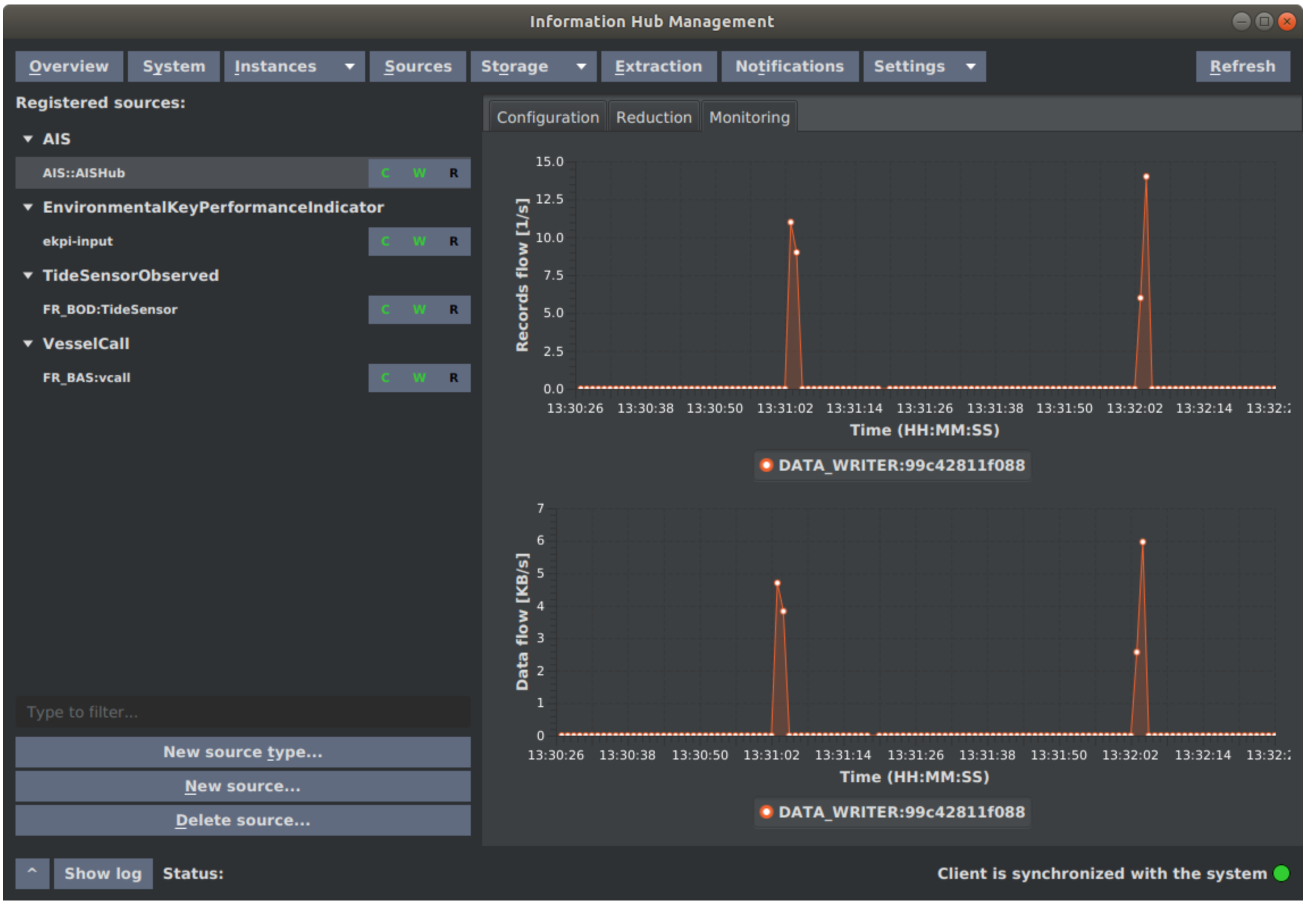
The **Configuration** tab consists of three sections. The **Basic configuration** section displays basic information about the source and following two checkboxes:

- **Collect source data**: if enabled, the source will be processed by a Data Collector instance (if available).
- **Archive source data**: if enabled, the source will be processed by a Data Writer instance (if available), i.e. stored to Elasticsearch.

The **Statuses and instances at different stages of processing** section shows if data from the Source is correctly processed in each of the stages: collection, archival and reduction. If processing is enabled and without errors, it shows the names of the corresponding instances that are processing the Source data.

The **Filters** section allows to set filters that will be used to configure the data source and filter the data received from it (does not apply to Orion data source).

The figure below shows management console for selected data source when **Monitoring** tab is selected:



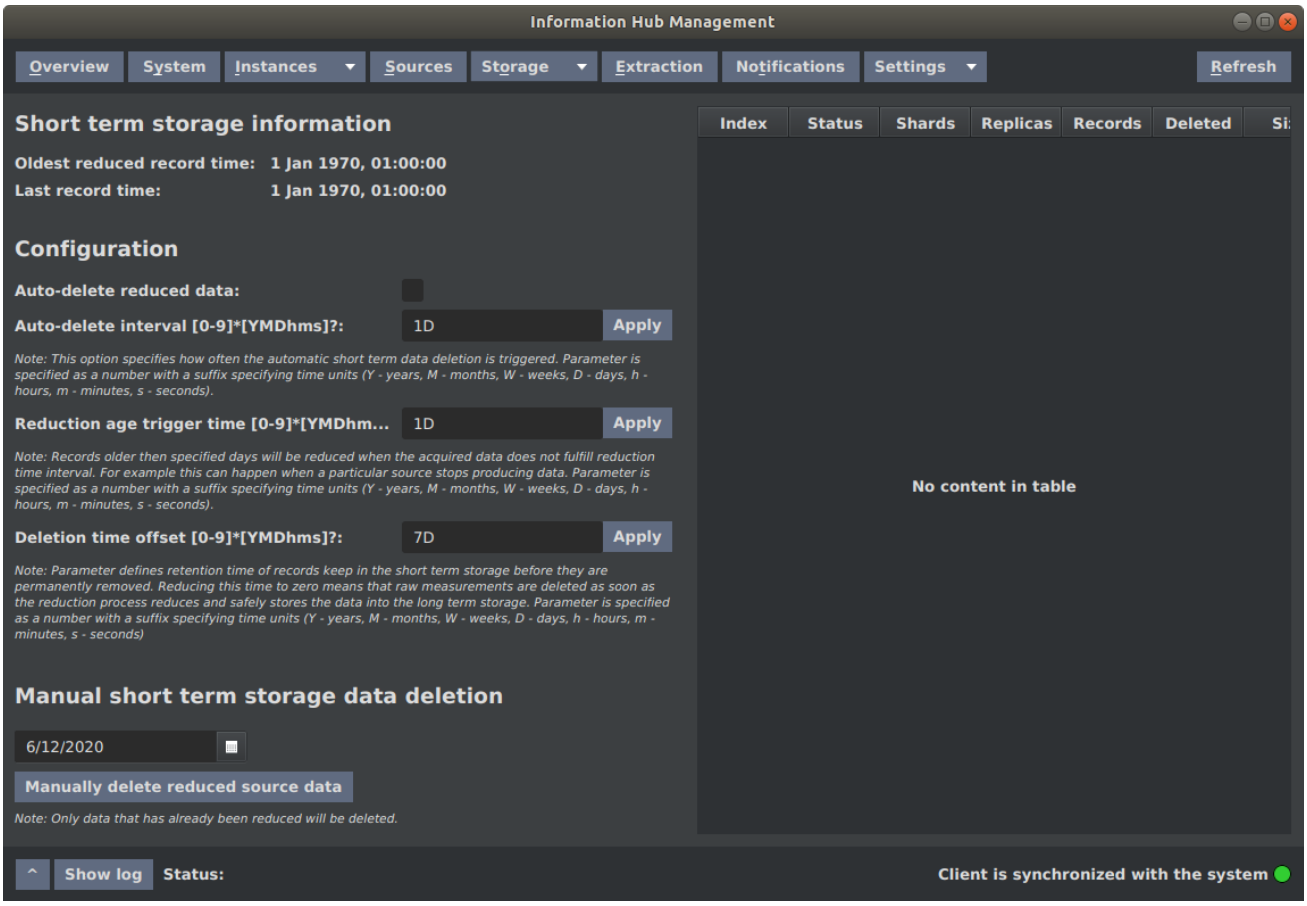
Client is synchronized with the system ●

Figure 63: Management Console when Monitoring is selected

The **Records flow** graph shows the number of records from the selected source being processed by Information Hub. The **Data flow** graph shows the bandwidth of data (in kB per second) from the selected source being processed by Information Hub.

4.4.2.4.6. Storage View

The **Storage** view enables you to configure short-term and long-term storages and reduction algorithms. Its capabilities exceed the needs of sources originating from the DAL, so we will not go into more details.



Short term storage information

Oldest reduced record time: 1 Jan 1970, 01:00:00

Last record time: 1 Jan 1970, 01:00:00

Configuration

Auto-delete reduced data: □

Auto-delete interval [0-9]*[YMDhms]?: 1D

Note: This option specifies how often the automatic short term data deletion is triggered. Parameter is specified as a number with a suffix specifying time units (Y - years, M - months, W - weeks, D - days, h - hours, m - minutes, s - seconds).

Reduction age trigger time [0-9]*[YMDhm... 1D

Note: Records older then specified days will be reduced when the acquired data does not fulfill reduction time interval. For example this can happen when a particular source stops producing data. Parameter is specified as a number with a suffix specifying time units (Y - years, M - months, W - weeks, D - days, h - hours, m - minutes, s - seconds).

Deletion time offset [0-9]*[YMDhms]?: 7D

Note: Parameter defines retention time of records keep in the short term storage before they are permanently removed. Reducing this time to zero means that raw measurements are deleted as soon as the reduction process reduces and safely stores the data into the long term storage. Parameter is specified as a number with a suffix specifying time units (Y - years, M - months, W - weeks, D - days, h - hours, m - minutes, s - seconds)

Manual short term storage data deletion

6/12/2020

Note: Only data that has already been reduced will be deleted.

Status:

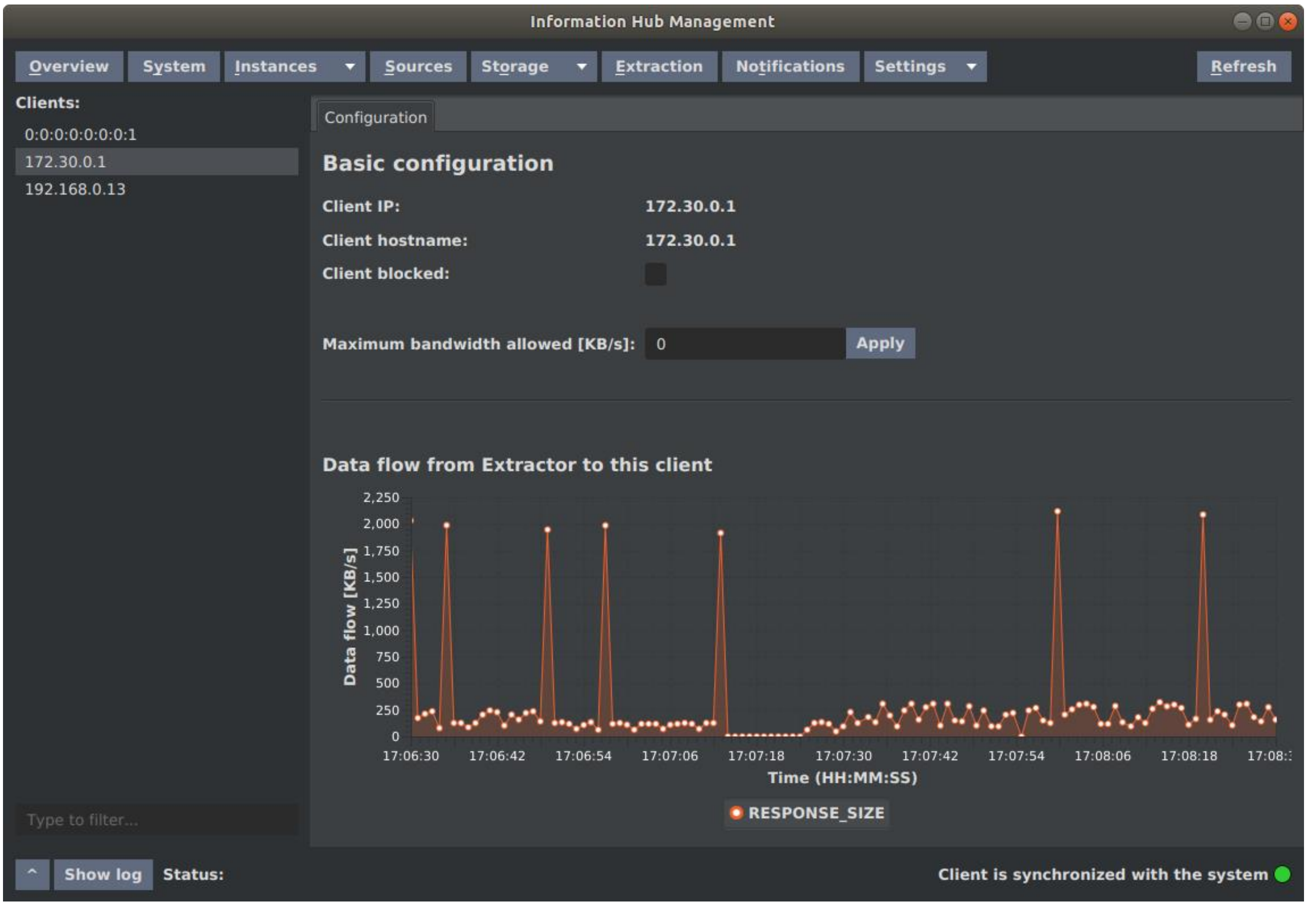
Client is synchronized with the system ●

Figure 64: Storage View

4.4.2.4.7. Extraction View

The **Extraction** view enables you to manage connected clients of Data Extractor instances. The panel on the left-hand side displays a list of clients that have been requesting data from Data Extractor instances. The right-hand side panel displays following information about the selected client:

- IP and hostname of the client.
- Client blocked: if checked, the client will be prevented from issuing requests to Data Extractor instances and will instead receive a 403-error message "You have no access".
- Maximum bandwidth allowed: if the client issues larger requests, their data flow will be limited to the specified bandwidth (in kB per second). The "Apply" button needs to be clicked to commit the setting.
- Client data flow: graph showing the recent data flow in kB per second from a Data Extractor instance to the selected client.



Clients:
0:0:0:0:0:0:0:1
172.30.0.1
192.168.0.13

Basic configuration

Client IP: 172.30.0.1

Client hostname: 172.30.0.1

Client blocked: □

Maximum bandwidth allowed [KB/s]: 0 Apply

Data flow from Extractor to this client

Status:

Client is synchronized with the system ●

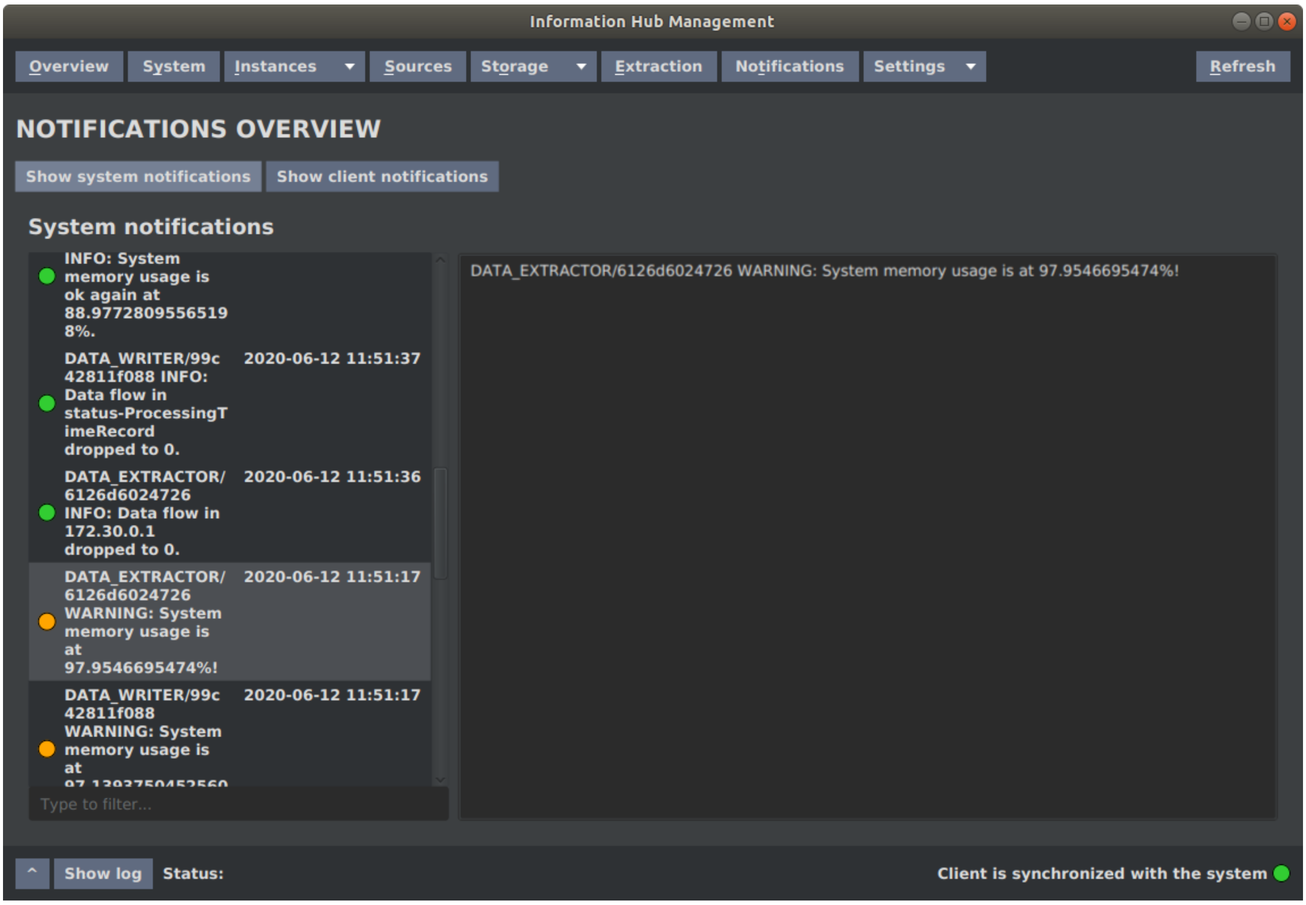
Figure 65: Extractions View

4.4.2.4.8. Notifications View

The **Notifications** view displays the system and client (administration console) notification messages. Using the buttons on the top you can switch between the lists of system and client notifications.

The left-hand panel displays a list of notifications which includes a short notification message, the message severity (red for errors, yellow for warnings and green for info messages) and the timestamp when the notification was generated. By clicking on any message, more detailed text (if available) is shown in the right-hand panel.

The filter box on the bottom of the left panel allows filtering of notifications by limiting the list to display only items that contain the entered text.



```
● INFO: System memory usage is ok again at 88.97728095565198%.
● DATA_WRITER/99c42811f088 INFO: Data flow in status-ProcessingTimeRecord dropped to 0.   2020-06-12 11:51:37
● DATA_EXTRACTOR/6126d6024726 INFO: Data flow in 172.30.0.1 dropped to 0.   2020-06-12 11:51:36
● DATA_EXTRACTOR/6126d6024726 WARNING: System memory usage is at 97.9546695474%!   2020-06-12 11:51:17
● DATA_WRITER/99c42811f088 WARNING: System memory usage is at 97.1393750452560   2020-06-12 11:51:17
```

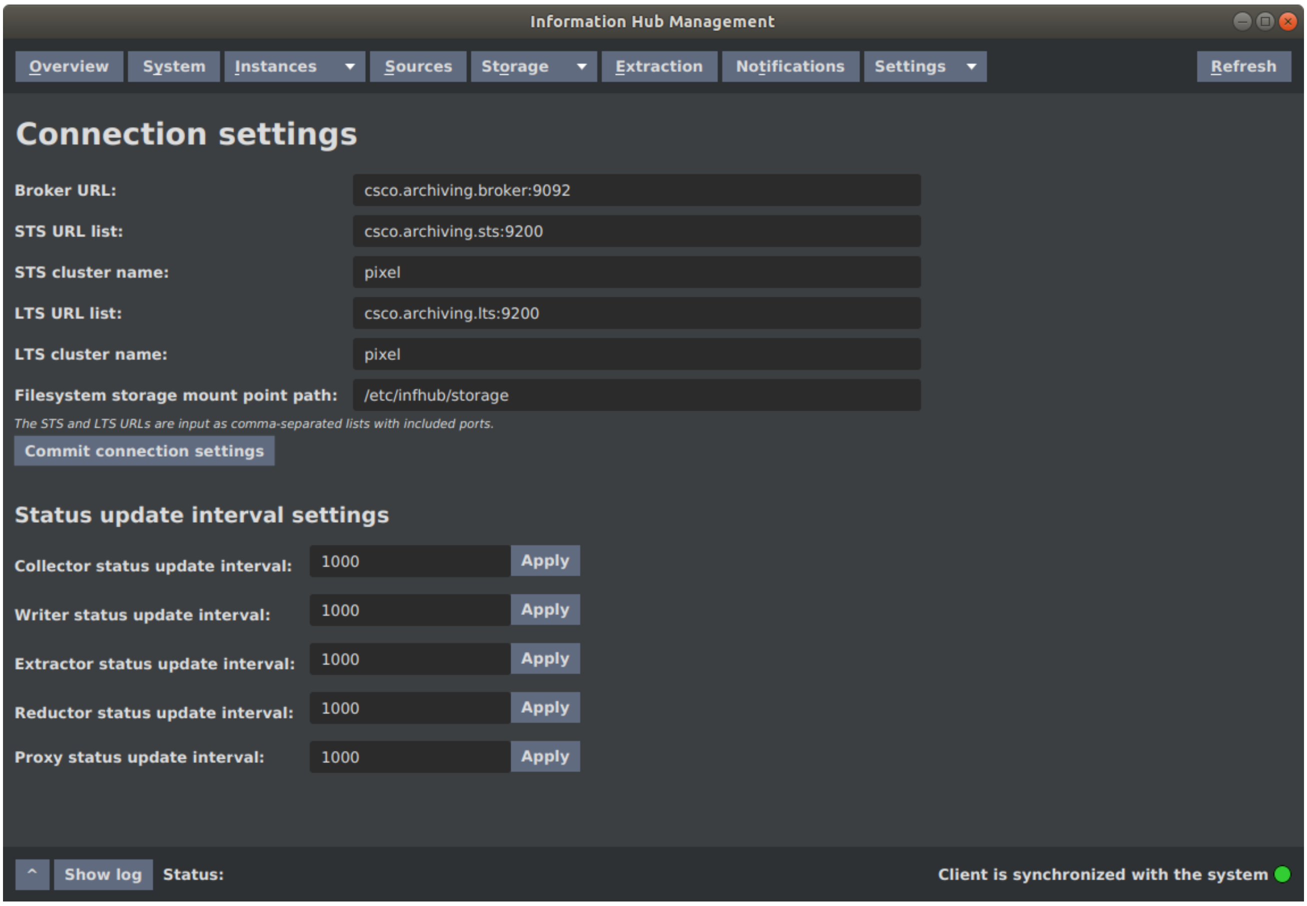
```
DATA_EXTRACTOR/6126d6024726 WARNING: System memory usage is at 97.9546695474%!
```

Figure 66: Notifications View

4.4.2.4.9. Settings View

The **Settings** view shows various configuration options for configuring Information Hub and management console. The **Connection Settings** panel depicted in figure below contains following elements:

- **broker URL**: address of the message broker (Apache Kafka) which is used for communication among Information Hub components. The address should be specified in the form *<IP or hostname>:<port>*.
- **STS URL list**: connection settings that are used by Data Writer, Data Extractor and Controller components to connect and exchange data with the short-term storage. The list contains addresses of the Short-Term Storage cluster nodes. The addresses should be specified in the form <*IP or hostname*>:<*port*> and the list items separated by a comma.
- **STS cluster name**: name of the short-term storage cluster, shared between all nodes.
- **LTS URL list**: connection settings that are used by Data Writer, Data Extractor and Controller components to connect and exchange data with the long-term storage. The list contains addresses of the long-term storage cluster nodes. The addresses should be specified in the form <*IP or hostname*>:<*port*> and the list items separated by a comma.
- **LTS cluster name**: name of the long-term storage cluster, shared between all nodes.
- **Filesystem storage mount point path**: when storing Source field data in binary form, this is the path in the filesystem that should be used for storing the binary data blocks. The binary storage system should therefore be mounted to this path on all Data Collector, Data Extractor and Data Proxy host machines.
- **Commit connection settings**: apply all changes from the input fields above.
- **Status update interval settings**: set how often services of each type send status messages. Consequently, affects the update frequency of all monitoring graphs in the GUI.



Connection settings

Broker URL: csco.archiving.broker:9092

STS URL list: csco.archiving.sts:9200

STS cluster name: pixel

LTS URL list: csco.archiving.lts:9200

LTS cluster name: pixel

Filesystem storage mount point path: /etc/infhub/storage

The STS and LTS URLs are input as comma-separated lists with included ports.

Commit connection settings

Status update interval settings

Collector status update interval: 1000 Apply

Writer status update interval: 1000 Apply

Extractor status update interval: 1000 Apply

Reductor status update interval: 1000 Apply

Proxy status update interval: 1000 Apply

Client is synchronized with the system ●

Figure 67: Settings View

4.4.3. PIXEL Operational Tools

4.4.3.1. Backend Interface

The Operational Tools are able to publish models, predictive algorithms and schedule them. Furthermore, there is also support for KPIs and events. The API has been specified as a REST API that includes a Swagger (Open API) interface to be tested. You can also use other developer tools such as Postman. The Swagger UI is very user friendly and allows to easily check all possible requests, its input parameters and the outputs. We will just provide a basic example for a dummy model in order to highlight the process, which should be considered as a scheme for all other requests (analogous process).

Open a web browser and go to **http://your-server-ip:8080/otpixel/doc** you should be able to see the Swagger UI of the application. Click first on **Authorize**, and enter your **apiKey**.

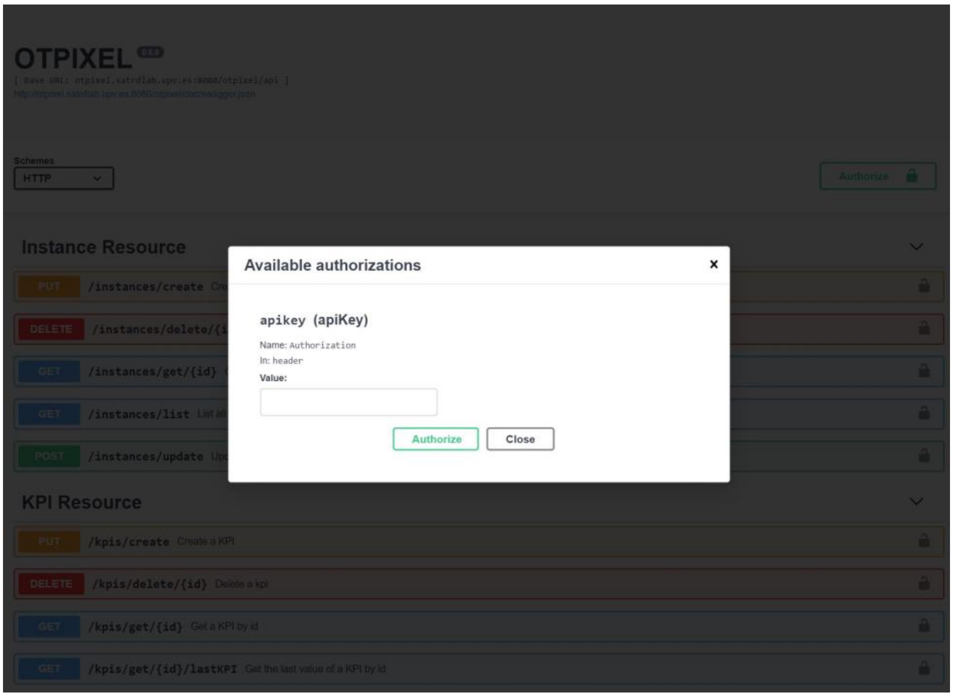
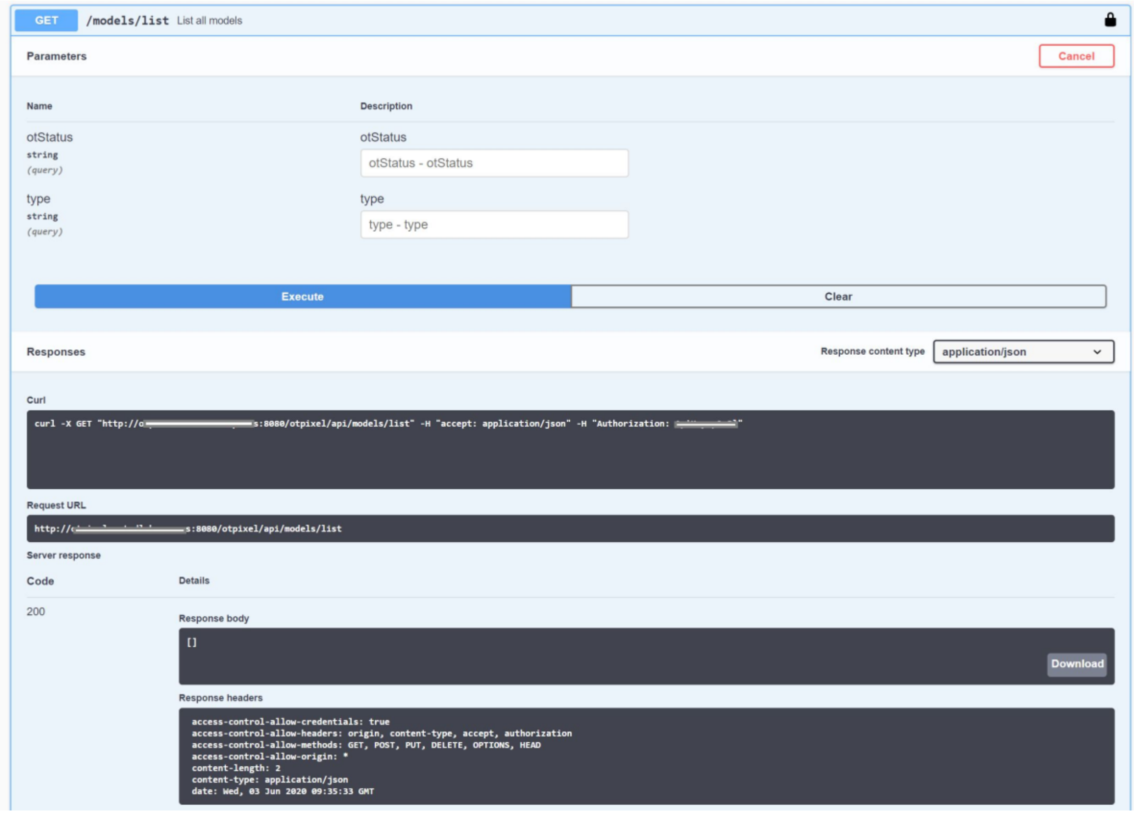

OTPIXEL
[Base URL: otpixel.satrdlab.upv.es:8080/otpixel/api]
http://otpixel.satrdlab.upv.es:8080/otpixel/doc/swagger.json
Schemes
HTTP
Authorize
Instance Resource
PUT /instances/create
DELETE /instances/delete/{i
GET /instances/get/{id}
GET /instances/list List all
POST /instances/update
KPI Resource
PUT /kpis/create Create a KPI
DELETE /kpis/delete/{id} Delete a kpi
GET /kpis/get/{id} Get a KPI by id
GET /kpis/get/{id}/lastKPI Get the last value of a KPI by id
Available authorizations
×
apikey (apiKey)
Name: Authorization
In: header
Value:
Authorize
Close

Figure 68: OT Swagger UI authentication

At the very beginning after installing the OT component, there is no data in Mongo (database), therefore any request will return an empty response. We will take as an example the 'models' resource. Click on **/models/list** and the options will expand.



```
curl -X GET "http://o[redacted]s:8080/otpixel/api/models/list" -H "accept: application/json" -H "Authorization: [redacted]"
```

```
http://[redacted]s:8080/otpixel/api/models/list
```

```
[]
```

```
access-control-allow-credentials: true
access-control-allow-headers: origin, content-type, accept, authorization
access-control-allow-methods: GET, POST, PUT, DELETE, OPTIONS, HEAD
access-control-allow-origin: *
content-length: 2
content-type: application/json
date: Wed, 03 Jun 2020 09:35:33 GMT
```

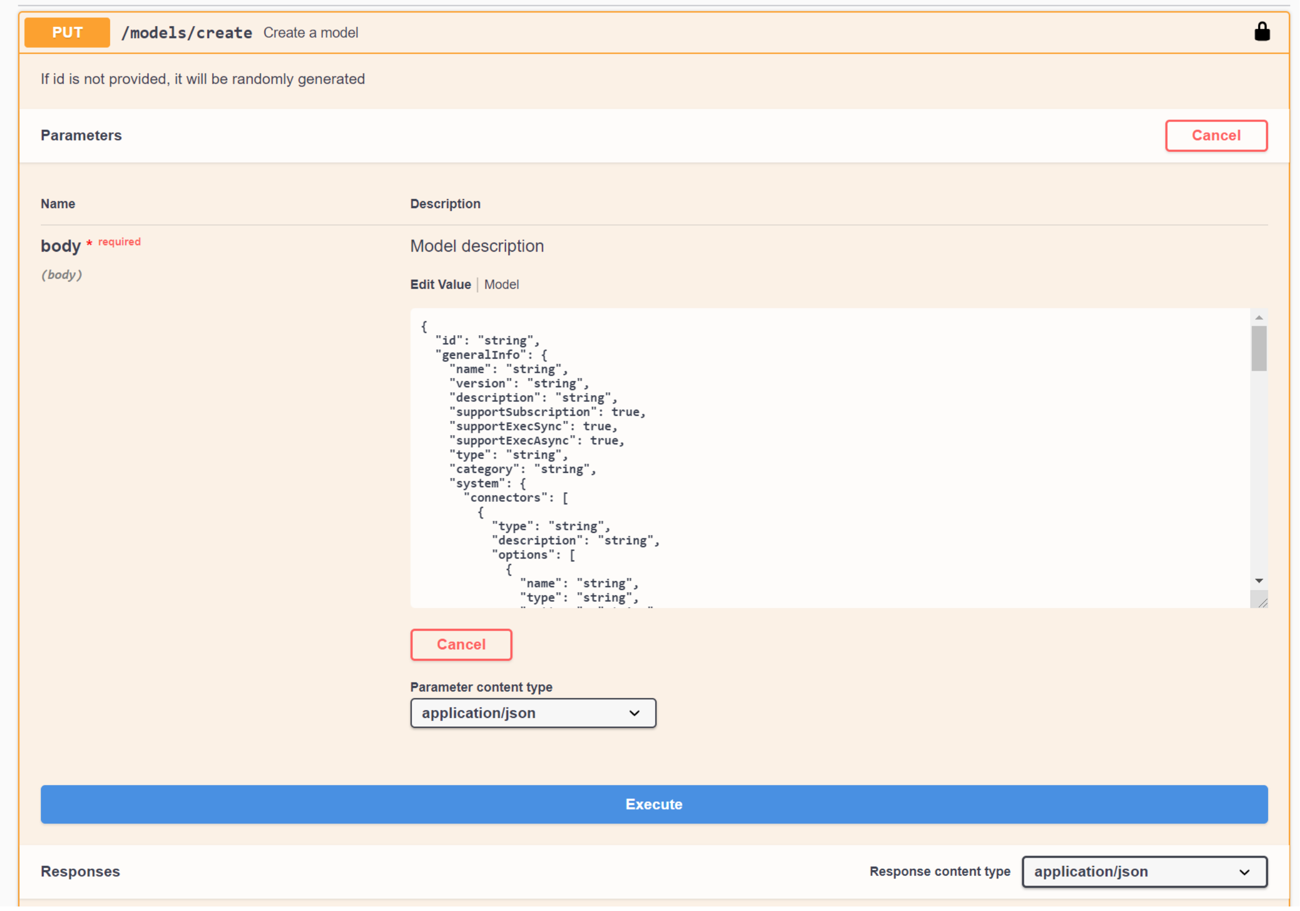
Figure 69: OT Swagger UI (list models). Empty response

Note here some optional parameters to be included in the request:

- **otStatus**: status of the models to be retrieved, which can be one of: created, deployed, error, deleted. If not given, all are provided.
- **type**: type to be considered: **model, pa**. If not given, all are provided.

Note also that you have an example of a CURL request. Finally, note that the response is an empty array as there are (yet) no models there.

To create a new one. Click on **models/create** and the options will expand.



If id is not provided, it will be randomly generated

body * required
(body)

Model description

```
{
  "id": "string",
  "generalInfo": {
    "name": "string",
    "version": "string",
    "description": "string",
    "supportSubscription": true,
    "supportExecSync": true,
    "supportExecAsync": true,
    "type": "string",
    "category": "string",
    "system": {
      "connectors": [
        {
          "type": "string",
          "description": "string",
          "options": [
            {
              "name": "string",
              "type": "string",
```

Parameter content type: application/json

Response content type: application/json

Figure 70: OT Swagger UI (create model)

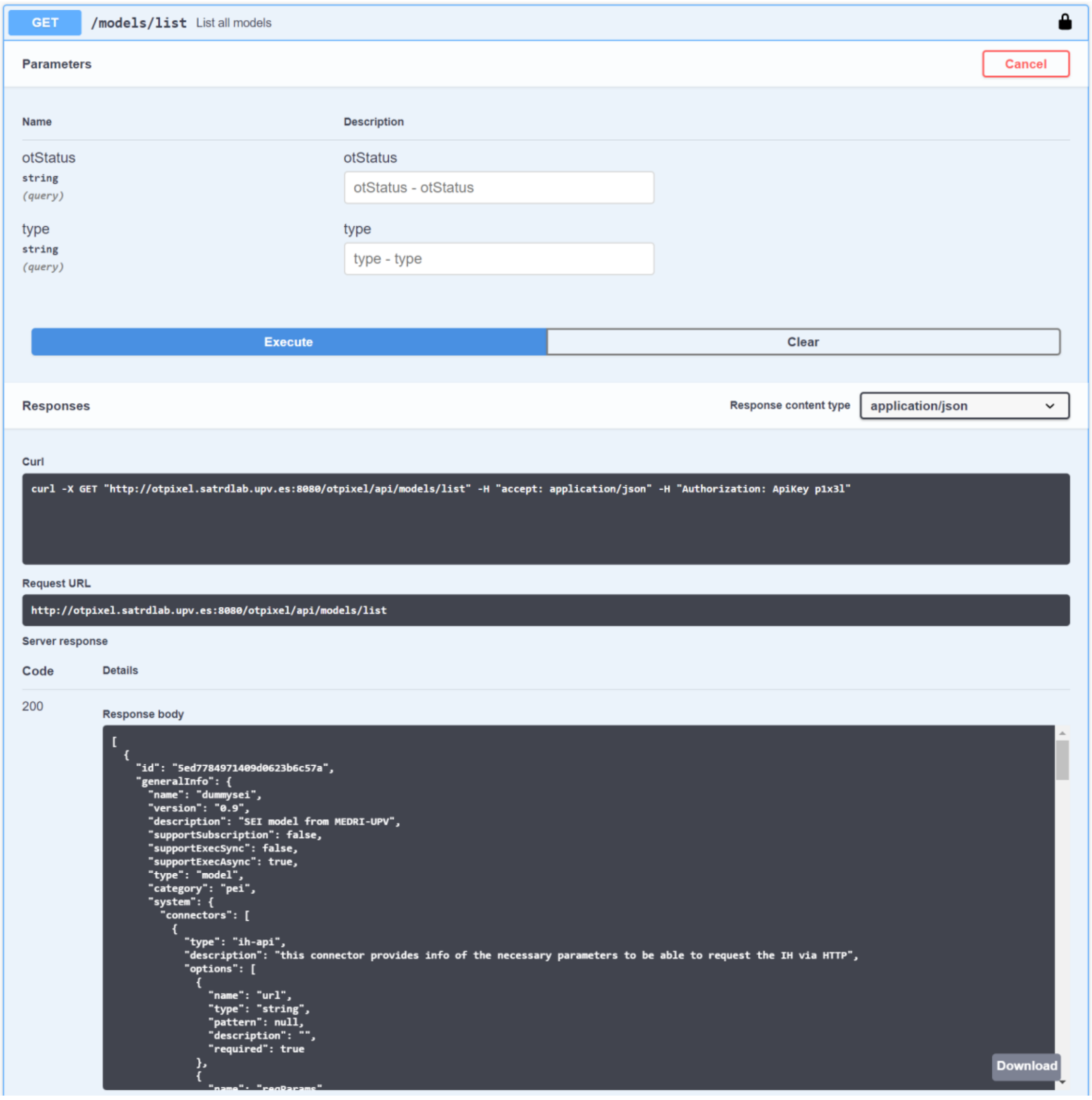
You can see a really complex body, but do not worry because there is no need to understand all info. You can just insert as body the following JSON (we will use a dummy model):

```
{
  "dockerInfo": {
    "dockerName": "pixelh2020/dummysei:0.1",
    "label": "getInfo"
  }
}
```

After pressing the **Execute** button, you should see the following response:

```
{
  "id": "5ed7784971409d0623b6c57a",
  "generalInfo": null,
  "dockerInfo": {
    "dockerName": "pixelh2020/dummysei:0.1",
    "label": "getInfo",
    "dockerRepo": null
  },
  "creation": 1591179337033,
  "otStatus": "created"
}
```

Right now, the model has been created in the Operational Tools. A backend process will retrieve the Docker image from **Dockerhub** and extract all description information. We can see this if we **list** the models again:



otStatus
string
(query)
otStatus
otStatus - otStatus

type
string
(query)
type
type - type

Response content type: application/json

Curl

```
curl -X GET "http://otpixel.satrdlab.upv.es:8080/otpixel/api/models/list" -H "accept: application/json" -H "Authorization: ApiKey p1x3l"
```

Request URL

```
http://otpixel.satrdlab.upv.es:8080/otpixel/api/models/list
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": "5ed7784971409d0623b6c57a",
    "generalInfo": {
      "name": "dummysei",
      "version": "0.9",
      "description": "SEI model from MEDRI-UPV",
      "supportSubscription": false,
      "supportExecSync": false,
      "supportExecAsync": true,
      "type": "model",
      "category": "pei",
      "system": {
        "connectors": [
          {
            "type": "ih-api",
            "description": "this connector provides info of the necessary parameters to be able to request the IH via HTTP",
            "options": [
              {
                "name": "url",
                "type": "string",
                "pattern": null,
                "description": "",
                "required": true
              },
              {
                "name": "reqParams"
```

Figure 71: OT Swagger UI (list models). Model example response

You can see now all information related to this model that has been imported through Dockerhub.

Other additional **CRUD** operations related to models are straightforward: **deleting a model**, **updating a model**, **getting a model** (by UUID).

The process with other resources (**instance**, **scheduledInstance** and **KPI**) is also straightforward in terms of CRUD operations. The KPI includes two additional functions:

- **/kpis/get/{id}/lastKPI**: Gets the last value of a KPI by id. It is supposed that the KPI is a time series changing throughout time. This data is stored in the Information Hub (Elasticsearch).
- **/kpis/get/{id}/stats**: Gets statistical info from a KPI between a given time interval (optional), such as: min, max, average and std. It also includes an array of KPI values (this is useful for the dashboard to print them on a graph).

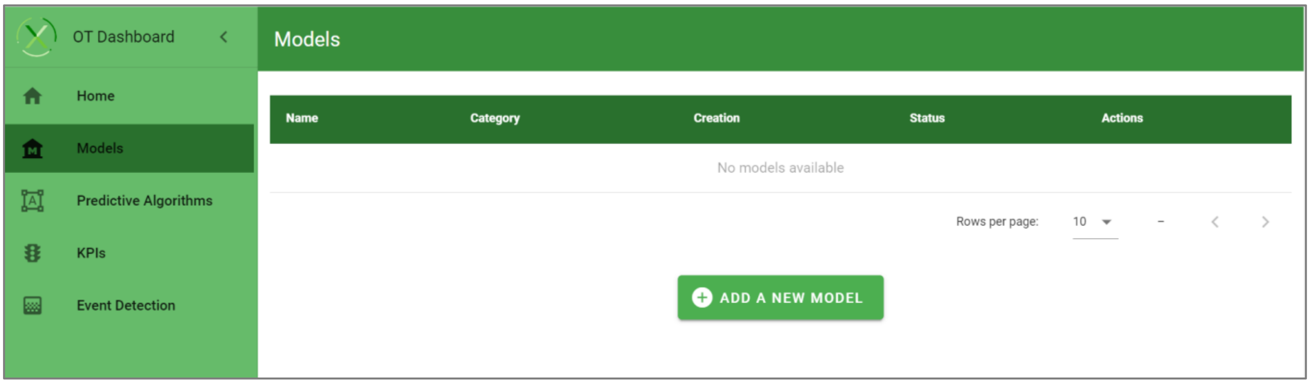
4.4.3.2. Graphical User Interface

The Operational Tools include a small basic UI that supports most of the functionalities of its API. It may serve as basis for your own development in case you intend to make your own project only considering this component of the PIXEL architecture, though the PIXEL Dashboard is intended to provide much more options and functionality.

4.4.3.2.1. Models

Creating a Model

If you want to create a new model, just click on the main (left) panel on **Models**. You should see a list of already published models, unless it is a fresh installation.

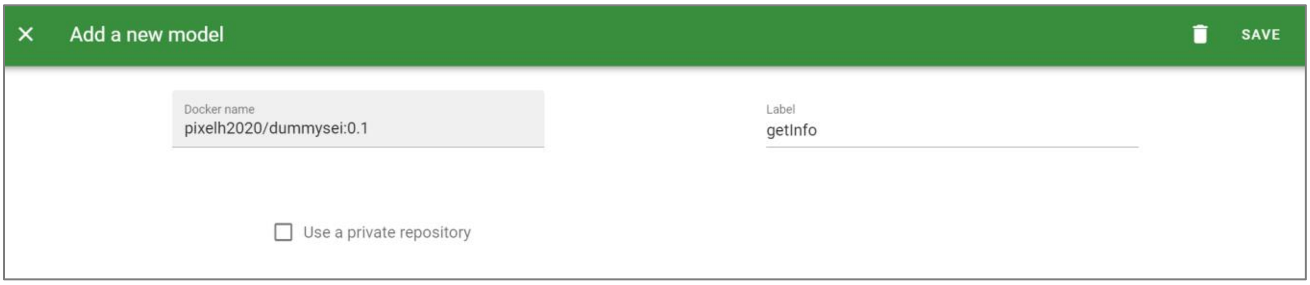


Name	Category	Creation	Status	Actions

No models available

Figure 72: Add a Model (Step 1)

Just click on **Add a New Model**. A basic form will appear asking for the name of the model in your Docker repository as well as the label where all descriptive information is included (also in the Docker image). If you do not have one by your side available, you can follow the process with a dummy example. Just enter the following values:



Docker name: pixelh2020/dummysei:0.1

Label: getInfo

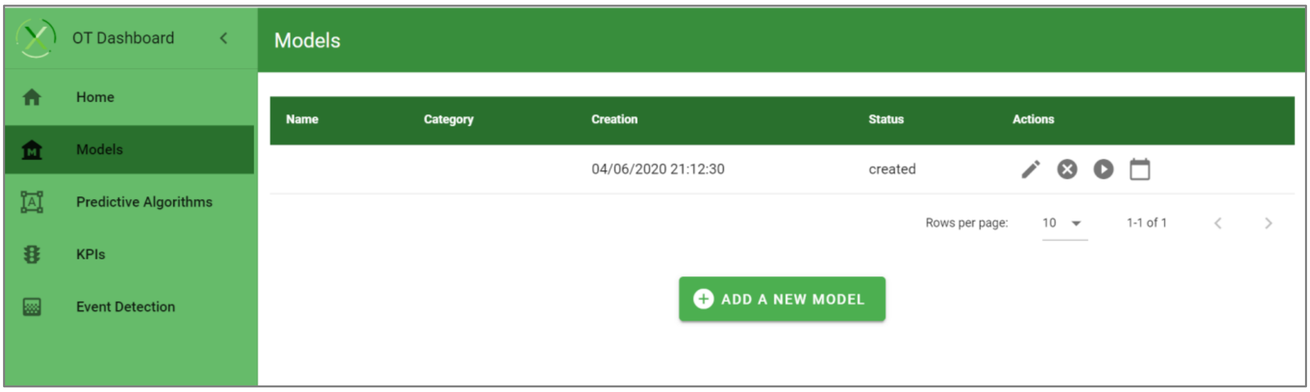
□ Use a private repository

Figure 73: Add a new model (Step 2)

As you may deduce, **pixelh2020** is an open (public) repository in Dockerhub, **dummysei:01** is the name and version of the Docker image to be used, and **getInfo** is the included label in the Docker image that described the model with a specific format defined in PIXEL. In a certain way, it is similar to a WSDL for web services.

The web form also includes the option to point to a private Docker repository, in that case, you will have to enter the credentials to access.

After clicking the **Save** button on the top right corner, you will see the model on the list as **created:**

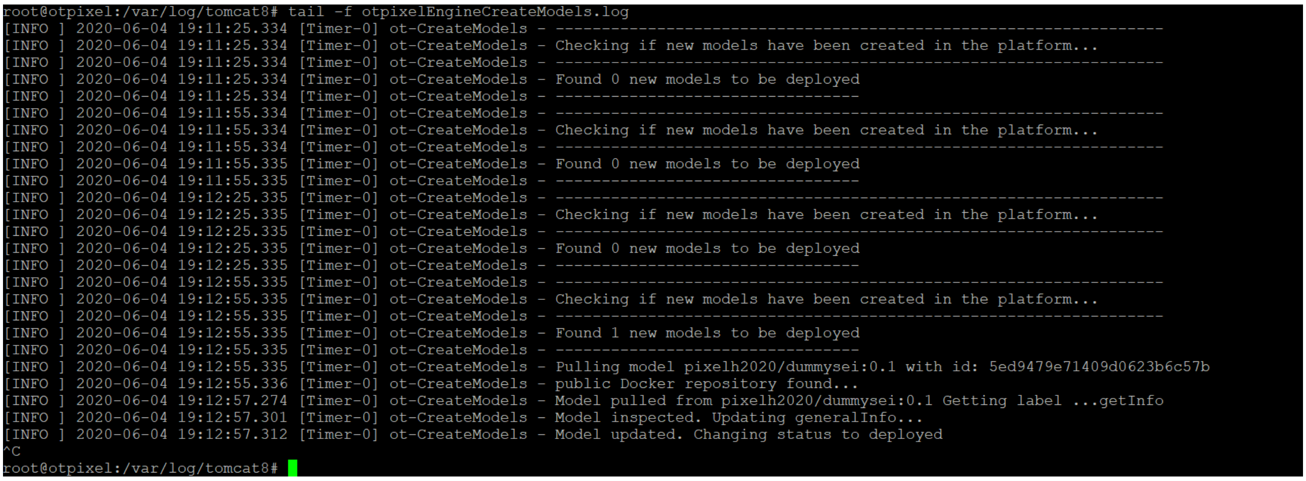


Name	Category	Creation	Status	Actions
		04/06/2020 21:12:30	created	

Rows per page: 10 1-1 of 1

Figure 74: OT GUI. Add a new model (Step 3)

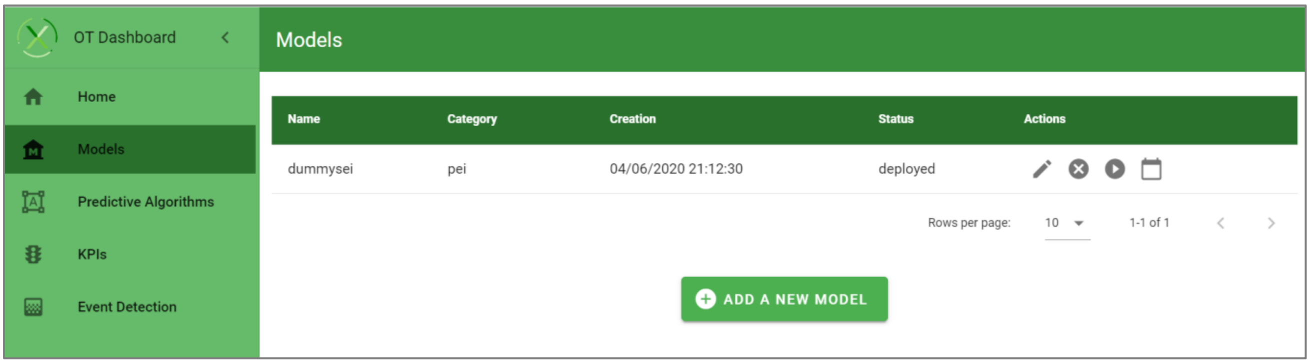
Note that there is still no name nor category for the model, as it needs to be first obtained (pulled) from the Docker repository. You can track this activity by monitoring the **/var/log/tomcat8/otpixelEngineCreateModel.log** file:



```
root@otpixel:/var/log/tomcat8# tail -f otpixelEngineCreateModels.log
[INFO ] 2020-06-04 19:11:25.334 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:11:25.334 [Timer-0] ot-CreateModels - Checking if new models have been created in the platform...
[INFO ] 2020-06-04 19:11:25.334 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:11:25.334 [Timer-0] ot-CreateModels - Found 0 new models to be deployed
[INFO ] 2020-06-04 19:11:25.334 [Timer-0] ot-CreateModels - ---------------------------------
[INFO ] 2020-06-04 19:11:55.334 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:11:55.334 [Timer-0] ot-CreateModels - Checking if new models have been created in the platform...
[INFO ] 2020-06-04 19:11:55.334 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:11:55.335 [Timer-0] ot-CreateModels - Found 0 new models to be deployed
[INFO ] 2020-06-04 19:11:55.335 [Timer-0] ot-CreateModels - ---------------------------------
[INFO ] 2020-06-04 19:12:25.335 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:12:25.335 [Timer-0] ot-CreateModels - Checking if new models have been created in the platform...
[INFO ] 2020-06-04 19:12:25.335 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:12:25.335 [Timer-0] ot-CreateModels - Found 0 new models to be deployed
[INFO ] 2020-06-04 19:12:25.335 [Timer-0] ot-CreateModels - ---------------------------------
[INFO ] 2020-06-04 19:12:55.335 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:12:55.335 [Timer-0] ot-CreateModels - Checking if new models have been created in the platform...
[INFO ] 2020-06-04 19:12:55.335 [Timer-0] ot-CreateModels - ------------------------------------------------------------------
[INFO ] 2020-06-04 19:12:55.335 [Timer-0] ot-CreateModels - Found 1 new models to be deployed
[INFO ] 2020-06-04 19:12:55.335 [Timer-0] ot-CreateModels - ---------------------------------
[INFO ] 2020-06-04 19:12:55.335 [Timer-0] ot-CreateModels - Pulling model pixelh2020/dummysei:0.1 with id: 5ed9479e71409d0623b6c57b
[INFO ] 2020-06-04 19:12:55.336 [Timer-0] ot-CreateModels - public Docker repository found...
[INFO ] 2020-06-04 19:12:57.274 [Timer-0] ot-CreateModels - Model pulled from pixelh2020/dummysei:0.1 Getting label ...getInfo
[INFO ] 2020-06-04 19:12:57.301 [Timer-0] ot-CreateModels - Model inspected. Updating generalInfo...
[INFO ] 2020-06-04 19:12:57.312 [Timer-0] ot-CreateModels - Model updated. Changing status to deployed
^C
root@otpixel:/var/log/tomcat8#
```

Figure 75: Log4j file for monitoring the creation of models

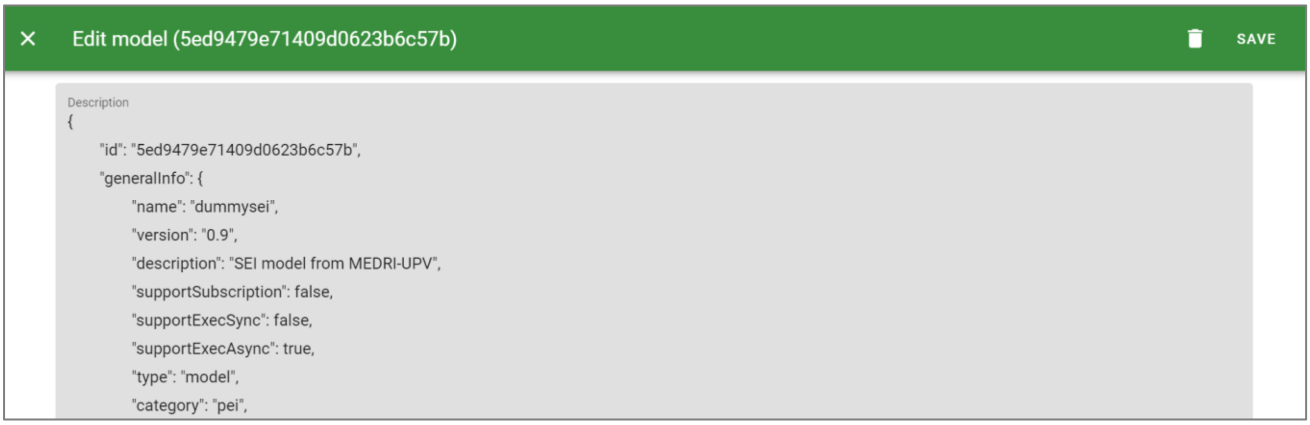
Now, if you refresh your browser, you should see that model has changed its status to **deployed**. Now there is a name and a category, which has been extracted from the given label of the Docker image.



Name	Category	Creation	Status	Actions
dummysei	pei	04/06/2020 21:12:30	deployed	

Figure 76: OT GUI. Add a new model (Step 4)

You should have noticed a list of actions represented by 4 icons: **edit**, **delete**, **run** and **schedule**. Clicking on the **Edit Model** icon will allow you to see the complete description of the model. We will not discuss the format, but basically it describes basic fields, connectors, inputs, outputs and logging configuration.



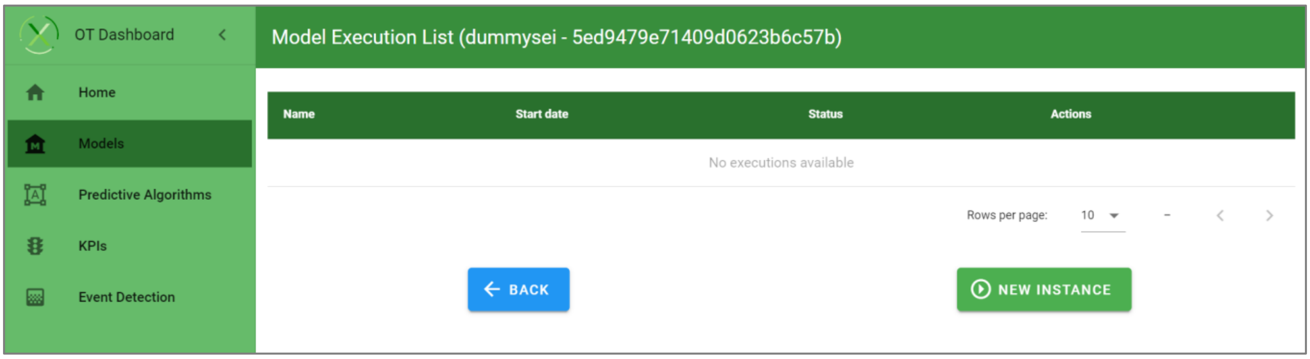
```
{
    "id": "5ed9479e71409d0623b6c57b",
    "generalInfo": {
        "name": "dummysei",
        "version": "0.9",
        "description": "SEI model from MEDRI-UPV",
        "supportSubscription": false,
        "supportExecSync": false,
        "supportExecAsync": true,
        "type": "model",
        "category": "pei",
```

Figure 77: OT GUI. Edit information from a published model

By clicking on the **Delete Model** icon, the model enters a **deleted** status. After a short while, if you refresh the browser the model will have disappeared. The other options (*run*, *schedule*) are commented on the next subsections

Running a Model

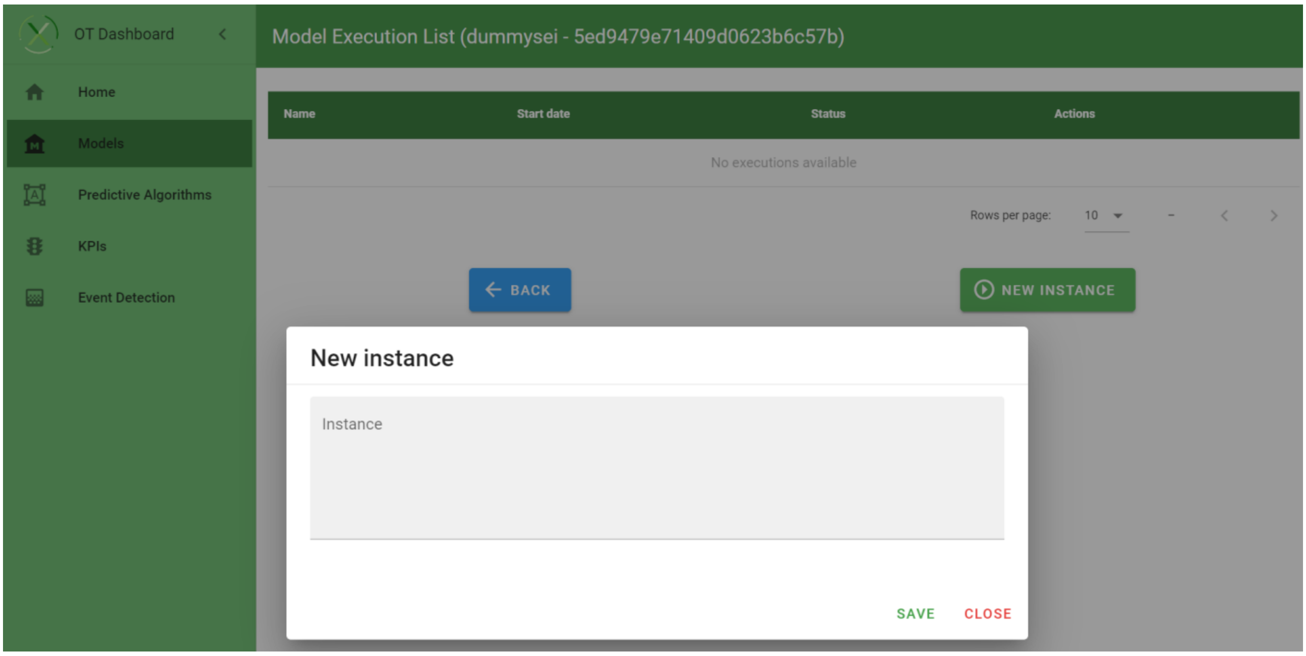
Once you have published and deployed a model (see previous step), you should be able to run the model. For that, just click on the **Run model** action button, and you should see a new page with a list of executions associated to that model. After a fresh installation, there will be no item in the list.



Name	Start date	Status	Actions
No executions available			

Figure 78: OT GUI. Create a new instance to run a model (Step I)

We can create a new execution by clicking on the **New Instance** button. A modal dialog appears where you will have to enter a JSON file describing the details of the execution.



Model Execution List (dummysei - 5ed9479e71409d0623b6c57b)

Name	Start date	Status	Actions
No executions available			

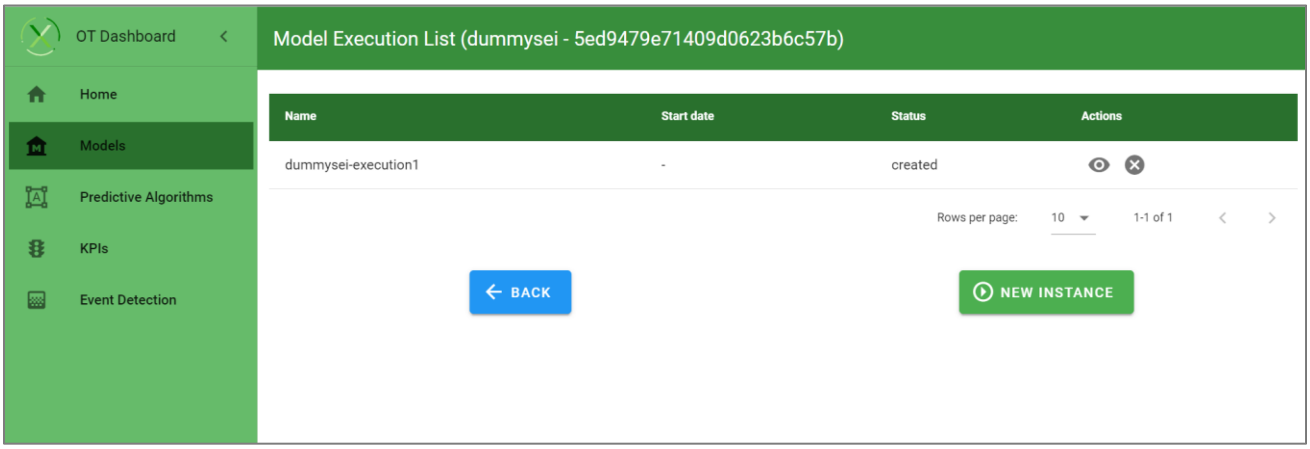
Rows per page: 10 -

New instance

Instance

Figure 79: OT GUI. Create a new instance to run a model (Step 2)

The introduction of data here is a particularization of the description of the model, with specific inputs and outputs, and varies from model to model. You should look at the specific model to enter valid data here. Once you do, just press the **Save** button in the modal. The new instance appears in the Menu with *status* **created**.



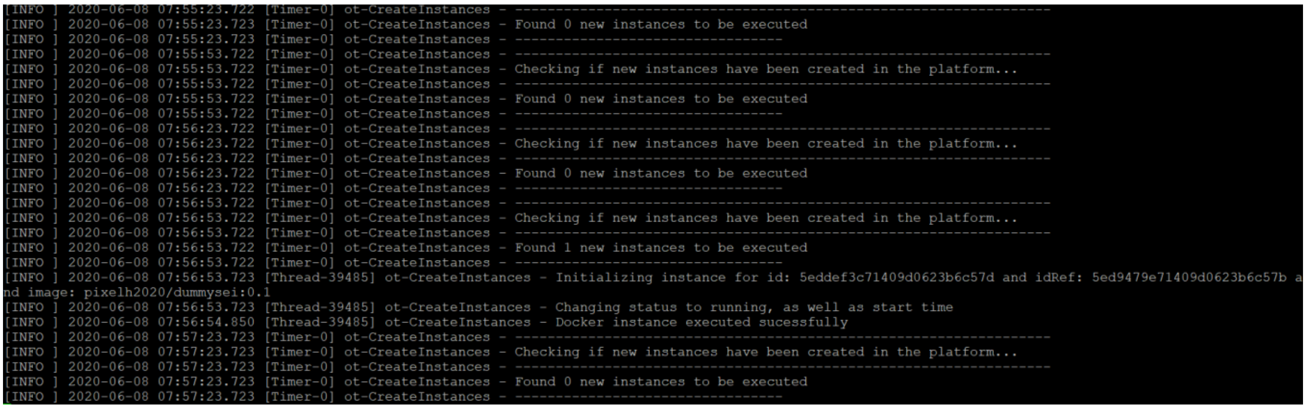
Model Execution List (dummysei - 5ed9479e71409d0623b6c57b)

Name	Start date	Status	Actions
dummysei-execution1	-	created	👁 ⊗

Rows per page: 10 1-1 of 1

Figure 80: OT GUI. Create a new instance to run a model (Step 3)

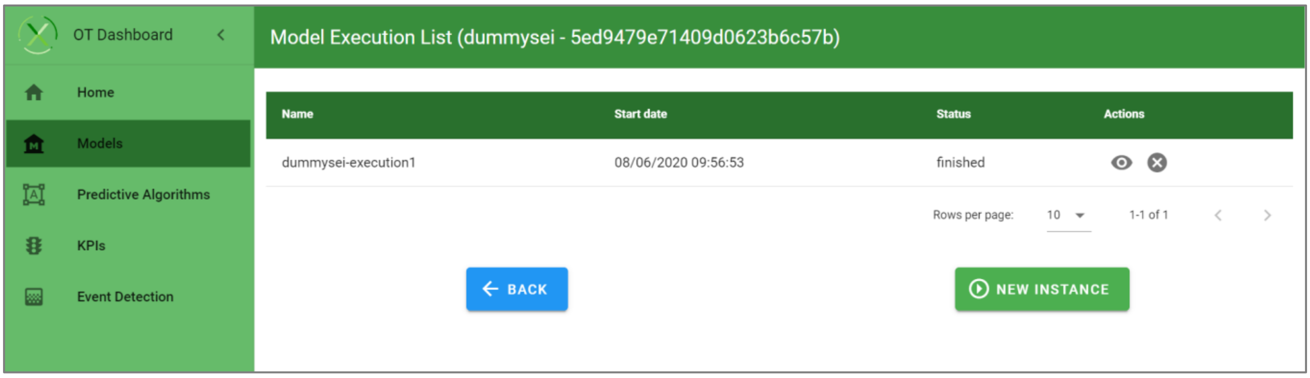
There is a backend process that periodically reads this table and runs the pending instances. You can track this activity by monitoring the **/var/log/tomcat8/otpixelEngineCreateInstances.log**:



```
[INFO ] 2020-06-08 07:55:23.722 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:55:23.723 [Timer-0] ot-CreateInstances - Found 0 new instances to be executed
[INFO ] 2020-06-08 07:55:23.723 [Timer-0] ot-CreateInstances - ---------------------------------
[INFO ] 2020-06-08 07:55:53.722 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:55:53.722 [Timer-0] ot-CreateInstances - Checking if new instances have been created in the platform...
[INFO ] 2020-06-08 07:55:53.722 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:55:53.722 [Timer-0] ot-CreateInstances - Found 0 new instances to be executed
[INFO ] 2020-06-08 07:55:53.722 [Timer-0] ot-CreateInstances - ---------------------------------
[INFO ] 2020-06-08 07:56:23.722 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:56:23.722 [Timer-0] ot-CreateInstances - Checking if new instances have been created in the platform...
[INFO ] 2020-06-08 07:56:23.722 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:56:23.722 [Timer-0] ot-CreateInstances - Found 0 new instances to be executed
[INFO ] 2020-06-08 07:56:23.722 [Timer-0] ot-CreateInstances - ---------------------------------
[INFO ] 2020-06-08 07:56:53.722 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:56:53.722 [Timer-0] ot-CreateInstances - Checking if new instances have been created in the platform...
[INFO ] 2020-06-08 07:56:53.722 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:56:53.722 [Timer-0] ot-CreateInstances - Found 1 new instances to be executed
[INFO ] 2020-06-08 07:56:53.722 [Timer-0] ot-CreateInstances - ---------------------------------
[INFO ] 2020-06-08 07:56:53.723 [Thread-39485] ot-CreateInstances - Initializing instance for id: 5eddef3c71409d0623b6c57d and idRef: 5ed9479e71409d0623b6c57b a
nd image: pixelh2020/dummysei:0.1
[INFO ] 2020-06-08 07:56:53.723 [Thread-39485] ot-CreateInstances - Changing status to running, as well as start time
[INFO ] 2020-06-08 07:56:54.850 [Thread-39485] ot-CreateInstances - Docker instance executed sucessfully
[INFO ] 2020-06-08 07:57:23.723 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:57:23.723 [Timer-0] ot-CreateInstances - Checking if new instances have been created in the platform...
[INFO ] 2020-06-08 07:57:23.723 [Timer-0] ot-CreateInstances - ------------------------------------------------------------------
[INFO ] 2020-06-08 07:57:23.723 [Timer-0] ot-CreateInstances - Found 0 new instances to be executed
[INFO ] 2020-06-08 07:57:23.723 [Timer-0] ot-CreateInstances - ---------------------------------
```

Figure 81: Log4j file for monitoring the creation of instances

After the execution, if you refresh your browser, you will see the details of the execution (instance) in the list.



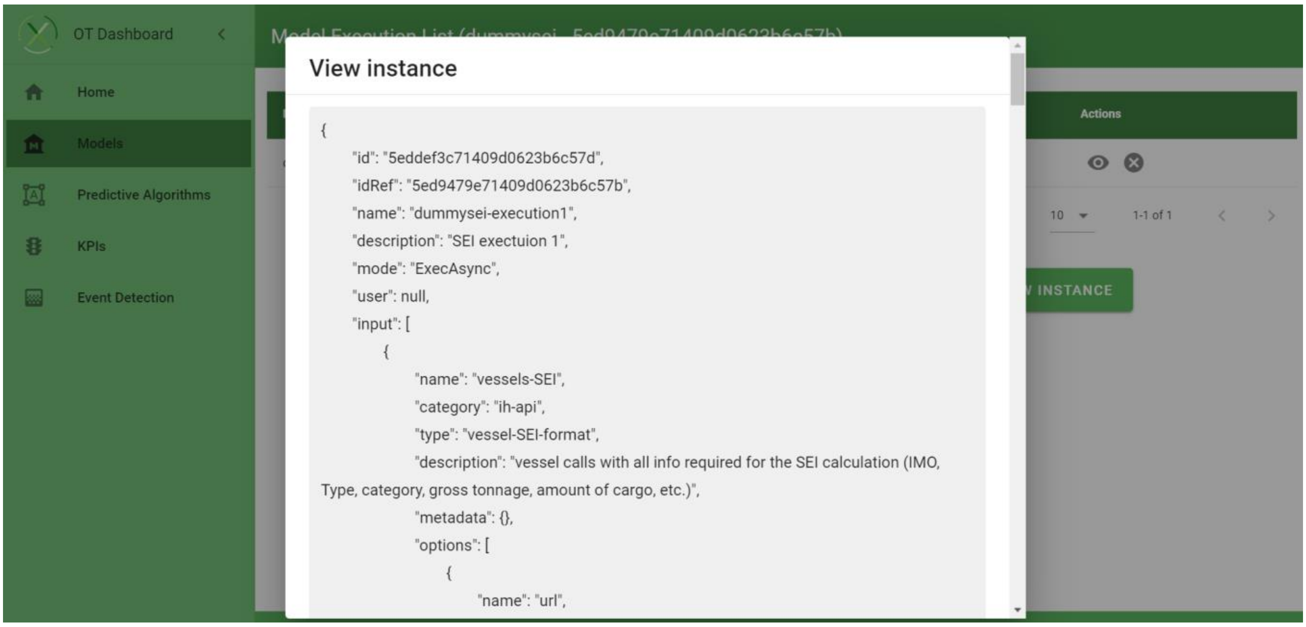
Model Execution List (dummysei - 5ed9479e71409d0623b6c57b)

Name	Start date	Status	Actions
dummysei-execution1	08/06/2020 09:56:53	finished	

Rows per page: 10 1-1 of 1

Figure 82: OT GUI. Create a new instance to run a model (Step 4)

Here you have two **action icons**. The **Delete instance** is obvious, whereas the *View instance* allows visualizing the details of the instance. It is pretty much the same as the input data provided when the instance was created, with some additional information added by the backend process (*creation time*, *start*, *otStatus*, *dockerId*). Note that the result of the execution is stored in Elasticsearch; the visualization of such result is model dependent as is provided by the PIXEL Dashboard.



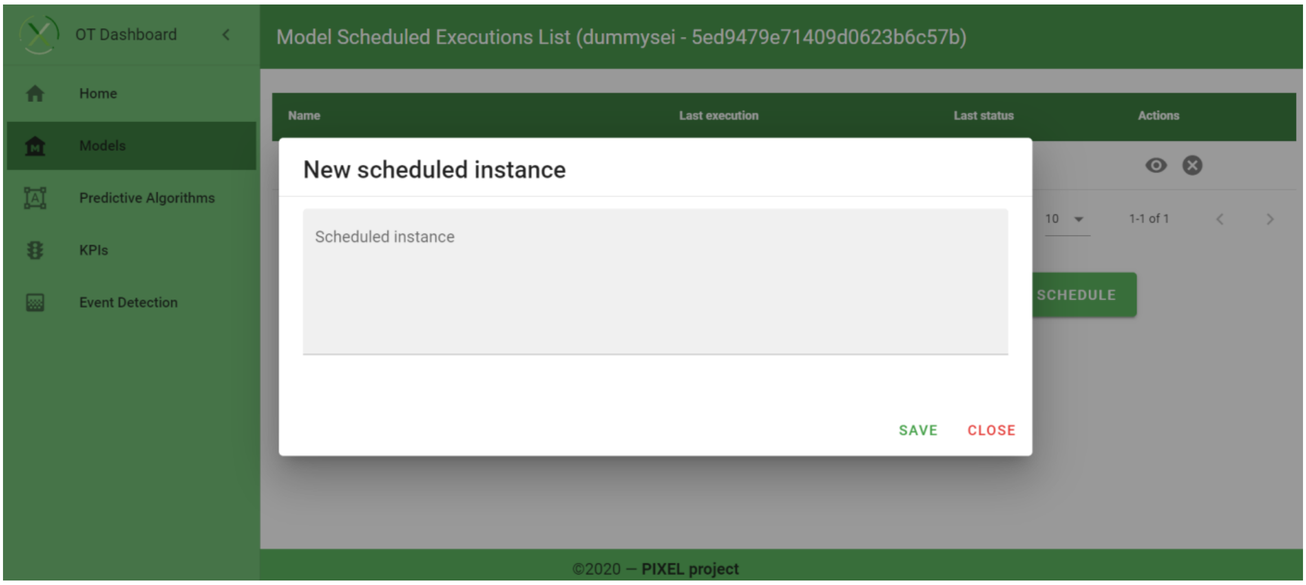
View instance

```
{
    "id": "5eddef3c71409d0623b6c57d",
    "idRef": "5ed9479e71409d0623b6c57b",
    "name": "dummysei-execution1",
    "description": "SEI exectuion 1",
    "mode": "ExecAsync",
    "user": null,
    "input": [
        {
            "name": "vessels-SEI",
            "category": "ih-api",
            "type": "vessel-SEI-format",
            "description": "vessel calls with all info required for the SEI calculation (IMO,
Type, category, gross tonnage, amount of cargo, etc.)",
            "metadata": {},
            "options": [
                {
                    "name": "url",
```

Figure 83: OT GUI. Create a new instance to run a model (Step 5)

Scheduling a Model

Some models are useful every day, every week, etc., and can be run automatically (scheduled), without any reason for user presence. The process of scheduling a model is analogous to the previous one (running a model), just click on the **Schedule model** action icon of the models list and you should be able to follow a similar process.



Model Scheduled Executions List (dummysei - 5ed9479e71409d0623b6c57b)

Name	Last execution	Last status	Actions

New scheduled instance

Scheduled instance

Figure 84: OT GUI. Create a new scheduled instance to run a model

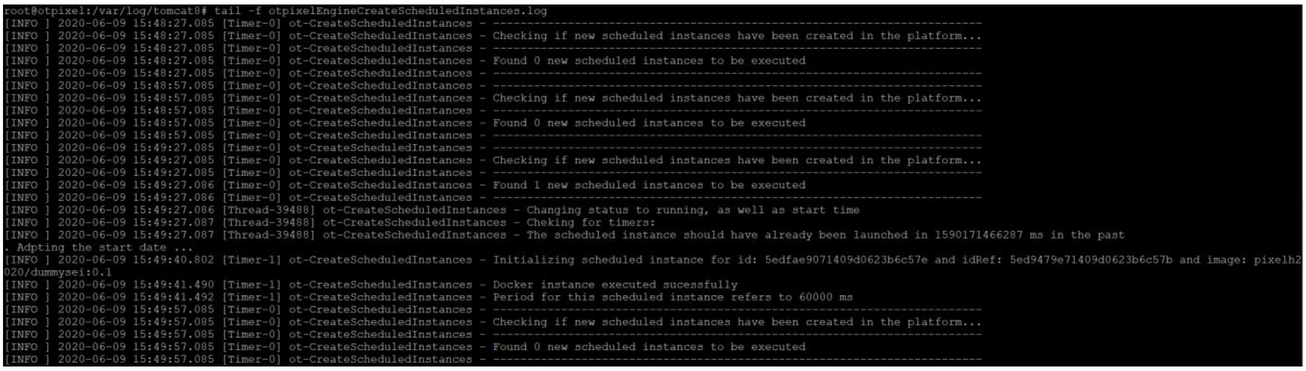
The only difference here resides in the fact that the model is going to be launched periodically in this case, not just once. Therefore, when we enter the JSON data of a scheduled instance, we need to include such data, which follows the structure:

```
"scheduleInfo": {
    "start": 1546300800,
    "unit": "minute",
    "value": 1
 }
```

The **start** field indicates (Unix time) when the model must be first launched, the **unit** filed represents the possible units (*second, minute, hour, day*) and the **value** field represents the number of units to wait between consecutive executions. In the example above, model will be run every minute.

The given *start* time should typically represent one timestamp in the future. However, if the given *start* time is any time in the past, the OT engine will recalculate the **nearest point of** time in the future as result of the **N-th multiple** of the given amount of time (here multiples are count every minute).

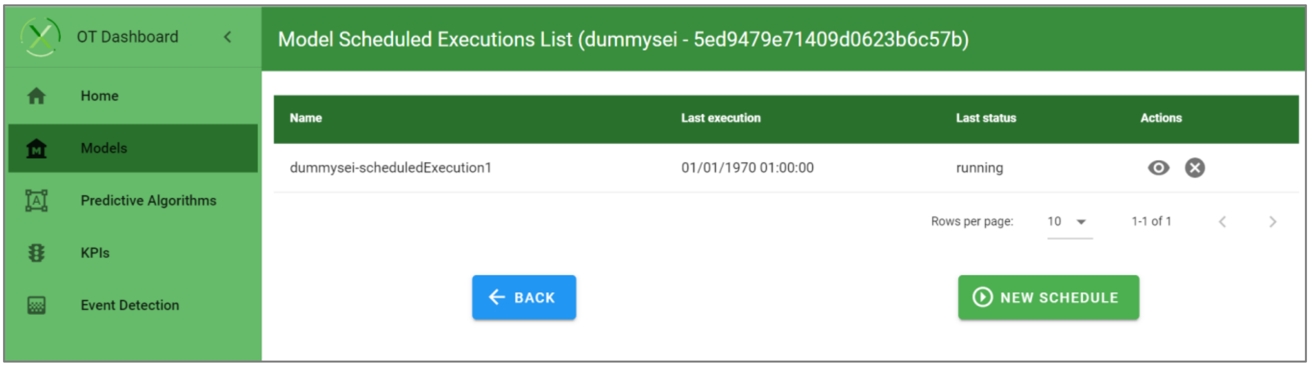
You can trace the backend process that periodically reads the corresponding table and runs the pending scheduled instances. The log is on **/var/log/tomcat8/otpixelEngineCreateScheduledInstances.log**:



```
root@otpixel:/var/log/tomcat8# tail -f otpixelEngineCreateScheduledInstances.log
[INFO ] 2020-06-09 15:48:27.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:48:27.085 [Timer-0] ot-CreateScheduledInstances - Checking if new scheduled instances have been created in the platform...
[INFO ] 2020-06-09 15:48:27.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:48:27.085 [Timer-0] ot-CreateScheduledInstances - Found 0 new scheduled instances to be executed
[INFO ] 2020-06-09 15:48:27.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:48:57.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:48:57.085 [Timer-0] ot-CreateScheduledInstances - Checking if new scheduled instances have been created in the platform...
[INFO ] 2020-06-09 15:48:57.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:48:57.085 [Timer-0] ot-CreateScheduledInstances - Found 0 new scheduled instances to be executed
[INFO ] 2020-06-09 15:48:57.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:49:27.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:49:27.085 [Timer-0] ot-CreateScheduledInstances - Checking if new scheduled instances have been created in the platform...
[INFO ] 2020-06-09 15:49:27.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:49:27.086 [Timer-0] ot-CreateScheduledInstances - Found 1 new scheduled instances to be executed
[INFO ] 2020-06-09 15:49:27.086 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:49:27.086 [Thread-39488] ot-CreateScheduledInstances - Changing status to running, as well as start time
[INFO ] 2020-06-09 15:49:27.087 [Thread-39488] ot-CreateScheduledInstances - Cheking for timers:
[INFO ] 2020-06-09 15:49:27.087 [Thread-39488] ot-CreateScheduledInstances - The scheduled instance should have already been launched in 1590171466287 ms in the past
. Adpting the start date ...
[INFO ] 2020-06-09 15:49:40.802 [Timer-1] ot-CreateScheduledInstances - Initializing scheduled instance for id: 5edfae9071409d0623b6c57e and idRef: 5ed9479e71409d0623b6c57b and image: pixelh2
020/dummysei:0.1
[INFO ] 2020-06-09 15:49:41.490 [Timer-1] ot-CreateScheduledInstances - Docker instance executed sucessfully
[INFO ] 2020-06-09 15:49:41.492 [Timer-1] ot-CreateScheduledInstances - Period for this scheduled instance refers to 60000 ms
[INFO ] 2020-06-09 15:49:57.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:49:57.085 [Timer-0] ot-CreateScheduledInstances - Checking if new scheduled instances have been created in the platform...
[INFO ] 2020-06-09 15:49:57.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
[INFO ] 2020-06-09 15:49:57.085 [Timer-0] ot-CreateScheduledInstances - Found 0 new scheduled instances to be executed
[INFO ] 2020-06-09 15:49:57.085 [Timer-0] ot-CreateScheduledInstances - ------------------------------------------------------------------------
```

Figure 85: Log4j file for monitoring the creation of scheduled instances

Now in the list of scheduled instances you should see the added scheduled instance. The **Last status** column should say running, unless there is an error (error trying to execute the Docker instance) in any of the executions.



Model Scheduled Executions List (dummysei - 5ed9479e71409d0623b6c57b)

Name	Last execution	Last status	Actions
dummysei-scheduledExecution1	01/01/1970 01:00:00	running	

Rows per page: 10 1-1 of 1

Figure 86: OT GUI. Create a new scheduled instance to run a model (II)

There is one final comment and relates to timing issues. If one of the inputs for the execution of the model is a **time dependent parameter**, e.g. current day of the execution, then this should be parametrized and interpreted by the OT engine. The user cannot provide here a fixed timestamp (otherwise this would provide the same result continuously). As example, suppose a model that requires as inputs a start time and an end time to make its internal calculation; this could be the case of getting vessels calls in a time window. If we want to run the model every day, then we need to parametrize this somehow in the JSON data structure. An example could be:

```
{
    "name": "start",
    "type": "datetime (Unix time)",
    "description": "start of calculation period",
    "value": "${DATE_DAY_init}"
}, {
    "name": "end",
    "type": "datetime (Unix Time)",
    "description": "end of calculation period",
    "value": "${DATE_DAY_last}"
}
```

Here, every time the model is executed, the OT engine previously interprets the parametrized date values (${}) and changes it with the corresponding operation. Currently the OT engine supports the following ones:

Table 23: Time parametrization options supported by the OT engine

FORMAT	Description (Unix format -millis)	Potential Use
${DATE_current}	Current date	Models started by triggers?
${DATE_DAY_init}	Date of the first second of the current day	PAS
${DATE_DAY_last}	Date of the last second of the current day	PAS
${DATE_WEEK_init}	Date of the first second of the current week	PEI
${DATE_WEEK_last}	Date of the last second of the current week	PEI

4.4.3.2.2. Predictive Algorithms

The management of predictive algorithms is completely analogous as for models in the previous section. Note that even the format (when invoking the API) is the same.

4.4.3.2.3. KPIs

Key Performance Indicators (KPIs) are special indicators set by port operators (it may differ from port to port) to better track, qualify and quantify the performance of their operations. The Operational Tools allow to create such KPIs, as long as they refer to specific data available in the Information Hub (Elasticsearch).

The data in the Information Hub has to comply with the KPI data model as defined by FIWARE (see https://fiware-datamodels.readthedocs.io/en/latest/KeyPerformanceIndicator/doc/spec/index.html for further information). Some fields of the model are mandatory, other are optional. For PIXEL we will potentially add new fields that include specific information (e.g. PEI).

In order to create a KPI at OT level, just click on the **KPIs** from the **Left Menu**. You should see a list of available KPIs (or an empty list, if it is a fresh installation).

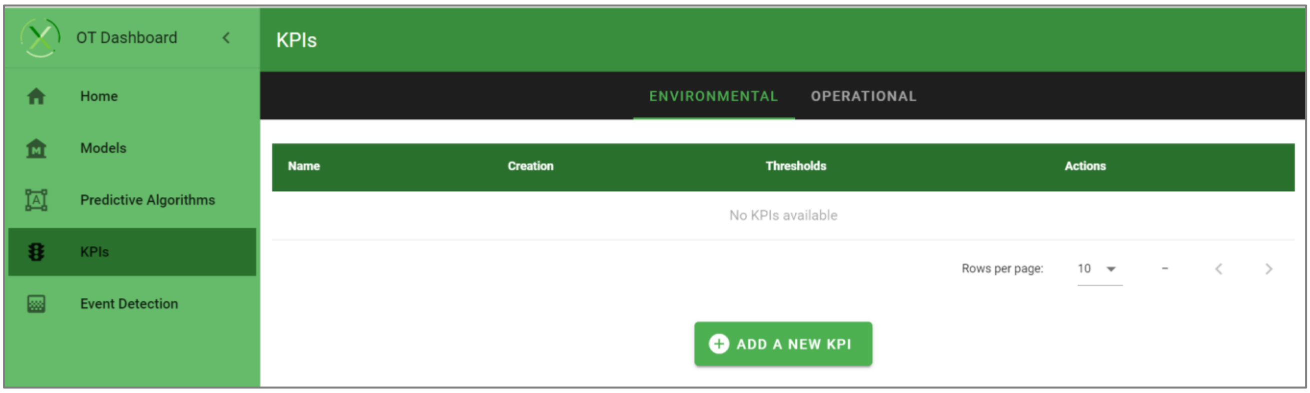
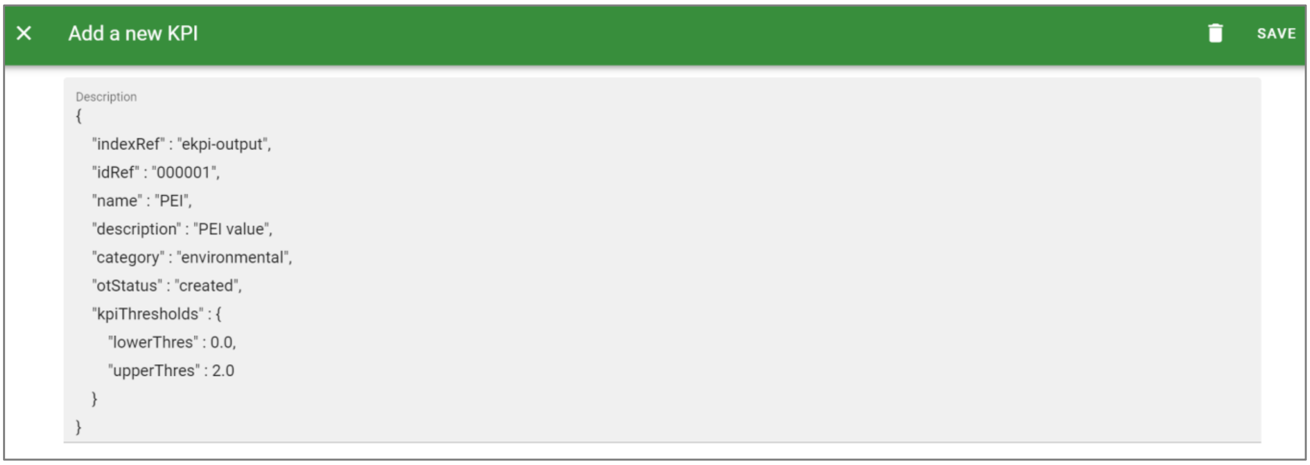


Figure 87: OT GUI. Create a new KPI (Step 1).

Click on the **Add a new KPI** button and a new modal will appear where you will have to enter the JSON description of the KPI. This is a representation of the data already available in the Information Hub; therefore the format is here not the FIWARE data model. A possible example will be the following:



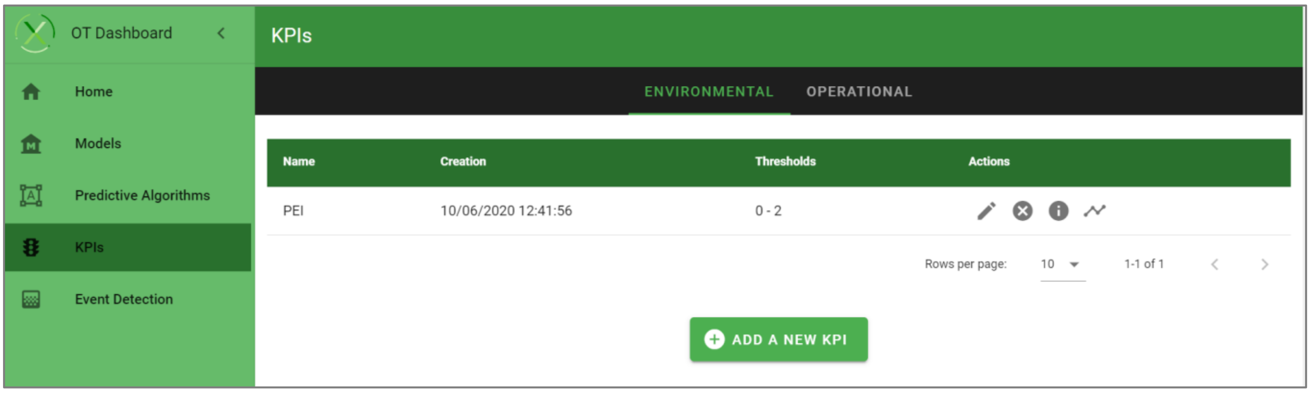
```
{
  "indexRef" : "ekpi-output",
  "idRef" : "000001",
  "name" : "PEI",
  "description" : "PEI value",
  "category" : "environmental",
  "otStatus" : "created",
  "kpiThresholds" : {
    "lowerThres" : 0.0,
    "upperThres" : 2.0
  }
}
```

Figure 88: OT GUI. Create a new KPI (Step 2).

Important fields to comment here are:

- **indexRef**: this refers to the Elasticsearch index to search for the info.
- **idRef**: the identifier of the specific KPI within the Elasticsearch index.
- **category**: associated category to classify your KPIs. Currently only environmental and operational have been identified as main categories.
- **kpiThresholds**: a set of thresholds (upper and lower) associated to this KPI. This is optional, but may allow later monitoring of KPIs, visualization and possible alarms.

After clicking on the **Save** button, you will see that the KPI is inserted in the list

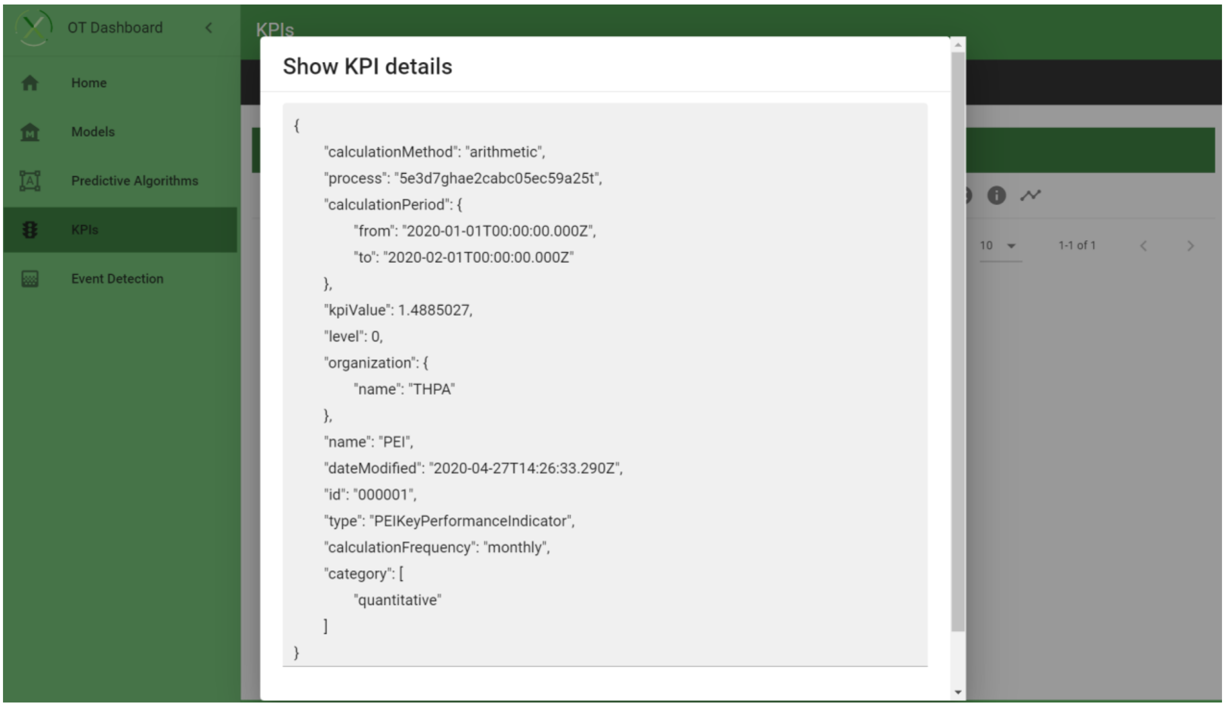


Name	Creation	Thresholds	Actions
PEI	10/06/2020 12:41:56	0 - 2	

Figure 89: OT GUI. Create a new KPI (Step 3).

Similar to the models, here there is a set of actions icons you may use:

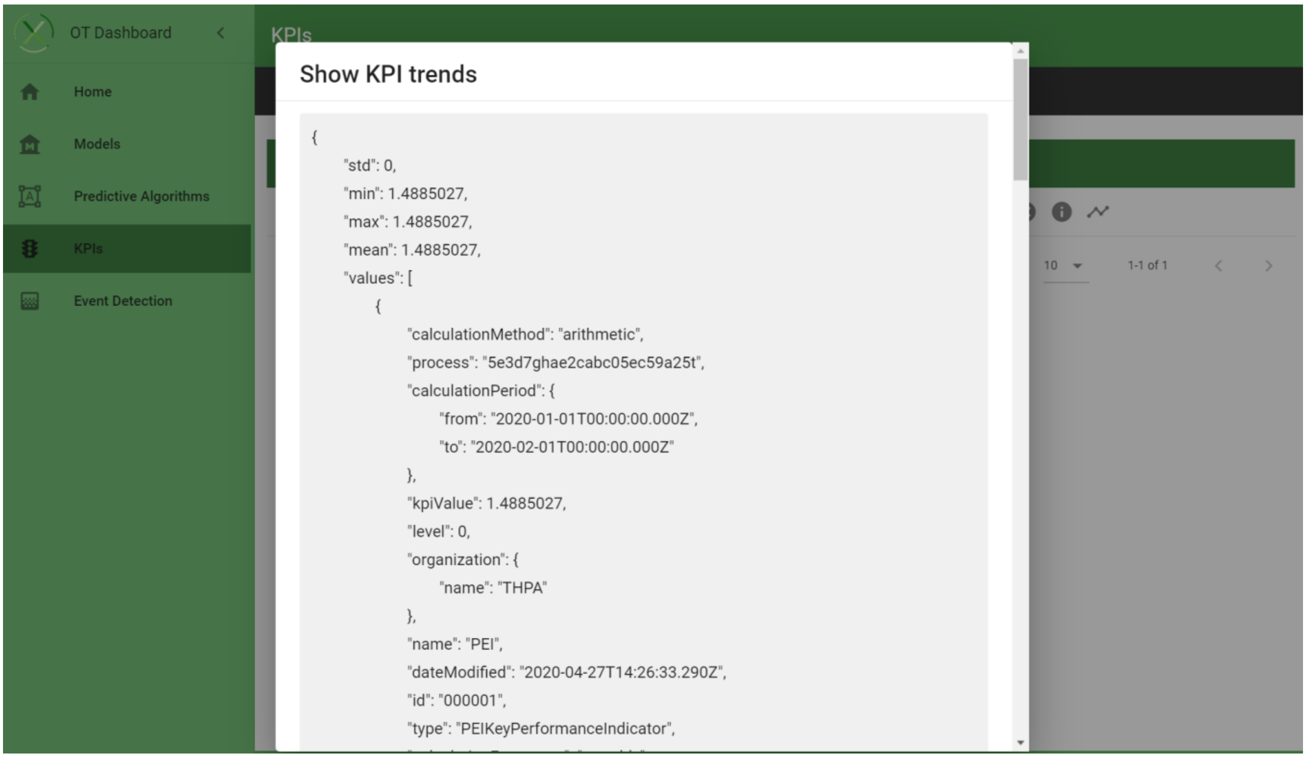
- The **Edit** icon will show you the description of the KPI (similar to the JSON structure given at creation time).
- The **Delete** icon allows you to delete the KPI. Here you are only deleting the KPI at OT level, the data is still available in the Information Hub, there is no deletion of data in Elasticsearch.
- The **Show details** icon allows you to inspect the content of the KPI that means, to retrieve the information from the Information Hub (Elasticsearch). In the Figure below you can see an example for the created KPI. The OT retrieves the information from the **last KPI** in Elasticsearch, as it is supposed to be a time series. Note that this structure is compliant with the FIWARE KPI data model. As there are several time fields in the JSON structure, the time field used for retrieving the last KPI is **calculationPeriod.from** (but you may change it at code level).



```
{
    "calculationMethod": "arithmetic",
    "process": "5e3d7ghae2cabc05ec59a25t",
    "calculationPeriod": {
        "from": "2020-01-01T00:00:00.000Z",
        "to": "2020-02-01T00:00:00.000Z"
    },
    "kpiValue": 1.4885027,
    "level": 0,
    "organization": {
        "name": "THPA"
    },
    "name": "PEI",
    "dateModified": "2020-04-27T14:26:33.290Z",
    "id": "000001",
    "type": "PEIKeyPerformanceIndicator",
    "calculationFrequency": "monthly",
    "category": [
        "quantitative"
    ]
}
```

Figure 90: OT GUI. Create a new KPI (Step 4).

- Finally, the **Show trends** icon allows to retrieve some trends of the given KPI (the API also supports an optional timeframe). The OT will provide some statistical info about the KPI values throughout time: mean standard deviation, max and min, as well as the set of KPIs as JSON structure for potential representation (this is offered in the Dashboard, not in this GUI). The Figure below represents an example for the created KPIs. Looking at the statistical info, one can easily deduce that the value has not change across time.



```
{
    "std": 0,
    "min": 1.4885027,
    "max": 1.4885027,
    "mean": 1.4885027,
    "values": [
        {
            "calculationMethod": "arithmetic",
            "process": "5e3d7ghae2cabc05ec59a25t",
            "calculationPeriod": {
                "from": "2020-01-01T00:00:00.000Z",
                "to": "2020-02-01T00:00:00.000Z"
            },
            "kpiValue": 1.4885027,
            "level": 0,
            "organization": {
                "name": "THPA"
            },
            "name": "PEI",
            "dateModified": "2020-04-27T14:26:33.290Z",
            "id": "000001",
            "type": "PEIKeyPerformanceIndicator",
```

Figure 91: OT GUI. Create a new KPI (Step 5).

4.4.3.2.4. Event Detection

There is no UI developed specifically for this purpose and it is considered further work. This is caused because this functionality at user level is offered via the PIXEL Dashboard. The reason for that is because both the Operational Tools and PIXEL Dashboard use a common element for alerting and notification based on ElastAlert.

4.4.4. PIXEL Integrated Dashboard and Notifications

Although the PIXEL platform has 5 components, users interact with it through the dashboard user interface.

The PIXEL Dashboard and Notification component provides a web User Interface to interact with the platform.

It is needed to have an account to access to the platform.

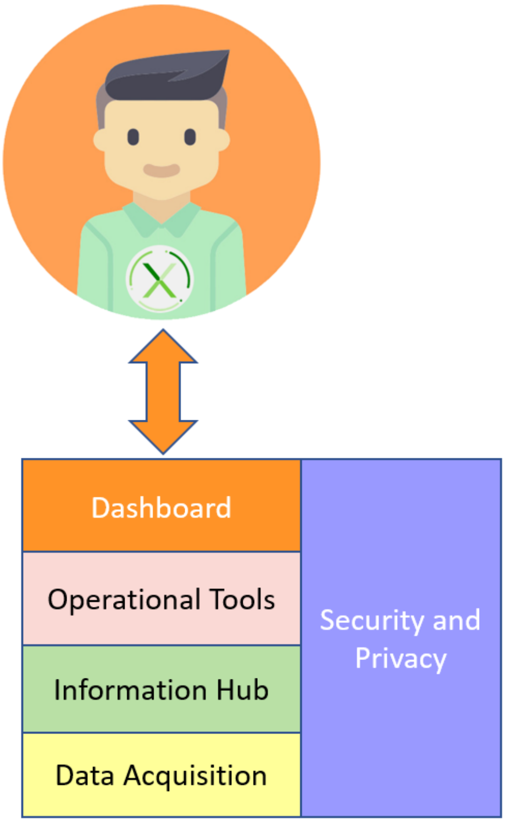
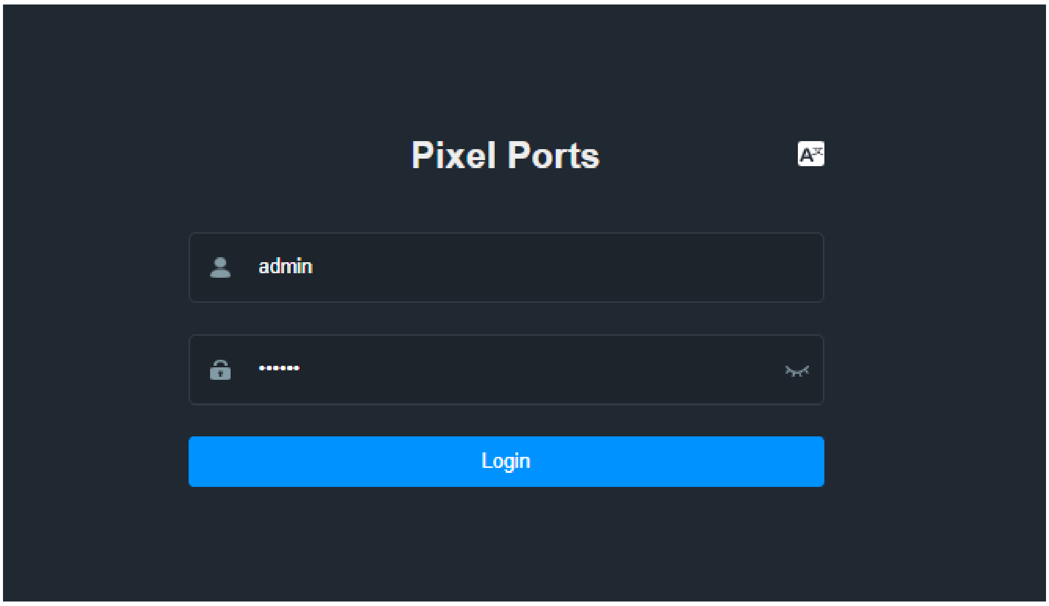


Figure 92: Platform User Interface

4.4.4.1. Login

After entering the URL of the PIXEL platform, a login form appears allowing you access the application. You need to have an account to enter into the application. Depending on your permissions you will have different functionalities available.


Pixel Ports

admin

••••••

Figure 93: Login page

4.4.4.2. Layout

The PIXEL platform has 3 main areas:

1- Menu: Provides access to different functionalities.
2- Header: Provides navigation and configuration properties.
3- Content: Provides the functionalities to interact with.

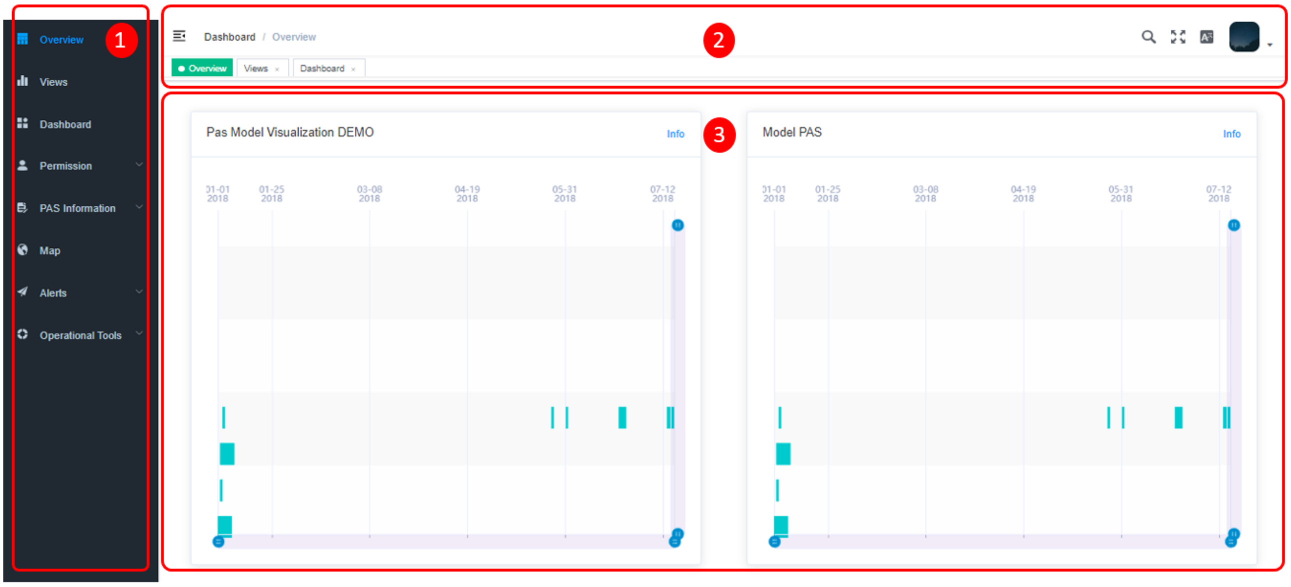


Figure 94: Layout

Header

The header provides a set of functionalities to navigate between different sections, search elements, language selection and profile information.

On the left-hand side, you will find the following functionalities:

- Condense / Extend menu.
- Breadcrumbs.
- Functionality button bar.

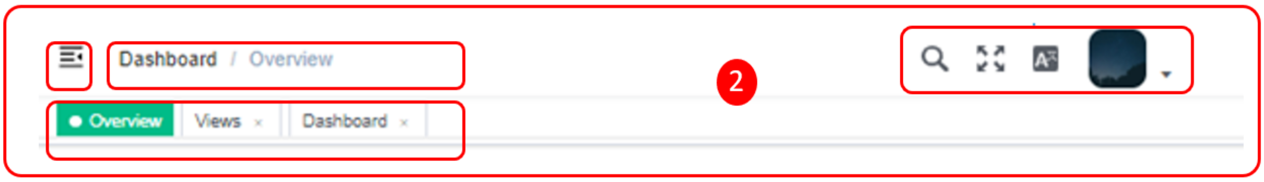


Figure 95: Header components

On the right-hand side, you will find the following functionalities:

1- Search functionality.
2- Language Selection.
3- Profile options.
4- Profile Details.
5- Logout.

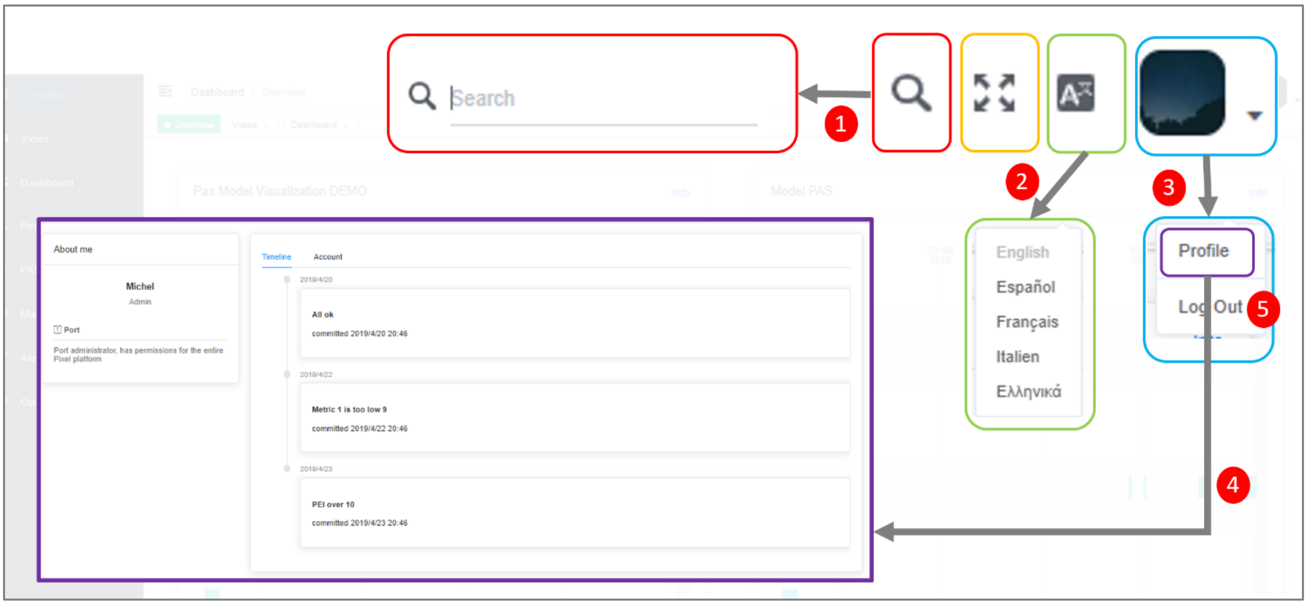


Figure 96: Header configuration options

Menu

The menu allows the user to access all the available functionalities of the platform.

By default, the menu appears in extended mode, but it can be extended through the compact / extend icon.

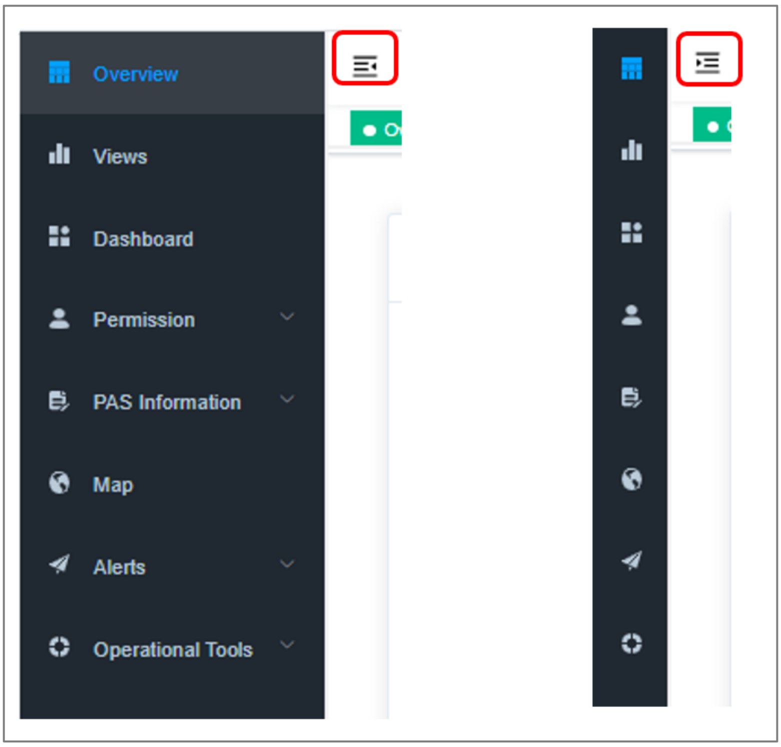


Figure 97: Extended /compact menu

The functionalities available are:

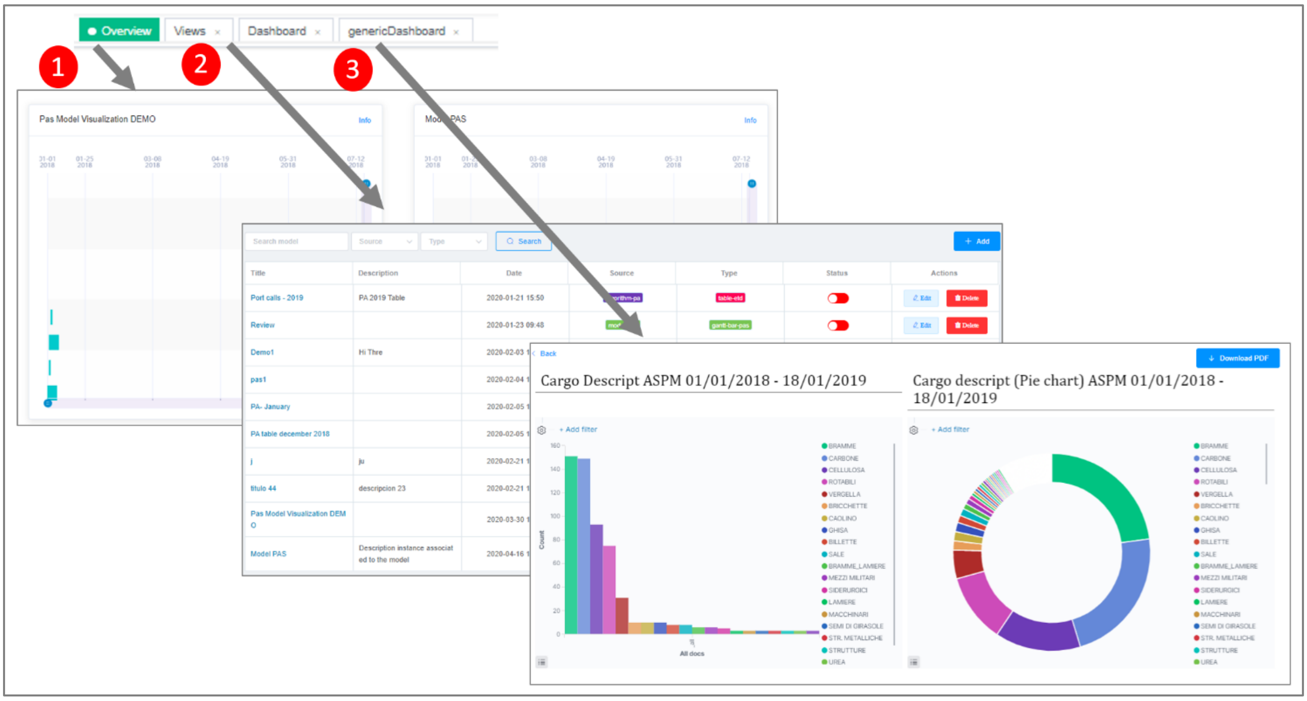
Table 24: Dashboard Functionalities summary

Name	Description
Overview	List of information visualization to control port operations
Views	Manage the visualizations to be shown in the overview functionality
Dashboard	Manage the creation of reports
Permission	Manage the roles and uses of the platform
PAS Information	Manage the Port Activity Scenario information
MAP	Geographical information system with real-time sensors
Alerts	Manage alerts definition and subscription
Operational Tools	Manage Models and Algorithms

Content

The content area is the largest display area and shows the content of the functionality selected in the menu.

At the top (Figure 98: Different content sections), there is a tab bar that allows quick access to previously opened content (1, 2, 3).



Title	Description	Date	Source	Type	Status	Actions
Port calls - 2019	PA 2019 Table	2020-01-21 15:50	orithm-pa	table-etd		Edit Delete
Review		2020-01-23 09:48	mod	gantt-bar-pas		Edit Delete
Demo1	Hi Thre	2020-02-03 1				
pas1		2020-02-04 1				
PA- January		2020-02-05 1				
PA table december 2018		2020-02-05 1				
j	ju	2020-02-21 1				
titulo 44	descripcion 23	2020-02-21 1				
Pas Model Visualization DEMO		2020-03-30 1				
Model PAS	Description instance associated to the model	2020-04-16 1				

Cargo Descript ASPM 01/01/2018 - 18/01/2019

Cargo descript (Pie chart) ASPM 01/01/2018 - 18/01/2019

Figure 98: Different content sections

4.4.4.3. Permission

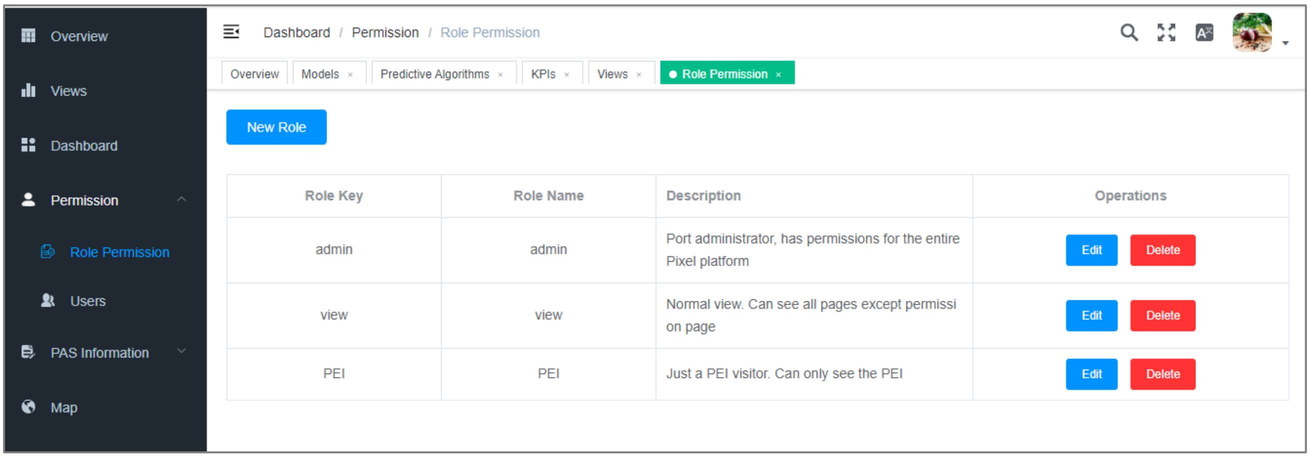
The permission functionality allows you to manage the roles and users of the platform. This functionality uses internally the security layer of PIXEL.

4.4.4.3.1. Role permission

The role permission functionality allows managing the roles of the platform “the different types of users that will use the platform and what actions they can execute”.

This functionality will only be available to administrators.

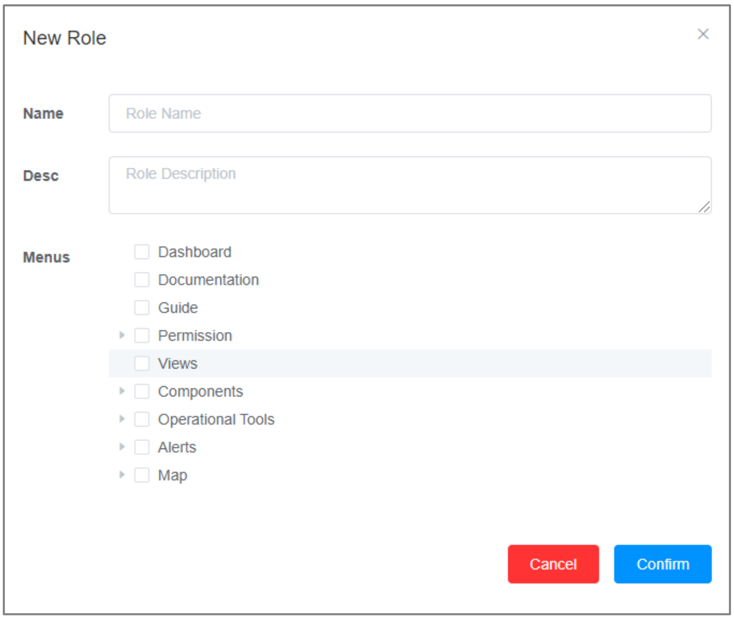
The list of roles shows the roles defined and provides the functionality to manage them (Create, Edit, Delete).



Role Key	Role Name	Description	Operations
admin	admin	Port administrator, has permissions for the entire Pixel platform	Edit Delete
view	view	Normal view. Can see all pages except permissi on page	Edit Delete
PEI	PEI	Just a PEI visitor. Can only see the PEI	Edit Delete

Figure 99: List of roles

If you want to create a new role, select the "New role" button. After selecting the button, a form appears that allows defining the characteristics of the new role.


New Role ×

Name: Role Name

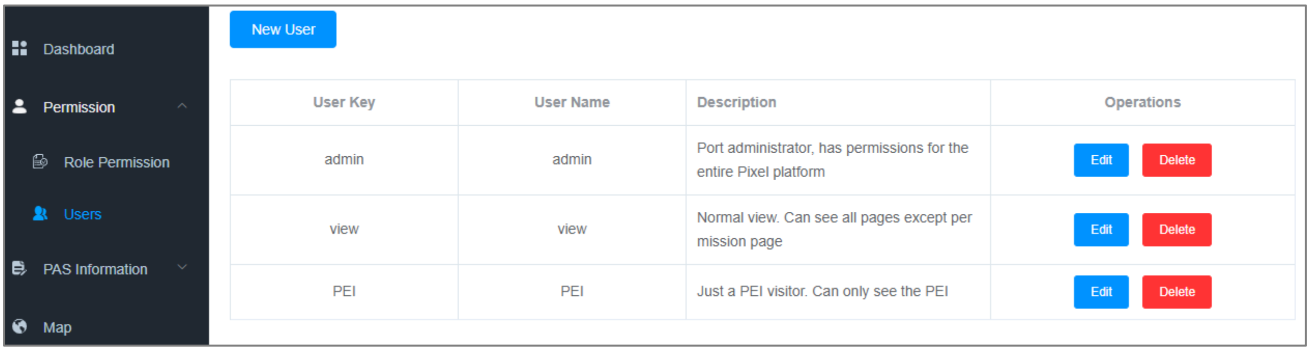
Desc: Role Description

Menus:
- □ Dashboard
- □ Documentation
- □ Guide
- □ Permission
- □ Views
- □ Components
- □ Operational Tools
- □ Alerts
- □ Map

Figure 100: Create a New Role

4.4.4.3.2. Users

The user functionality allows managing the users that can access to the platform. The list of users provides a list of users created.



User Key	User Name	Description	Operations
admin	admin	Port administrator, has permissions for the entire Pixel platform	Edit Delete
view	view	Normal view. Can see all pages except per mission page	Edit Delete
PEI	PEI	Just a PEI visitor. Can only see the PEI	Edit Delete

Figure 101: List of users

If you want to create a new user, select the "New user" button. After selecting the button, a form appears that allows defining the properties of a new user.

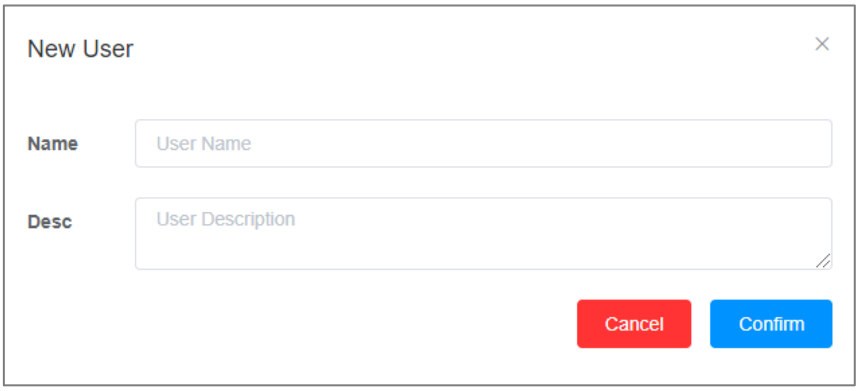


Figure 102: Create new user

4.4.4.4. Overview and Views

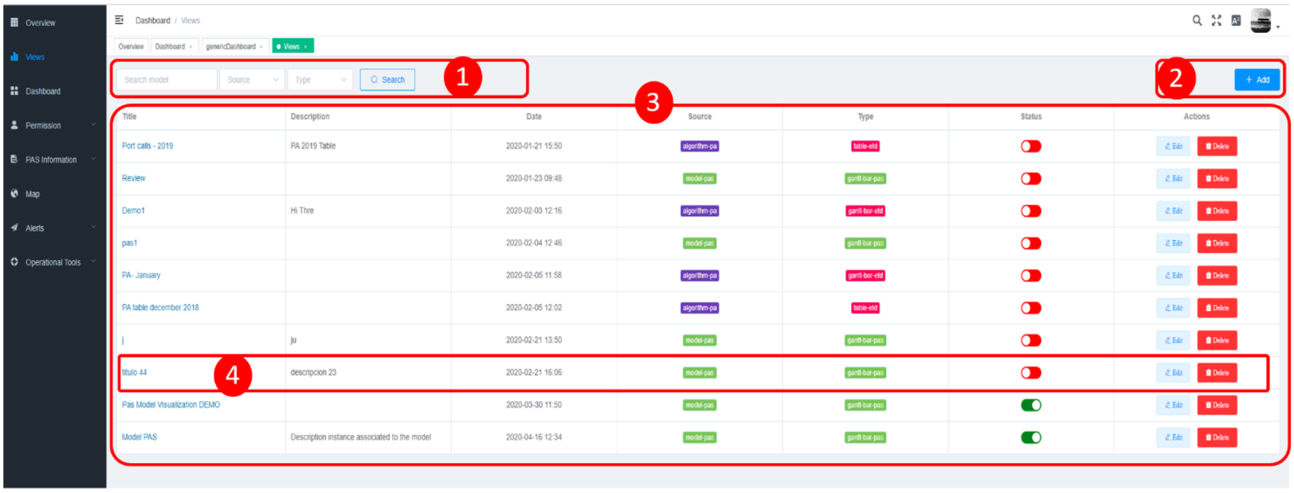
These functionalities allow defining and showing the most suitable visualizations to monitor the port activity depending on the specific needs of each port.

Note: Most of the visualizations that a user can create are related with model executions. Before creating these visualizations, the user has to add a model and run it form the operational tools functionality.

4.4.4.4.1. Views

List of Views - Visualizations

From the list of views, the user can manage the most appropriate visualizations to control the port activity. The list of views provides a search mechanism (1) to filter by name, source or type the views showed in the table. The Add (2) button allows creating a new visualization. (3) Shows a list of the different visualizations. For each visualization (4), the user can change the visualization status. Only the enabled visualization will appear in the overview.



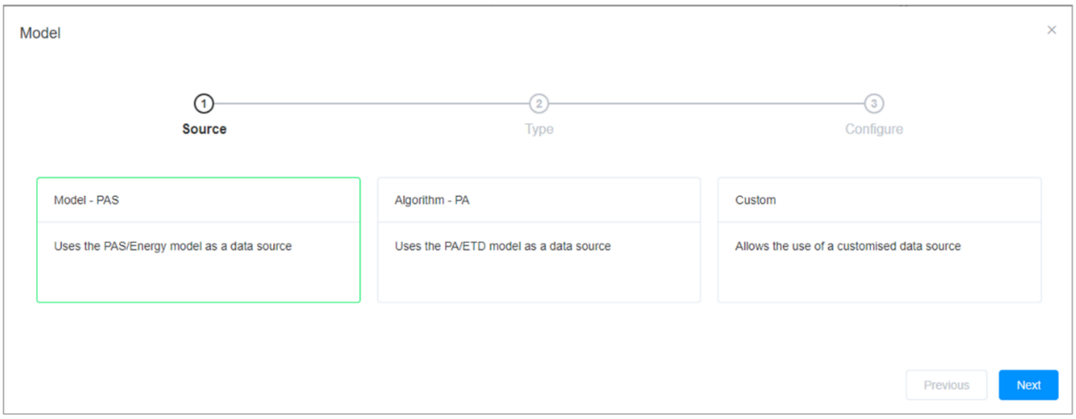
Title	Description	Date	Source	Type	Status	Actions
Port calls - 2019	PA 2019 Table	2020-01-21 15:50	algorithm-pa	table-etd	○	Edit Delete
Review		2020-01-23 09:48	model-pas	gantt-bar-pas	○	Edit Delete
Demo1	Hi Thre	2020-02-03 12:16	algorithm-pa	gantt-bar-etd	○	Edit Delete
pas1		2020-02-04 12:46	model-pas	gantt-bar-pas	○	Edit Delete
PA- January		2020-02-05 11:58	algorithm-pa	gantt-bar-etd	○	Edit Delete
PA table december 2018		2020-02-05 12:02	algorithm-pa	table-etd	○	Edit Delete
j	ju	2020-02-21 13:50	model-pas	gantt-bar-pas	○	Edit Delete
titulo 44	descripcion 23	2020-02-21 16:06	model-pas	gantt-bar-pas	○	Edit Delete
Pas Model Visualization DEMO		2020-03-30 11:50	model-pas	gantt-bar-pas	●	Edit Delete
Model PAS	Description instance associated to the model	2020-04-16 12:34	model-pas	gantt-bar-pas	●	Edit Delete

Figure 103: List of visualizations

Create a new view for and specific model

The add visualization functionality has three steps.

- 1st Step: the user chooses the model for which he wants to create a new visualization. Currently the platform provides visualization for the PAS model and for the Predictive ETD algorithm. Moreover, the user can create a custom visualization.



Model - PAS

Uses the PAS/Energy model as a data source

Algorithm - PA

Uses the PA/ETD model as a data source

Custom

Allows the use of a customised data source

Figure 104: Create Visualization - Step 1

- 2nd Step: In the second step, the user chooses the visualization most appropriated for the analysis he wants to perform.

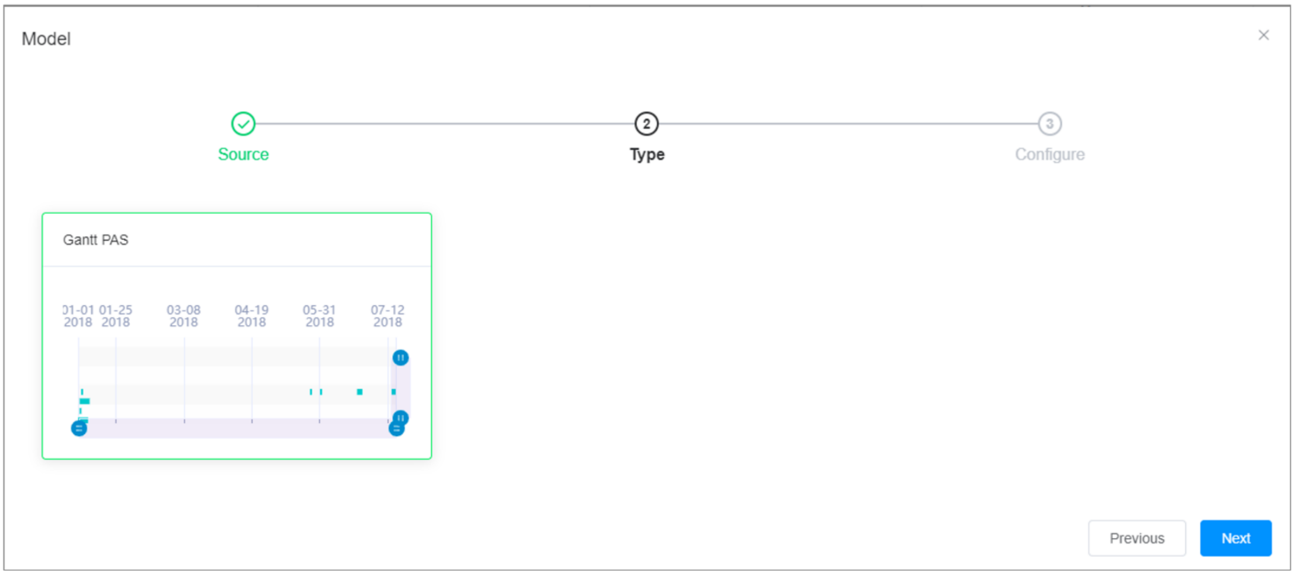
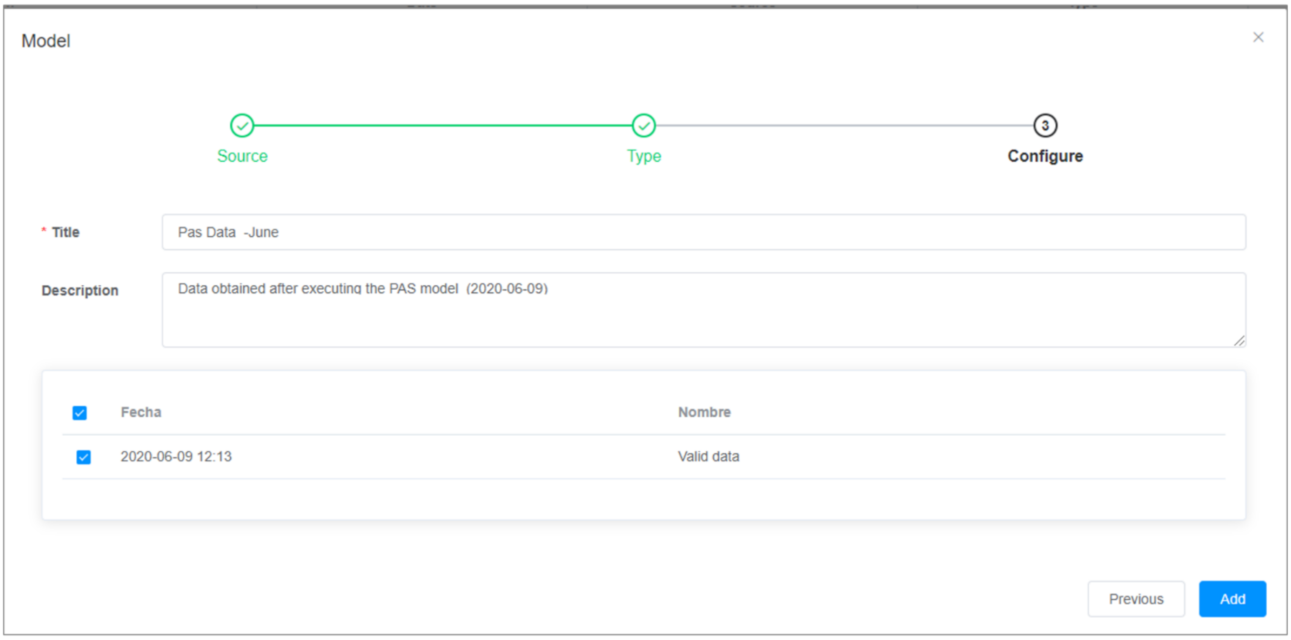


Figure 105: Create Visualization - Step 2

- 3rd Step: In the third step, the user defines the name of the visualization and selects the proper model execution to be shown.



* Title: Pas Data -June

Description: Data obtained after executing the PAS model (2020-06-09)

☑	Fecha	Nombre
☑	2020-06-09 12:13	Valid data

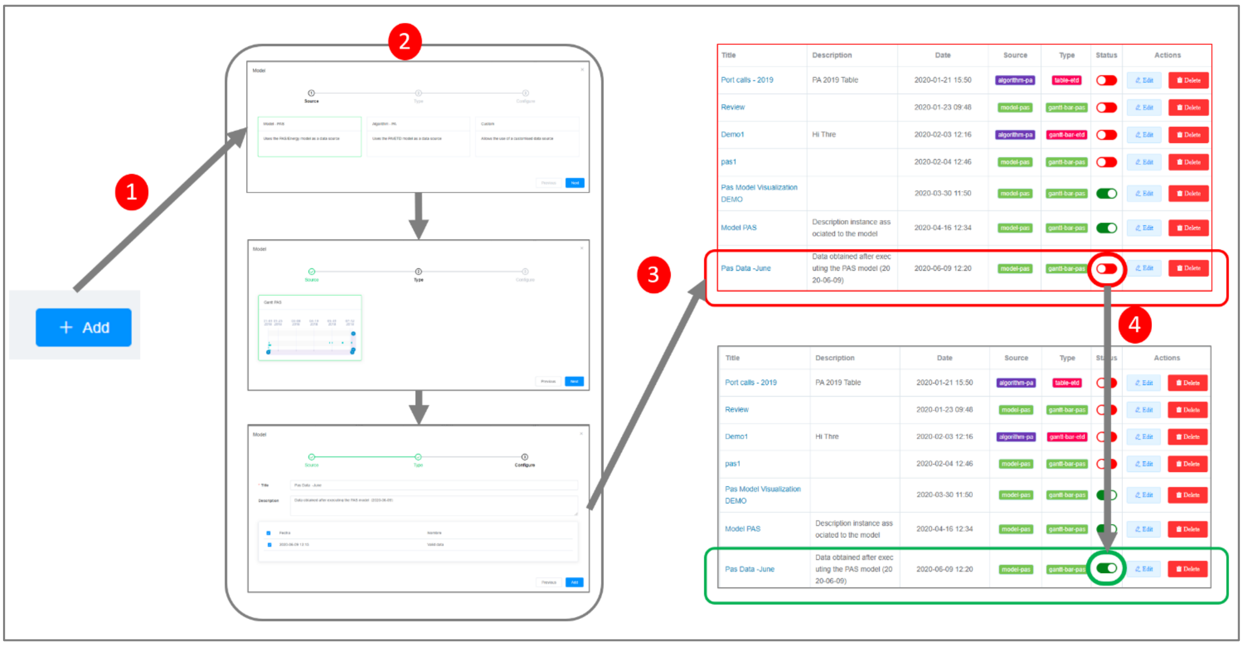
Figure 106: Create Visualization - Step 3

Process of creating and visualizing a new view

To show a new visualization on the overview, you need to follow these steps:

1. In the list view, click the add button.
2. Complete the three steps for creating a new visualization.
3. Look for the visualization created.

4. Change the status from disabled to enabled.



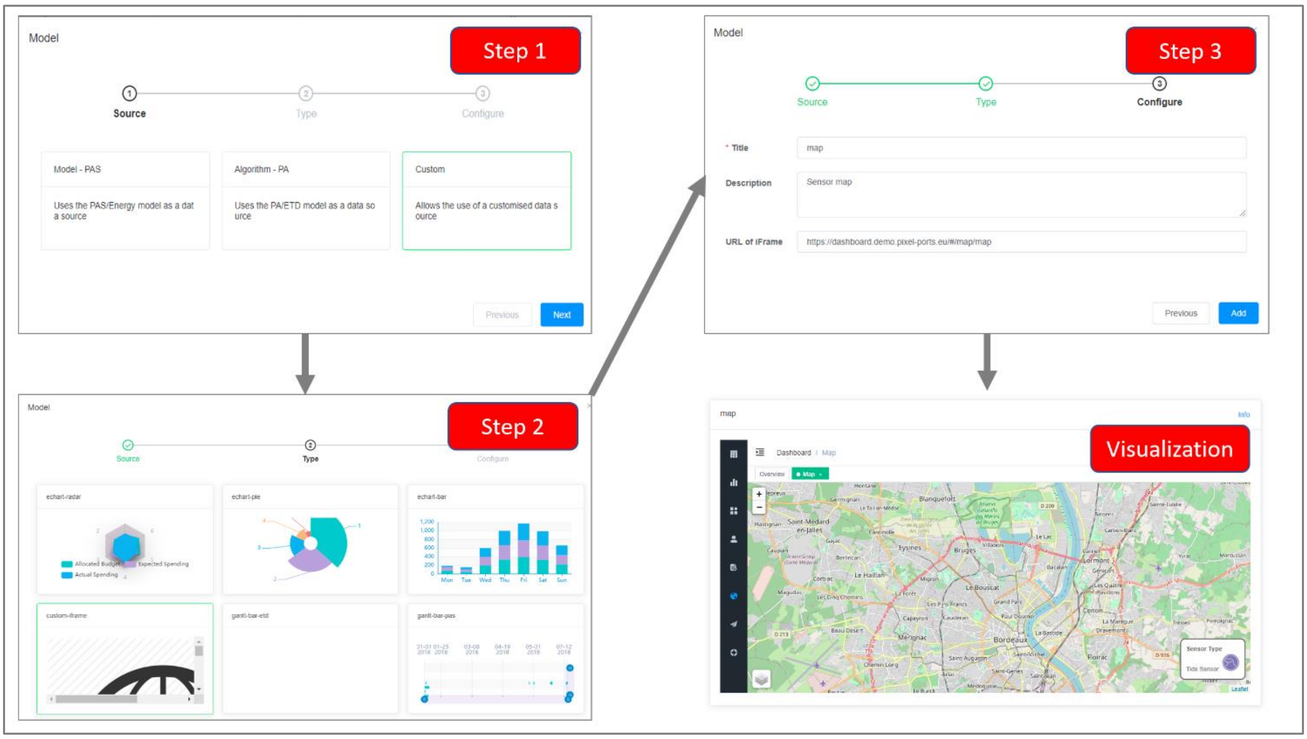
Title	Description	Date	Source	Type	Status	Actions
Port calls - 2019	PA 2019 Table	2020-01-21 15:50	algorithm-pa	table-etd	○	Edit Delete
Review		2020-01-23 09:48	model-pas	gantt-bar-pas	○	Edit Delete
Demo1	Hi Thre	2020-02-03 12:16	algorithm-pa	gantt-bar-etd	○	Edit Delete
pas1		2020-02-04 12:46	model-pas	gantt-bar-pas	○	Edit Delete
Pas Model Visualization DEMO		2020-03-30 11:50	model-pas	gantt-bar-pas	●	Edit Delete
Model PAS	Description instance associated to the model	2020-04-16 12:34	model-pas	gantt-bar-pas	●	Edit Delete
Pas Data -June	Data obtained after executing the PAS model (2020-06-09)	2020-06-09 12:20	model-pas	gantt-bar-pas	○	Edit Delete

Title	Description	Date	Source	Type	Status	Actions
Port calls - 2019	PA 2019 Table	2020-01-21 15:50	algorithm-pa	table-etd	○	Edit Delete
Review		2020-01-23 09:48	model-pas	gantt-bar-pas	○	Edit Delete
Demo1	Hi Thre	2020-02-03 12:16	algorithm-pa	gantt-bar-etd	○	Edit Delete
pas1		2020-02-04 12:46	model-pas	gantt-bar-pas	○	Edit Delete
Pas Model Visualization DEMO		2020-03-30 11:50	model-pas	gantt-bar-pas	●	Edit Delete
Model PAS	Description instance associated to the model	2020-04-16 12:34	model-pas	gantt-bar-pas	●	Edit Delete
Pas Data -June	Data obtained after executing the PAS model (2020-06-09)	2020-06-09 12:20	model-pas	gantt-bar-pas	●	Edit Delete

Figure 107: Create Visualization Process

Create a new custom view

If a user wants to create a custom visualization, independent from any model, in the process of creating a new visualization, the user has to select the custom option in the step 1. In the next figure, you can see how a user can create a custom visualization to show in an iframe the content of another web.



Step 1 – Model – Source:

- Model - PAS: Uses the PAS/Energy model as a data source
- Algorithm - PA: Uses the PA/ETD model as a data source
- Custom: Allows the use of a customised data source

Step 3 – Model – Configure:

- * Title: map
- Description: Sensor map
- URL of iFrame: https://dashboard.demo.pixel-ports.eu/#/map/map

Figure 108: Custom Visualizations

4.4.4.4.2. Overviews

The overview is the main control panel to monitor and analyse the port activities. The overview will show all the enabled views in a table of two columns. In the Figure 108: Custom Visualizations you can see the layout of the visualizations showed in the overview.

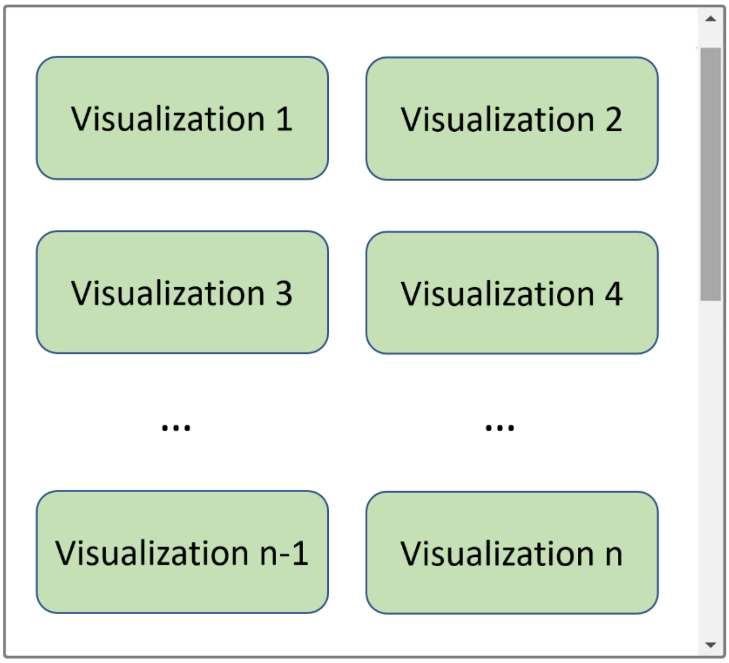


Figure 109: Overview Layout

Next figure shows an example of the overviews with 3 visualizations.

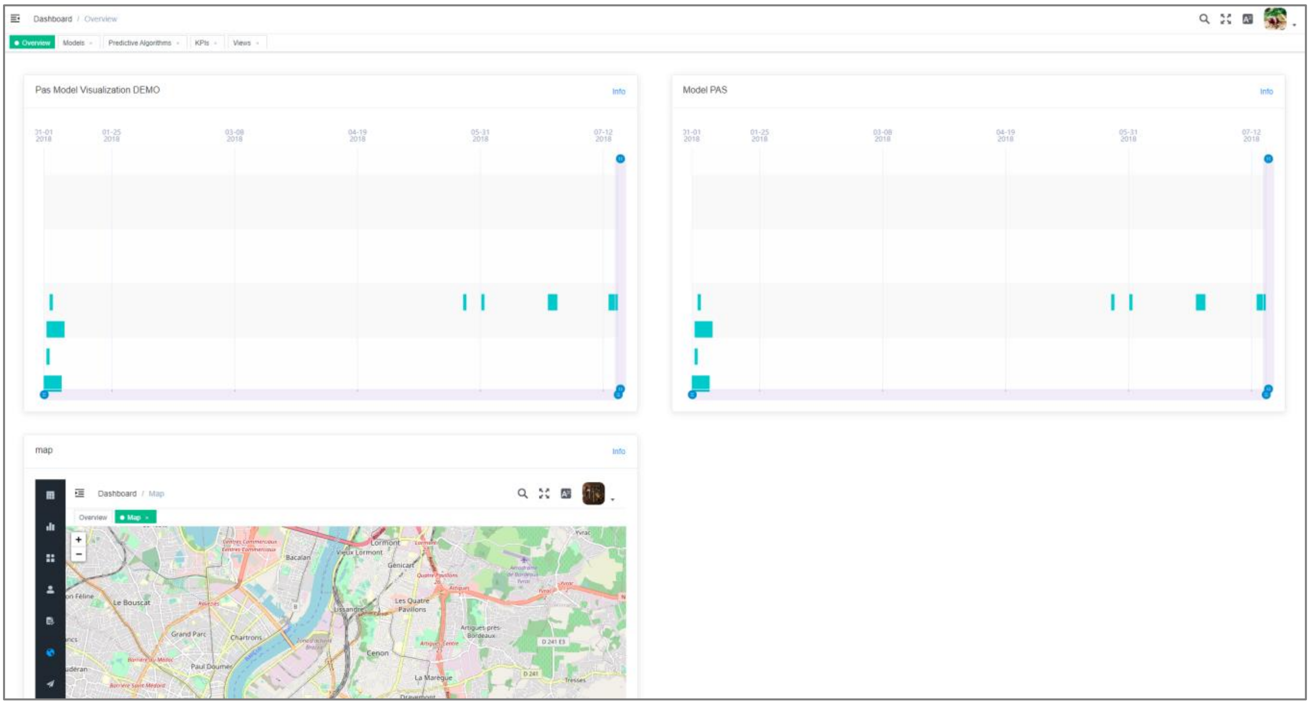
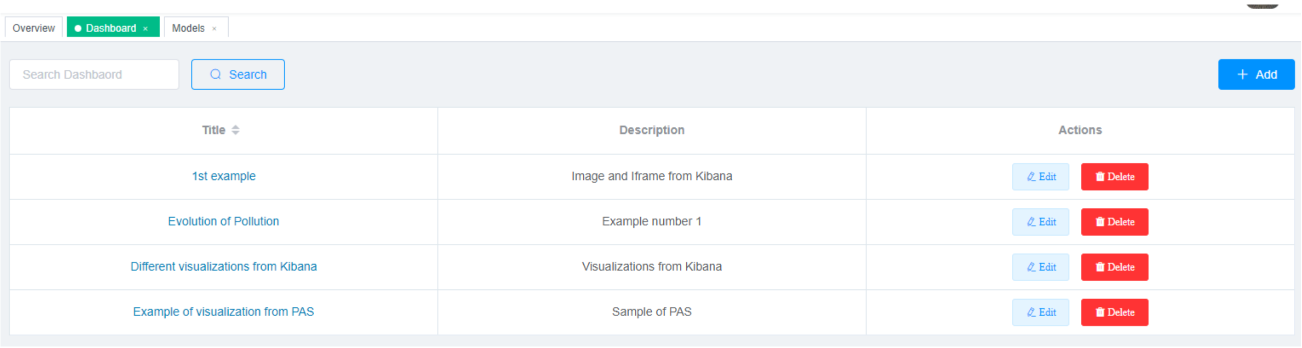


Figure 110: Example of overview with 3 visualizations

4.4.4.5. Dashboard – Reporting

This functionality allows the users to create flexible dashboards and reports thanks to a mechanism that lets the user insert different types of information in any place/size of the report. During the creation of the dashboard the user can drag and drop and resize any visual component to make the most adequate report for each case.

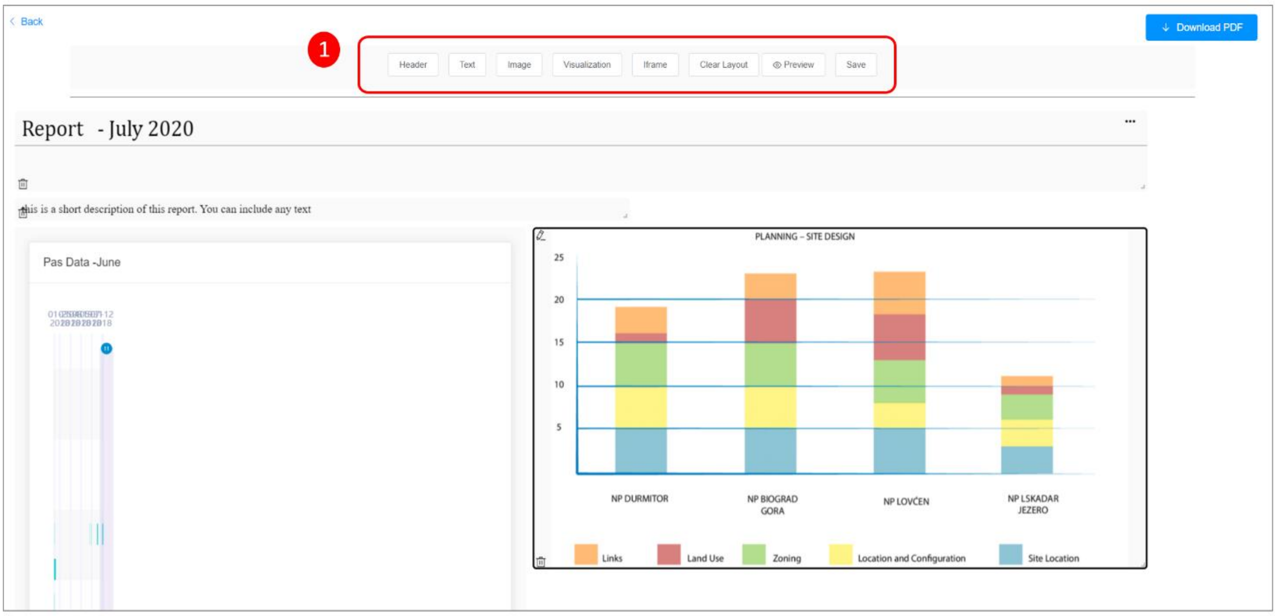


Title	Description	Actions
1st example	Image and Iframe from Kibana	Edit Delete
Evolution of Pollution	Example number 1	Edit Delete
Different visualizations from Kibana	Visualizations from Kibana	Edit Delete
Example of visualization from PAS	Sample of PAS	Edit Delete

Figure 111: Dashboard Management

Dashboard creation/edition

If a user wants to create a new dashboard (click Add button) or edit an existing one (click Edit button).



Report - July 2020

this is a short description of this report. You can include any text

Figure 112: Dashboard Creation

Dashboard visualization and Print

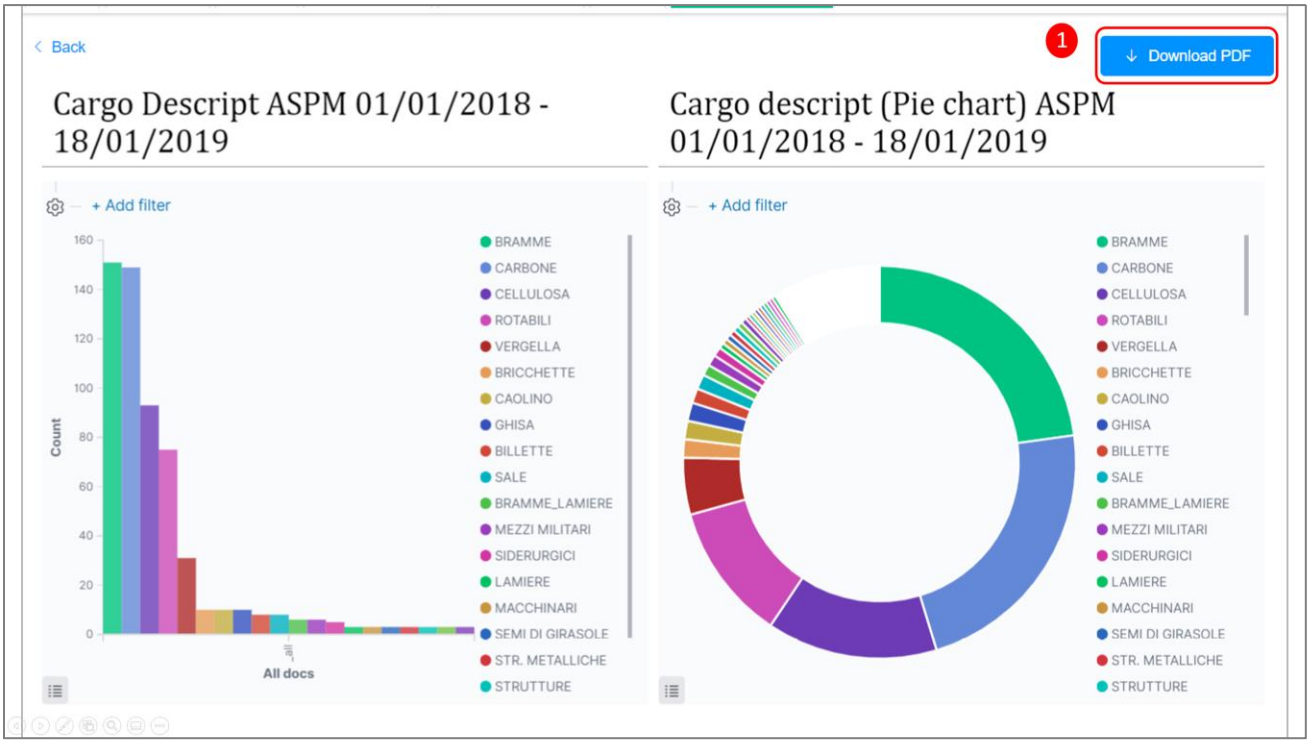


Figure 113: Dashboard Visualization

4.4.4.6. PAS Information

The PAS Information allows defining the information needed to use the Port Activity Scenario model (PAS Model).

The information has been divided in 3 sections:

- Rules: Cargo Categories, Shift works and priorities information.
- Resources: Machines and Areas information.
- Supplier Chain: Supplier chain information.

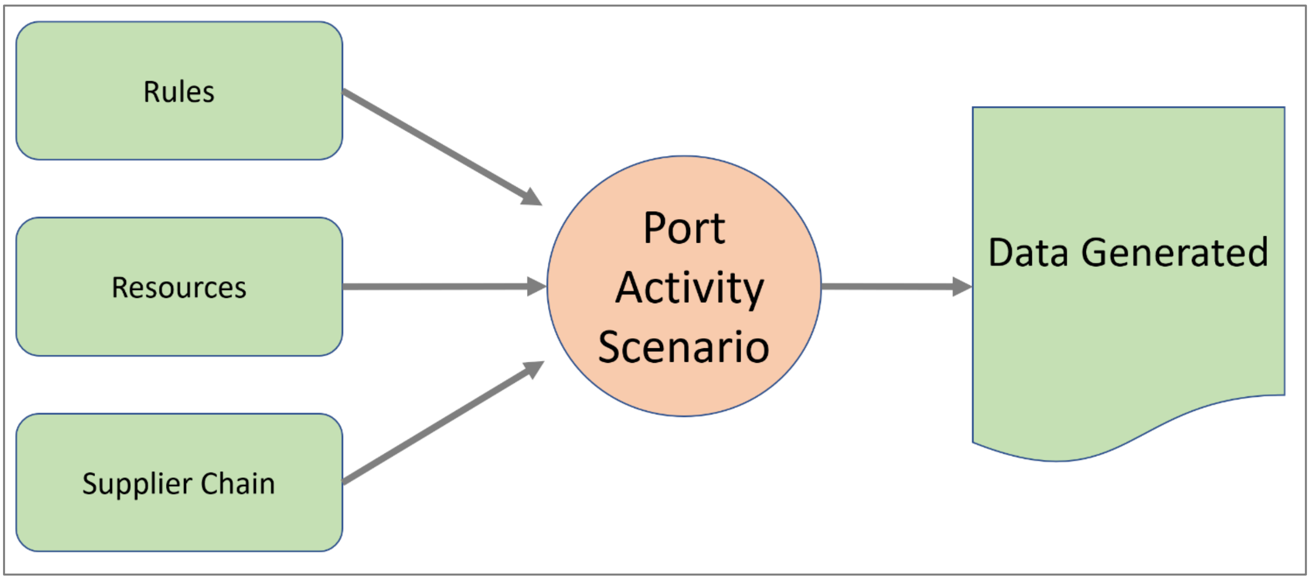
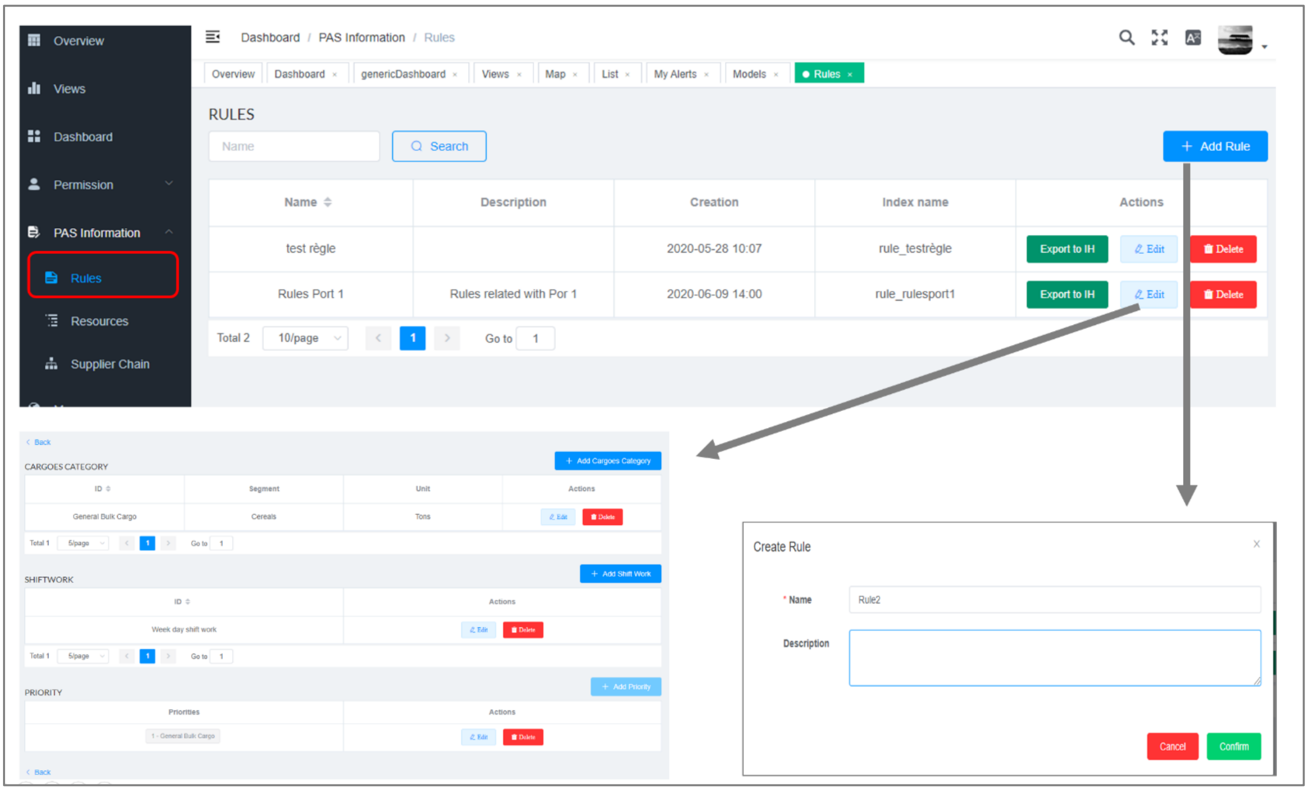


Figure 114: PAS inputs and outputs

4.4.4.6.1. Rules

The rules functionality allows creating the information related with cargo types, shift works and priorities. It is possible to create several rules. After finalising the definition of the rules, it is possible to publish the information (export to Information Hub) to be used by the PAS Model.

After creating a new rule (Add Rule button), the user can complete the content of the rule through the Edit button that opens a new page. From this page, it is possible to define cargo categories, shift works and priorities.



Name	Description	Creation	Index name	Actions
test règle		2020-05-28 10:07	rule_testrègle	Export to IH Edit Delete
Rules Port 1	Rules related with Por 1	2020-06-09 14:00	rule_rulesport1	Export to IH Edit Delete

Total 2 10/page 1 Go to 1

ID	Segment	Unit	Actions
General Bulk Cargo	Cereals	Tons	Edit Delete

Total 1 5/page 1 Go to 1

ID	Actions
Week day shift work	Edit Delete

Total 1 5/page 1 Go to 1

Priorities	Actions
1 - General Bulk Cargo	Edit Delete

* Name: Rule2

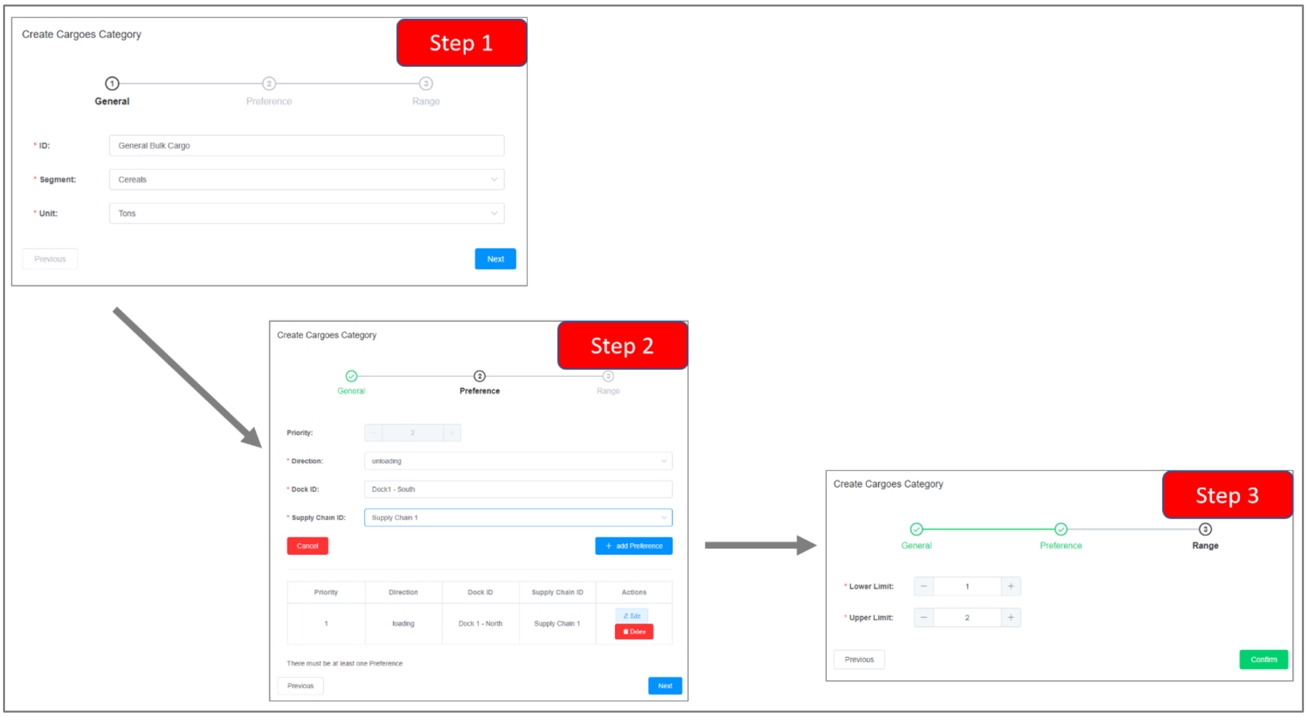
Description:

Figure 115: Rules List

Cargoes Category

The form for the creation of charge categories is divided into 3 steps:

- In the Step 1 general property information is filled in.
- In the step 2 preference properties are filled in.
- In the step 3 range property information is filled in.



Create Cargoes Category

① General — ② Preference — ③ Range

* ID: General Bulk Cargo

* Segment: Cereals

* Unit: Tons

Previous | Next

Create Cargoes Category

General — ② Preference — ③ Range

Priority: 2

* Direction: unloading

* Dock ID: Dock1 - South

* Supply Chain ID: Supply Chain 1

Cancel | + add Preference

Priority	Direction	Dock ID	Supply Chain ID	Actions
1	loading	Dock 1 - North	Supply Chain 1	Edit Delete

There must be at least one Preference

Previous | Next

Create Cargoes Category

General — Preference — ③ Range

* Lower Limit: 1

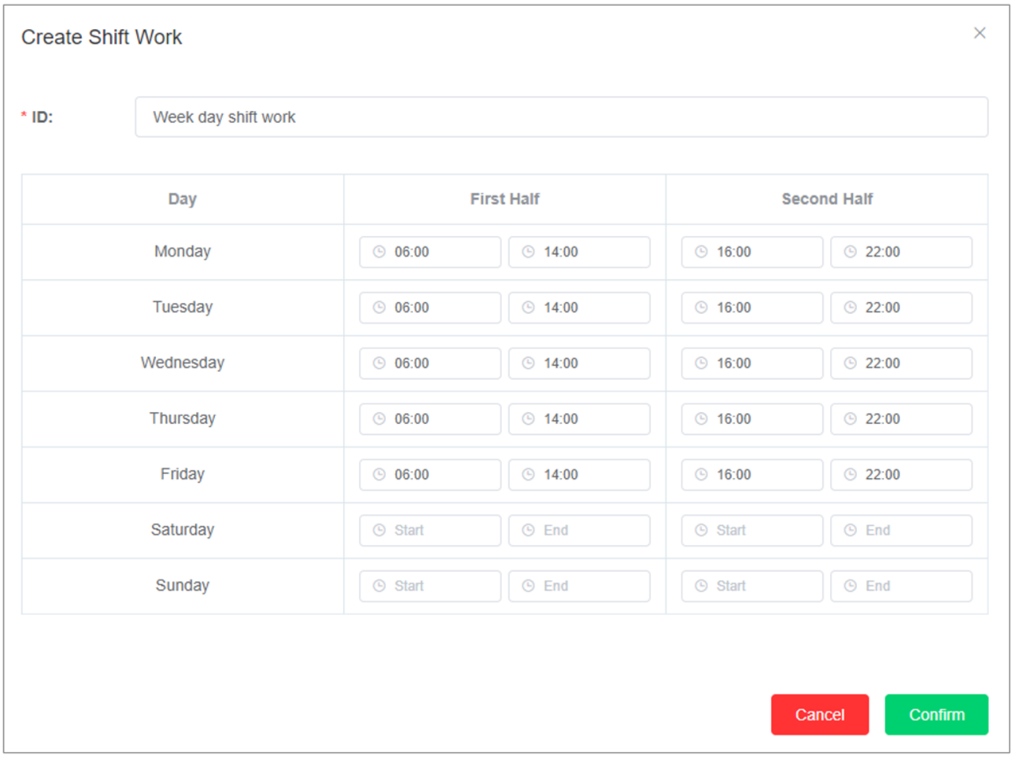
* Upper Limit: 2

Previous | Confirm

Figure 116: Create Cargoes Category

Shift work

The form for the creation of Shift works allows you to create different work schedules for a port that will later be assigned to different activities.



Create Shift Work

* ID: Week day shift work

Day	First Half		Second Half	
Monday	06:00	14:00	16:00	22:00
Tuesday	06:00	14:00	16:00	22:00
Wednesday	06:00	14:00	16:00	22:00
Thursday	06:00	14:00	16:00	22:00
Friday	06:00	14:00	16:00	22:00
Saturday	Start	End	Start	End
Sunday	Start	End	Start	End

Cancel | Confirm

Figure 117: Create Shift Work

Priorities

The priorities form allows you to define the priorities for attending to the different cargo categories.

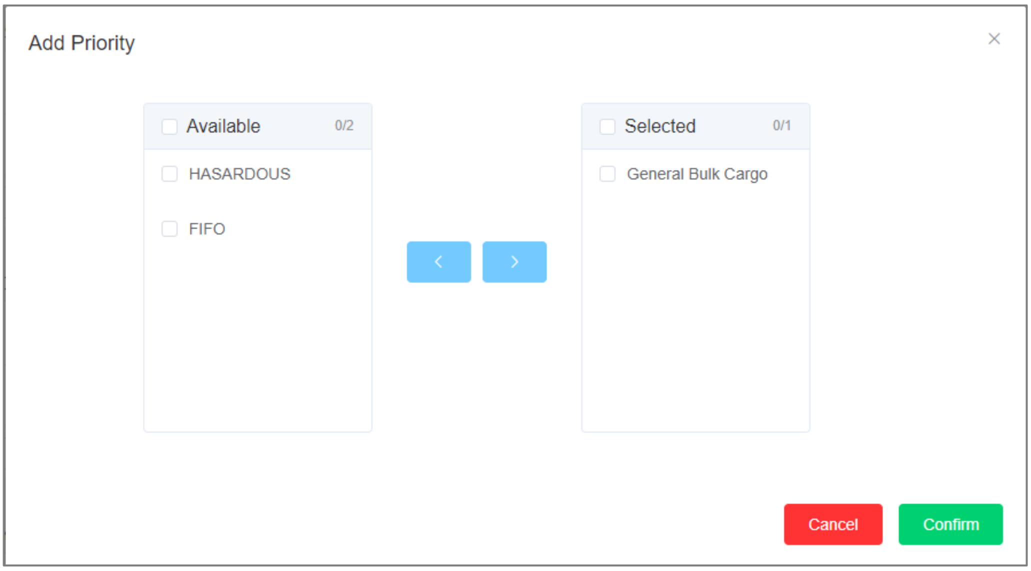
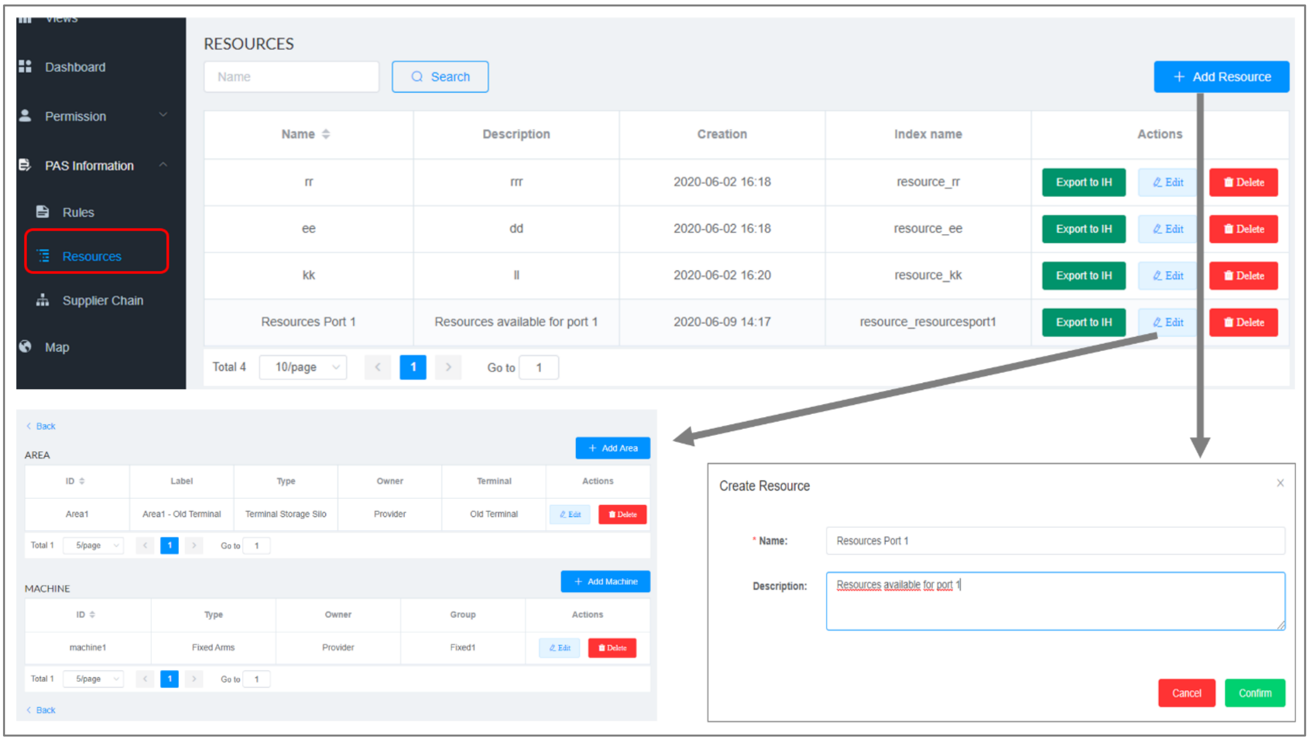


Figure 118: Add Priority

4.4.4.6.2. Resources

The resources functionality allows creating the information related with areas and machines. It is possible to create several resources. After finalising the definition of the resources, it is possible to publish the information (export to Information Hub) to be used by the PAS Model.

After creating a new resource (Add Resource button), the user can complete the content of the resource through the Edit button that opens a new page. From this page, it is possible to define areas and machines.



Name	Description	Creation	Index name	Actions
rr	rrr	2020-06-02 16:18	resource_rr	Export to IH Edit Delete
ee	dd	2020-06-02 16:18	resource_ee	Export to IH Edit Delete
kk	ll	2020-06-02 16:20	resource_kk	Export to IH Edit Delete
Resources Port 1	Resources available for port 1	2020-06-09 14:17	resource_resourcesport1	Export to IH Edit Delete

ID	Label	Type	Owner	Terminal	Actions
Area1	Area1 - Old Terminal	Terminal Storage Silo	Provider	Old Terminal	Edit Delete

ID	Type	Owner	Group	Actions
machine1	Fixed Arms	Provider	Fixed1	Edit Delete

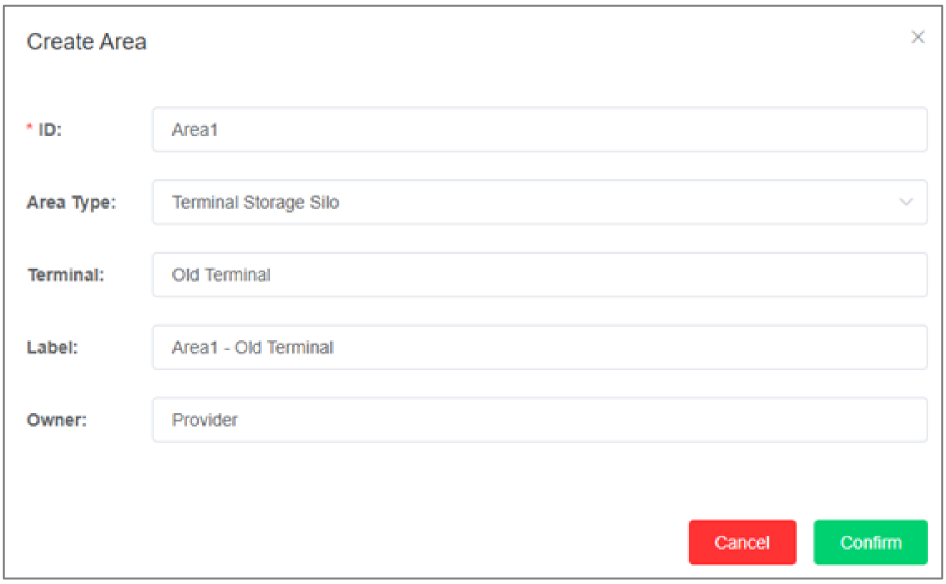
* Name: Resources Port 1

Description: Resources available for port 1

Figure 119: Resources List

Area

The area form allows you to define a new port Area where a where port activities take place.



Create Area

* ID: Area1

Area Type: Terminal Storage Silo

Terminal: Old Terminal

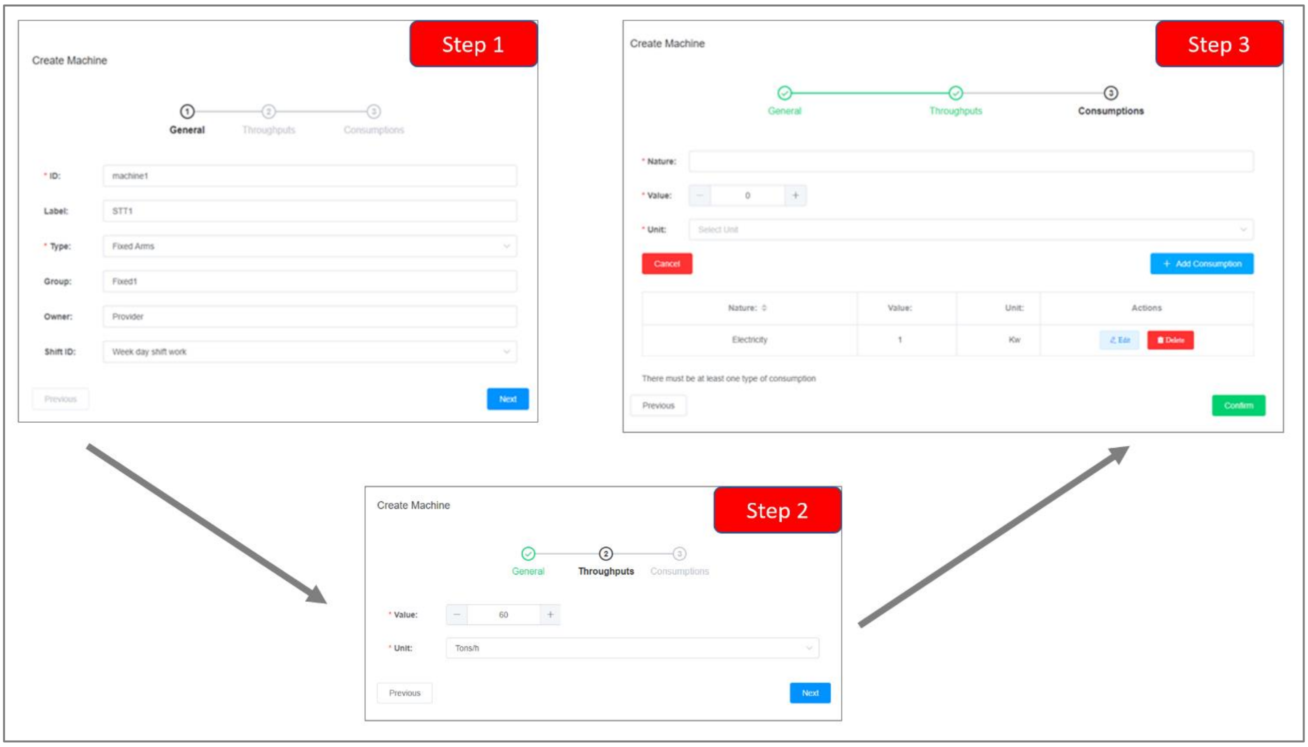
Label: Area1 - Old Terminal

Owner: Provider

Figure 120: Create Area

Machine

The Add Machine form allows you to create a new machine and it is divided into 3 steps. In the Step 1 general property information is filled in. In the step 2 throughput properties are filled in. In the step 3 range consumption information is filled in.



Step 1

Create Machine

① General ② Throughputs ③ Consumptions

* ID: machine1

Label: STT1

* Type: Fixed Arms

Group: Fixed1

Owner: Provider

Shift ID: Week day shift work

Step 2

Create Machine

✓ General ② Throughputs ③ Consumptions

* Value: 60

* Unit: Tons/h

Step 3

Create Machine

✓ General ✓ Throughputs ③ Consumptions

* Nature:

* Value: 0

* Unit: Select Unit

Nature:	Value:	Unit:	Actions
Electricity	1	Kw	Edit Delete

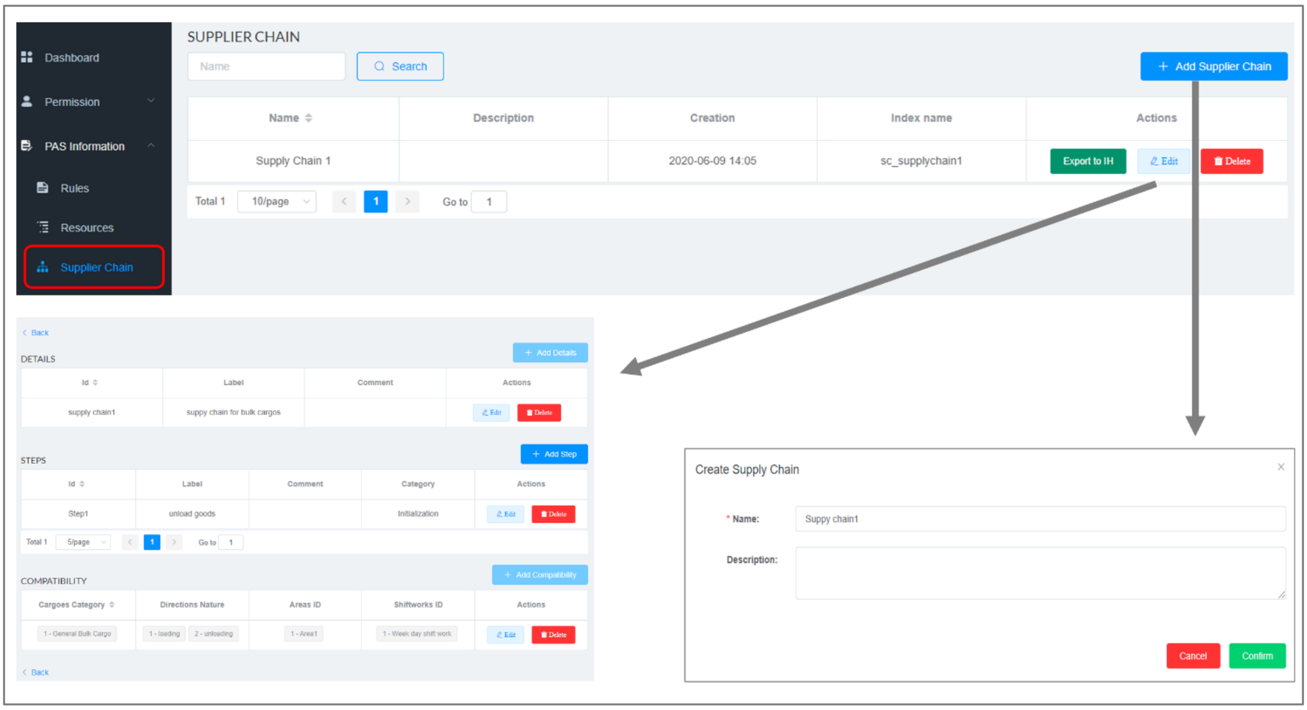
There must be at least one type of consumption

Figure 121: Create Machine

4.4.4.6.3. Supplier Chain

The supplier chain functionality allows creating the information related with a supply chain and its steps. It is possible to create several supply chains. After finalising the definition of the supply chains, it is possible to publish the information (export to Information Hub) to be used by the PAS Model.

After creating a new supply chain (Add Supplier Chain), the user can complete the content of the supply chain through the Edit button that opens a new page. From this page, it is possible to define the details, steps, and compatibilities of a supply chain.



SUPPLIER CHAIN

Name	Description	Creation	Index name	Actions
Supply Chain 1		2020-06-09 14:05	sc_supplychain1	Export to IH Edit Delete

Total 1 10/page 1 Go to 1

DETAILS

Id	Label	Comment	Actions
supply chain1	suppy chain for bulk cargos		Edit Delete

STEPS

Id	Label	Comment	Category	Actions
Step1	unload goods		Initialization	Edit Delete

Total 1 5/page 1 Go to 1

COMPATIBILITY

Cargoes Category	Directions Nature	Areas ID	Shiftworks ID	Actions
1 - General Bulk Cargo	1 - loading 2 - unloading	1 - Area1	1 - Week day shift work	Edit Delete

Create Supply Chain

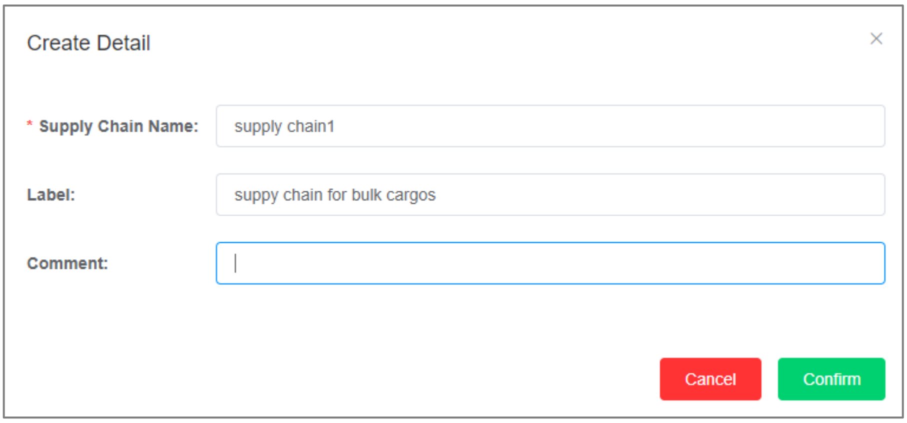
* Name: Suppy chain1

Description:

Figure 122: Supplier Chain List

Detail

The detail form allows you to define the detail properties of the supply chain.



Create Detail

* Supply Chain Name: supply chain1

Label: suppy chain for bulk cargos

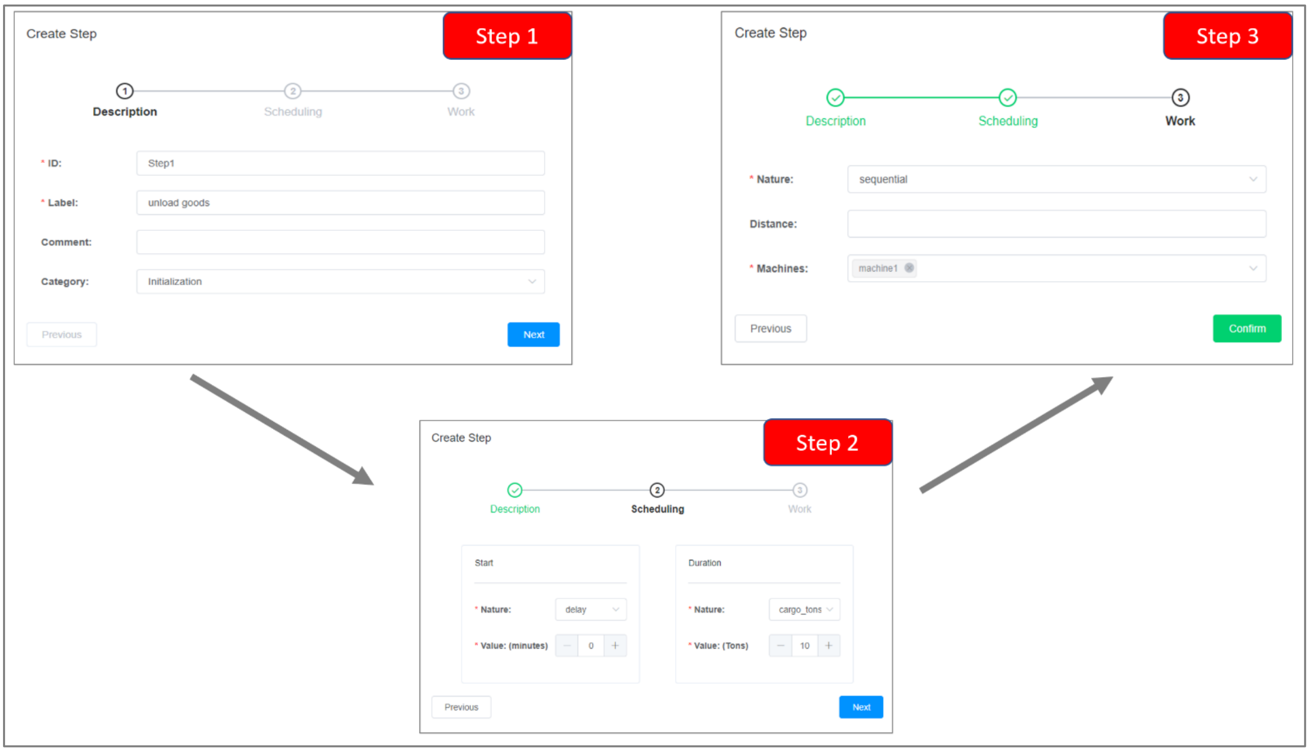
Comment:

Figure 123: Create Supply Chain Details

Steps

The Add Steps form allows you to create the different steps of the supply chain and it is divided into 3 steps:

- In the Step 1 general property information is filled in.
- In the step 2 scheduling properties are filled in.
- In the step 3 work information is filled in.



* ID:	Step1
* Label:	unload goods
Comment:	
Category:	Initialization

Start

* Nature:	delay
* Value: (minutes)	0

Duration

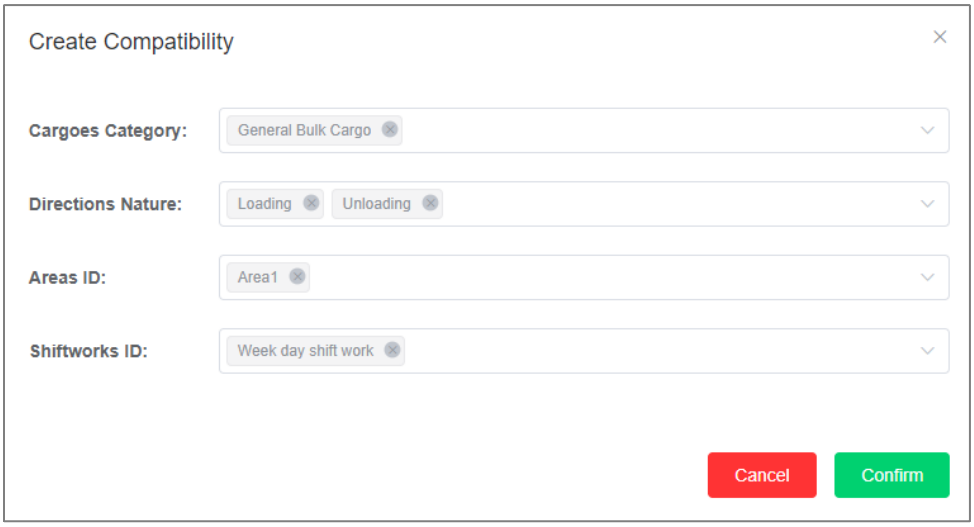
* Nature:	cargo_tons
* Value: (Tons)	10

* Nature:	sequential
Distance:	
* Machines:	machine1

Figure 124: Create Supply Chain Steps

Compatibility

The compatibility form allows you to define the elements (cargo categories, direction, areas and shift works) compatible with the supply chain.

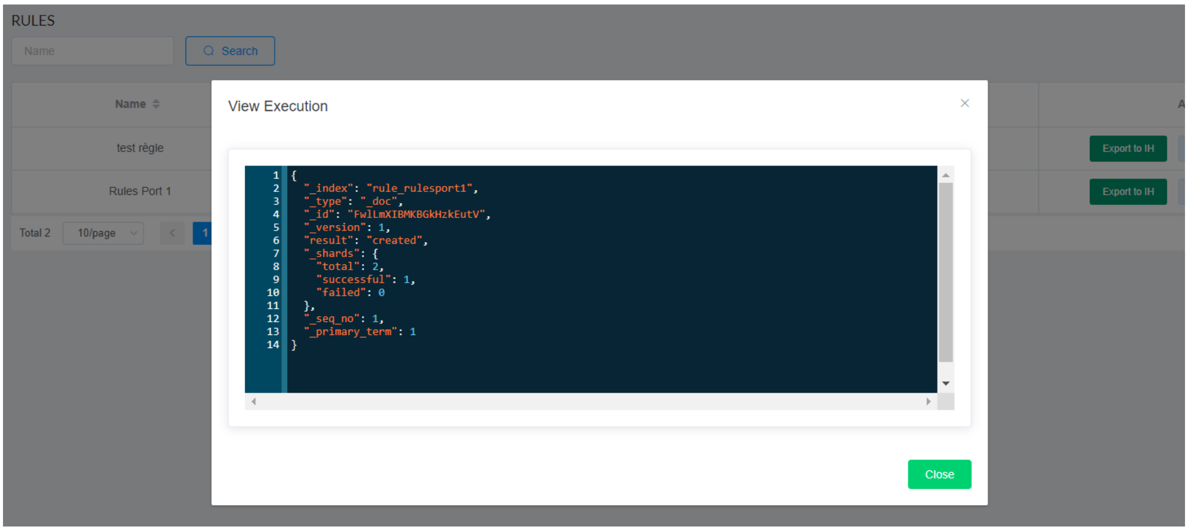


Cargoes Category:	General Bulk Cargo
Directions Nature:	Loading, Unloading
Areas ID:	Area1
Shiftworks ID:	Week day shift work

Figure 125: Create Supply Chain Compatibility

4.4.4.6.4. Publish – Export to Information Hub

After defining rules, resources, and supply chains, you should publish this information to be available for the PAS Model. To publish the information created you only need to press the “Export to IH” button available in list of rules, resources, and supply chains. After executing the action, you will see the execution result with the name of the index created. You will need to specify the name of the index when you run a new instance of the model.



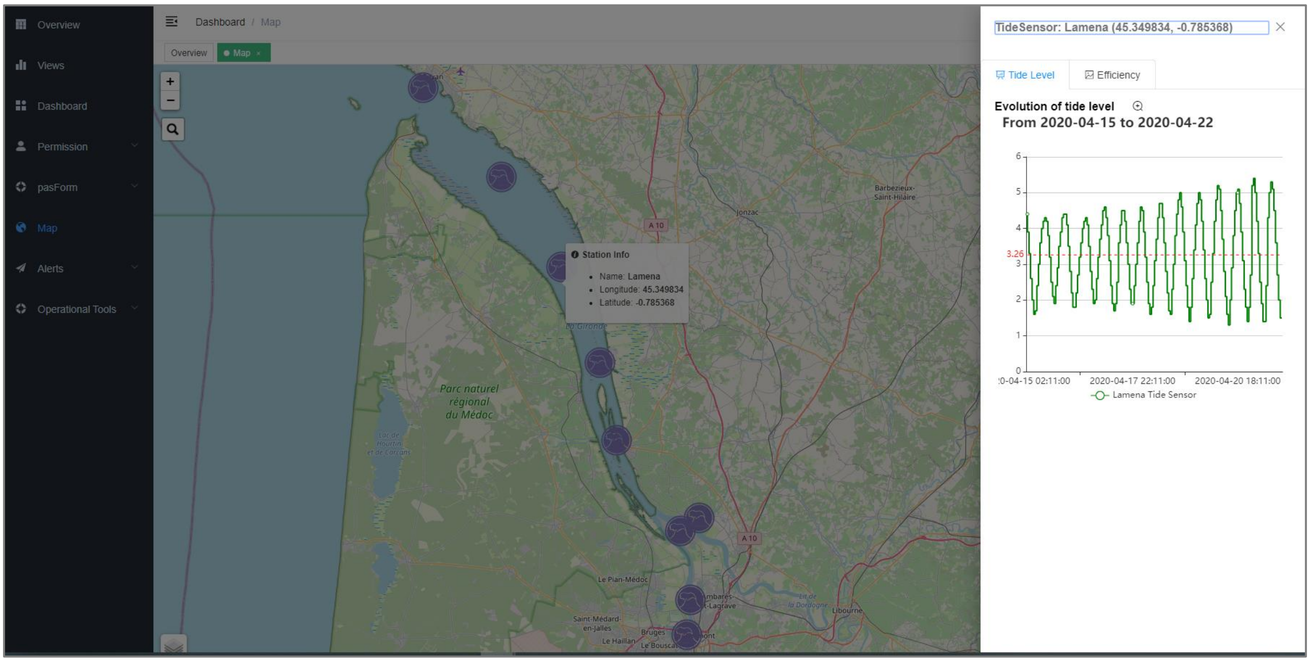
```
{
  "_index": "rule_rulesport1",
  "_type": "_doc",
  "_id": "FwlLmXIBMKBGkHzkEutV",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 1,
  "_primary_term": 1
}
```

Figure 126: Index created

Every time that a user updates any information related with the Port Activity scenario it is needed to use export the information (Export to IH) to have the new data available for the PAS model.

4.4.4.7. Map

The maps functionality allows the user to see the location of several devices or sensors. It is possible to filter by a specific type of device. In the figure, you can see the location in the map of the tide sensors that measure the tide level. If you select (click on the icon) a sensor, a new panel appears showing details of the sensor and the captured values.



Station Info

- Name: Lamena
- Longitude: 45.349834
- Latitude: -0.785368

TideSensor: Lamena (45.349834, -0.785368)

Evolution of tide level

From 2020-04-15 to 2020-04-22

Figure 127: Map view with tide sensors

4.4.4.8. Operational Tools

PIXEL is a flexible analytical platform; through the operational tools (OP Tools) you can install new analytical functionalities and perform as many analyses as you need.

In PIXEL the analytical functions have been grouped in 2 types. The functionalities based on models (1) and the functionalities based on predictive algorithms (2). Both can be executed on demand (3) or can be programmed to be executed periodically (4). After the executions (5), graphs (6) or KPIs (7) can be created to show the results of the model/algorithm executions.

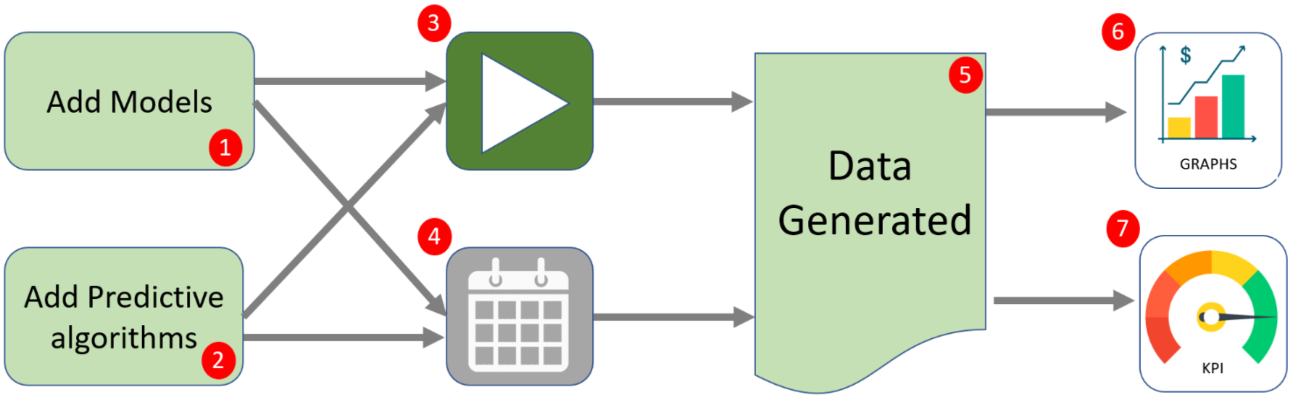
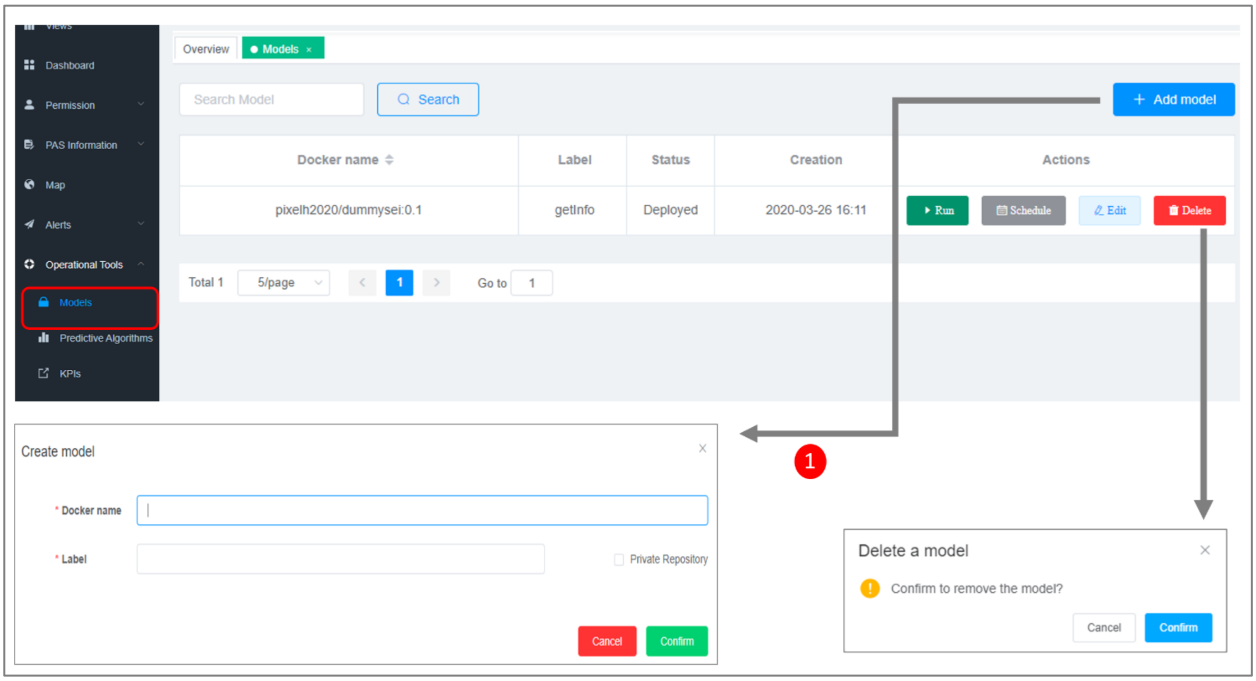


Figure 128: Models /algorithms execution and visualization

4.4.4.8.1. Models

The models functionality is part of the OP Tools and allows you to manage the models added/installed in the platform. The Add model button allow us to create/ add a new model into the PIXEL platform, the information needed is the label or name of the model and the Docker name. All the models are encapsulated into Docker components.



Docker name	Label	Status	Creation	Actions
pixelh2020/dummysei:0.1	getInfo	Deployed	2020-03-26 16:11	Run Schedule Edit Delete

Create model

* Docker name: |

* Label:

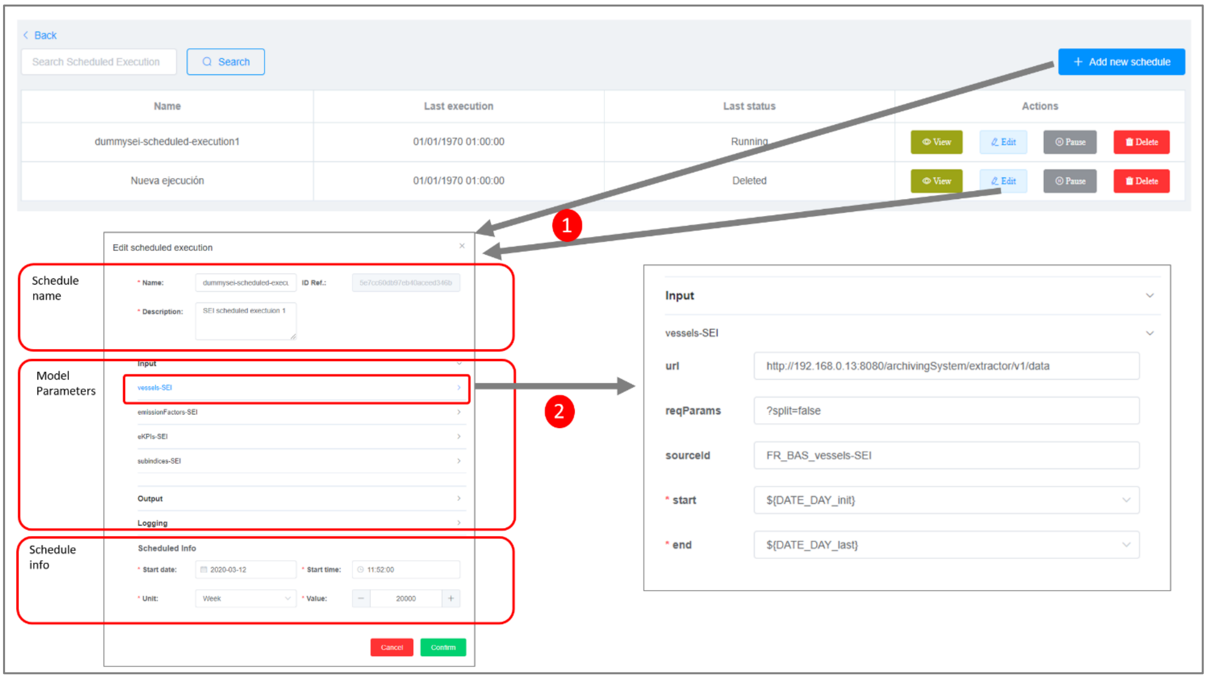
Delete a model

Confirm to remove the model?

Figure 129: Models Management

Schedule

The schedule functionality allows you to manage the scheduled executions of a model. To Add a new schedule (1) you have to provide the name of the schedule, the parameters of the model and the schedule information. Each model has different Parameters (2), in the example we have chosen the “vessels-SEI” property.



Name	Last execution	Last status	Actions
dummysei-scheduled-execution1	01/01/1970 01:00:00	Running	View Edit Pause Delete
Nueva ejecución	01/01/1970 01:00:00	Deleted	View Edit Pause Delete

Edit scheduled execution

* Name: dummysei-scheduled-execu ID Ref.: 5e7cc60db97eb40aceed346b

* Description: SEI scheduled exectuion 1

Input
- vessels-SEI
- emissionFactors-SEI
- eKPIs-SEI
- subindices-SEI

Output

Logging

Scheduled Info

* Start date: 2020-03-12 * Start time: 11:52:00

* Unit: Week * Value: 20000

Input

vessels-SEI

url: http://192.168.0.13:8080/archivingSystem/extractor/v1/data

reqParams: ?split=false

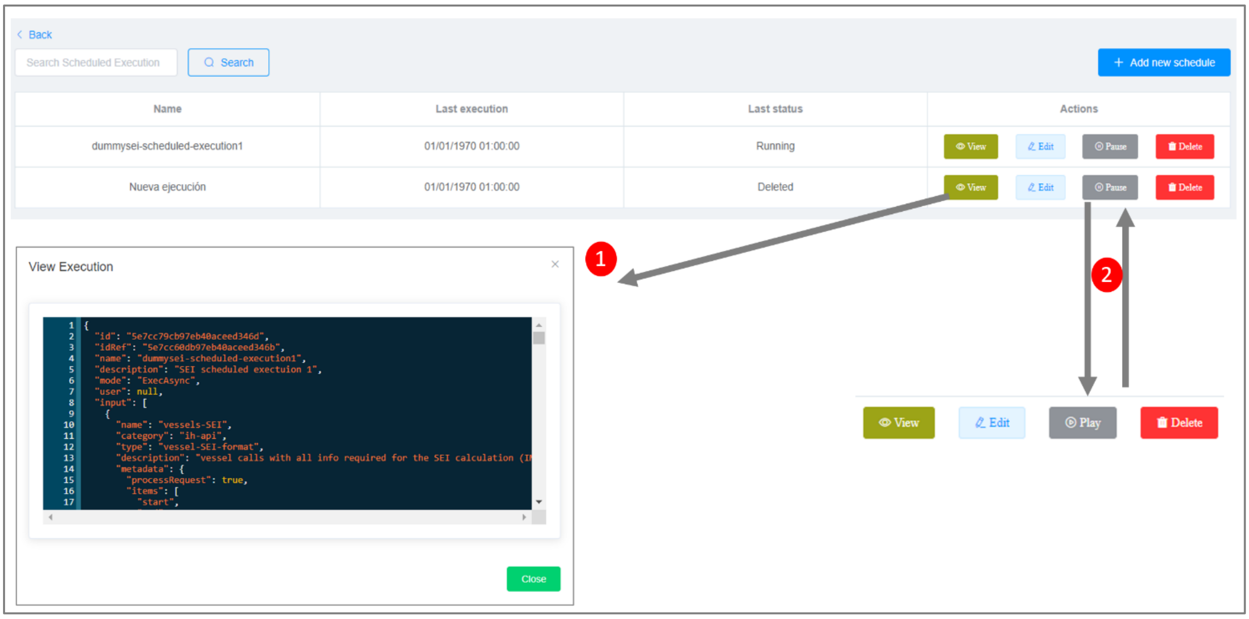
sourceId: FR_BAS_vessels-SEI

* start: ${DATE_DAY_init}

* end: ${DATE_DAY_last}

Figure 130: Model Execution Schedule

After the model is executed, it is possible to verify the execution results from the view button (1). At any moment, the user can pause / run (play) (2) the scheduled executions of a model.



Name	Last execution	Last status	Actions
dummysei-scheduled-execution1	01/01/1970 01:00:00	Running	View Edit Pause Delete
Nueva ejecución	01/01/1970 01:00:00	Deleted	View Edit Pause Delete

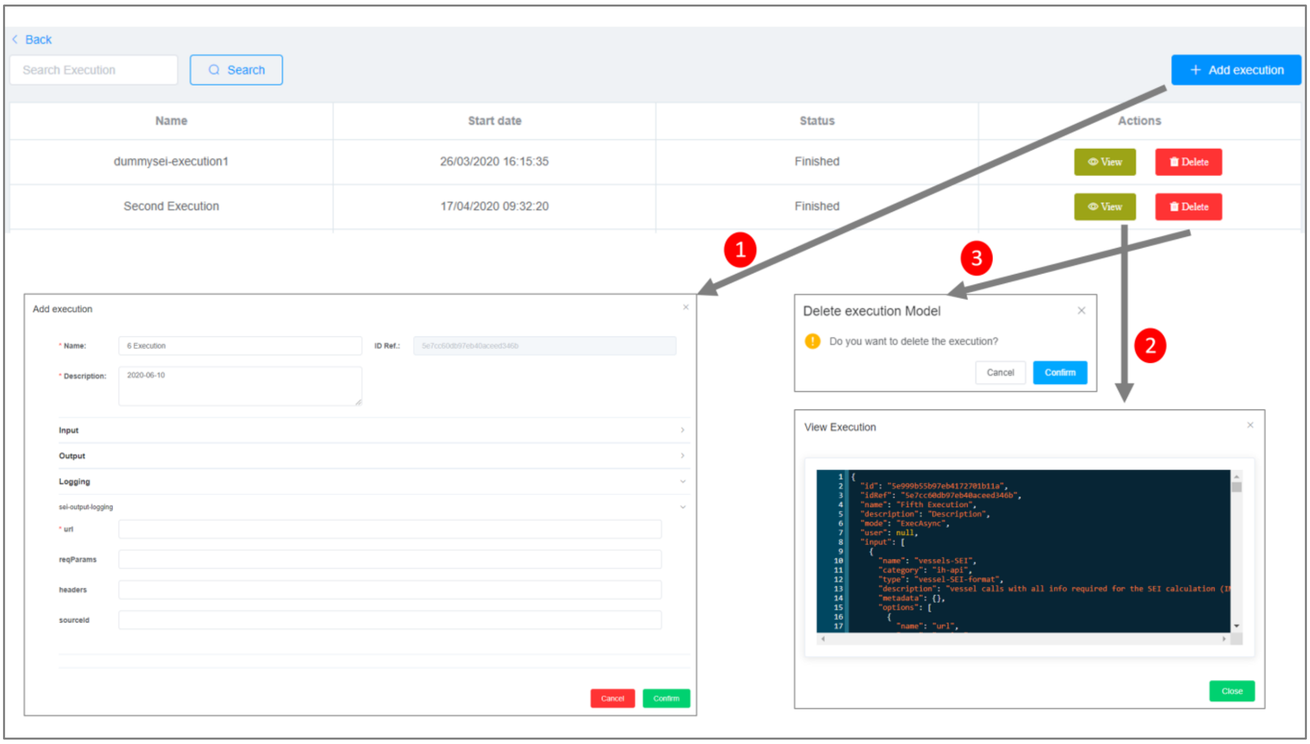
View Execution

```
{
  "id": "5e7cc79cb97eb40aceed346d",
  "idRef": "5e7cc60db97eb40aceed346b",
  "name": "dummysei-scheduled-execution1",
  "description": "SEI scheduled exectuion 1",
  "mode": "ExecAsync",
  "user": null,
  "input": [
    {
      "name": "vessels-SEI",
      "category": "ih-api",
      "type": "vessel-SEI-format",
      "description": "vessel calls with all info required for the SEI calculation (I
      "metadata": {
        "processRequest": true,
        "items": [
          "start",
```

Figure 131: Model Execution Result

Run Model

The Run functionality allows you to manage the on demand executions of a model. To Add a new Run (1) you have to provide the name of the run and the parameters of the model. Each model has different Parameters. As soon as you confirm the run, the model is executed. The status property of the table will show the “Finished” value when the execution ends. From the view (2) button, it is possible to view the result of the execution.



Name	Start date	Status	Actions
dummysei-execution1	26/03/2020 16:15:35	Finished	View Delete
Second Execution	17/04/2020 09:32:20	Finished	View Delete

* Name: 6 Execution

ID Ref.: 5e7cc60db97eb40aceed346b

* Description: 2020-06-10

* url

reqParams

headers

sourceId

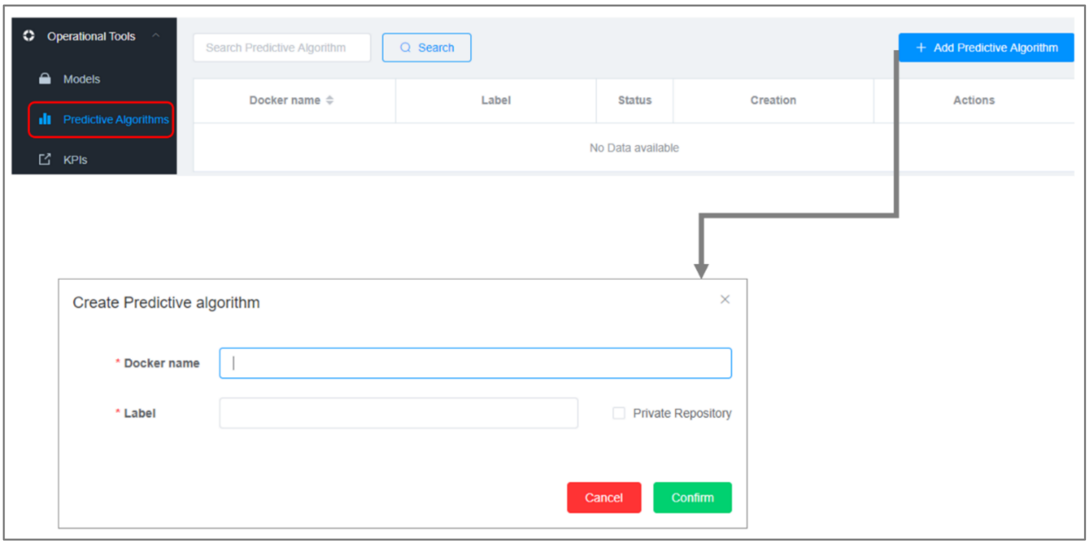
Do you want to delete the execution?

```
{
  "id": "5e999b55b97eb4172701b11a",
  "idRef": "5e7cc60db97eb40aceed346b",
  "name": "Fifth Execution",
  "description": "Description",
  "mode": "ExecAsync",
  "user": null,
  "input": [
    {
      "name": "vessels-SEI",
      "category": "ih-api",
      "type": "vessel-SEI-format",
      "description": "vessel calls with all info required for the SEI calculation (I
      "metadata": {},
      "options": [
        {
          "name": "url",
```

Figure 132: Model runs

4.4.4.8.2. Predictive Algorithms

The Predictive Algorithms functionality is part of the OP Tools and allows you to manage the algorithms added/installed in the platform. The functionality is equivalent to the model functionality, but in this case instead of running Artificial Intelligent Models, it runs predictive algorithms.


Operational Tools
Models
Predictive Algorithms
KPIs
Search Predictive Algorithm
Search
+ Add Predictive Algorithm
Create Predictive algorithm
Cancel
Confirm

Docker name	Label	Status	Creation	Actions
No Data available				

* Docker name

* Label

□ Private Repository

Figure 133: Predictive Algorithm management

4.4.4.8.3. KPIs

The KPI functionality allows you to create different KPI related with model execution. There are two types of KPIs, environmental KPIs and Operational KPI. You can define as many KPIs (1) as you need and show the trends of them (2).

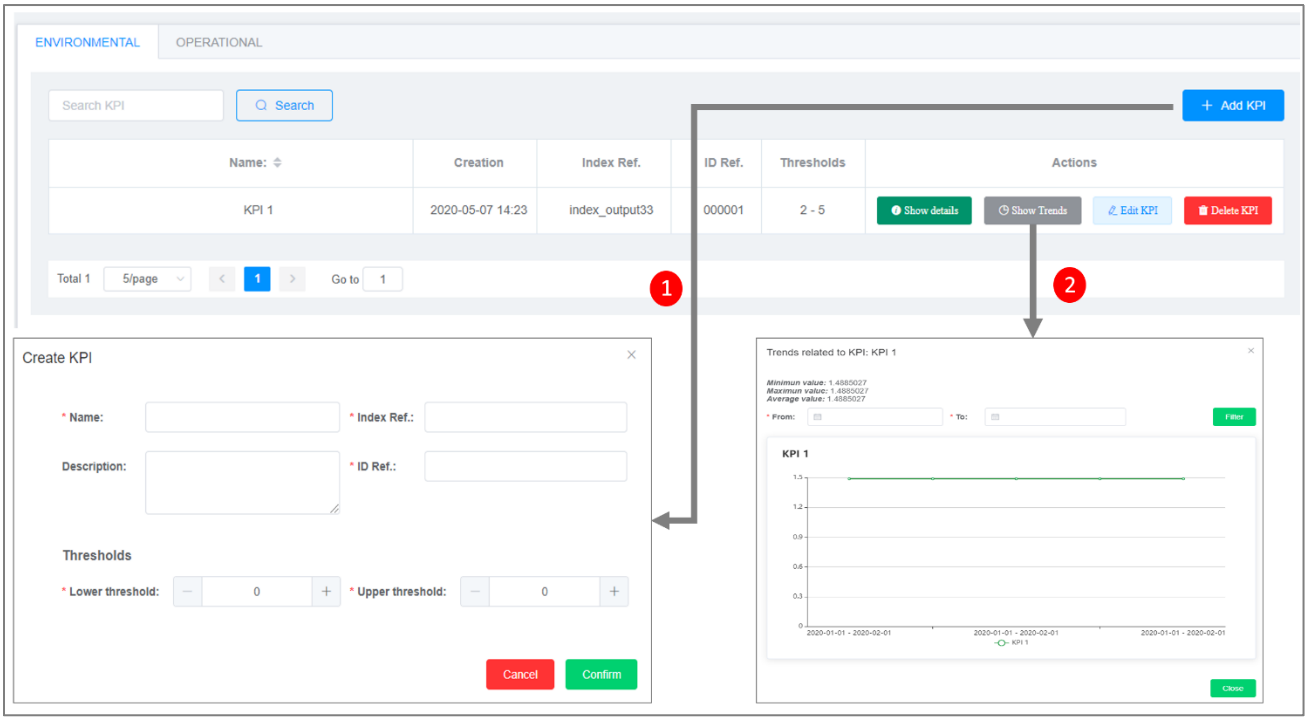


Figure 134: KPI Management

4.4.5. PIXEL Security

We use standard FIWARE Security Components and architecture to implement the Security Layer, so you can refer to the FIWARE official documentation to how to use those components:

- KeyRock,
- Wilma
- AuthZForce

KeyRock offer an API to manage it: http://<keyrock>:<port>/v1

Keyrock also offer a WEB UI to manage it more friendlily: http://<keyrock>:<port>

FIWARE also propose full tutorials to manage KeyRock and Wilma.

For the security we use 2 main features of FIWARE Security solutions:

- OAuth2: to authenticate users, agents and API Consumers.
- Authorizations control based on roles and permissions of Keyrock.

4.4.5.1. OAuth2 mechanism

The full documentation on OAuth2 features implemented with FIWARE is available on the official documentation.

On PIXEL to authenticate Data Source that will push data to a Daemon NGSI Agent, we use the grant=password mechanism:

Table 25: OAuth2 mechanism

Authorization Basic
We authenticate against an application, we need 2 information for that application • *Client id* • *Client secret*

Then we can combine them to create the *Authorization Basic* token:

- *base64(client_id:client_secret)*

Access Token Request

```
POST /oauth2/token HTTP/1.1
Host: id.<pilot>.port-pixel.eu
Authorization: Basic <authorization basic token>
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=<user email>&password=<user password>
```

user email and password have to be URL Encoded, the token is valid 1 hour. To refresh it you can authenticate again, or use the refresh token.

Permanent Access Token Request

For some client who can’t implement the OAuth2 mechanism, we can provide them a permanent token (it is still possible to revoke it)

```
POST /oauth2/token HTTP/1.1
Host: id.<pilot>.port-pixel.eu
Authorization: Basic <authorization basic token>
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=<user email>&password=<user password>
&scope=permanent
```

user email and password have to be URL Encoded

Access Token Response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"bearer",
    "expires_in":3600,
    "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
}
```

Refresh Token

```
POST /oauth2/token HTTP/1.1
Host: id.<pilot>.port-pixel.eu
Authorization: Basic <authorization basic token>
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=<refersh_token>
```

Token Verification

```
curl https://id.<pilot>.port-pixel.eu/user?access_token=<access_token>
    {
      "organizations": [
        {
```

```
        "id": "13e88767-7473-472d-9c33-110c5bed2a57",
        "name": "test_org",
        "description": "my org",
        "website": null,
        "roles": [
          {
            "id": "9c4e8db4-a56b-4731-bfc6-7dd8fb2fbea3",
            "name": "test"
          }
        ]
      }
    ],
    "displayName": "My User",
    "roles": [
      {
        "id": "9c4e8db4-a56b-4731-bfc6-7dd8fb2fbea3",
        "name": "test"
      }
    ],
    "app_id": "ff03921a-a772-4220-9854-e2d499ae474a",
    "isGravatarEnabled": false,
    "email": "myuser@test.com",
    "id": "myuser",
    "authorization_decision": "",
    "app_azf_domain": "",
    "username": "myuser"
  }
```

4.4.5.2. Authorizations

The second FIWARE security mechanism we use is the Authorizations solution. It relies on role/permission architecture.

The main object is the Application that regroups the authorization information. Then we can define permissions and roles. Roles are a set of permissions.

And finally we can trust user and organization (group of user) on the Application with assigned roles.

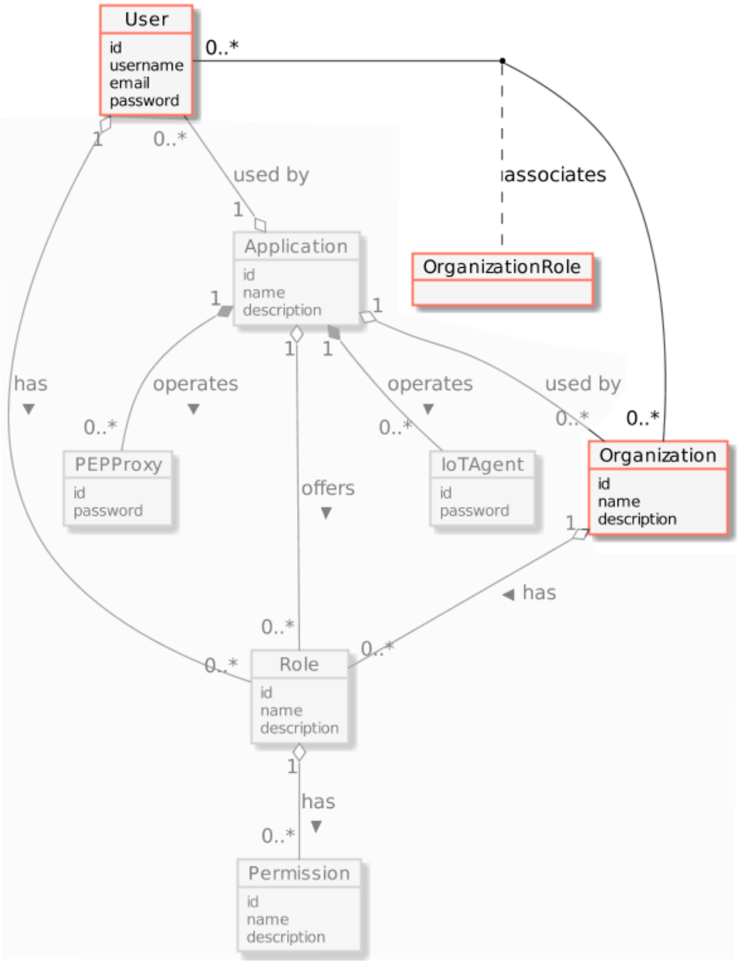


Figure 135: Diagram of Authorization mechanism

We use two kind of permission:

- Basic permission: they rely on the HTTP verbs and the resource path (could be a regex).
- XACML rules using AuthZForce.

On PIXEL we didn't interact directly with AuthzForce or KeyRock as PDP (Policy Decission Point). We use KeyRock as a PAP (Policy Administration Point) and we use Wilma as PEP (Policy Execution Point).

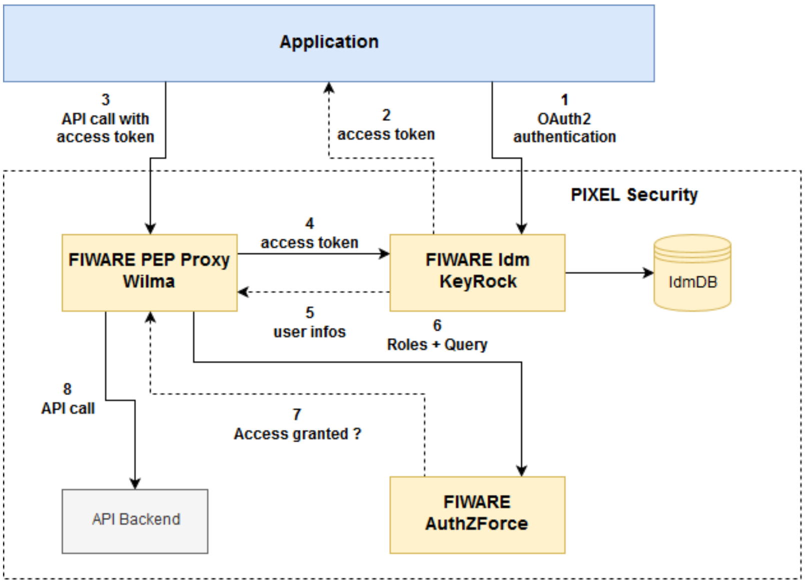
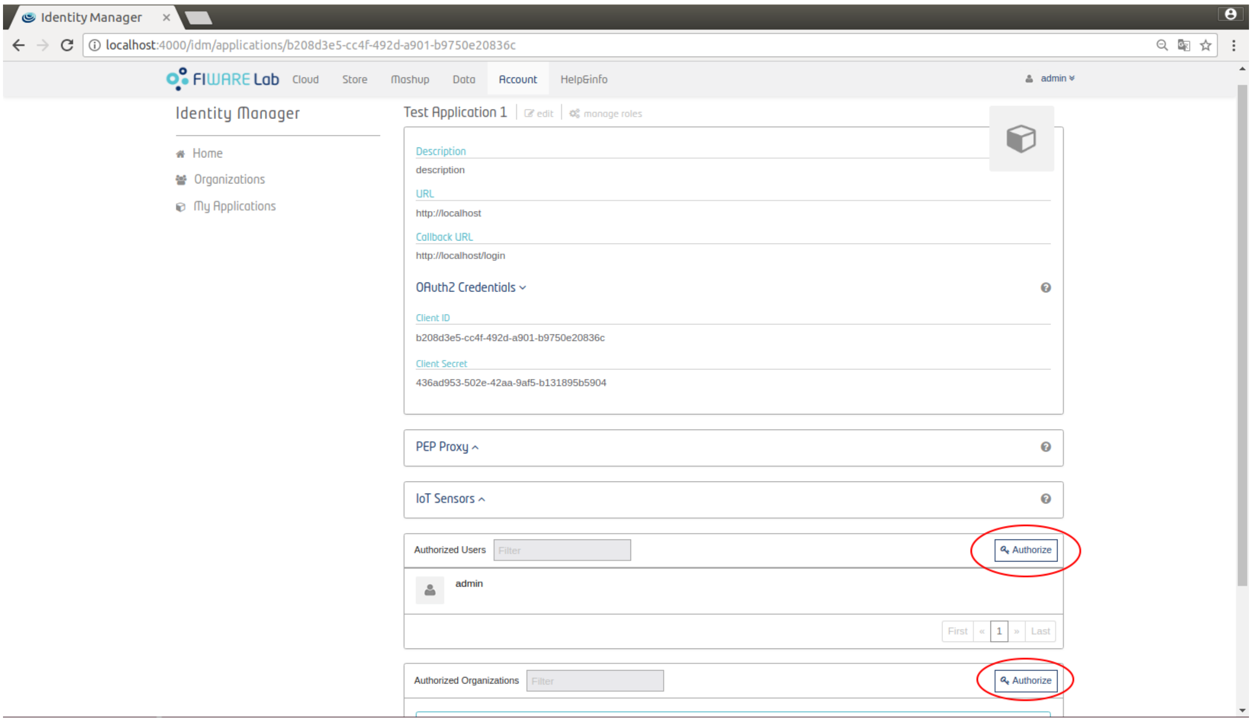


Figure 136: Interactions among different components of Pixel Security Layer

In order to manage the permissions on KeyRock, you have two solutions:

- KeyRock API : That offer all features to manage each objects
- KeyRock UI : That offer a friendly way to do the job



localhost:4000/idm/applications/b208d3e5-cc4f-492d-a901-b9750e20836c

Test Application 1

Description
description

URL
http://localhost

Callback URL
http://localhost/login

Client ID
b208d3e5-cc4f-492d-a901-b9750e20836c

Client Secret
436ad953-502e-42aa-9af5-b131895b5904

Authorized Users
admin

Authorized Organizations

Figure 137: Managing permissions on KeyRock

4.4.5.3. Data Tracking and Security

In order to track where the data come from and to implement security control for sensible data, all the information that enters through the DAL is tagged with two special attributes:

- Source
- DataProvider

The Source field is an URN created and managed by PIXEL to be a unique identifier of the DataSource that provide the data, for example: **urn:pixel:DataSource:dummies**.

The catalogue of DataSource is managed in the ORION Database. All the DataSource known in PIXEL platform or stored in object of type DataSource. We also stored the schema of each Data Model and SourceModelRelation that provide the information on which DataSource provide which DataModel.

- **DataSource format is**

```
{
    "id": "urn of the data source",
    "type": "DataSource",
    "name": {
        "type": "Text",
        "value": "the source name if it is not an urn"
    }
}
```

- **DataModel format is**

```
{
    "id": "type name as declared in the orion entity",
    "type": "DataModel",
    "schema": {
        "type": "StructuredValue",
        "value": an object containing the json schema
    },
    "schemaUrl": {
        "type": "string",
        "value": "an url to the schema"
    },
    "schemaEncoded": {
        "type": "STRING_URL_ENCODED",
        "value": "a text version URL encoded of the schema if it contains forbidden
characters"
    }
}
```

schemaUrl is mandatory, **schema** should be provide for compatibility with previous version, **schemaEncoded** has to be present only if schema contains forbidden chars.

- **SourceModelRelation format is:**

```
{
    "id": "urn of the relation datasource/dataModel",
    "type": "SourceModelRelation",
    "source": {
        "type": "Text",
        "value": "urn/id of the DataSource"
    },
    "model": {
        "type": "Text",
        "value": "Data Model provide by the DataSource"
    }
}
```

That information is mainly used by information Hub to decide the data to import from Orion.

5. Conclusions and Future work

5.1. Conclusion

This document represents the second version of the "PIXEL data acquisition, information hub and data representation report". It contains the final version of the PIXEL platform including technical documentation and software components. A lot of effort has been done during these months to adapt the preliminary version of the platform presented in the D6.3 to its architectural design.

This deliverable will be used by pilots to install the PIXEL platform and learn how to use it.

5.2. Future work

This document has been written based on 22 months of work (4 – 26). Although the current version of the platform is the final version, is almost sure that during the development of the different pilots some problems will arise, and it will be needed to proceed with some minor changes to this version of the platform. These changes and adjustments will be developed within the task T7.1 with the support of the partners involved in WP6.