MIDIH
MANUFACTURING·INDUSTRY
DIGITAL·INNOVATION·HUBS

Grant Agreement No. 767498
Innovation Action Project
H2020-FOF-12-2017

D4.3
MIDIH Open CPS/IOT Components

WP4 - Open Platform architecture, development, integration and testing

Version:	1.5
Due Date:	30/06/2018
Delivery Date:	23/07/2018
Type:	OTHER
Dissemination Level:	PU
Lead partner:	ATOS
Authors:	All Partners (See List of Contributors below)
Internal reviewers:	Sergio Gusmeroli (Engineering), Christian Schwede (FhG IML)

Disclaimer

MIDIH has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement no. 767498.

Version Control:

Version	Date	Author	Author's Organization	Changes
0.1	11/05/2018	ATOS	Jesús Benedicto	First version of the TOC
1.0	14/06/2018	ATOS	Jesús Benedicto	TOC refinement and Contributions to Chapters 2 and 3.
1.2	05/07/2018	ATOS	Jesús Benedicto	Factsheets Added First draft
1.3	09/07/2018	ENG	Sergio Gusmeroli	Internal review
1.4	18/07/2018	FHG	Christian Schwede	Internal review
1.5	23/07/2018	ENG EITD	Sergio Gusmeroli Susanne Kuehrer	Final editing Submission to EC

Annexes:

Nº	File Name	Title

Contributors:

Contributor	Partner
Jesús Benedicto Cirujeda	ATOS
Sergio Gusmeroli	ENG
Javier Hitado	ATOS
Angelo Marguglio	ENG
Felipe Roca	HOPU
Nadia Scandelli	CEFRIEL
Christian Schwede	FHG
Nenad Stojanovic	NISSATECH

Deliverable Title: MIDIH Open CPS/IOT Components

Deliverable Number D.4.3

Keywords: Smart Factory, Automation, Analytics, Data-at-rest, Data-in-motion, Open Digital Platforms, Digitalization, Industrial Internet of Things, Architecture, FIWARE4Industry, Siemens MindSphere, OPC UA

Executive summary

The D4.3 "MIDIH Open CPS/IOT Components v1" is a public document delivered in the context of WP4 and is accompanying the first prototypes of the MIDIH components generated in the context of MIDIH tasks WP4.2, WP4.3 and WP4.4.

- ***T4.2 Edge-oriented Local Clouds for Factory Automation***, responsible for providing the MIDIH Open Source reference architecture with a distributed cloud architecture (such as Fog Computing, Local Clouds or generically Edge-oriented). For this purpose, the usage of the Edge Computing Node as part of FogFlow project based on FIWARE integrating a subset of the Data in Motion reference implementation, has been the objective.
- ***T4.3 Brownfield Integration via Open APIs***, responsible to address brownfield integration and interoperability with proprietary solutions and standards. Main objective is implementing gateways between MIDIH platform and proprietary but open commercial solutions in the field of IOT. In this first prototype, interoperability between the two protocols NGSI (FIWARE) and OPC-UA (SIEMENS Mindsphere IoT platform) has been the objective of the development.
- ***T4.4 Industrial IoT and Analytics Platform***, responsible for developing the MIDIH platform for IoT and Analytics, having taken into account both data-in-motion and the data-at-rest. Main challenge is implementing this platform for analytics based on FIWARE and on worldwide known Open Source projects and Foundations, such as APACHE; and providing interoperability model between the two different technologies.

The output of these tasks delivers a collection of functional components implementing the MIDIH reference architecture (D4.1) for the Cross-Border Industrial Experiments in WP5 (and Open Calls winners). In the spirit of an Innovation Action, the developed open source components enhance, extend or integrate existing Open Source solutions available on the market or developed within previous European research projects such as BEinCPPS (Phase II I4MS).

Table of Contents

List of Figures

1 Introduction

This prototype includes results of development activities for edge-oriented factory automation, brownfield interoperability and Industrial Analytics, three tasks of WP4 (WP4.2, WP4.3 and WP4.4 respectively) responsible to develop open source components to implement the MIDIH Functional and Modular Reference Architecture in support to all the cross-border experiments in the CPS/IOT domain.

As detailed in the DoA, main objectives are:

- To integrate in the Reference Architecture, open source components aiming at developing local cloud edge-computing nodes for distributed Factory Automation.
- To integrate open APIs enabling the MIDIH Architecture to interoperate with existing legacy systems, in particular those implementing IOT and Analytics functionality.
- To integrate in the MIDIH Architecture additional Open Source components to finally implement an original MIDIH solution for Industrial CPS/IOT Data Analytics.

The architecture reflected in Figure 1, which is detailed in the deliverable “D4.1. Functional and Modular Architecture of MIDIH CPS/IOT System v1” shows which functional components of the architecture are covered by the different tasks in this first prototype, and where the developments have been focused.

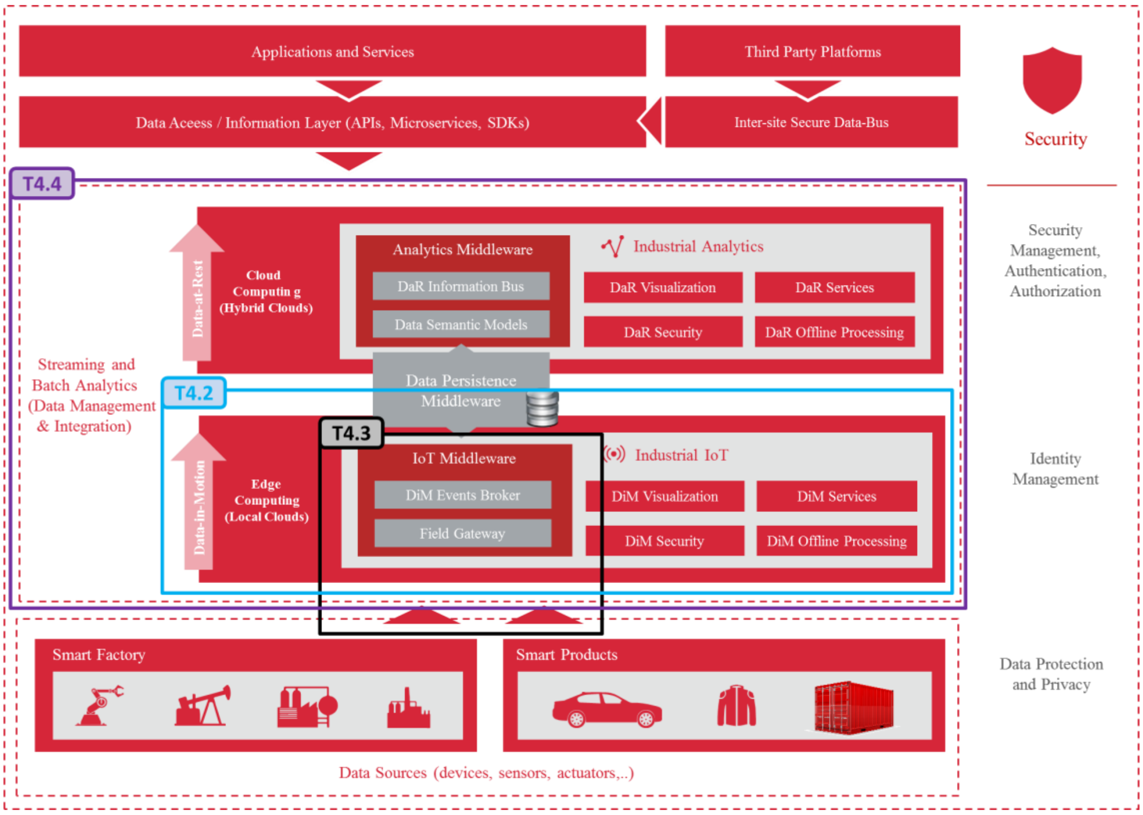

Applications and Services
Third Party Platforms
Data Aceess / Information Layer (APIs, Microservices, SDKs)
Inter-site Secure Data-Bus
Security
Security Management, Authentication, Authorization
Identity Management
Data Protection and Privacy
T4.4
T4.2
T4.3
Streaming and Batch Analytics (Data Management & Integration)
Data-at-Rest
Cloud Computin g (Hybrid Clouds)
Analytics Middleware
DaR Information Bus
Data Semantic Models
Industrial Analytics
DaR Visualization
DaR Services
DaR Security
DaR Offline Processing
Data Persistence Middleware
Data-in-Motion
Edge Computing (Local Clouds)
IoT Middleware
DiM Events Broker
Field Gateway
Industrial IoT
DiM Visualization
DiM Services
DiM Security
DiM Offline Processing
Smart Factory
Smart Products
Data Sources (devices, sensors, actuators,..)

Figure 1 – MIDIH Reference Architecture

One chapter is devoted to each of the tasks and the development made within them, including the same organization:

- general information of the Composite Outcome(s)
- a short description of each Component that made up the Composite Outcome(s)
- the background and foreground for these components

In addition, for each specific component that made up the Composite Outcome(s), a factsheet is provided in the Annexes following this schema:

- general information of the component
- functional description and architecture
- manuals both technicians (installation, configuration) and end-users (usage)

2 Edge-oriented Local Clouds for Factory Automation (T4.2)

Main objectives of this task in the first iteration are:

- The implementation of the MIDIH Edge Computing Node as part of FogFlow[1] project based on FIWARE (https://www.fiware.org/), in order to have a highly distributed cloud architecture (such as Fog Computing, Local Clouds or generically Edge-oriented).
- The deployment of a subset of the **Data in Motion** reference implementation into the Edge Computing Node.

The concept of Edge Computing provides a new way of processing data produced by network nodes. This architecture is based on the idea of "processing at the edge", that is to say, to bring the media, technology, intelligence and services closer to the data source where we will be able to process the data immediately.

It is obvious how quickly data is processed with this architecture because we place processing capability right next to the data, assuming minimal data transfer time, as opposed to cloud processing.

On the other hand, another highlight of this architecture is the **security/privacy** it offers in the field device, by bringing processing capabilities closer to the data producers. A more secure environment is achieved since the data is processed and stored in the processing node, increasing the difficulty to intercept raw data by third party entities. Additionally, thanks to the encapsulation obtained, it is possible to protect personal data such as the MAC of a smartphone. The aim of this practice is to respect and comply with the new data protection regulation (GDPR), which is a solution to be considered.

Regarding communication to other architecture components, these are encrypted with HTTPS communication. From the architecture deployment process to the data exchange between components such as the data exchanges between IoT Brokers and IoT Discovery, all these communications are protected against the interception of third party's entities.

The concept of Data in Motion takes on great importance in Edge Computing, Data in Motion is all data that is continuously sent; once it has been received and stored, it becomes Data at Rest.

[1] https://fogflow.readthedocs.io/en/latest/

In Edge Computing data is collected and processed at the node but then sent to the cloud, hence the value of the Data in Motion concept.

Another important feature is the fact of "filtering" the data before sending it to the cloud: with this architecture, the data can be pre-processed on the edge node to summarize the information or carry out a selection of the most relevant data. This results in less data being sent to the cloud and avoids overloads.

Continuing with the advantages provided by the Edge Computing, since the Edge Node is a processing point, depending on the use case, it could be capable to process data and retain data during general network failures or communication cuts. In fact, when the data is processed in the cloud, an Internet connection is necessary, but in the case processing takes place in the edge, it may not be necessary to be connected to a network continuously. This means that the processing of important data obtained on industrial spaces can continue even if something happens with the network connection.

Although also cheaper hardware could be used, the D4.3 edge experimentation will take place with the so-called Smart Spot in conjunction with a processing node which provides high processing and storage capabilities. The processing node currently used is a MinnowBoard with 2GB of RAM, an Intel® Quad-Core processor Atom™ E3845 and 240GB SSD. The processing node could be replaced by any other hardware such as a BeagleBone Black (BBB), Raspberry PI, … The following picture (Figure 2) illustrates the described hardware where the black board is the Smart Spot core board and the blue board is the Minnowboard, the SSD hard disk is placed under the Smart Spot core board. The illustrated devices also include an extra board for 4G communications that will be included only if the use case requires it.

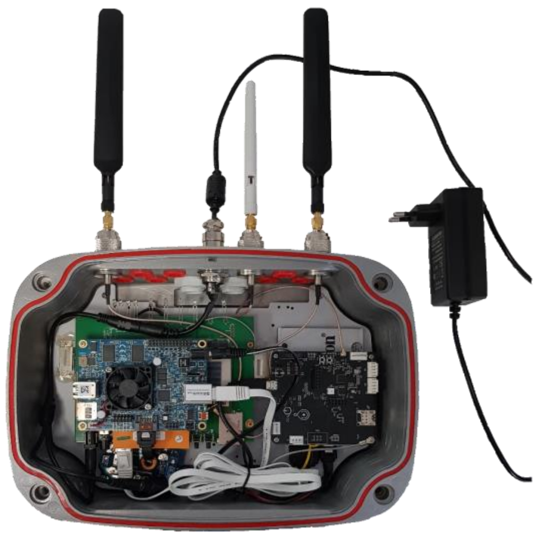


Figure 2 – SmartSpot device

Specifically, the processing node specifications are:

Hardware component	Features
MinnowBoard Turbot Quad-Core	• Quad-Core Intel® Atom™ E3845 • 4/4 Cores

	• 2MB On-chip cache • 2Gb DDR3L SDRAM Size
Storage	240 Gb SSD
OS	Open Source Operating System Debian 8.10
I/O ports	• 1x USB2.0 • 1x USB3.0 • microHDMI port • 1 x 1Gb Ethernet RJ45 • UART pins

2.1 Architecture of the FogFlow

FogFlow (https://fogflow.readthedocs.io) is a framework which allows to easily deploy Edge Computing, it was born out of the necessity to process the large amount of data that is produced today by connected devices (by 2020 the number of 30 billions of such devices is estimated). This framework has been created and developed by NEC company.

FogFlow framework architecture is composed by three layers, shown in Figure 3; the first one is in charge of managing the services provided by the framework as a docker; the second one is in charge of the communication between the different contexts, and the last one is in charge of data processing. The tasks that directly interact with sensors or IoT Devices can be found in this layer. These layers are detailed on section 2.2.

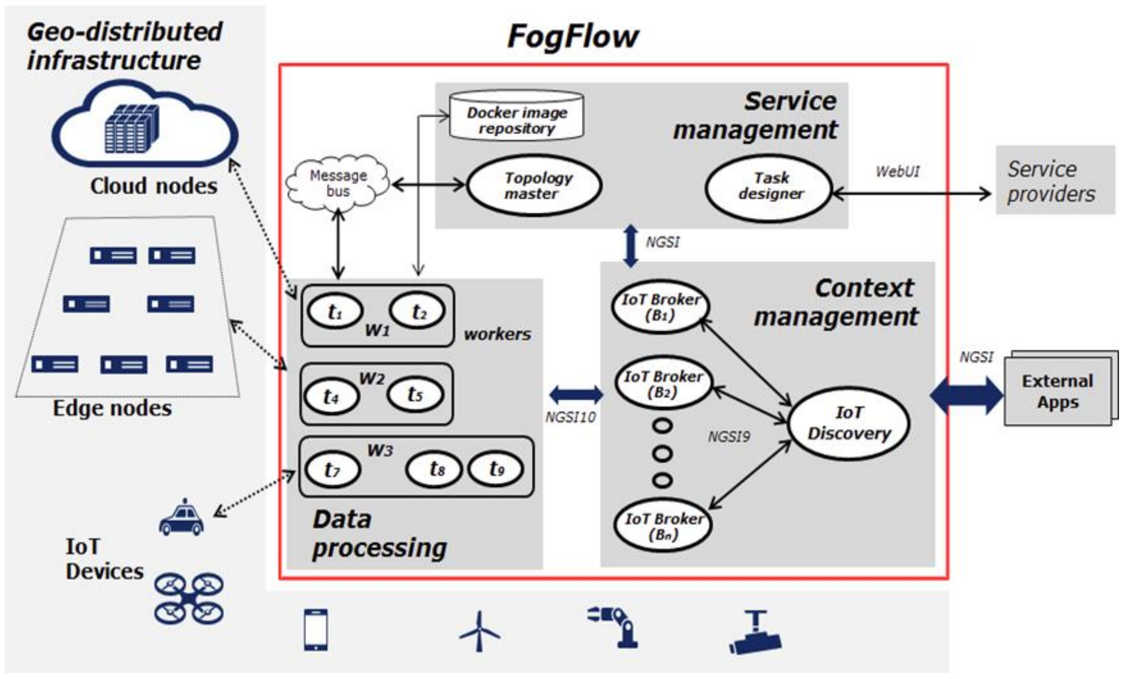


Figure 3 – General view of the Fog Flow framework

2.2 FogFlow Framework

This section presents the functionality of the three layers that composes the FogFlow architecture in detail.

- Service management: collects service requirements, processes them and creates a specific execution plan that is executed on the cloud and at the fog nodes. The main components of this layer are the Task Designer and the Topology Master that will be explained in the "Edge-oriented Local Clouds for Factory Automation – FogFlow" Annex A.
- Context management: manages all the information from different contexts, making it accessible and recognizable through fully flexible queries and subscription interfaces. The main components of this layer are the IoT Discovery and IoT Broker.
- Data processing: In this layer the data processing tasks are executed, dependant on the devices that are transmitting the information and collected through the interfaces provided by the context management layer.

2.3 MIDIH Developments

During the project, a first implementation of an IoT protocol related bridge will be designed and implemented in order to support the integration of external devices with communication protocols such as LwM2M[2] (LightWeight Machine to Machine) into the local IoT Broker of the Edge node.

After the integration of data from IoT devices into FogFlow, an interesting function to be implemented is a microCEP (Complex Event Processing), purely based on Fog Flow software or via external components such as the CEP FIWARE enabler in order to locally process the data streams allowing to act in synch with the input stream. This allows to explore possible cases of data stream management such as the interaction of machine vibrations/impacts, exposure to contaminants, ...

[2] https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/

3 Brownfield Integration via Open APIs (T4.3)

Main objective of this task in this first prototype has been to define and implement a more structured approach to interconnect FIWARE with a commercial OPC-UA based platform such as Siemens Mindsphere. For this, interoperability between the two protocols: NGSI and OPC-UA has been developed.

The solution implemented is based on several developments described in the following subchapters.

3.1 Architecture of the OPC-UA Connector for Siemens MindSphere

This section is devoted to describing the OPC-UA Connector for Siemens MindSphere. This Connector is an Open Source component intended to enable capturing data from IoT sensors and OPC UA devices on the shop floor, transforming them into context entities supported by FIWARE to finally be injected into the Siemens Mindsphere IoT platform, allowing the integration of data between the MIDIH platform and the Mindsphere IoT platform.

The following picture (Figure 4) reflects the functional architecture and the components that made up the overall solution.

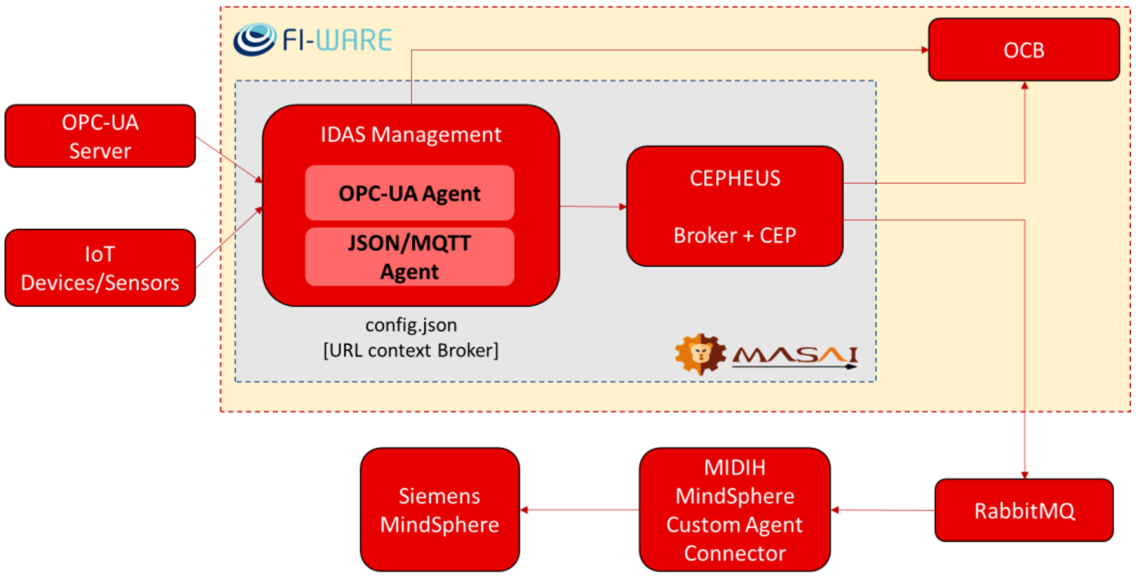


Figure 4 – MIDIH OPC-UA Connector for Siemens MindSphere Architecture

Several developments have been carried out in order to achieve this solution:

- FIWARE OPC UA Agent: Main objective has been to develop an IoT Agent, interfacing between OPC UA and FIWARE NGSI protocols. For this purpose, a component to connect the IoT Devices which implements the OPC-UA standard connection technology, with a NGSI Publish/Subscribe Context Broker such as the FIWARE Orion Context Broker (OCB) has been developed.

- MASAI: The objective has been to improve the functionalities provided by this open source component in order to integrate the FIWARE OPC UA Agent and extend communication mechanisms to interact with additional message brokers. This allow to send context entities information to an AMQP broker (RabbitMQ) and later retrieve this information through a custom development that acts as an intermediary between the MIDIH platform and MindSphere.
- MIDIH MindSphere Custom Agent Connector: This is a custom development integrating a specific library provided by Siemens that allows to connect with the Mindsphere platform in a secure and controlled way.

The MASAI and the MIDIH MindSphere Custom Agent Connector are covered together in the subchapter "3.3 – MIDIH MindSphere Custom Agent".

3.2 FIWARE OPC UA Agent

This section introduces the **FIWARE OPC UA Agent**, an Open Source component intended to enable capturing data from OPC UA devices on the shop floor and provide them to the upper levels of a FIWARE-based system. Therefore, the focus of this component is on the communication from field devices implementing an OPC UA server to the FIWARE environment of the MIDIH Open Platform, allowing the communication to the Orion Context Broker, the gateway to FIWARE (OCB). For the first version of the FIWARE OPA UC Agent, it is assumed that the underlying devices expose an OPC UA service API through an *OPC UA TCP Binding* or uses specific libraries to communicate with the middleware. For convenience, in the picture (Figure 4) we indicated these pre-existing components with the *OPC UA Server* box

3.2.1 Background components

The FIWARE OPC UA Agent is based on the reference implementation of the FIWARE Backend Device Management Generic Enabler, **IDAS**[3], delivered by Telefonica I+D. IDAS provides a collection of *Agents* – i.e., independent processes that are typically executed in the proximity of IoT devices and that are responsible for bridging a specific IoT protocol to the NGSI standard (e.g. the IDAS distribution includes off-the-shelf Agents for LwM2M and MQTT). To this end, IDAS links the NGSI southbound API of the FIWARE Orion Context Broker to the northbound API of the IoT application stack, by providing a software library (the *IoT Agent Li*b)) for developing custom Agents that may extend the bridging capabilities of IDAS to other protocols.

On the south bound interface, another exploited background asset is the **OPC-UA JavaScript Library**: among all the components available in the Node.js platform, it is the most interesting, as it offers both a client component (used in the OPC-UA JavaScript Wrapper) and a server one, that can be used to send data from the IoT devices in the shop floor to the OPC-UA JavaScript Wrapper.

[3] https://catalogue-server.fiware.org/enablers/backend-device-management-idas

3.2.2 Foreground components

The **FIWARE OPC UA Agent** component is a Node.js service which can efficiently publish any OPC UA data model exposed by automation devices (*address space* in OPC UA jargon) as a NGSI *context* in the FIWARE Context Broker, keeping both in sync: updates on one side are automatically reflected on the other. The mapping between the two domains are easily defined by means of a configuration file, so that no software coding is required to integrate different OPC UA devices.

3.3 MIDIH MindSphere Custom Agent

The **MIDIH MindSphere Custom Agent** is an Open Source component based on FIWARE and intended to enable capturing data from IoT and OPC UA devices on the shop floor, allowing the integration of this data inside the Siemens Mindsphere IoT platform. This development provides interoperability of the MIDIH platform with third-party tools and solutions. The complete solution developed covers data capture, processing and subsequent transformation of these data, before integrating them into the Mindsphere platform.

3.3.1 Background components

The MIDIH MindSphere Custom Agent is based on MASAI, which is an open source component based on FIWARE Generic Enablers that provides an infrastructure to support continuous data collection from IoT based resources and normally is installed at companies' premises level. Its mission is to gather the data from the shop floor IoT devices and provide it to any Data Collector Platform or Framework.

FIWARE exposes to developers Data Context elements or entities (JSON objects) with attributes and metadata with a uniform REST API, which allows MASAI to be an open source software stack aiming to bring Data-level interoperability to the complexity of standards and protocols in the world of IoT today. MASAI is able to expose all IoT devices information and commands using the Data Context API (NGSI9/10 - Next Generation Service Interface) for communicating devices with NGSI brokers or any other piece that uses the NGSI protocol. Therefore, MASAI is dedicated for the collection of data from the shop floor acting as a middleware between the IoT data producers in the shop floor and the data consumers, which can be external services.

In addition, the open cloud platform from Siemens, MindSphere, is a powerful IoT operating system with data analytics and connectivity capabilities, tools for developers, applications, and services. It helps to evaluate and utilize the manufacturer's data and to gain breakthrough insights. Driving the performance and optimization of their assets.

MindSphere is a commercial platform which requires the payment of a license for its use.

3.3.2 Foreground components

The MIDIH MindSphere Custom Agent comprises several developments that will allow the MIDIH platform interoperate with the Siemens MindSphere. For this, it provides the capabilities of capturing data from different IoT and OPC UA devices, transforming them into context entities supported by FIWARE and finally inject them for further analysis into the MindSphere through a specific connector. For this purpose, the FIWARE OPC UA Agent, also developed in T4.3, has been integrated in MASAI allowing it to capture data from OPC-UA devices, in addition to the IoT devices already covered. It has also been developed a custom Mindsphere connector, the MIDIH MindSphere Custom Agent Connector, responsible for capturing data provided by a message broker (RabbitMQ) and integrating them within the Mindsphere platform.

4 Industrial IoT and Analytics Platform (T4.4)

MIDIH T4.4 responsible for the Data Analytics Platform (Figure 5), has developed two basic lanes for Industrial IoT and Industrial Analytics, one FIWARE-based and the other based on worldwide known Open Source projects and Foundations, such as APACHE. The former represents an EU-funded reference implementation, fully compatible with any Context Information data source; the latter aims at achieving interoperability with existing worldwide projects using other Open Source middleware. At the same time, it has to allow and facilitate as main challenge the interoperability model and implementation between the two Lanes.

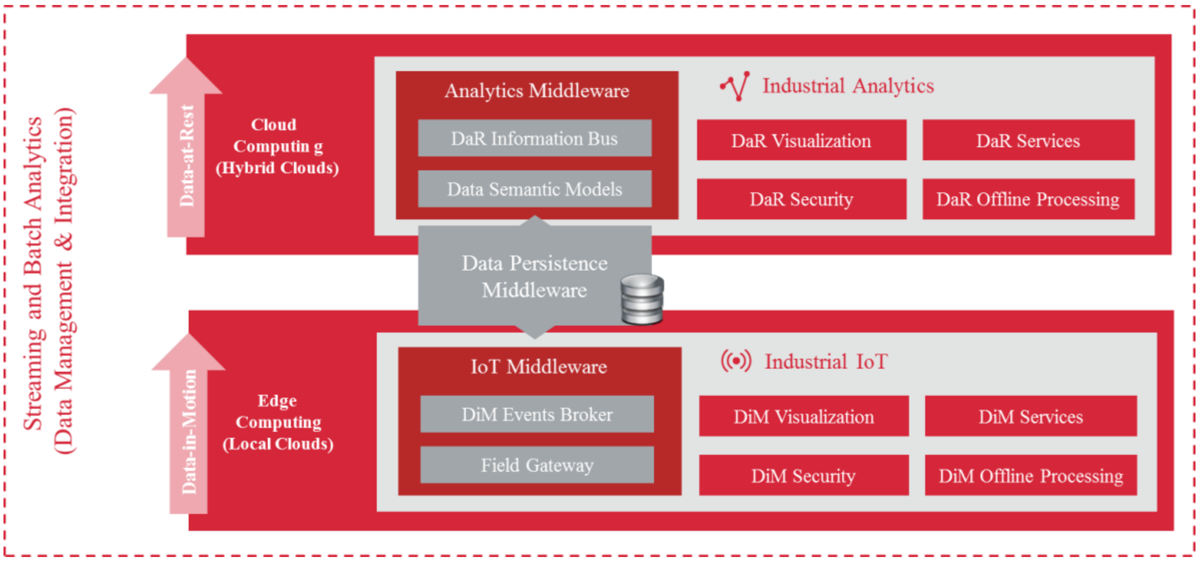


Figure 5 – MIDIH Industrial IoT and Analytics Platform

4.1 Industrial IoT and Analytics Platform – Apache lane

The Platform covers an advanced set of functionalities in order to support the challenging manufacturing scenarios the project is targeting. On the application and data processing levels, the Platform will offer a rich feature set for cloud enablement and generic data treatment, which can be used to rapidly assemble end-to-end IoT applications for industrial systems automation, predictive maintenance, and remote monitoring.

The following picture shows the initial APACHE components selected to cover the functionalities offered by the Industrial IoT and Analytics Platform.

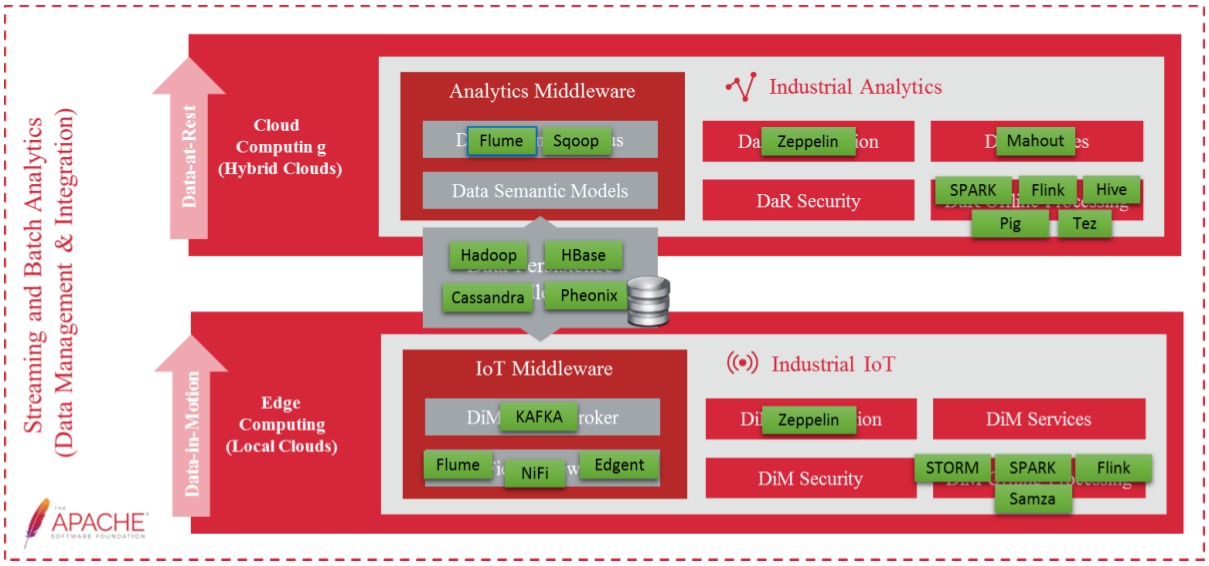


Figure 6 – MIDIH Industrial IoT and Analytics Platform APACHE implementation

4.1.1 Background components

D2Lab (https://d2lab.nissatech.com/) is the proprietary framework by NISSATECH for developing big data analytics solutions. It can be treated as a personalized diagnostic laboratory for industrial cases. The framework has been applied for the development of several efficient and scalable systems for data-driven quality control. Figure 7 shows the conceptual architecture of the framework.

The main innovation is related to the data-driven management of usual/unusual behavior that enables us to treat unusualness as first-class citizens and consequently to provide support for the whole life cycle of the usual/unusual (anomalous) behavior.

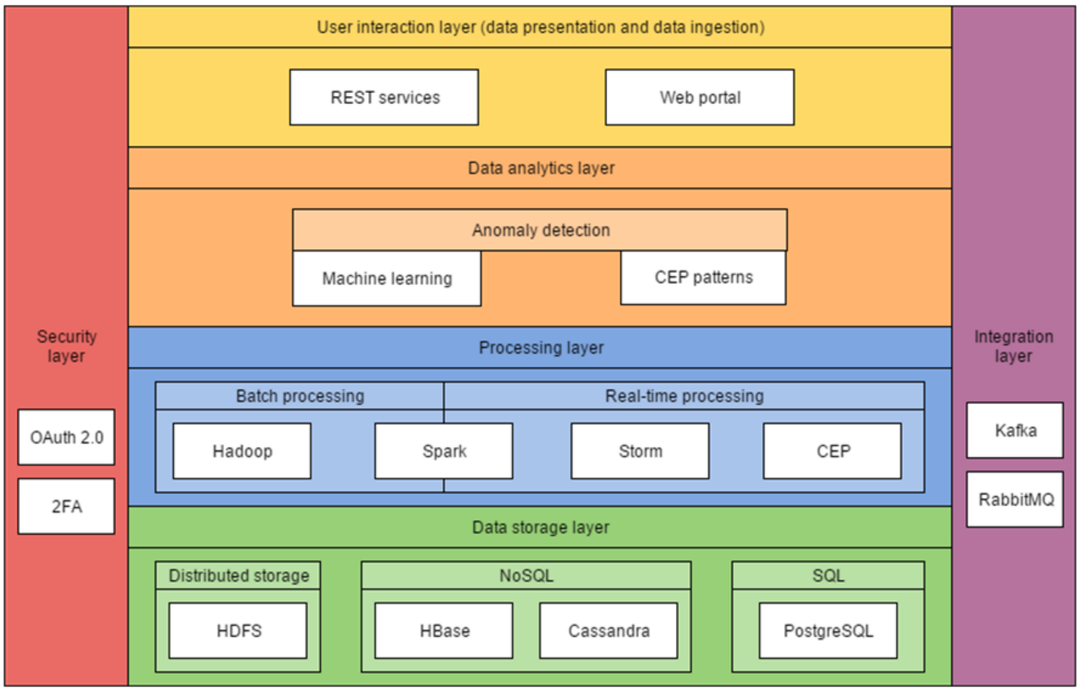


Figure 7 – D2lab conceptual architecture

The system consists of the following components:

- Data storage layer for storing training data, meta-data, models and results of detection;
- Processing layer provides different execution engines for scalable, distributed and parallel algorithm execution;
- Data analytics layer implements different machine learning and pattern recognition algorithms on top of processing layer and technologies such as Hadoop, Spark and CEP;
- User interaction layer is used for data ingestion (most often through REST services) and presentation of data and enables the user to interact with the system and experiment with different algorithms and parameters through the web portal;
- Integration layer provides flexible and scalable integration of components in the system;
- Security layer makes the interactions secure through different concepts such as OAuth protocol, two factor authentication and HTTPS supported data transfer.

4.1.2 Foreground components

The main contribution is the set up and configuration provided to create a common, reliable and valuable platform starting from separated microservices.
The data-processing (Apache) pipeline and the identification of the most suitable components for each one of the steps inside this pipeline is the main contribution. It will allow MIDIH's cross-border industrial experiments to develop an end-to-end solution for their complex industry cases.

4.2 Industrial IoT and Analytics Platform – FIWARE lane

The FIWARE lane is made up of a set of micro-services, aptly-named Generic Enablers (GE), that combined can provide valuable functionalities. This project aims to provide useful tools that will be deployed for manage and monitor industrial scenarios the project is targeting. As is depicted in Figure 8, the tools must cover from the data gathering, here named as Data-In-Motion, to the data storage and treatment, named Data-at-rest.

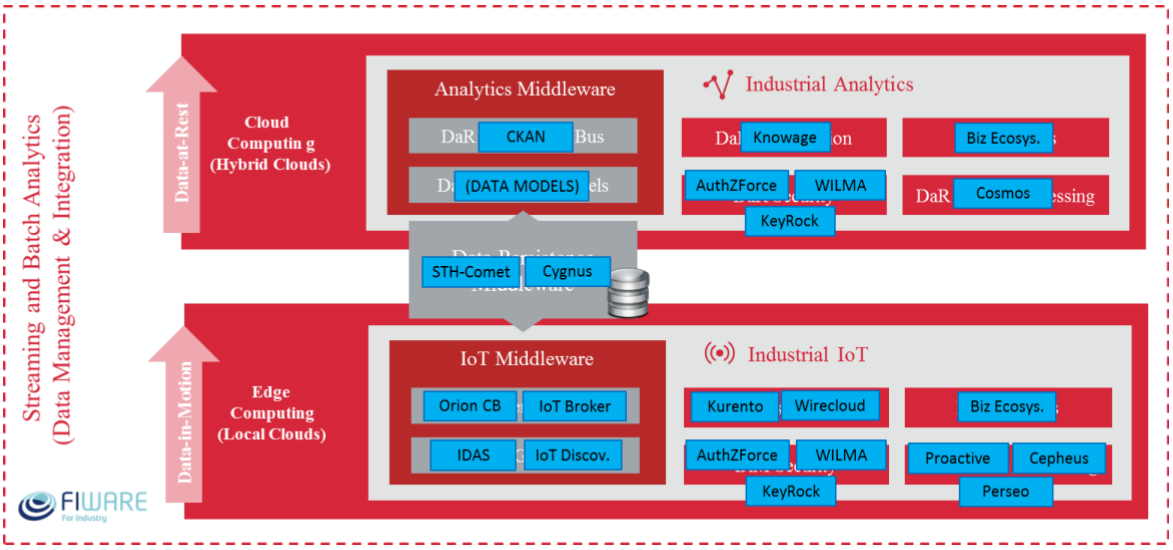


Figure 8 – MIDIH Industrial IoT and Analytics Platform FIWARE implementation

4.2.1 Background components

The project starts from the well-defined FIWARE IoT Stack (http://fiware-iot-stack.readthedocs.io/en/latest/). This architecture fits the majority of the functionalities depicted in the previous Figure 8, and serves as base architecture for the FIWARE lane deployment. Besides this, other FIWARE components are added, extending the functionality in order to cover the entire set of capabilities required. For instance, tools like Knowage, CKAN or Kurento are open source tools that can be used within the FIWARE ecosystem.

4.2.2 Foreground components

As described, the FIWARE IoT Stack is focused in data gathering from IoT devices. The innovation presented is the architecture's evolution from an IoT-based scenario to a well-defined, two-layer, industrial scenario. This new scenario, with a new and more developed architecture, must serve all the steps from data gathering or data storage to treatment and processing. Therefore, a more detailed and ambitious architecture will be presented as result.

Besides this, a complete integration between the aforementioned APACHE lane and this FIWARE lane will be provided as an interesting output.

5 Conclusions and Future Outlook to D4.4

Deliverable D4.3 provides complete information about the components developed in tasks WP4.2, WP4.3 and WP4.4 in order to implement the MIDIH reference architecture, providing a clear vision of the work carried out during this first period.

With regard task ***WP4.2***, the introduction of Edge Computing capabilities into the architecture provides to the ecosystem new features such as local data processing, reducing the number of messages between the IoT Device and the cloud, since the data provided by sensors can be pre-processed, aggregated, privatized,… resulting in network decongestion, increasing data processing speed and security.

It is quite possible that the usage of this data processing will be an extended practice in the future, when the amount of data produced by millions of devices is too large to be processed in the cloud.

In the second iteration some new features will be implemented, the most interesting is a microCEP (Complex Event Processing) based on FogFlow or FIWARE, in order to locally process the data streams allowing act in consequence with the input stream.

About task ***WP4.3***, besides addressing brownfield integration and interoperability with legacy systems and machines, providing mechanisms to send information through gateways to a condition monitoring system in order to get detailed information about the status of machines, in this first period, a more structured approach to interconnect FIWARE with a commercial OPC-UA based platform such as SIEMENS MINDSPHERE has been developed. Focusing on developing interoperability between the two protocols NGSI and OPC-UA.

For the next iteration (D4.4), the objective will be more focused in the domain of Smart Supply Chain. For this, MIDIH will leverage on the existing open source FIWARE implementation of the IDS RA and extend it with a Distributed Ledger layer to trace and control B2B transactions along the value chains, as required by some of our MIDIH Industrial Cases. Implementing the MIDIH Inter-Site Secure Data-Bus.

In task ***WP4.4***, the Industrial IoT and Analytics Platform is oriented towards reusing the open source components for realizing challenging industrial scenarios. It is important to emphasize that the usage of open source components is a huge advantage for developing this kind of complex system (esp. related to the performances and the costs). However, there are plenty of open source components and the selection of the most proper ones is a non-trivial task. Indeed, in the Apache big data stack there are hundreds of components that have similar characteristics and the details can be the differentiator. Plenty of components can be found in FIWARE (open source) repository as well. It implies a huge importance of a proper requirements analysis that will clarify the details of the intended usage of the system that is to be developed. Indeed, based on the well specified requirements it would be possible to define the best selection of the open source components.

On the other hand, in the data analytics community, there are already well-established processing pipelines (corresponding to different reference architectures). These pipelines enable an efficient organization of the processing, starting from the data ingestion (how the data can be collected) till the data visualization. We defined such a pipeline based on the previous industrial experience and the preliminary (high level) requirements. The selection of the relevant open source components is done, and the first implementation is ready.

For the next period we plan to work on a) the application of the existing prototypes on selected use cases in order to demonstrate the usefulness of the platform (MVP) and b) a more specific use case requirement analysis in order to understand the details which are important for the technical implementation. These steps will support also the further improvement of the platform, related to the selection of the open source components and the selection and configuration of the steps in the data analytics pipeline.

Annex A. Factsheet for Edge-oriented Local Clouds for Factory Automation – FogFlow

1 Edge-oriented Local Clouds for Factory Automation – FogFlow

1.1 INTRODUCTION

FogFlow is a framework based on fog computing, it is based on the idea of extreme programming. The FogFlow technology was developed at NEC Laboratories Europe, whose main creator was Bin Cheng, a NEC employee expert of Fog Computing. FogFlow offers on-demand processing of contextual information, enhances the possibilities offered by FIWARE to bring computing closer to the IoT devices that control and collect data of real-world interest.

FogFlow is an open source system and respects the NGSI standard also used by FIWARE environment for data transmission, this framework has been approved by the FIWARE Foundation as generic enabler in the open source FIWARE ecosystem.

1.2 FUNCTIONAL DESCRIPTION

1.2.1 Overview

One of the more interesting FogFlow capabilities is carry out the processing of data produced by IoT Devices in the local environment, with the aim of bringing all the necessary computing resources to the extreme to avoid having to resort to the cloud. FogFlow can be applied to any kind of work, it contains a complex task control system, consultation and management of the context data produced by the different tasks, which are controlled by IoT Discovery.

1.2.2 Architecture and Specification

The framework follows a three-tier structure, the service layer that is in charge of high-level services such as docker deployment and two important elements of FogFlow:

- Task Designer: Web tool to visually manage, create and delete the tasks that will be deployed on the fog/cloud nodes.
- Topology Master: Component that based on the configured application topology, will decide when, where and which of the tasks are deployed, whether in the cloud or at the ends

The second layer is the context layer, where all context data is located, this layer is in charge of manage, store and distribute the data through the distributed application. This layer is mainly composed by the following two components:

- IoT Discovery: it is in charge of processing context information with its id, attributes, metadata, etc., and allowing other elements to query and subscribe to the data. It is used by the local IoT Broker to query for entities located in other IoT Brokers contained on different edge nodes.
- IoT Broker: it is in charge of manage the local context entities that can be produced by nearby IoT Devices integrated on the Edge Node, providing a single view of all the entities that provides input streams to the tasks deployed on the edge node. It can be also used to provide output streams to be consumed by tasks deployed on other Edge nodes or in the Cloud.

Finally, there is the data processing layer, which, as its name suggests, where the data from input streams is processed, optionally producing output streams after process or analyse the input. This layer is mainly composed by the following two components:

- Worker: coinciding with the topology master, each worker launches his tasks in docker containers on his local machine. Defines the inputs and outputs of tasks and manages them according to their priority.
- Operator: The operators are the objects that contain all the data processing logic of a service topology. The way in which data is processed in FogFlow is through tasks, as we have said previously on this document, but the way in which FogFlow deals with tasks is through the operators. An operator can be implemented using python or JavaScript, in the FogFlow documentation there is a developed and explained example of how to implement an operator.

The Figure 9 illustrates how the different interactions between the Worker, IoT Broker and deployed tasks occurs. This interaction is further explained in detail next.

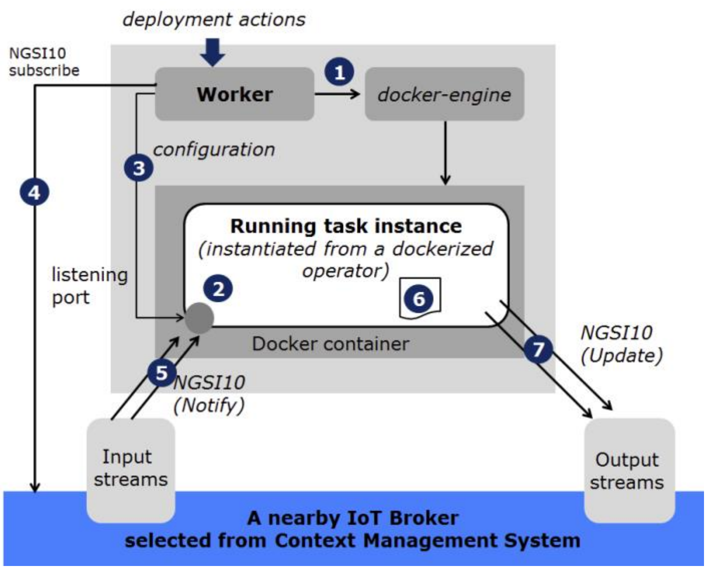


Figure 9 – Data Flow Processing

The Worker informs the docker that it wants to create a new container with an instance of a task, the inputs and outputs are configured for the task. Once this is done the task starts listening on the given port waiting to receive incoming data streams. The Worker can send a configuration object to the task through the listening port. From the IoT Broker you can make subscription requests for the task through NGSI10 messages. When the task receives incoming flows, processes the data and sends outgoing data flows, all subscriptions are notified at that time.

Next picture (Figure 10) illustrates the full deployment of the FogFlow system, you can see three parts: a first node that corresponds to the cloud, formed by the IoT Discovery, the IoT Broker, the Worker, the Task Designer and the Topology Master, which have been explained above, but has an element called RabbitMQ, a message bus based on AMQP for the exchange of control commands between the Topology Masters and the Workers. Another node that corresponds to the border node, where the IoT Devices are located along with the worker and the IoT Broker. Finally, we have a Docker Registry that is in charge of managing the images of the operators' registration couplers.

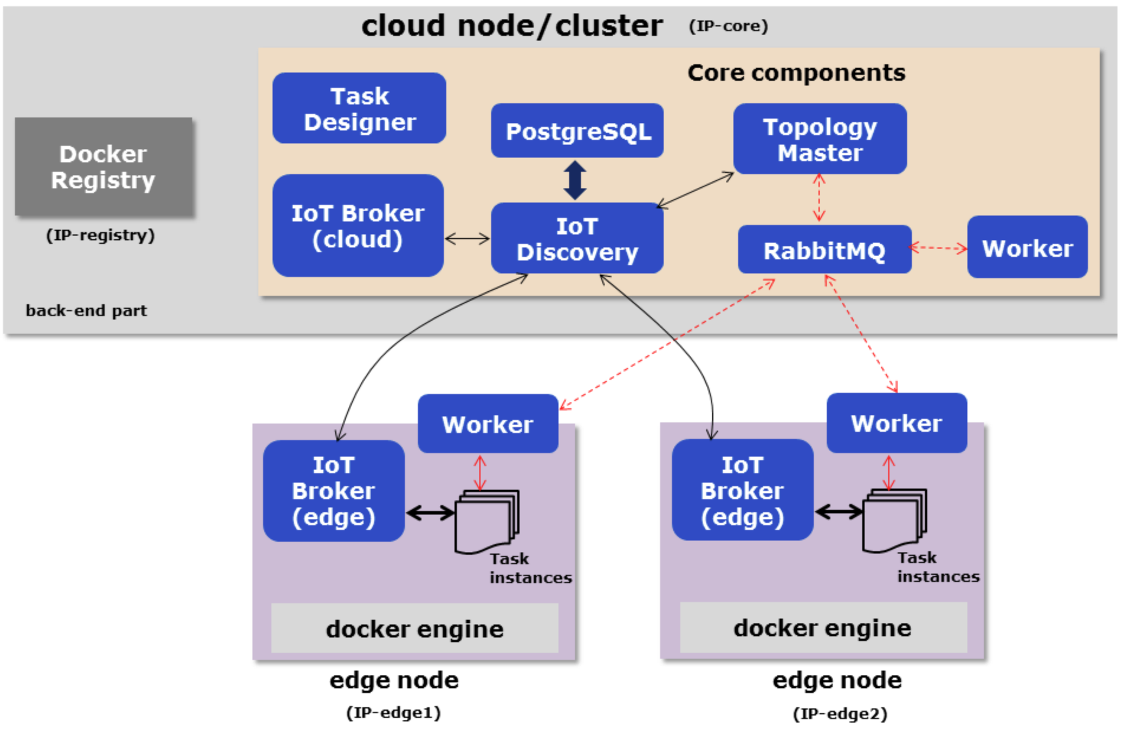


Figure 10 – Deployment view

1.3 MANUALS

1.3.1 Installation

The installation of FogFlow requires a large number of steps, so the installation will be divided into several sections.

1.3.1.1 Previous required dependencies

- git: the following commands are required to install git:

```
- sudo apt-get update
- sudo apt-get install git
```

- docker: the commands for installation are:

```
- curl fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
- sudo add-apt-repository “deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -
cs) stable”
- sudo apt-get update
- apt-cache policy docker-ce
- sudo apt-get install ~~-y docker-~~ce
- sudo systemctl status docker #Check that it works
```

- golang: the instructions to install golang need for the installation can be found on the official web page https://golang.org/doc/install, once installed check the installation using the following command:

```
- go version          #output go version go1.9 linux/amd64
```

- nodejs: to download and install nodejs, enter in the official page https://nodejs.org/en/download/ and after installation check that it is well installed using the following commands:

```
- nodejs -v           #output v6.10.2
- npm -v              #output 3.10.10
```

1.3.1.2 Construction of all FogFlow components

Previous steps:

- Change the environment variable GOPATH:

```
- export GOPATH=”/home/smartfog/go”
```

- To check out the code repository:

```
- cd /home/smartfog/go/src/
- git clone https://github.com/smartfog/fogflow.git
```

Build IoT Discovery:

- Build the native executable program

```
- cd /home/smartfog/go/src/fogflow/discovery
- go get
- go build
```

- Build the docker image:

```
- go get
- CGO_ENABLED=0 go build -a
docker build -t "fogflow/discovery" .
```

Build IoT Broker:

- Build the native program

```
- cd /home/smartfog/go/src/fogflow/broker
- go get
- go build
```

- Build docker image

```
- go get
- CGO_ENABLED=0 go build -a
docker build -t "fogflow/broker" .
```

Build Topology Master:

- Build the native executable program

```
- cd /home/smartfog/go/src/fogflow/master
- go get
- go build
```

- Build the docker image:

```
- go get
- CGO_ENABLED=0 go build -a
docker build -t "fogflow/master" .
```

Build Worker:

- Build the native executable program:

```
- cd /home/smartfog/go/src/fogflow/worker
- go get
- go build
```

- Build the docker image:

```
- go get
- CGO_ENABLED=0 go build -a
docker build -t "fogflow/worker" .
```

Build Task Designer:

- Install third-party library dependencies:

```
- cd /home/smartfog/go/src/fogflow/designer
- npm install
```

- Build the docker image:

```
- npm install
- docker build -t "fogflow/designer" .
```

Install docker-compose:

```
- sudo curl L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-‘uname -
s’-’uname -m’ ~~o usr/local/bin/docker-~~compose
- sudo chmod +x /usr/local/bin/docker-compose
- docker-compose --version
```

1.3.2 Configuration

To carry out the initialization of the FogFlow elements it is necessary to change some configuration files, as described below.

1.3.2.1 fogflow/deployment/fog/docker-compose.yml

In this file it is necessary to change the indicated IPs with a comment:

```
version: “2”
services:
  broker:
      image: fogflowe/broker
      ports:
        - 8070:8070
  worker:
      image: fogflow/worker
      environment:
        - myip=192.168.1.1 #Specify the IP address of the fog node
        - discoveryURL=”http://192.168.0.1:8070/ngsi” #IP address of the cloud node
        -rabbitmq=”amqp://admin:mypass@192.168.0.1:5672/” #IP address of the cloud node
      volumes:
        - /var/run/docker.sock:/var/run/docker.sock
```

1.3.2.2 file of IoT Broker (Edge)

For the correct functioning of IoT Broker the following elements of the file must be tailored:

- “host”: to be the IP address of the fog node
- “discoveryURL”: change it to the accessible IP address of IoT Discovery in the cloud
- physical_location: set the geo-location of the fog node

1.3.2.3 file of IoT Worker (Edge)

The following changes must be made in the IoT Worker configuration file:

- “my_ip”: to be the IP address of the fog node
- “message_bus”: to be the HOST_IP address of the RabbitMQ in the cloud
- “iot_discovery_url”: change it to the accessible IP address of IoT Discovery in the cloud
- physical_location: set the geo-location of the fog node

1.3.3 Usage

This section explains the way to start the different elements of FogFlow, both in the cloud and in the edge node.

1.3.3.1 Cloud services

After the installation of docker-compose, the cloud services can be started with the following commands:

```
- export HOST_IP=AAA.BBB.CCC.DDD #private or public IP of the host
- cd fogflow/deployment/core
- docker-compose up
```

To check that the cloud services are working correctly, go to https://HOST_IP:8080 in the browser, where a section appears to check the installation of all the components, among other things.

1.3.3.2 Edge node

Two different kinds of Edge Node deployments can be achieved, first one uses both the Worker and the IoT Broker while it is also possible use just the Worker.

- <u>Run Worker and IoT Broker at the fog node</u>: This point can be achieved in two ways, using dockers or compiled executables:
  - Start dockerized Worker and IoT Broker by using docker compose script
    - The .yml file (explanation in the previous section 1.3.2.1) must be made for the use of the docker-compose.
    - Start the docker-compose.yml:

      ```
      - cd fogflow/deployment/fog
      - docker-compose up
      ```
  - Start native Worker and IoT Broker on the fog node.
    - if it has not been compiled from the source code, it needs to download the executable files, currently ARM and Linux:

      ```
      //for ARM-based fog node
      - wget
      https://github.com/smartfog/fogflow/blob/master/deployment/fog/download/arm/fog-
      arm6.zip
      //for Linux-based fog node (64bits, x86 processor architecture)
      - wget
      https://github.com/smartfog/fogflow/blob/master/deployment/fog/download/linux64/fog
      -linux64.zip
      ```
    - Unzip the zip file:

      ```
      - unzip fog-x.zip
      ```

  - Change the configuration file of IoT Broker (Edge), as explained in the previous section 1.3.2.2.

  - Start IoT Broker (Edge)

```
- cd fog-arm6/broker
- ./broker
```

  - Change the configuration file of Worker, as explained in the previous section 1.3.2.3.

  - Start Worker (Edge):

```
- cd fog-arm6/worker
- ./worker
```

- <u>Run only Worker at the light-weight fog node</u>
  - if it has not been compiled from the source code, it need to download the executable files, currently ARM and Linux:

```
- wget
```

  - Unzip the download zip file

```
- unzip fog-x.zip
```

  - Change the configuration file of Worker, as explained in the previous section 1.3.2.3.

  - Start Worker (Edge):

```
- cd fog-arm6/worker
- ./worker
```

After the previous explained installation, it is possible include security on the architecture, to do this or receive an introduction to the development of tasks, operators, subscriptions, …, please refer to the official web site[4] or the repository[5].

1.3.4 Licensing

FogFlow is licensed under BSD-4-Clause[6].

[4] https://fogflow.readthedocs.io/en/latest/index.html
[5] https://github.com/smartfog/fogflow
[6] https://directory.fsf.org/wiki/License:BSD_4Clause

Annex B. Factsheet for FIWARE OPC UA Agent

1 FIWARE OPC UA Agent

1.1 INTRODUCTION

OPC Unified Architecture (UA) is a communication framework, using a service-oriented architecture (SOA) paradigm, for the exchange of information models. It is a feature-rich standard designed to address the needs for systems integration in a large variety of application domains (not only for Industrial Automation). The resulting infrastructure is platform independent, scalable and with high-performance. The rich information model can represent complex relationships of data and its semantics. Specially designed transport protocols offer highest communication speed and interoperability. Furthermore, OPC UA provides security mechanisms like authentication, authorization, encryption and data integrity based on the latest cryptographic standards such as PKI, AES, and SHA.

FIWARE, instead, is an Open Source platform for the realization of innovative applications for the Future Internet. FIWARE is based on elements called Generic Enablers (GE), which constitute the base blocks of the platform ready to be used. The various Generic Enablers are divided into various Technical Chapters according to the sector of use. One of these chapter is dedicated to the Internet of Things, providing GEs to allow Things to become context resources, in order to ease the interaction between applications based on FIWARE and objects of real life. The Next Generation Service Interfaces (**NGSI**), a standard developed and maintained by the Open Mobile Alliance (**OMA**), has been adopted by FIWARE for the management and the exchange of resources information.

1.2 FUNCTIONAL DESCRIPTION

1.2.1 Overview

The **FIWARE Orion Context Broker**[7] represents the core data handling enabler in any FIWARE application, therefore a renovate interest has been focusing on the creation of a bridge between an OPC UA and NGSI in order to connect OPC UA devices to FIWARE-based ecosystems**.** To this end, the FIWARE Device Backend Gateway (**IDAS**[8]) allows to simplify the management and integration of devices supporting several IoT protocols with a modular architecture based on the concept of "IoT Agents". It collects data from devices using heterogeneous protocols and translates them into NGSI entities, i.e. the FIWARE platform standard language. In order to achieve this connection with OPC UA, a new FIWARE OPC UA Agent will be developed as specified in the rest of the document. In this context, a new component is needed to be able to seamlessly interact with OPC UA servers, being able to convert OPC data stream to NGSI requests. This effectively allows OPC UA devices to communicate with FIWARE components

[7] http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
[8] http://catalogue.fiware.org/enablers/backend-device-management-idas

without ever speaking NGSI directly. This component will definitely enhance the FIWARE IDAS GE, intended to provide the ability to communicate using as many protocols as possible, by providing a new support targeting industrial applications.

1.2.2 Architecture and Specification

The **FIWARE OPC UA Agent** is a software component that connects automation systems (Field Environment), which implement the OPC UA[9] standard connection technology, to the Future Internet Platform's information bus – i.e., the Publish/Subscribe Context Broker[10].

The Publish/Subscribe Context Broker, of which the Orion Context Broker (**OCB**) is the reference implementation, is a Generic Enabler of the FIWARE platform that exposes a standard interface for applications to interact with field devices and with each other. Information producers and consumers integrate themselves with the Publish/Subscribe Context Broker through the OMA NGSI API.

The FIWARE OPC UA Agent is a module of IDAS, the reference implementation of the FIWARE Backend Device Management GE[11]. It translates OPC UA *address spaces* into NGSI *contexts* (the FIWARE standard data exchange model) without exposing the underlying OPC UA binary communication protocol to applications.

The FIWARE OPC UA Agent is also able to deal with security aspects of the FIWARE platform (e.g. enforcing authentication and authorization on the communication channel) and provide other common services.

For the first version of the FIWARE OPA UC Agent (see Figure 11), it is assumed that the underlying devices expose an OPC UA service API through an *OPC UA TCP Binding* or uses specific libraries to communicate with the middleware. For convenience, we indicated these components with the *OPC UA Server* blocks, which are anyway not part of the developed component.

[9] https://opcfoundation.org/about/opc-technologies/opc-ua/

[10] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.ContextBroker

[11] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.IoT.Backend.DeviceManagement

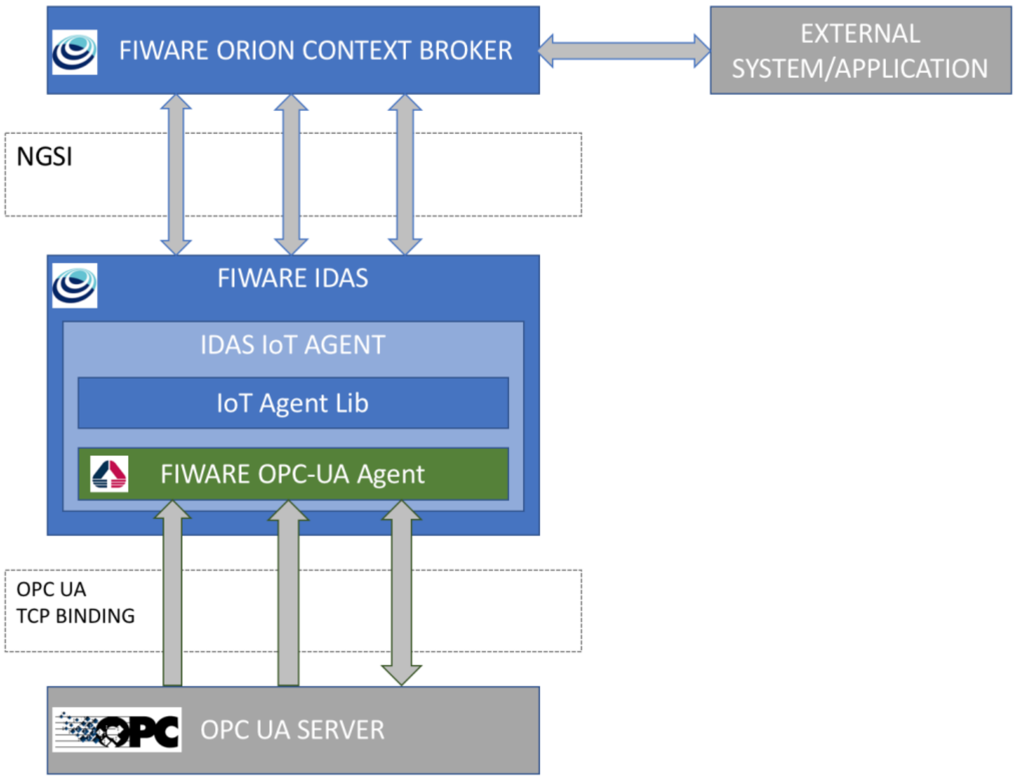


Figure 11 – FIWARE OPC UA Agent abstract architecture

1.2.2.1 *HW/SW Prerequisite*

FIWARE IDAS and Node.js are the most relevant technological prerequisites, since they constitute the grounding technologies needed to implement the FIWARE OPC UA Agent.

- **Node.js**: it is an asynchronous event driven JavaScript runtime, created with the goal of building scalable network applications. Thanks to the abstraction from the operative system, it runs as applicative platform on the major operative systems (Windows, OSX, Linux) and their derivatives for prototyping boards (e.g. Raspberry Pi, BeagleBone). Node.js can host both server as well as client applications, making possible to have a common environment for deployment.
    - *Node.js: Open Source*
    - *Tool downloads and documentation: http://nodejs.org*
- **OPC-UA JavaScript Library**: among all the components available in the Node.js platform, it is the most interesting, as it offers both a client component (used in the OPC-UA JavaScript Wrapper) and a server one, that can be used to send data from the IoT devices in the shopfloor to the OPC-UA JavaScript Wrapper.
    - *Node-opcua: Open Source (MIT)*
    - *Tool downloads and documentation: https://www.npmjs.com/package/node-opcua*
    - *Dependencies: Node.js*
- The **IDAS Agent Framework** is a Node.js module. It provides a simple object model that Node.js services can leverage to easily integrate with the FIWARE Orion Context Broker (OCB). The module does not assume any particular standard or technology on the south

side: developers are required to implement the southbound interface from scratch. However, northbound communications with the OCB are entirely encapsulated by JavaScript objects and methods.

- *Open Source, GNU Affero General Public License v3.0*
- *IDAS Generic Enabler main site:* *http://catalogue.fiware.org/enablers/backend-device-management-idas*
- *OMA NGSI protocol documentation:* *http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/ngsi-v1-0*
- *Software and documentation repository:* *https://github.com/telefonicaid/iotagent-node-lib*
- *Dependencies: Node.js (see paragraph above), MongoDB (optional, used for persisting device configuration information)*

1.2.2.2 MIDIH Specific developments

The FIWARE OPC UA Agent connects the OPC UA world and the NGSI Context Broker using two libraries: on one side NodeOPCUA SDK[12] is used for dialogue with devices, on the other IotAgent Node Lib[13] (part of IDAS) that allows the communication with the OCB.

The main functionalities of the component are:

- **Connect physical devices to FIWARE-based systems.** Supports devices that use the OPC UA standard API and connects them to an instance of FIWARE Orion Context Broker (OCB).
- **Manage NGSI Context Entities.** Automatically creates one Context Entity per physical connected device.

Each device will be mapped as a Context Entity following some rules. Basically, the user will provide an Entity Name and an Entity Type for a given device. The Entity Type will define some named Entity Attributes. Individual fields of an OPC UA device's *AddressSpace* will be mapped to specific Entity Attributes of the matching Entity Type. The agent could provide a graphical tool in order to facilitate the user experience during the addresses configuration; in this way the configuration of the agent should run easily and intuitively.

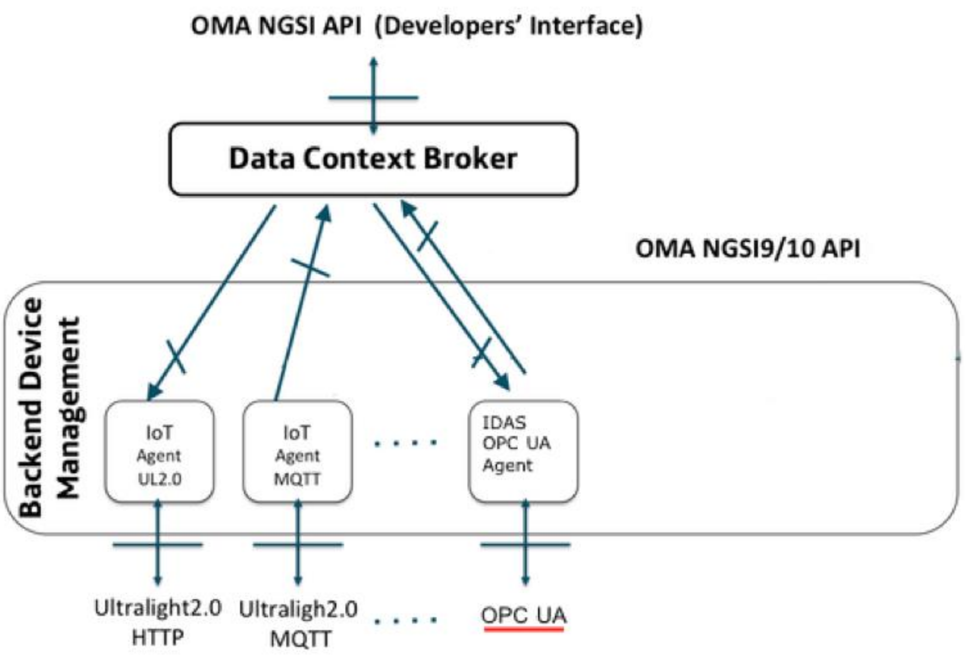
Fields can have the following different behaviors:

- **Active attributes.** Fields that are pushed from the device to the Agent. This field's changes will be sent to the Context Broker as *updateContext* requests over the device entity.
- **Lazy attributes.** Some sensors will be passive and will wait for the Agent to request for data. For those fields, the Agent will register itself in the Context Broker as a Context Provider (for all the lazy fields of that device), so if any component asks the Context Broker for the value of that sensor, its request will be redirected to the Agent. Updates over this field will be transformed into commands over the device by the Agent.

[12] http://node-opcua.github.io/
[13] https://github.com/telefonicaid/iotagent-node-lib

- **Commands**: In this case, the interaction will begin by setting an attribute in the device's entity, for which the IoT Agent will be registered as Context Provider. The IoT Agent will return an immediate response to the Context Broker and will be held responsible of contacting the device to perform the command itself, updating special *status* and *info* attributes in the entity as soon as it has any information of the command progress.

It is worth to notice that firstly the implementation will focus on active attributes, since they fit the needs to implement a publish/subscribe model. Further developments (currently out of scope) will be needed to handle also lazy attributes and commands more suitable for a client/server communication model.



These are the features that the FIWARE OPC UA Agent exposes north-bound:

- **Device registration.** Multiple devices will be connected to the Agent, each one of those mapped to a Context Broker entity. The OPC UA Agent will register itself as a Context Provider for each device when it provides commands or lazy attributes.
- **Device information update.** Whenever a device has new fields to publish, it should send the information to the Agent with OPC UA protocol. This message should, in turn, be sent as an *updateContext* request to the Context Broker, where the fields will be updated in the device entity.
- **Device command execution and value updates.** As a Context Provider, the Agent should receive update operations from the Context Broker and relay them to the corresponding device (decoding it using its ID and Type, and other possible metadata). These commands will arrive as *updateContext* operations from the Context. (Command execution PENDING; value updates available[14])

[14] https://github.com/telefonicaid/iotagent-node-lib/issues/572

These are the main OPC UA related capabilities provided by the integrated library and used internally to the FIWARE OPC UA agent:

- **Discovery Service Set** to get automatically information about OPC UA servers in the network.
- **Secure Channel Service Set** to manage a secure communication channel with devices.
- **Session Service Set** to manage a communication session OPC UA.
- **View Service Set** to browse OPC UA nodes.
- **Attribute Service Set** to manage OPC UA attributes (i.e., read and write an attribute).
- **MonitoredItems Service Set** for OPC UA monitoring and triggering (i.e., about a device or component of interest).
- **Subscription Service Set** to manage the subscriptions.

1.3 MANUALS

1.3.1 Installation

All the source code and a packed docker image can be retrieved at:

https://github.com/is3labengrd/iotagent-opcua

https://github.com/is3labengrd/idas-opcua-agent-support-tool

The installation process can be started by using the following command, providing the capability to download and install all the required dependencies:

```
npm install
```

1.3.2 Configuration

The configuration of the FIWARE OPC UA Agent can follow a twofold approach: automatic or manual.

The **Auto Configuration** (using of Mapping Tool) can be used to configure the properties file in order to set parameters of the North side (Context Broker), agent server and South side (OPC UA endpoint).

Using of Auto Configuration creates a mapping for all OPC UA objects (except those with namespace to ignore matching): all OPC UA variables will be configured as active attributes whereas all OPC UA methods will be configured as commands. It is possible modify configuration output (config.json file in same path) manually in order to drop some attributes/command, add lazy attributes and enable the command polling.

```
#SOUTHBOUND CONFIGURATION (OPC UA)
#Namespace to ignore
namespace-ignore=2,7
#OPC UA Endpoint
endpoint=opc.tcp://localhost:4334/UA/CarServer
```

```
#NORTHBOUND CONFIGURATION (ORION CB)
context-broker-host=192.168.56.101
context-broker-port=1026
fiware-service=opcua_car
fiware-service-path=/demo

#AGENT SERVER CONFIGURATION
server-base-root=/
server-port=4041
device-registry-type=memory
provider-url=http://192.168.56.1:4041
device-registration-duration=P1M
log-level=INFO
#MongoDB Agent Config
mongodb-host=192.168.56.101
mongodb-port=27017
mongodb-db=iotagent
mongodb-retries=5
mongodb-retry-time=5

#DATATYPE MAPPING OPCUA --> NGSI
OPC-datatype-Number=Number
OPC-datatype-Decimal128=Number
OPC-datatype-Double=Number
OPC-datatype-Float=Number
OPC-datatype-Integer=Integer
OPC-datatype-UInteger=Integer
OPC-datatype-String=Text
OPC-datatype-ByteString=Text
#END DATATYPE MAPPING OPCUA --> NGSI

#Administration Services
api-port=8080
#End Administration Services

#POLL COMMANDS SETTINGS
polling=false
polling-commands-timer=3000
pollingDaemonFrequency=20000
pollingExpiration=200000
#END POLL COMMANDS SETTINGS
```

By using the **Manual Configuration** (editing config.json file), instead, the user can achieve the following objectives:

- To define active attributes
  - set the active object in active section array of type object
  - set the mapping object in mappings array of contexts
- To define lazy attributes
  - set the lazy object in lazy section array of type object
  - set the mapping object in mappings array of contextSubscriptions (set object_id to null and inputArguments to empty array)
- To define commands attributes
  - set the command object in commands section array of type object
  - set the mapping object in mappings array of contextSubscriptions (object_id is the parent object of the method)
- To define poll commands
  - set polling to true to enable or to false to disable poll commands
  - set polling Daemon Frequency and Expiration in ms
  - set polling-commands-timer in ms to execute the poll commands automatically

1.3.3 Usage

The agent can be started by using the following command:

node index.js

It's possible to redirect the output log on a file by using:

node index.js > out.log

1.3.4 Licensing

The FIWARE OPC UA Agent is released under the **GNU Affero General Public License v3.0**[15].

[15] http://www.gnu.org/licenses/agpl-3.0.en.html

Annex C. Factsheet for MIDIH MindSphere Custom Agent

1 MIDIH MindSphere Custom Agent

1.1 INTRODUCTION

MindSphere is a commercial platform offered as a Service (PaaS) for scalable, global IoT connectivity and application development. With the objective of facilitate connectivity, Siemens provides a suite called MindConnect, which offers numerous possibilities for connecting machines, systems, fleets, machines and products to MindSphere, regardless of the manufacturer and based on open standards for connectivity, including OPC-UA. The MindSphere connectivity suite is available as hardware and software. There are several hardware solutions, one tested is called ***MindConnect Nano,*** which is a plug-and-play solution that enables manufacturers to read out data from their asset and preprocess it for transfer to MindSphere in a simply and reliably way. The encrypted data is then transferred securely to MindSphere, where it is available for their analysis.

As a software, Siemens offers the ***MindConnect LIB***, a library encoded in C programming language that provides connectivity functions to enable communication with the MindSphere Platform, offering the possibility to self-programme embedded devices and to connect with the MindSphere platform. It allows connecting third-party products securely and encrypting data communication. The ***MindConnect LIB*** can be integrated into software, building a "Custom Agent", which connects and transfers data to Mindsphere through a secure internet protocol in order to enable cloud-based applications and services. A Custom Agent can be programmed for any common OS (such as Windows 32/64 Bit, Linux32/64Bit, Mac OS X, etc.).

Main objective of the MIDIH MindSphere Custom Agent, is to provide the mechanisms in order to gather data from OPC UA devices, integrate them inside the MIDIH platform (NGSI) and provide interoperability with the MindSphere platform, allowing data coming from the MIIDH platform to be injected into this platform, for their storage and subsequent analysis.

1.2 FUNCTIONAL DESCRIPTION

1.2.1 Overview

FIWARE[16] is an open source standard platform integrating many components making it easier to develop smart applications. For this, the FIWARE platform provides a rather simple yet powerful set of APIs (Application Programming Interfaces) that ease the development of Smart Applications in multiple vertical sectors. For this purpose, FIWARE exposes to developers Data Context elements or entities (JSON objects) with attributes and metadata with a uniform REST API, which allows developers creating software stacks aiming to bring data-level interoperability to the complexity of standards and protocols in the world of IoT today. These apps will be able to expose IoT devices information and commands using the Data Context API (NGSI9/10 -Next Generation Service Interface) for communicating devices with NGSI brokers or any other piece which uses the NGSI protocol.

[16] https://www.fiware.org

The **MASAI** component is made up by three separate modules/subcomponents; two of them based on FIWARE Generic Enablers implementations focused on IoT purposes, the FIWARE Backend Device Management GE (IDAS)[17] and the FIWARE IoT Data Edge Consolidation GE (Cepheus)[18], plus an additional component which is a MQTT broker called Mosquitto[19]. MASAI is dedicated for the collection of data from the shop floor acting as a middleware between the IoT data producers in the shop floor and the data consumers, which can be external services or applications through a NGSI broker, normally the FIWARE Orion Context Broker (OCB) or any other type of broker. These three sub-components tackle three different issues: communication, management of the devices and handling of data.

- Protocol Adaptation enables **communication** between both IoT systems and any Data Collector Platform or Framework, granting interoperability and adaptation between different protocols and the matching between consumers and providers of data.
- For **managing** the different types of IoT devices that could be available at the companies, the Backend Device Management encloses generic information about the devices and addresses their security and connectivity.
- Supporting **data handling** functionalities, MASAI is also granted with a sub component that ensures that the data obtained in the IoT world is pre-filtered before being passed to the Data Collector Platform, reducing the flow or the quantity of inaccurate data.

The following Figure 12 presents the High-Level Architecture of MASAI:

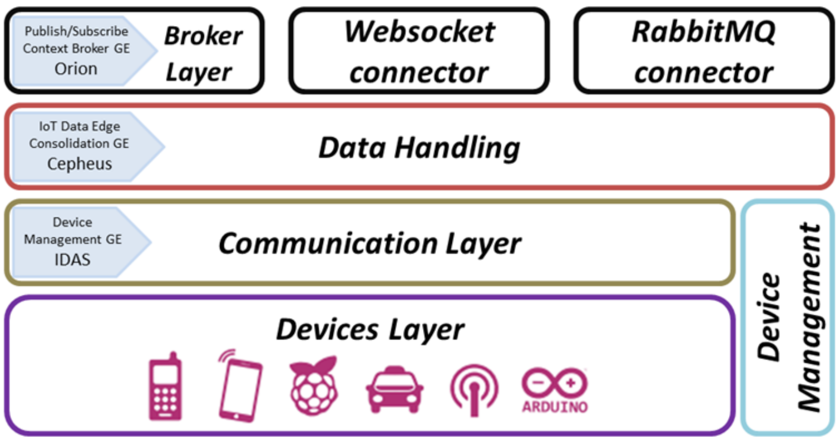


Figure 12 – MASAI Overview

The **MindConnect LIB** is a collection of functionalities provided as source code which allows software developments that integrates it, to connect with and transmit data from devices, systems and equipment to the Siemens MindSphere. Connections are made in a secure way using SSL/TLS in order to protect client's transferred data.

The MIDIH MindSphere Custom Agent solution provided, allows to collect data from both IoT devices and OPC-UA servers, convert them into FIWARE context entities and finally inject them

[17] https://catalogue-server.fiware.org/enablers/backend-device-management-idas
[18] https://fiware-cepheus.readthedocs.io/en/latest/
[19] https://mosquitto.org/

into the Siemens MindSphere platform. The solution contemplates MASAI, including the different IoT Agents, and a custom development integrating the **MindConnect LIB**.

1.2.2 Architecture and Specification

The following picture (Figure 13) depicts the MASAI detailed architecture, where we can identify three different layers: Communication, Data Handling and Broker layer:

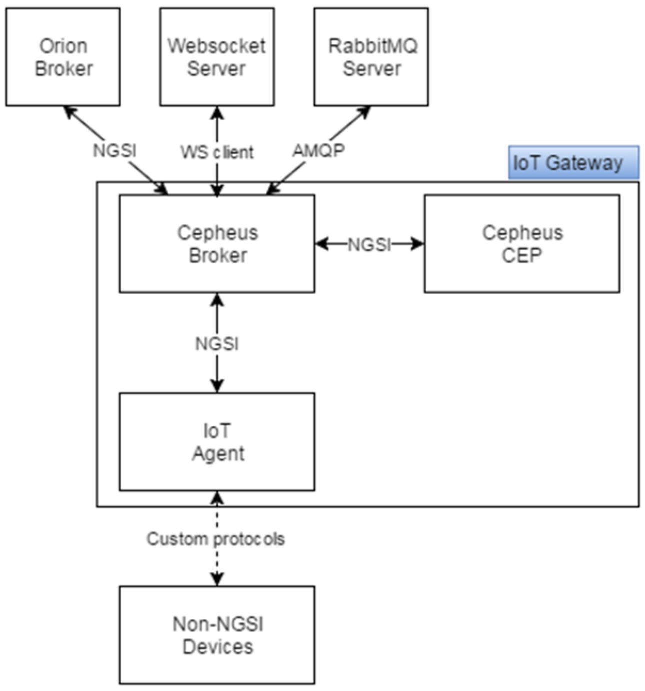


Figure 13 – MASAI Architecture

The different layers of this architecture are:

- **Communication Layer**: This layer is based on the Backend Device Management GE, which is used to connect IoT devices (or gateways) to FIWARE-based ecosystems, as long as it translates IoT-specific protocols into Next Generation Service Interface (NGSI) context entities that are the FIWARE standard data exchange model. Using this component, devices will be represented in a FIWARE platform as NGSI entities in a Context Broker. This means that anyone can query or subscribe to changes of device parameters status by querying or subscribing to the corresponding NGSI entity attributes at the Context Broker. Additionally, you may trigger commands to your actuation devices just by updating specific command-related attributes in their NGSI entities representation at the Context Broker.
- **Data Handling Functionalities**: In the MASAI infrastructure it is the entry point of data from IoT resources. That means that it will then consume a huge amount of data coming from external sources. It is mandatory to provide a module where data could be filtered, aggregated and merged in real-time allowing applications to consume only to

specific/value-added data or pre-processed. Data Handler, also known as Data Edge Consolidation GE, will be responsible for providing such functionalities. Data handling layer will then ensure that the data obtained in the IoT world is pre-filtered before sent to upper layers, reducing the quantity of inaccurate or flux of data. Moreover, Data Handler addresses the need for filtering, merging, aggregating and consolidating real-time data being collected from the heterogeneous sources, thus providing pre-processed and aggregated data to upper layers of the infrastructure. Data handler pre-processes the data based on understanding the context of the device and data processing constraints defined by the end users. Based on the context of the device i.e. the type of measurement the device is making, IoT Data Edge Consolidation GE - Cepheus can filter unexpected faulty data. User specific constraints are used for data selection and aggregation. In performing the second type of pre-processing, Data Handler will internally take advantage of the use of a Generic Enabler implementation (GEi) called Cepheus, which addresses the need to process data in real time. This GEi uses **Esper technology**[20] (open-source event processing java library).

- **Broker Layer**: This layer can be considered as the central point for data communication. It will use the Publish/Subscribe Orion Context Broker. It will mediate between data producers (i.e. sensors) and data consumers (i.e. data analytics). If data storage is required, a software associated to Orion such as Cygnus that might be found as useful. Cygnus implements a connector for context data coming from Orion Context Broker and aimed to be stored in a specific persistent storage, such as Apache Hadoop HDFS, CKAN or MySQL.

In the communication layer, MASAI integrates the ***JSON/MQTT IoT Agent***[21] and the ***FIWARE OPC UA Agent*** in order to retrieve data from the shop floor and allows to distribute the captured data both through the Orion context Broker and/or through a specific topic defined in a messaging broker. In this case the selected broker has been the RabbitMQ[22], which acts as an intermediary for messages between the MIDIH platform and the ***MIDIH MindSphere Custom Agent Connector***, responsible to inject data into the MindSphere platform through the ***MindConnect LIB*** (see Figure 14).

[20] http://www.espertech.com/esper
[21] https://github.com/telefonicaid/iotagent-json
[22] https://www.rabbitmq.com/

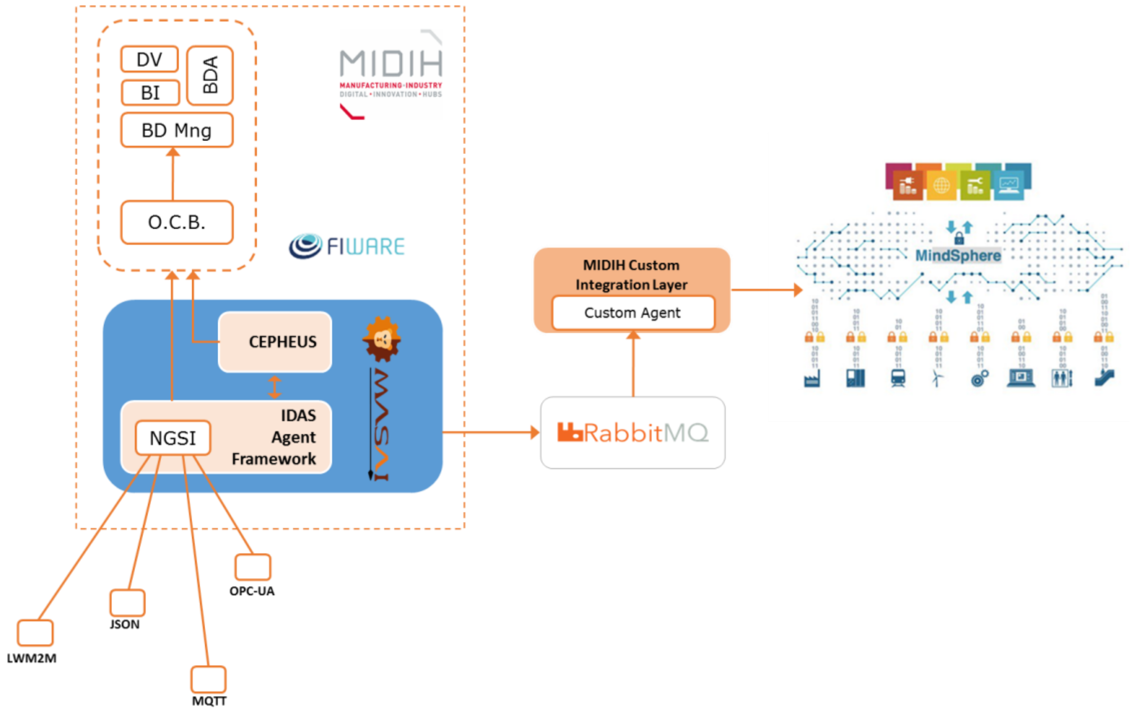


Figure 14 – MIDIH MindSphere Custom Agent functional architecture

1.2.2.1 HW/SW Prerequisite

The installation of **MASAI** requires the following system requirements:

- Java Development Kit (JDK) (version 1.8 or higher)
- Apache Maven (version 3.3.9 or higher)
- node (version 4.3.2)
- MQTT Mosquitto broker (version 1.4.7) (out-of-the-box setup)
- Git (version 1.9.1)

Optional but recommended

- Curl (version 7.19.7)

To get started with **MindSphere** and to start developing applications, the first step would be to obtain an MindAccess Developer account. The MindAccess Developer is the entry to MindSphere and gives us the ability, tools and documentation we need to get started. Exist three kind of access level (S, M, L) which come with different settings and features. The **MindConnect LIB** is only available in combination with the MindAccess Developer offering, and it is necessary to pay a fee in order to have access to the MindSphere platform.

The version of the MindConnect LIB used in this first prototype is the version 2.0.

MindConnect LIB Requirements:

- For building the MindConnect LIB is necessary the cross-make tool "cmake (3.5.2 and newer)" and appropriate toolchain including compiler, linker, etc. for desired target environment.

- The current version of the MindConnect LIB works only (together with Libcurl V7.52.1 and OpenSSL V1.0.2k.
- The Libcurl installation must be configured to use OpenSSL as the TLS v1.2 implementation.

Before build the MindConnect LIB, the **MIDIH MindSphere Custom Agent Connector** requires an open source C-language AMQP client library for use with the RabbitMQ broker. This library is the "rabbitmq-c"

- *Software and documentation repository* https://github.com/alanxz/rabbitmq-c

Finally, an instance of **RabbitMQ** is necessary to publish messages to the MIDIH MindSphere Custom Agent Connector.

1.2.2.2 MIDIH Specific developments

The MIDIH MindSphere Custom Agent comprises several developments that will allow the MIDIH platform interoperate with commercial solutions. For this purpose, the overall workflow and the components to achieve interoperability has been identified, and then it has proceeded to the implementation.

The functionalities provided by the component are:

- connect IoT and OPC UA devices to FIWARE-based systems
- capture data from these devices
- preprocess the collected data in order to clean and detect incorrect data before transforming it into NGSI Context Entities
- Publish this information in a Message Broker, the Orion Context Broker (RabbitMQ)
- retrieve the information and map it with the aspects (variables) defined for the specific Asset (device) in MindSphere
- inject them into the MindSphere for further analysis.

In order to retrieve the information from the Message Broker (RabbitMQ), translate this information to the MindSphere data models and inject them into MindSphere, the *MindSphere Custom Agent Connector* has been developed from scratch. This component is responsible for acting as an intermediary between the MIDIH platform and MindSphere, and to integrate the connectivity library provided by Siemens, the MindConnect LIB.

1.3 MANUALS

1.3.1 Installation

The MASAI source code can be retrieved at:

https://github.com/ARI-MR/MASAI

In order to set up MASAI, is needed to build and configure separately these different components: the Mosquitto MQTT broker, The FIWARE Backend Device Management GE – IDAS together with the FIWARE OPC UA Agent and the FIWARE IoT Data Edge Consolidation GE – Cepheus.

For the next version of the prototype, MASAI will be delivered as a docker image in order to facilitate its deployment.

1.3.2 Configuration

These are the steps to follow to configure the different components:

MQTT broker (Mosquitto)

To setting up MQTT Mosquitto broker for implementing MQTT communication perform the following steps:

1. Login to Ubuntu Linux machine where the Mosquitto broker will be deployed with root privileges (sudo)
2. Add the mosquitto repository by given below commands

```
$ sudo apt-get update
$ sudo apt-get install mosquitto mosquitto-clients
```

3. Ensure that Mosquitto broker is running

```
$ sudo service mosquitto status
```

For the above command, expected result is mosquitto start/running with and specific process id, that will vary from machine to machine.

The mosquito is installed as a service running in the background, so once it is installed, automatically will be up and running.

To install the FIWARE components, just clone the MASAI project, in the MASAI folder:

```
$ git clone https://github.com/ARI-MR/MASAI.git masai/
```

Once the MASAI project is cloned, you need to install each component in the following way:

FIWARE Backend Device Management GE – IDAS

To install the IDAS IoT Agent for JSON (MQTT transport) component, you only need to prepare iotagent-node-lib as a dependency of iotagent-mqtt. For this, move to the folder:

```
$ cd masai/iotagent-node-lib/
```

And execute the command below:

```
$ sudo npm install
```

Do the same for the iotagent-mqtt. Move to the folder:

```
$ cd masai/iotagent-mqtt/
```

And execute the given below command:

```
$ sudo npm install
```

Finally, copy the iotagent-node-lib into node_modules folder of iotagent-mqtt:

```
$ sudo cp -r iotagent-node-lib/ iotagent-mqtt/node_modules/
```

In order to start the IoT Agent, from the masai/iotagent-mqtt folder, type:

```
$ bin/iotagentMqtt.js
```

This will start the JSON IoT Agent in the foreground. Use standard linux command nohup to start it in background.

When started, the IoT Agent gets the configuration from the config.js file located in the root folder, if no arguments are indicated at startup.

Therefore, before starting the iotagent-mqtt, check config.json file, where it has to be set where the MQTT broker is running and in which port will the agent be running:

```
config.mqtt = {
   host: 'localhost',
   port: 1883,
   defaultKey: 'ATOS',
   thinkingThingsPlugin: true
}
server: {
   port: 4041
}
```

The most relevant information located in this file is the host and port of the MQTT broker, the API Key and Device ID of the device to simulate.
To have detailed information about the different parameters that can be indicated in the config file, please have a look to the iotagent-json “manual for Installation & Administration Guide”, located here: https://goo.gl/QfIHJZ.
For more generic information about the component, please visit the URL: https://goo.gl/1qcm5r.

FIWARE IoT Data Edge Consolidation GE – Cepheus

To install the Cepheus GE, which is composed by two subcomponents: the Cepheus-Broker a lightweight NGSI broker and Cepheus-CEP, a Complex Event Processor; first you need to compile both components in order to generate the executables .jar files.

Move to the folder:

```
$ cd masai/fiware-cepheus
```

And execute the given below command:

```
$ sudo mvn clean install
```

Two executable files will be created:

- *cepheus-broker-1.0.1-SNAPSHOT.jar* in folder “masai/fiware-cepheus/cepheus-broker/target” and
- *cepheus-cep-1.0.1-SNAPSHOT.jar* in folder “masai/fiware-cepheus/cepheus-cep/target”

Once those files have been created, it is needed to execute them; first the Cepheus-broker and secondly the Cepheus-CEP.

For this, move to the target folders and execute the following commands:

```
$ cd masai/fiware-cepheus/cepheus-broker/target/
$ sudo java -jar cepheus-broker-1.0.1-SNAPSHOT.jar
$ cd masai/fiware-cepheus/cepheus-cep/target/
$ sudo java -jar cepheus-cep-1.0.1-SNAPSHOT.jar
```

You can modify some application settings before building the components editing the application.properties files located at "*src/main/resources/*" folder.

The most relevant application properties to modify are:

- server.port: port used, 8080 by default
- data.path: path to store data, /tmp/ by default
- logging.level.com.orange.cepheus.cep: log level, INFO level by default

For more specific information of this GE, please visit the URL: https://goo.gl/Q41b9B

Additionally, if you need to modify any attribute of the IoT Agent or of the Cepheus CEP, you can proceed with the following instructions:

How to restart MASAI IoT Agent:

```
$ sudo lsof -i :4041 --> GET PID from instruction
$ sudo kill -9 {$PID}
```

Move to the folder where the iotagent-mqtt is installed:

```
$ cd masai/iotagent-mqtt
```

If necessary, modify config.js file. Then start the service again:

```
$ bin/iotagentMqtt.js
```

And to modify the CEPEHUS properties, you can proceed in this way:

How to restart MASAI CEP Broker:

```
$ sudo lsof -i :8081 --> GET PID from instruction
$ sudo kill -9 {$PID}
```

How to restart MASAI CEP Engine:

```
$ sudo lsof -i :8080 --> GET PID from instruction
$ sudo kill -9 {$PID}
```

modify *application.properties* of broker and cep if needed, then start the service again. Respective file can be found at:
broker: *masai/fiware-cepheus/cepheus-broker/src/main/resources/application.properties*
cep: *masai/fiware-cepheus/cepheus-cep/src/main/resources/application.properties*

```
$ cd masai/fiware-cepheus/
$ sudo mvn clean install
```

```
$ cd masai/fiware-cepheus/cepheus-broker/target/
$ sudo java -jar cepheus-broker-1.0.1-SNAPSHOT.jar
```

```
$ cd masai/fiware-cepheus/cepheus-cep/target/
$ sudo java -jar cepheus-cep-1.0.1-SNAPSHOT.jar
```

For the configuration of the ***FIWARE OPC UA Agent***, you can consult it in the component's factsheet, in Annex B.

MIDIH MindSphere Custom Agent Connector

The parameters to configure in the MIDIH MindSphere Custom Agent Connector are located in the "_setup_configuration" method. The relevant parameters to complement before compiling the executable file are: the hostname, port, user_agent, proxy_type and the initial_access_token. Of course, the certificate for the tenant is mandatory.

```
static void _setup_configuration(mcl_configuration_t **configuration)
{

   (*configuration) = mcl_new_configuration();
   (*configuration)->mindsphere_hostname = "https://mindconnectcom.apps.mindsphere.io";
   (*configuration)->mindsphere_port = 443;
   //J-
    (*configuration)->mindsphere_certificate =
   //Root certificate of Mindsphere
    "-----BEGIN CERTIFICATE-----\n"
    ....
    "-----END CERTIFICATE-----";
   //J+
   (*configuration)->security_profile = MCL_SECURITY_SHARED_SECRET;
   (*configuration)->proxy_hostname = "http://127.0.0.1";
   (*configuration)->proxy_port = 8888;
   (*configuration)->proxy_type = MCL_PROXY_HTTP;
   (*configuration)->proxy_username = NULL;
   (*configuration)->proxy_password = NULL;
   (*configuration)->proxy_domain = NULL;
   (*configuration)->max_http_payload_size = 16384;
   (*configuration)->user_agent = "custom agent v1.0";
   (*configuration)->initial_access_token = "InitialAccessToken";
   (*configuration)->initial_registration_uri =
"https://mindconnectcom.mindsphere.io/api/register";
   (*configuration)->tenant = "minddemo";
   (*configuration)->store_path = "registrationInformation.txt";
}
```

In addition, there is a xml file for configure the RabbitMQ parameters (*"rabbitmq_config.xml"*). Is needed to set where the broker is running and other mandatory information such as the port, queuename, username and password.

```
<general>
  <masai>
    <rabbitmq_hostname>IP</rabbitmq_hostname>
    <rabbitmq_port>port_number</rabbitmq_port>
    <rabbitmq_exchange>exchange_name</rabbitmq_exchange>
    <rabbitmq_queuename>queuename</rabbitmq_queuename>
    <rabbitmq_username>user</rabbitmq_username>
    <rabbitmq_password>pass</rabbitmq_password>
  </masai>
</general>
```

Siemens MindSphere Platform

Mindsphere provides several Apps in order to manage and configure the tenant (user account) – see Figure 15:

- ***IoT Data Modeler***, available for the configuration of the account. Allow us for organization and user management, and for configuring Assets.
- ***Fleet Manager*** is the standard app for the analysis, visualization and monitoring the Assets and Aspects previously defined in the IoT Data Modeler.

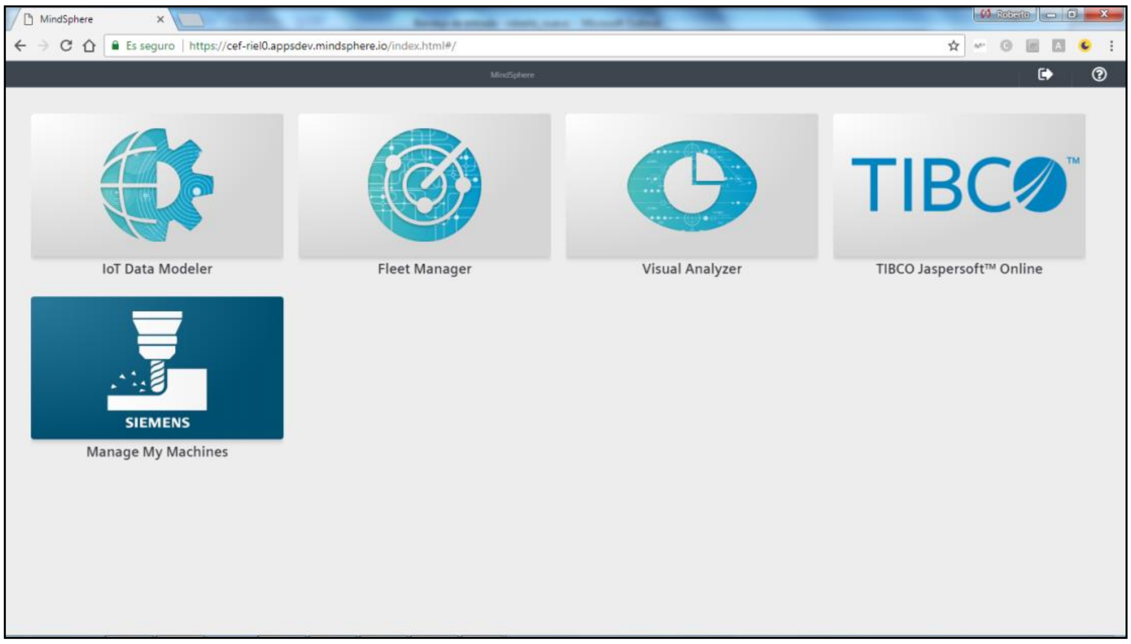


Figure 15 – MindSphere MindApps

In MindSphere Context, an Asset is a logical representation of a machine or a system. For each Asset a cluster of multiple data sources which are required for a specific analysis (e.g. evaluation of machine status) can be created (the so-called Aspects). Aspects are the playground for meaningful analysis.

The first action we must take in MindSphere is configure an Asset (Device), providing the general configuration (Name, description and storage location), the Data Sources for this Asset (the source of the data which is imported to MindSphere) and the Asset aspects, which are the context of the imported data. This process is carried out through the *IoT Data Modeler App*.

Once the Asset has been created, the *Fleet Manager App*, provides an overview of Assets configured in MindSphere and the possibility to quickly search for relevant Assets based on various criteria.

1.3.3 Usage

Once **MASAI** has been installed and pre-configured, to start working with it, you must configure the different modules that compose this component (if it is required), ideally in this specific order:

1. Configure and execute the IoT Agents
2. Configure and execute the Cepheus-Broker and Cepheus-CEP
3. Provisioning devices to the IoT Agent
4. Start generating data/events

1.3.3.1 IoT Agent Configuration and execution

The IoT Agent MQTT-JSON protocol uses plain JSON objects to send information formatted as key-value maps over an MQTT transport. All the topics used in the protocol are prefixed with the APIKey of the device group and the Device ID of the device involved in the interaction. The API Key is a secret identifier shared among all the devices of a service, and the DeviceID is an ID that uniquely identifies the device in a service.
All the configuration of the IoT Agent can be done modifying a single file, config.js. The default values in the config.js file located at the root folder of the component should meet the needs of this tutorial.
These values are the following:

```
var config = {};
config.mqtt = {
  host: 'localhost',
  port: 1883,
      defaultKey: 'ATOS ' ,
  thinkingThingsPlugin: true
} ;
config.iota = {
  logLevel: 'DEBUG',
  contextBroker: {
    host: 'localhost',
    port: '8081'
  } ,
  server: {
  port: 4041
  } ,
  deviceRegistry: {
         type: 'memory'
  } ,
  types: {},
  service: 'Hannover',
  subservice: '/Messe',
  providerUrl: 'http://localhost:4041',
  deviceRegistrationDuration: 'P1M',
  defaultType: 'Thing'
} ;
module.exports = config;
```

MASAI IoT Agent Configuration example file

Here we can see the different values, for example for the apiKey which is an alphanumerical string used to group devices logically:

apiKey: ATOS

And for the service we are going to deploy:

service: Hannover
subservice: /Messe

It's important to note that the configuration type of the *deviceRegistry* is set to memory. This means all the contents of the Device Registry will be wiped out from memory when the IoT Agent restarts. This is meant to be used in testing environments and it will force you to provision again all your devices once you have restarted the agent. For persistent registries, check the documentation located here: https://goo.gl/VXhhSZ, to see how to connect the IoT Agent to a MongoDB instance.

Once the configuration has been adapted to our requirements, we need to start the IoTAgent-mqtt executing the following command:

```
$ bin/iotagentMqtt.js
```

The agent should be now listening in the port 4041(by default). You can check it with a netstat command:

```
$ netstat -ntpl | grep 4041
```

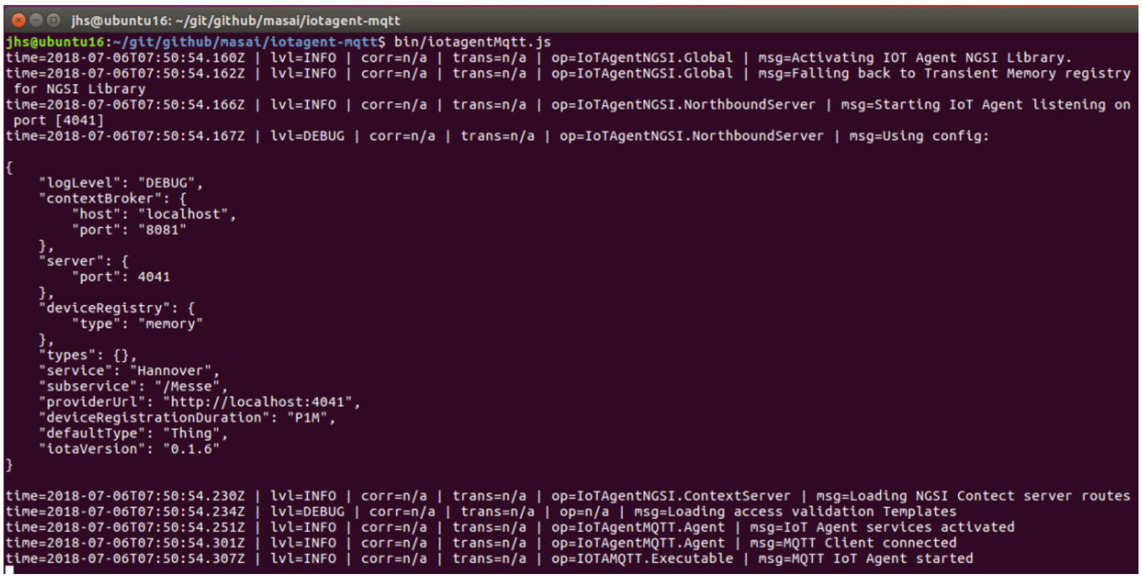
the output should be like this:

```
tcp       0      0 0.0.0.0:4041          0.0.0.0:*          LISTEN      23604/node
```

Also, it is possible to get the version directly calling the API:

```
$ curl http://localhost:4041/iot/about
```

Once executed the IoT Agent, the console will show something like this:



```
jhs@ubuntu16: ~/git/github/masai/iotagent-mqtt
jhs@ubuntu16:~/git/github/masai/iotagent-mqtt$ bin/iotagentMqtt.js
time=2018-07-06T07:50:54.160Z | lvl=INFO | corr=n/a | trans=n/a | op=IoTAgentNGSI.Global | msg=Activating IOT Agent NGSI Library.
time=2018-07-06T07:50:54.162Z | lvl=INFO | corr=n/a | trans=n/a | op=IoTAgentNGSI.Global | msg=Falling back to Transient Memory registry
 for NGSI Library
time=2018-07-06T07:50:54.166Z | lvl=INFO | corr=n/a | trans=n/a | op=IoTAgentNGSI.NorthboundServer | msg=Starting IoT Agent listening on
 port [4041]
time=2018-07-06T07:50:54.167Z | lvl=DEBUG | corr=n/a | trans=n/a | op=IoTAgentNGSI.NorthboundServer | msg=Using config:

{
    "logLevel": "DEBUG",
    "contextBroker": {
        "host": "localhost",
        "port": "8081"
    },
    "server": {
        "port": 4041
    },
    "deviceRegistry": {
        "type": "memory"
    },
    "types": {},
    "service": "Hannover",
    "subservice": "/Messe",
    "providerUrl": "http://localhost:4041",
    "deviceRegistrationDuration": "P1M",
    "defaultType": "Thing",
    "iotaVersion": "0.1.6"
}

time=2018-07-06T07:50:54.230Z | lvl=INFO | corr=n/a | trans=n/a | op=IoTAgentNGSI.ContextServer | msg=Loading NGSI Contect server routes
time=2018-07-06T07:50:54.234Z | lvl=DEBUG | corr=n/a | trans=n/a | op=n/a | msg=Loading access validation Templates
time=2018-07-06T07:50:54.251Z | lvl=INFO | corr=n/a | trans=n/a | op=IoTAgentMQTT.Agent | msg=IoT Agent services activated
time=2018-07-06T07:50:54.301Z | lvl=INFO | corr=n/a | trans=n/a | op=IoTAgentMQTT.Agent | msg=MQTT Client connected
time=2018-07-06T07:50:54.307Z | lvl=INFO | corr=n/a | trans=n/a | op=IOTAMQTT.Executable | msg=MQTT IoT Agent started
```

MASAI Agent console

1.3.3.2 *Cepheus-Broker and Cepheus-CEP Configuration and execution*

Once the executable files for both components have been created, the Cepheus-Broker and the Cepheus-CEP, it is not possible to modify the default configuration of these components.

If we want to change values of any parameter, it must be done in the respective configuration files of both components, called *application.properties*, located in the *src/main/resources* folders of each of the components. Once the values of the attributes have been changed, we need to rebuild the components again from the source, as indicated in the "*IoT Hub Installation and Administration Guide*" document.

Alternatively, it is also possible to modify the application properties when executing the components through command line parameters:

```
java -jar cepheus-cep.jar --property=value
java -jar cepheus-cep.jar --property=value
```

Example:

```
$ sudo java -jar cepheus-broker-1.0.1-SNAPSHOT.jar --server.port=8080 --data.path=/var/cepheus/
```

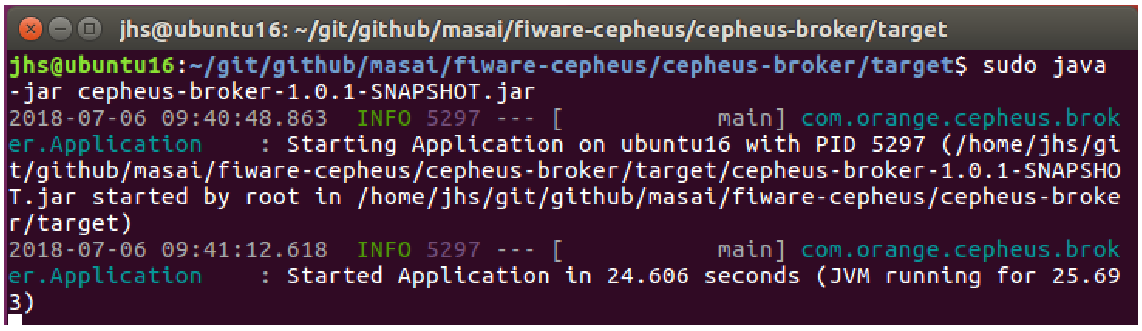
Once the configuration has been adapted to our requirements, and the executable files have been created, you need to execute both components; first the Cepheus-broker and secondly the Cepheus-CEP.

For this, move to the target folders and execute the following commands, each of them in a separate console:

```
$ cd masai/fiware-cepheus/cepheus-broker/target/
$ sudo java -jar cepheus-broker-1.0.1-SNAPSHOT.jar
```

```
$ cd masai/fiware-cepheus/cepheus-cep/target/
$ sudo java -jar cepheus-cep-1.0.1-SNAPSHOT.jar
```

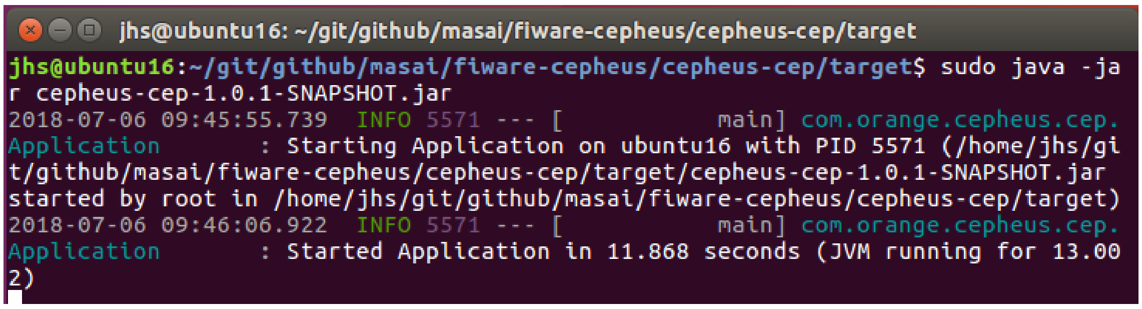
Once executed and started, the Cepheus-Broker console will look like this:



```
jhs@ubuntu16: ~/git/github/masai/fiware-cepheus/cepheus-broker/target
jhs@ubuntu16:~/git/github/masai/fiware-cepheus/cepheus-broker/target$ sudo java
-jar cepheus-broker-1.0.1-SNAPSHOT.jar
2018-07-06 09:40:48.863  INFO 5297 --- [           main] com.orange.cepheus.brok
er.Application     : Starting Application on ubuntu16 with PID 5297 (/home/jhs/gi
t/github/masai/fiware-cepheus/cepheus-broker/target/cepheus-broker-1.0.1-SNAPSHO
T.jar started by root in /home/jhs/git/github/masai/fiware-cepheus/cepheus-broke
r/target)
2018-07-06 09:41:12.618  INFO 5297 --- [           main] com.orange.cepheus.brok
er.Application     : Started Application in 24.606 seconds (JVM running for 25.69
3)
```

Cepheus-Broker console

And the console for the Cepheus-CEP in this way:



```
jhs@ubuntu16: ~/git/github/masai/fiware-cepheus/cepheus-cep/target
jhs@ubuntu16:~/git/github/masai/fiware-cepheus/cepheus-cep/target$ sudo java -ja
r cepheus-cep-1.0.1-SNAPSHOT.jar
2018-07-06 09:45:55.739  INFO 5571 --- [           main] com.orange.cepheus.cep.
Application       : Starting Application on ubuntu16 with PID 5571 (/home/jhs/gi
t/github/masai/fiware-cepheus/cepheus-cep/target/cepheus-cep-1.0.1-SNAPSHOT.jar
started by root in /home/jhs/git/github/masai/fiware-cepheus/cepheus-cep/target)
2018-07-06 09:46:06.922  INFO 5571 --- [           main] com.orange.cepheus.cep.
Application       : Started Application in 11.868 seconds (JVM running for 13.00
2)
```

Cepheus-CEP console

1.3.3.3 *Provisioning devices to the IoT Agent*

Next step is to provision the IoT Agent with devices, but first, it is needed to understand how Cepheus-CEP and Cepheus-Broker interact with each other. The data sent by the IoT Agent will not be sent directly to the Cepheus-CEP but will transit though the Cepheus-Broker.
It has to be configured the Cepheus-CEP with the type of events that it is going to receive, in this case from the Cepheus-Broker and also the kind of update messages for this specific Context Entity. This is done through the administration API that exposes the Cepheus-CEP.
The endpoint path for this API is "*/v1/admin/config*" and it accepts two verbs:

GET v1/admin/config

This endpoint returns the actual configuration as a JSON object with a 200 Ok status code.

$ curl -H 'Accept: application/json' http://localhost:8080/v1/admin/config curl -H 'Accept: application/json' http://localhost:8080/v1/admin/config

POST v1/admin/config

This endpoint applies a new configuration given in the body as JSON.

$ cat cepheus.json | curl -H 'Accept: application/json' -H 'Content-Type: application/json' -d @- http://localhost:8080/v1/admin/config

Once the new configuration has been successfully applied to the CEP, the configuration is persisted on disk. If the Cepheus-cep is later restarted, it will automatically load the last configuration on start-up.

DELETE v1/admin/config

This endpoint removes the current configuration. It will return a 200 Ok on a successful operation.

$ curl -X DELETE http://localhost:8080/v1/admin/config

GET v1/admin/statements

This endpoint returns the list of the statements as a JSON object with a 200 Ok status code.

$ curl -H 'Accept: application/json' http://localhost:8080/v1/admin/statements

The CEP configuration is a simple JSON object containing the complete description of the behaviour of the CEP engine (a set of EPL statements) and the mapping between the NGSI Context Entities and CEP Events.
As we can see in the following CEP configuration file, additionally to the basic information about the host, the service and servicePath, there are three different sections:

- **In**: in this section, translated to the configuration format of the Cepheus-CEP, it is indicated that the Cepheus-CEP will accept an object of type Room, with attributes: temperature, humidity and type as input, from a specific provider located in the URL "*http://localhost:8081*", which is the URL where the Cepheus-Broker is running.
- **Out**: Section where are indicated the events that are going to be generated by the CEP in form of NGSI Context entities. In this case will be Context Entities called Alerts, and also it has a subsection called brokers to indicate where send the update for this Context Entity to the Cepheus-Broker.
- **Statements**: Here we can indicate the EPL rules to trigger the Alerts. We would like that the system sends and Alert when the room temperature is over > 40°C.

```
{
 "service":"Hannover",
 "servicePath":"/Messe",
 "host":"http://localhost:8080",
 "in":[
   {
   "id":"Room.*",
   "type":"Room",
   "isPattern":true,
   "providers":[
     "http://localhost:8081"
     ] ,
   "attributes":[
     { "name":"temperature", "type":"double" },
     { "name":"humidity", "type":"double" },
     { "name":"Device", "type":"string" }
     ]
   }
   ] ,
 "out":[
   {
   "id":"Alert1",
   "type":"Alert",
   "attributes":[
     { "name":"temperature", "type":"double" },
     { "name":"humidity", "type":"double" },
     { "name":"Device", "type":"string" }
     ] ,
   "brokers": [
      {
     "url":"http://localhost:8081",
     "serviceName":"DemoIoT",
     "servicePath":"/"
      }
     ]
     }
   ] ,
 "statements":[
  "INSERT INTO Alert SELECT temperature, humidity, Device FROM Room Where temperature > 15"
   ]
}
```

MASAI CEP configuration json file

Therefore, when we call the Cepheus-CEP API to configure it with this file, the Cepheus-CEP will subscribe to the broker for this data and therefore be notified when new data is received.

Additionally, in case it is necessary to send data to a RabbitMQ broker, i.e. to feed Siemens Mindsphere with shop floor data, a new out broker will have to be defined in the cep configuration as follows:

```
{
       "rabbitMQHost": "${rabbitMQHost}",
       "rabbitMQUsername": "${rabbitMQUsername}",
       "rabbitMQPassword": "${rabbitMQPassword}",
       "rabbitMQQueueName": "${rabbitMQQueueName}",
       "rabbitMQExchange": "${rabbitMQExchange}"
}
```

Once the Cepheus-CEP is configured, it is time to provisioning the IoT Agent with a new device.

The following example shows a device provisioning process using curl commands made through the provisioning API exposed by the IoT Agent, where devices can be preregistered.

```
        $ curl localhost:4041/iot/devices -s -S --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'Fiware-Service: Hannover' --header 'Fiware-ServicePath: /Messe' --data "@agent.json"
```

This command could be used for the device specified ("*device1*") in the json below, called *agent.json*:

```
{"devices": [
  {"device_id": "device1",
   "entity_name": "Room1",
   "entity_type": "Room",
   "protocol": "MQTT",
   "timezone": "Europe/Madrid",
   "attributes": [
            {
            "name": "temperature",
            "type": "double"
            } ,
            {
            "name": "humidity",
            "type": "double"
            } ,
            {
            "name": "Device",
            "type": "string"
            }
        ]
      }
  ]
}
```

MASAI IoT Agent Configuration file example

This command will create the simplest kind of device, with three attributes: temperature, humidity and device.

Other interesting endpoints offered by the IoT Agent API to recover specific information about devices are:

API: /iot/devices/{deviceId} – GET Operation: Returns all the information about a particular device, in our case, of the *device1* device:

```
$ curl -H 'content-type: application/json' -H 'Fiware-Service: Hannover' -H 'Fiware-ServicePath: /Messe' -vX GET http://localhost:4041/iot/devices/device1
```

API: /iot/devices – GET Operation: Returns a list of all the devices in the device registry with all its data:

```
$ curl -H 'content-type: application/json' -H 'Fiware-Service: Hannover' -H 'Fiware-ServicePath: /Messe' -vX GET http://localhost:4041/iot/devices
```

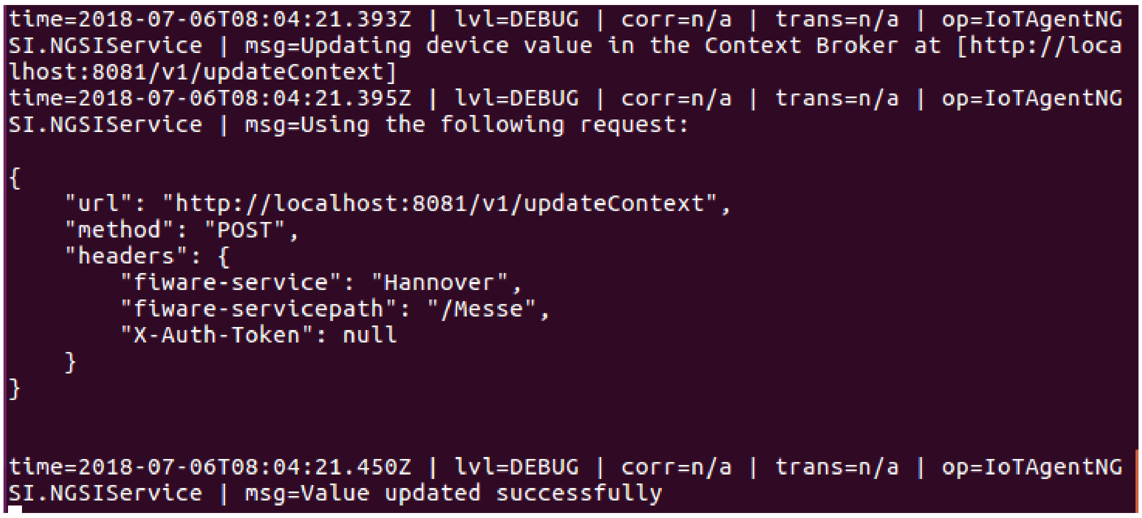
API: /iot/about – GET Operation: Returns information generic information about IoT Agent:

```
$ curl -H 'content-type: application/json' -H 'Fiware-Service: Hannover' -H 'Fiware-ServicePath: /Messe' -vX GET http://localhost:4041/iot/about
```

API : /iot/devices/{deviceId} - DELETE Operation: Remove a device from the device registry.

```
$ curl -H 'content-type: application/json' -H 'Fiware-Service: Hannover' -H 'Fiware-ServicePath: /Messe' -vX DELETE http://localhost:4041/iot/devices/device1
```

You have a detailed description of the configuration and provisioning API here: *https://goo.gl/a1LeKR*


```
time=2018-07-06T08:04:21.393Z | lvl=DEBUG | corr=n/a | trans=n/a | op=IoTAgentNG
SI.NGSIService | msg=Updating device value in the Context Broker at [http://loca
lhost:8081/v1/updateContext]
time=2018-07-06T08:04:21.395Z | lvl=DEBUG | corr=n/a | trans=n/a | op=IoTAgentNG
SI.NGSIService | msg=Using the following request:

{
    "url": "http://localhost:8081/v1/updateContext",
    "method": "POST",
    "headers": {
        "fiware-service": "Hannover",
        "fiware-servicepath": "/Messe",
        "X-Auth-Token": null
    }
}

time=2018-07-06T08:04:21.450Z | lvl=DEBUG | corr=n/a | trans=n/a | op=IoTAgentNG
SI.NGSIService | msg=Value updated successfully
```

IoT Agent console once has been provisioned with a device

If you prefer, you can run the *configure.sh* file, located in the “demo” folder, in a terminal to automatize the CEP configuration and the IoT Agent provisioning. Immediately you can check the different consoles, for the Cepheus-Broker and Cepheus-CEP to check the logs and see the Rooms temperature sent to the CEP and the CEP reacting to the events sending back alerts when the temperature is over 15 ºC.

To do so, move to the demo folder and execute the command:

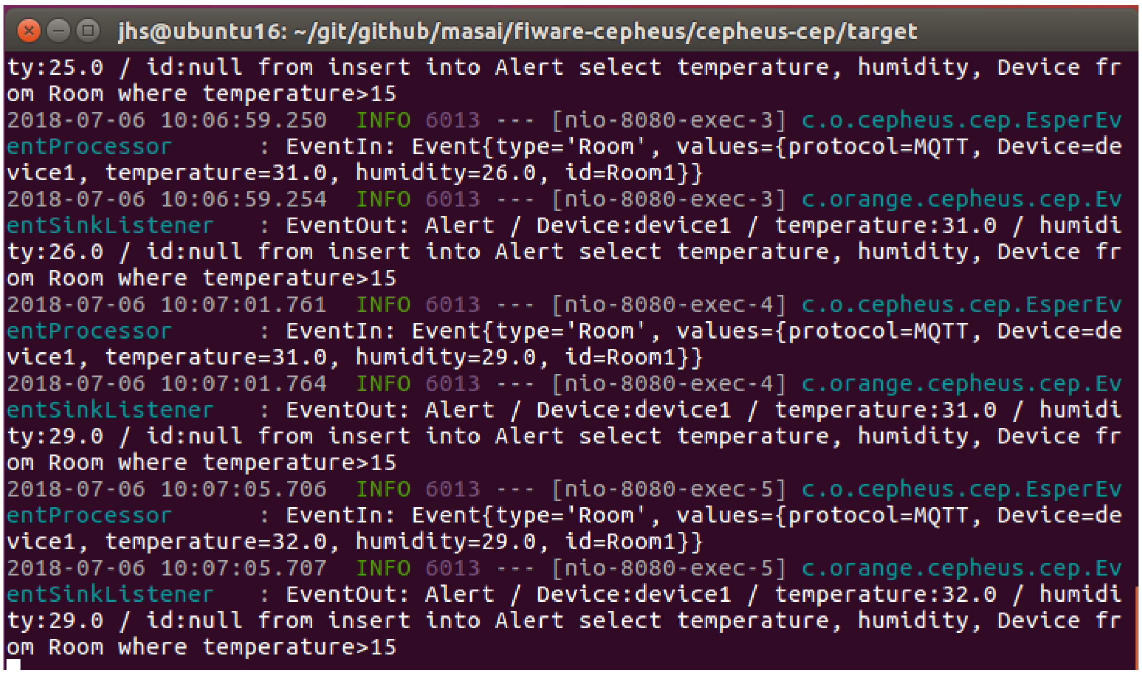
```
$ ./configure.sh
```

The script first sends the *cepheus.json* file to Cepheus-CEP, and then sends the device file to the IoT Agent in order to provision it.

Once the script is executed, we can generate data with for instance a java client using some mqtt client libraries to generate data with a command like this:

```
$ mosquitto_pub -h localhost -t /ATOS/device1/attributes -m
'{"temperature":"32","humidity":"29"}'
```

Going back to the terminal where we it is launched the CEP, and once the simulated data is generated, we should see temperatures and humidity as "EventIn" being logged. After a few seconds, the "EventOut" logs will show the CEP triggering the Alerts when the temperature is over 15 ºC.



```
jhs@ubuntu16: ~/git/github/masai/fiware-cepheus/cepheus-cep/target
ty:25.0 / id:null from insert into Alert select temperature, humidity, Device fr
om Room where temperature>15
2018-07-06 10:06:59.250  INFO 6013 --- [nio-8080-exec-3] c.o.cepheus.cep.EsperEv
entProcessor        : EventIn: Event{type='Room', values={protocol=MQTT, Device=de
vice1, temperature=31.0, humidity=26.0, id=Room1}}
2018-07-06 10:06:59.254  INFO 6013 --- [nio-8080-exec-3] c.orange.cepheus.cep.Ev
entSinkListener     : EventOut: Alert / Device:device1 / temperature:31.0 / humidi
ty:26.0 / id:null from insert into Alert select temperature, humidity, Device fr
om Room where temperature>15
2018-07-06 10:07:01.761  INFO 6013 --- [nio-8080-exec-4] c.o.cepheus.cep.EsperEv
entProcessor        : EventIn: Event{type='Room', values={protocol=MQTT, Device=de
vice1, temperature=31.0, humidity=29.0, id=Room1}}
2018-07-06 10:07:01.764  INFO 6013 --- [nio-8080-exec-4] c.orange.cepheus.cep.Ev
entSinkListener     : EventOut: Alert / Device:device1 / temperature:31.0 / humidi
ty:29.0 / id:null from insert into Alert select temperature, humidity, Device fr
om Room where temperature>15
2018-07-06 10:07:05.706  INFO 6013 --- [nio-8080-exec-5] c.o.cepheus.cep.EsperEv
entProcessor        : EventIn: Event{type='Room', values={protocol=MQTT, Device=de
vice1, temperature=32.0, humidity=29.0, id=Room1}}
2018-07-06 10:07:05.707  INFO 6013 --- [nio-8080-exec-5] c.orange.cepheus.cep.Ev
entSinkListener     : EventOut: Alert / Device:device1 / temperature:32.0 / humidi
ty:29.0 / id:null from insert into Alert select temperature, humidity, Device fr
om Room where temperature>15
```

Cepheus-CEP console showing EventIn and EventOut info

1.3.3.4 MIDIH MindSphere Custom Agent Connector Configuration and execution

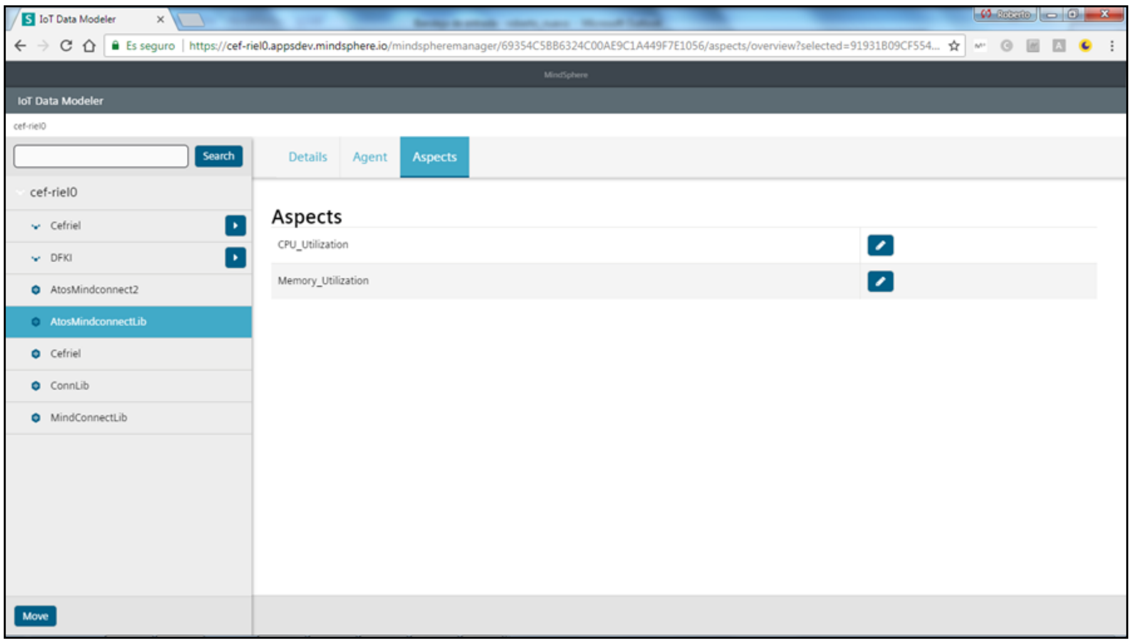
Once the Onboarding Process, necessary for connecting a new Device to MindSphere, that comprises the configuration of the device and assignment to an Asset ID, has been completed through the *IoT Data Modeler App*. It is necessary to perform different steps in order to start working with it:

1. Configure the industrial data of the Asset, for this we need to configure the Aspects, which are combined, pre-configured data and form the context for the evaluation of industrial processes.
2. Define the Data Sources, a Data Source is a physical element of a device that can be monitored by MindSphere. For example, an OPC UA Server
3. Define the Data Points or Tags, Data Points or Tags are all recordable values of the Aspects, e.g. sampling rate, temperature, pressure.

Summarizing, Assets generate Aspects within the industrial process and forward these Aspects to MindSphere. Then, the analysis and evaluation of complex industrial processes are performed on the basis of these Aspects.

These previous steps are done from the MIDIH MindSphere Custom Agent Connector as well as the configuration of the RabbitMQ and the specific queue to which it will subscribe to receive the data for the different Assets.

The configuration of the Aspects for Data Sources (OPC UA server) is carried out in the *IoT Data Modeler App*. Through this application we can show the Aspects and their variables created.

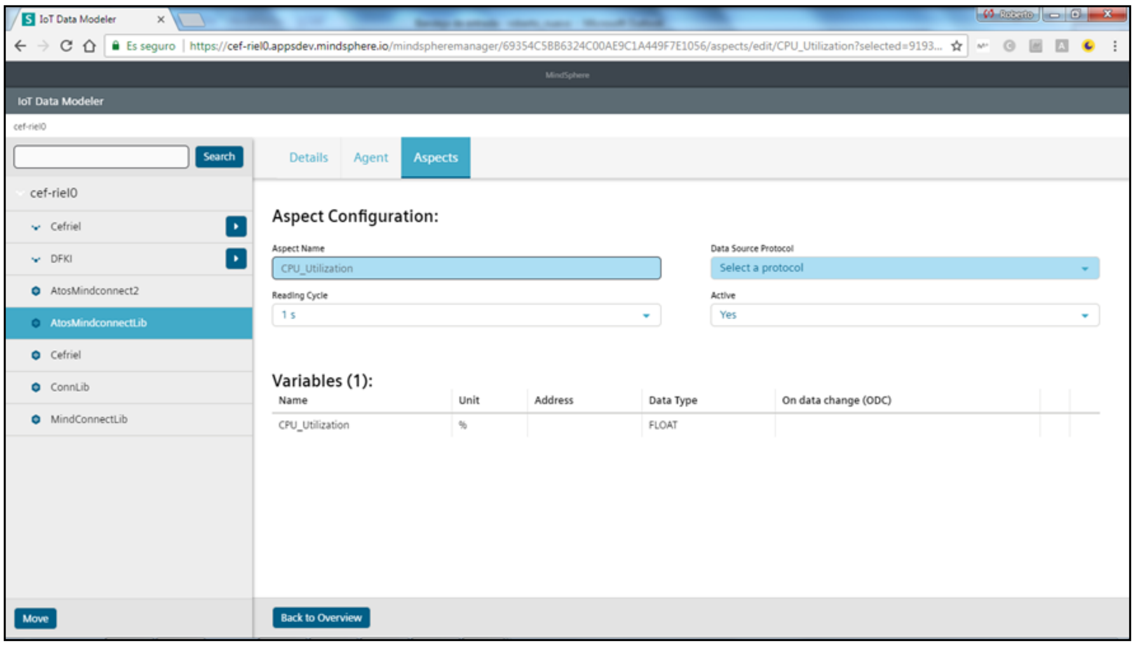


Aspects

CPU_Utilization

Memory_Utilization

Figure 16 – Aspects in the IoT data Modeler MindApp



Aspect Configuration:

Aspect Name: CPU_Utilization

Data Source Protocol: Select a protocol

Reading Cycle: 1 s

Active: Yes

Variables (1):

Name	Unit	Address	Data Type	On data change (ODC)
CPU_Utilization	%		FLOAT	

Figure 17 – Aspect Configuration in the IoT data Modeler MindApp

The values received from the defined variables of an Asset can be viewed through the Fleet Manager MindApp.

The Fleet Manager MindApp has the option of exporting the Aspects of the Assets from MindSphere thought a specific expansion called “Export”. This offers the following functions:

- Selection of the desired time period for exporting the Aspects
- Saving the exported data to a CSV or JSON file

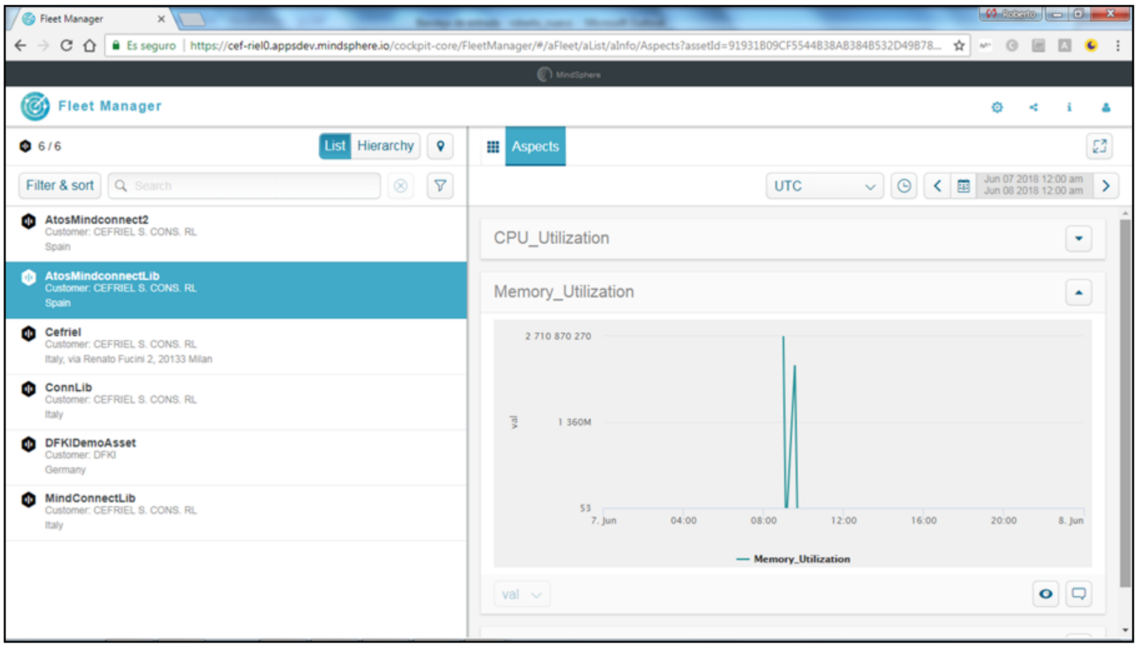


Figure 18 – Visualization of the values of a variable/tag from an Aspect in the Fleet Manager MindApp

To start working the MindSphere Custom Agent Connector, move to the folder where it is located and execute the command:

MIDIHMindSphereCustomAgentConnector

1.3.4 Licensing

MASAI and the MIDIH MindSphere Custom Agent Connector are released under the **GNU Affero General Public License v3.0**[23].

[23] http://www.gnu.org/licenses/agpl-3.0.en.html

Annex D. Factsheet for Industrial IoT and Analytics Platform – Apache line

1 Industrial IoT and Analytics Platform – Apache line

1.1 INTRODUCTION

The Industrial IoT and Analytics Platform is a SaaS solution for resolving complex industry cases using the power of data analytics. It implements a complex, end-to-end data processing pipeline, starting from the collection of data until the visualization of the complex data-driven models.

The main advantage is its low barrier nature: the user should provide "only" an access to a relevant data source (or a set of sources) and the Platform will be able to generate useful data-driven services. Indeed, in the nutshell of the data processing is an unsupervised learning approach that minimizes the involvement of the domain experts. However, the domain knowledge can be injected in the various steps of the processing pipeline.

The Platform is based on the Apache technologies.

1.2 FUNCTIONAL DESCRIPTION

1.2.1 Overview

In the first prototype we focus on the clustering process, i.e. on the analysis of the system behavior based on past data and the creation of the model of its behavior (usual, unusual). This model is used for a better real-time situational awareness, e.g. for the detection of different anomalies.

1.2.2 Architecture and Specification

The most important functionalities are:

1. collection and transfer of all types of data (Data Ingestion)
2. real-time analytics that processes huge streaming data in order to predict and detect events based on underlying patterns and correlations (processing Data in Motion) and
3. storage layer for persisting all type of data, like past data, meta-data, models (Data Persistance)
4. data-analytics services on multidimensional and complex data, including exploratory analysis, multivariate analysis, predictive analytics and deep learning (processing Data at Rest)
5. visualization services to enable users to contextualize, understand and apply results for better decision making.

There is also the need for the orchestration of the tasks in order to achieve a common goal (Workflow Management) – see Figure 19

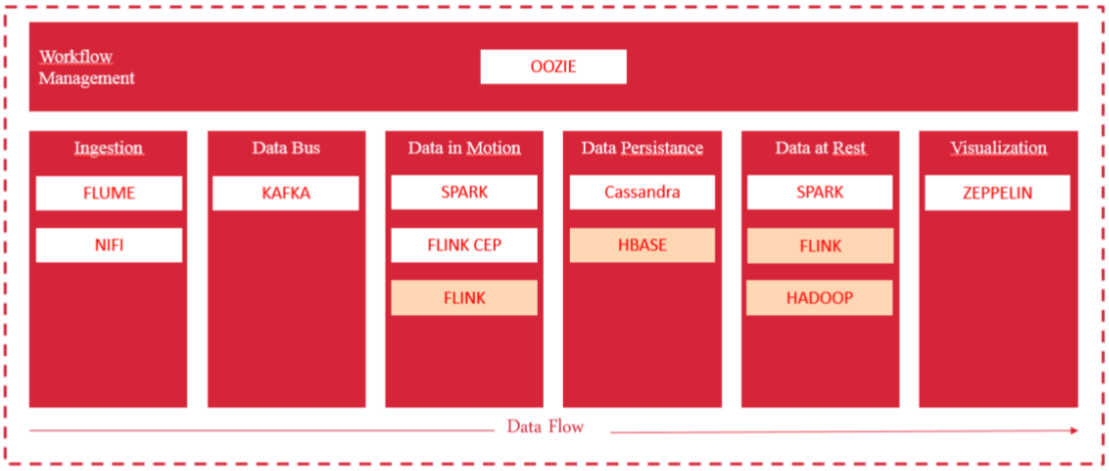


Figure 19 – Industrial IoT and Analytics Platform Pipeline

In the following, we provide the description of these layers.

1.2.2.1 Data Ingestion

This is the integration layer that provides flexible and scalable integration of components in the system. It is based on Apache Kafka that is a persistent distributed message broker, also acting as a queue. It's the main backbone of the event-driven components of the system, transmitting signals, triggers and notifications across the backend. Messages are always binary serialized, so any sort of message can be put into the message payload. However, we will use always JSON messages.

Adapters are components responsible for retrieving and parsing raw input files from clients and storing the resulting instance. Before storing the instances, anomaly detection is performed using the Anomaly detector and based on the result an instance can be stored as usual or unusual (anomaly).

1.2.2.2 Data on Move processing

This is the layer responsible for the real-time data processing (analytics)

Detection result worker listens the broker for messages regarding detection algorithm execution, stores those results and triggers certain side-effects (if needed).

Anomaly detector performs anomaly detection on new instances based on previously generated (newest) model. It receives questionable instance and the model, compares them and returns a detection result. More on the Anomaly detector API will be given in the following.

1.2.2.3 Data Persistence

Choosing and designing the database required a lot of thought in order to support efficient storage and retrieval of many instances of time series data, as well as all generated results. The storage model represents a crucial part of the system and will be described in this section.

We found these to be some of the important aspects to consider when choosing a specific database solution for our storage:

- we are dealing with large amounts of data with a loose structure, comprised only of a few different entities

- there isn't a need for typical linking and joins found in relational database solutions
- majority of data that will be stored are parameter time series with many values which do not fit well with the relational model, where we would be forced to store these values in separate rows and later use joins to retrieve them all
- we would benefit from easy horizontal scalability in the case of clients with a lot of data

Mentioned aspects and already having experience with successfully storing time series in Cassandra guided us towards using it in this system as well.

Another solution is HBase, which is the Hadoop database. As so, there is no worries about integration like in case of Cassandra. HDP, Cloudera, or PaaS providers like Amazon offer HBase hand in hand with Hadoop. Moreover, it is a practice to run MR jobs with HBase as input/output. It is a mature project, which means solid documentation and has active community, like Cassandra. It is column oriented and it is not strange to use it for time-series storage.

SQL is an ordinary relational database. In practice this may represent multiple separate databases, or even separate database servers altogether. It should store user account data, project information, project settings, file, model and test metadata.

1.2.2.4 Data at Rest processing

This layer provides different execution engines for scalable, distributed and parallel algorithm execution. It implements different machine learning and pattern recognition algorithms on top of processing layer and technologies such as Hadoop, Spark and CEP;

Hadoop is the pivot point of the system which runs parallel, distributed batch training jobs using map-reduce algorithms. It is directly controlled through Oozie and data used as input for the jobs is stored in HBASE.

Clustering

All heavy work of model training is done here. Clustering is an unsupervised machine learning method which identifies existing patterns in the data and groups instances based on their similarity. As the result of the clustering procedure a model and a set of clusters are generated. Clusters can be used to analyze existing patterns in the data, while the model can be used in real-time, for real-time anomaly detection. The clustering procedure is performed periodically and, in each iteration a new model and set of clusters is generated.

Hadoop is the pivot point of the system which runs parallel, distributed batch training jobs using map-reduce algorithms. It is directly controlled through Oozie and data used as input for the jobs is stored in HBASE.

Job Management Service

Hadoop (training) job management will be controlled by this service. When a user schedules periodic training using the web portal or starts training immediately, the call will be delegated to this service. The service will keep track of the scheduling and accordingly configure Oozie.

Data worker (and workers in general)

Workers in the whole project are in general components with tasks simple to describe, and usually do one thing and one thing well. Although the workers are drawn as separate items, it is not important for them to actually be implemented like that. They may be separate actors, applications, classes which can be decided at implementation time.

The data worker has the task to read files from the Kafka broker (both training and testing files) and load the file raw payload and metadata to HBASE and the relational database, respectively. The exact procedure will be explained in detail in the next chapter.

Job progress worker

Job progress worker is a worker listening the broker for training state and progress messages, and the worker will be persisting those messages to the database or reacting activating certain side-effects on status/progress points (if needed).

1.2.2.5 Interaction

User interaction layer is used for data ingestion (most often through REST services) and presentation of data and enables the user to interact with the system and experiment with different algorithms and parameters through the web portal;

The web portal is the main visual interaction point between the client and the system. The web portal is there for the user to set up their account, project, observe detection results and training status. The web portal uses multiple backend services for its operations and has relatively thin operational logic.

This web service interacts using a set of REST endpoints with JSON payloads with both the web portal, and the client. The clients will mainly utilize calls for uploading training and detection files. Other endpoints are for tracking training job progress, checking file processing status and observing detection results.

User Web Service is used for user management only. That includes: account creation and account management, permissions enforcement, billing and similar issues.

1.2.2.6 Workflow management

Oozie is a server-based tool for organizing Hadoop workflows (setting up jobs, starting, stopping, scheduling etc). It's necessary for managing Hadoop jobs through an API. It will execute tasks forwarded through the job management service, which abstracts its options.

1.2.2.7 HW/SW Prerequisite

Hardware-related prerequisites are presented in the following table:

Compute/HW capacity	
CPU (Ghz)	1x8core(2Ghz), 1x16core(2Ghz), 3x16core(2Ghz)
RAM (GB)	1x16GB, 1x16GB, 1x32GB
Distributed SSD (GB)	1x500GB, 1x1TB, 3x1TB
Distributed HDD (Magnetic)	/
Outgoing data transfer (GB)	50GB
IP (Number of IPs required)	5

VLAN (number of VLANs required)	1

1.2.2.8 MIDIH Specific developments

The data-processing (Apache) pipeline is the main contribution. It enables the development of an end-to-end solution for complex industry cases.

1.3 MANUALS

1.3.1 Installation

The system runs as Software as a Service. It requires a proper set-up in the computing (cloud) infrastructure (Hadoop, Oozi)

1.3.2 Configuration

There are three main configuration issues, which should be discussed separately for each use case:

1) How the data will be transferred from the data sources to the processing pipeline
2) What kind of adapters is required for the transformation of the collected data in a proper format
3) What is the most appropriate structure of the data storage

1.3.3 Usage

1.3.3.1 File upload

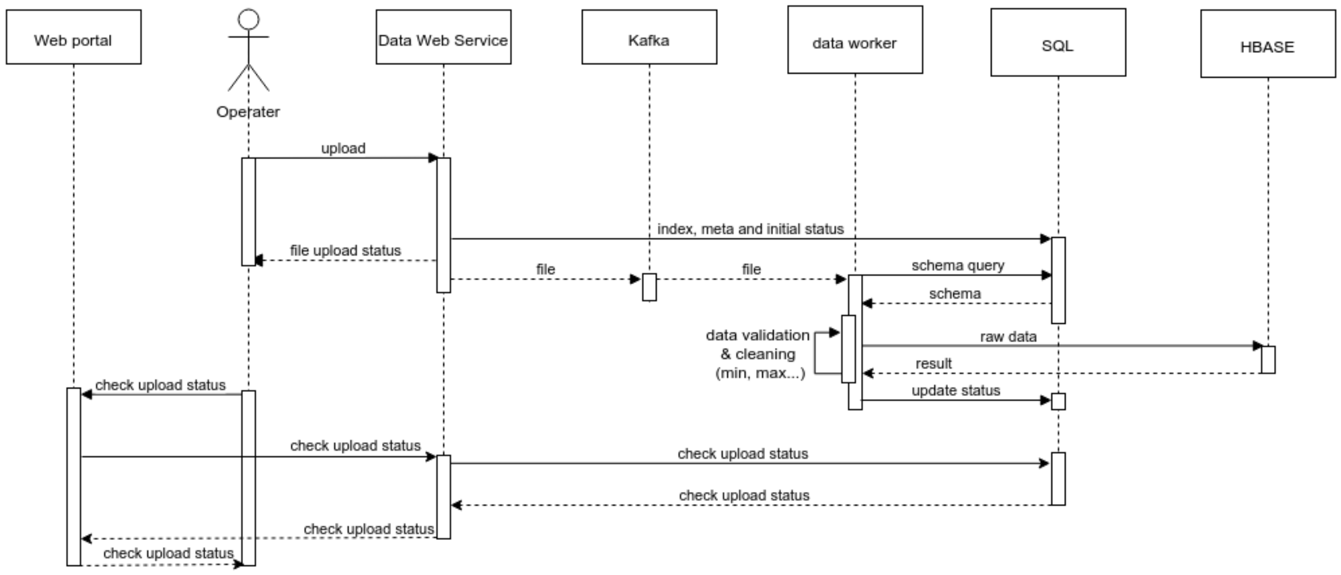
First and foremost, the user should be able to upload a file to the system, either for training or testing. Presumably, that will be done using a REST API, and in a more or less automatized way on the client side. Since the system should be ready to accept a larger number of files ("big data"), an asynchronous method has been developed. It's shown in the figure below.

Each kind of upload (training and detection) is executed in the same way. The operator, in the broadest sense as an automated client or human input, uploads the file to the *Data Web Service*, using a REST API. As soon as the file is received from the service and stored in some temporal storage, the user gets an approval and an id which bill be used for later reference.

However, the file is not exactly stored now in the operational databases. Currently the file is only in the temporal storage, which is relatively fast and simple. The web service puts the file into the Kafka queue for further processing, and also writes an entry into the relational database that a file has been received and it should be processed. That way we can track each file, and its state (has the processing failed, reason of the failure, failed for an internal reason, parsing error, corrupted file etc.). The data worker reads files from the queue, one by one or in smaller batches, and steadily loads those files into the relational database (where mostly metadata about the files is stored) and the raw data from the file into HBASE. The worker shall accordingly change the file status in the relational database.

Before writing the files, the worker will read the file schema from the relational database, for that project. The schema describes the format of the file, and imposes certain limitations, such as the minimal and maximal values that can be present for each parameter in the file. The worker will execute filtering and cleaning on the file according to that schema.

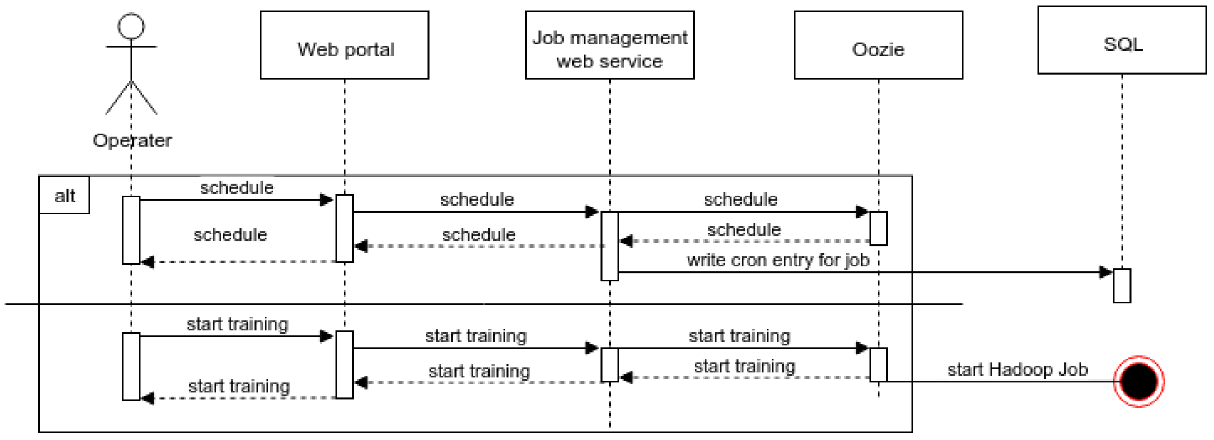
The worker can also publish a message to the broker as a trigger to any component interested that the file has been stored. This message is described in a later section.



To confirm what has actually happened with the uploaded file, the user may and should (at a bit later time) go to the web portal and check the state of the upload. The web portal will call the data web service which will relatively simply read the state from the relational database.

1.3.3.2 Training and training scheduling

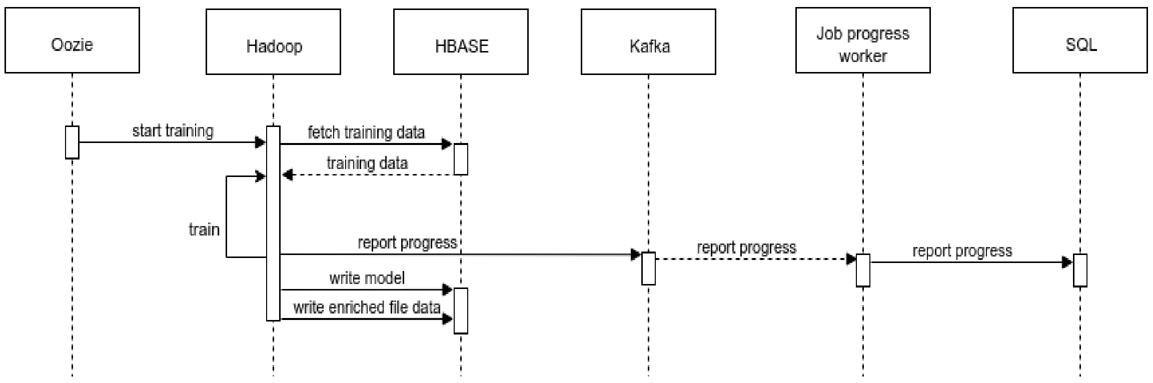
The user of the system will be able to schedule the training to start at a specified time or recurring or even start it one time only. Those functions will be setup from the portal, and the detail sequence is shown in the figure.



A user can choose one out of two operations with the scheduling mechanism. They can schedule using the web portal, which will use the *job management rest service* backend to schedule with

Oozie, and also write that information to SQL to the appropriate project configuration, as a *cron*[24] string.

Alternatively, we can also use Oozie to start the job immediately. Nevertheless, after the instruction sent to Oozie, the training process is the same, as in the figure.

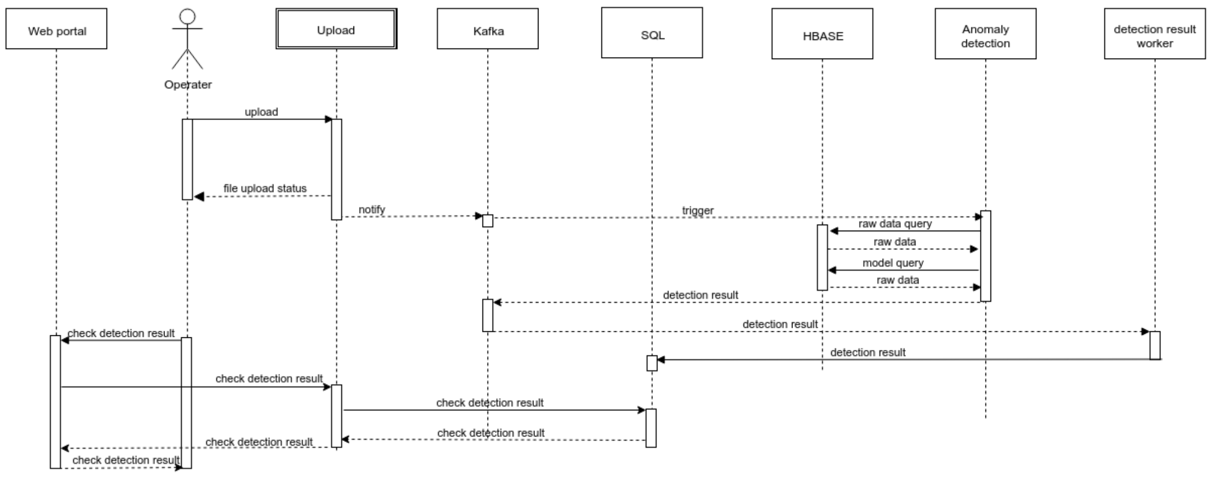


When Oozie hits a specific time to start the training process, it will send an instruction to Hadoop with the job configuration. Each time the job progress changes, starts, stops, fails or is somewhere in between (for example on each 5% progress) it will publish a message on the broker with that job id and the current progress. That message can be consumed by any interested party, but most importantly by the job progress worker which will update that information to the job record in the relational database. That status should be observable through the web portal for the user to have a sense of progress.

[24] http://www.nncron.ru/help/EN/working/cron-format.htm

1.3.3.3 Detection

Naturally, the system should provide the option to detect anomalies in the client's files.



Please note that the *Upload* task is a complex one and described on the *File upload* chapter. After the upload sequence is finished, a trigger is dispatched using Kafka to the *Anomaly Detection* component that a file is ready for detection. The trigger contains elementary file information such as file id and project id. The anomaly detection component can read the project configuration (i.e. algorithm, distance) based on that reference, load the model from HBASE and apply the algorithm on the new file. The detection result is published to the broker for any interested component, but most importantly, the *detection result worker* which specifically listens to these messages and writes the detection result into the database.

The user may check the results of the anomaly detection progress from the portal at a later point when the detection is finished, which should in general be in a very short time since the anomaly detection does not take too long (matter of seconds).

1.3.4 Licensing

The Industrial IoT and Analytics Platform is released under the Apache License 2.0[25].

[25] https://www.apache.org/licenses/LICENSE-2.0

Annex E. Factsheet for Industrial IoT and Analytics Platform – FIWARE line

1 Industrial IoT and Analytics Platform – FIWARE line

1.1 INTRODUCTION

The Industrial IoT and Analytics Platform is a decoupled, stateless and micro-service-based solution that data collection, transformation, storage and processing in industry focused scenarios. The platform has been designed using a set of well-defined components provided by FIWARE, commonly named Generic Enablers (GE), and trying to cover the data flow, storage and processing.
The platform's main advantages are:

- its open source nature, most of the components that made up the platform belongs to the FIWARE foundation, therefore are completely open;
- its simply but potent configuration, the platform can be launched through Docker rewriting a reduced set of configuration files;
- and its capability to be adapted to custom scenarios the services provides a huge set of properties allowing the developer to set a personalized scenario.

1.2 FUNCTIONAL DESCRIPTION

1.2.1 Overview

This first version covers the main functionalities of both, Data-in-Motion and Data-at-Rest, lanes. The platform is completely based on FIWARE components, taking as the codebase the FIWARE IoT Stack architecture. Currently, the services deployed will cover the real-time data scenarios, especially the IoT related, in a more accurate way than the data processing. In next releases, the services and uses cases will evolve to cover more industrial related scenarios.

1.2.2 Architecture and Specification

As depicted in Figure 20 the main expected functionalities are directly related with FIWARE components. A brief description of the capabilities of each one of the functions presented can be found in the equivalent section in Annex B. The following sections aim to provide the description of these layers.

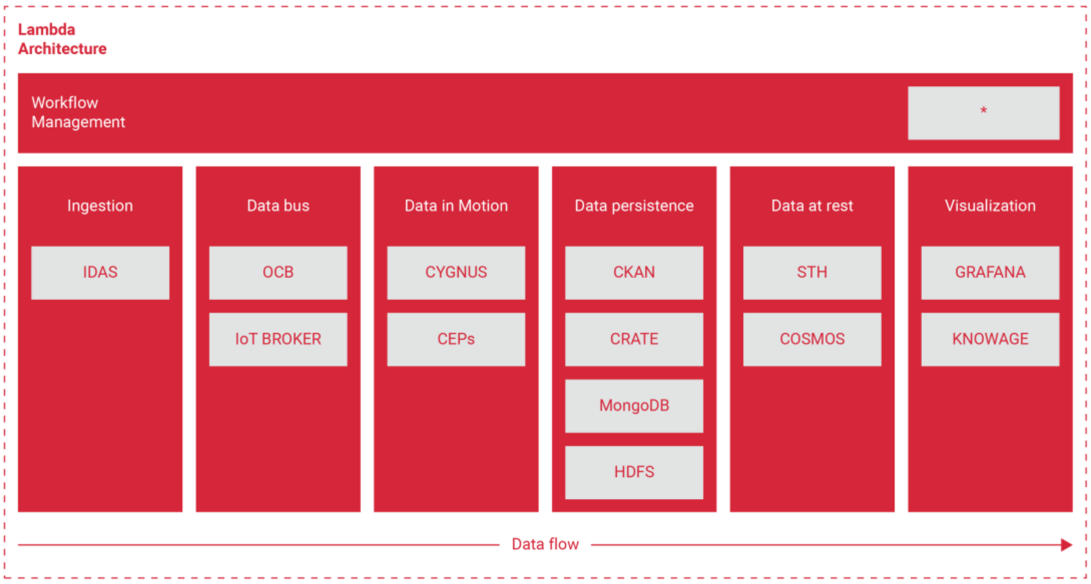


Figure 20 – FIWARE Pipeline for Industrial IoT and Analytics Platform

In the following, we provide the description of these layers.

1.2.2.1 Data Ingestion

The Data Ingestion layer must provide an easy but flexible way to gather and collect data from non-FIWARE devices allowing other services or consumer to interact with them in a transparent fashion. Currently FIWARE provides different IoT Agents, also named IDAS, allowing MQTT, HTTP or OMA LwM2M devices to be interoperable within the FIWARE architecture.

As described, the IoT Agents are in essence bridges that performs translations between, commonly, IoT related protocols to OMA NGSI, the protocol used by the majority of FIWARE components.

The IoT Agent component is made up of two servers and two clients, for the IoT protocol and OMA NGSI. As double server it can receive requests from FIWARE components and data from non-FIWARE devices and as double client it can request data from non-FIWARE devices and receive requests from FIWARE components, such as Orion.

1.2.2.2 Data in Motion Processing

The component responsible for the real-time data processing, in this context called Data-in-motion, in FIWARE is Orion Context Broker. Orion Context Broker is in charge of manage all the lifecycle of context information including queries, registrations and deletions. Furthermore, Orion Context Broker allows the consumer to subscribe itself to a context information. Orion Context Broker will notify the consumer when any modification occurs.

More information about the component can be found in the following link (https://fiware-orion.readthedocs.io/en/master/).

This context information is structured following the formats of FIWARE data models. These data models aim to achieve portability among applications. More information about data models can be found in the following link (https://www.fiware.org/developers/data-models/).

1.2.2.3 Data at Rest Processing

This component allows managing of historical raw and aggregated time series information of entity attributes. The historical raw shows the values that the attribute of an entity has taken over time, and an aggregate shows the minimum, maximum, sum, or sum of the squares of the values. Both taken by an attribute in a determined period and the times it has been recorded.

Knowage is an analysis tool that allows representing data processed in a graphical view. Knowage provides adapters with some databases for data storage and data access through REST services among others, although these are the most interesting in the FIWARE infrastructure. With the data collected, Knowage can generate cockpits drawing tables, text, graphics, and histograms of obtained values.

The graphic interface allows: users management, roles, views, menus, although the configuration of data sources and data set (database connection and data selection). Knowage is an evolution of SpagoBI.

1.2.2.4 Data Persistence

The persistency of data is presented in FIWARE in an almost transparent way for the developer. Three components are in place to achieve data storage:

- Orion Context Broker, a service that allows to manage context information to provide information to clients and request information to the devices.
- Cygnus is a service that allows the user to persist the data Orion is collecting and notifying into databases. Cygnus is based on Apache Flume.
- and of course, the database you choose for your deployment. As depicted in Figure 6 the current platform will provide adapters to MongoDB, CKAN, CrateDB and HDFS.

In order to clarify, Cygnus obtains data from Orion through an Orion's subscription. Using the REST API that Orion provides, the developer can create a subscription, when a new value is notified to Orion, it sends this data to Cygnus. Finally, Cygnus will insert this data into the configured databases.

1.2.2.5 Interaction

The easiest way to interact with the platform is through the tools that provides a graphical interface. Knowage and Grafana are those tools. Both allow developers to generate queries to database showing data result graphically. The connection between tools and database is over data source, these have drivers to connect with more recognized databases. Once connected, the queries can be perform directly providing real-time and big data representations.

The services offer an REST API allowing the user to interact with the platform's components in a simple and well-known fashion. The REST API uses the OMA NGSI representation and uses JSON format in the body section for send and receive requests.

Other interesting way of communication and integration with the platform is through the publish/subscribe mechanism. One of the most valuable features of Orion Context Broker is its subscription mechanism. This allows a consumer to subscribe itself to context information for receive asynchronous notifications when entities are updated or created. All of this can be configured in a fine-grained fashion specifying the entities, types or attribute in which the consumer is interested. Orion Context Broker is in charge of create, update, retrieve and delete subscriptions. Those are stored in a MongoDB database.

1.2.2.6 HW/SW Prerequisite

The software requirements are simply Docker. Even when all the tools can be installed over the most popular linux distro, Docker and its tool docker-compose provides a simple and transparent way of deployment.

On the other hand, an estimation of the hardware resources required for running the presented platform can be found in the following table:

Compute/HW capacity	
CPU (Ghz)	8core(2Ghz)
RAM (GB)	32GB
Hard disk Storage (GB)	2TB
Outgoing data transfer (GB)	50GB
IP (Number of IPs required)	1

1.2.2.7 MIDIH Specific developments

There are no specific developments so far. The main contribution is the set up and configuration provided to create a common, reliable and valuable platform starting from separated microservices.

1.3 MANUALS

1.3.1 Installation

The platform runs using Docker as deployment tool. In this way, the solution can be launched and tested in multi-platform environments, from a cloud architecture to a desktop computer.

All the files required to launch will be provided, and the configuration parameters are showed in the following sections.

1.3.2 Configuration

1.3.2.1 Data Ingestion

The IoT Agents can be deployed in docker through a file called “docker-compose.yml”. The minimum valuable scenario must have at least an Orion, a database and an IotAgent configured. It is necessary to have installed docker and docker-compose in the computer.

The IoT Agents offer a REST API that allows to perform CRUD operations directly to the device. The IoT Agents can be configurated in two ways.

- First, using its REST API for configure it. In the body request must have the configuration of IoT Agent in JSON format.
- On the other hand, a previous configuration can be performed, at launch time, through a file called “config.json”. This file must contain configuration values such as the protocol and how to map the data from device protocol representation to OMA NGSI.

Other values that must be configured in the component are the IoT related protocol sever port, which IP and server protocols they will use, the delayed observation time, the formats that the protocol accepts, the writing format and types of devices.
The mapping of devices data configuration in the NGSI representation needs the context broker, the server, the type of device registry, how to connect with database, the mapping from device protocol to NGSI, the service, the subservice, the provider url and the device registration duration.

1.3.2.2 Data in Motion processing

Data representation
An entity represents any identified physical entity. An entity is formed by set of attributes following a data model FIWARE format. These attributes have a specific value that can be updated at any time. Orion Context Broker is in charge of create, update, retrieve and delete an entity or list of entities. Furthermore, it allows managing entities by their type, attributes or geographic position.

Data isolation
Orion Context Broker stores data separating it into services providing a lightweight multiservice model. We can define in which service the entities will be stored by using HTTP header '*Fiware-Service*' in the request. If we don't define the service in the header, the default service '/' will be used.
Orion Context Broker also supports defined scopes. From the '*Fiware-ServicePath*' sent to Orion Context Broker as a header in the HTTP request, we can define the scope in which we are going to work. Those scopes can be hierarchical.

Component configuration

A set of options are allowed when launching the Orion Context Broker in command line for the management of the database, activating the log, managing the connection with Orion Context Broker, among others. We can see the different options in the following link (https://fiware-orion.readthedocs.io/en/master/admin/cli/index.html).

In addition, Orion Context Broker can be launched as a Docker service. To launch it, Orion Context Broker needs to be connected to a MongoDB database, which can be added of two ways:

- We can launch a MongoDB image together with Orion Context Broker from a Docker Compose.
- Orion Context Broker can be launched as a docker container independently to MongoDB database. The database may be running on localhost, located in another docker container or running on a different host.

1.3.2.3 Data at Rest processing

The way of configuring STH through docker needs to set the following environmental variables:

- STH_HOST, the STH host will be the IP where start the service,
- DB_URI, DB_USERNAME, DB_PASSWORD, those parameters are in charge of connect STH with the database.

Other useful variables like SHOULD_STORE or TRUNCATION_EXPIRE_AFTER, can help to store only the information necessary. More information about STH configuration here in the following link (http://fiware-sth-comet.readthedocs.io/en/r5_fiware/running/).

Knowage must be executed with MySQL, the simplest way is with docker. Docker permits connect Knowage to MySQL using containerized versions of both services. The values that must contain in the configuration file are: MYSQL_USER, MYSQL_PASSWORD, MYSQL_DATABASE, MYSQL_ROOT_PASSWORD. The environment variable to keep in mind is PUBLIC_ADDRESS, without it settled up, Knowage will show error messages. All remaining settings are established in the graphical interface (users, roles, menus).

1.3.2.4 Data Persistence

Cygnus is based on Apache Flume and lets you configure their agents through two files: *cygnus_instance.conf* which configures its not Flume parameters and is only necessary when Cygnus acts as a service and *agent.conf* to configure all Flume parameters internal to the Cygnus configuration. The *agent.conf* configuration file is required.

The Cygnus Agent's parameters can be configured:

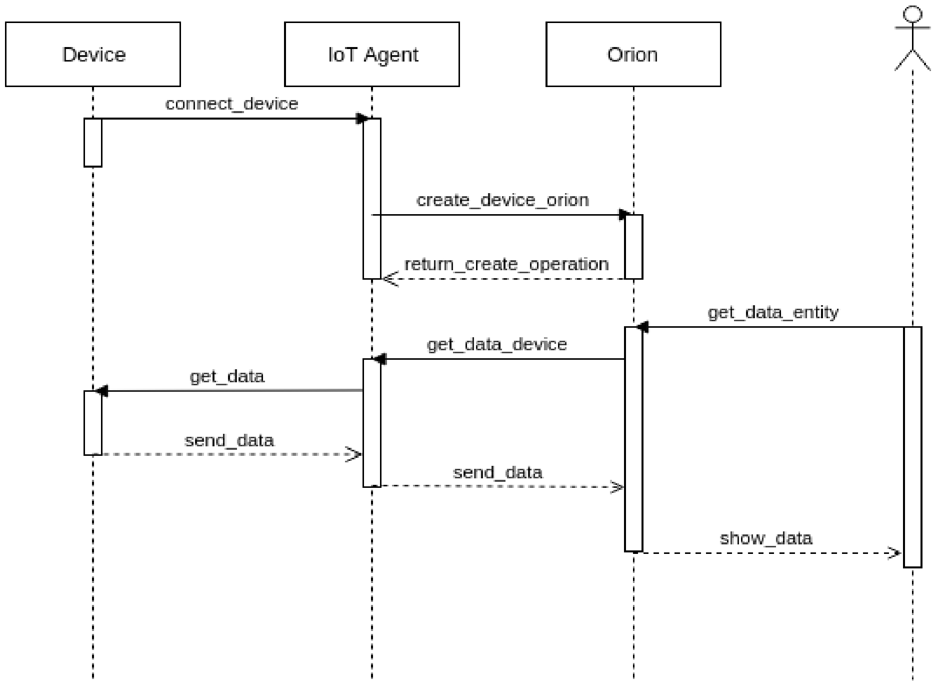
- using Cygnus's REST API, methods are provided to retrieve, create, delete and update Flume's parameters.
- through the *agent.conf*, using this file parameters like the connection with the HTTP source, its handler and its interceptors, the connection with MongoDB and its channel for communication and the connection with STH and the channel for its communication can be configured.

1.3.3 Usage

1.3.3.1 Data Ingestion

Currently Ultratlight 2.0 and JSON are supported regardless they use HTTP or MQTT as transport protocol. Besides this, OMA LWM2M and Thinking Things Open are also available protocols

IoT Agent needs Orion Context Broker and a database to work properly as defined in configuration section. The following diagram serves as a flow example of the component usage. First, can be seen how a non-FIWARE device is connected for the first time to the FIWARE architecture, after that the diagram shows how a user can request data from Orion Context Broker, and the process flow followed until the data is retrieved.



1.3.3.2 Data in Motion Processing

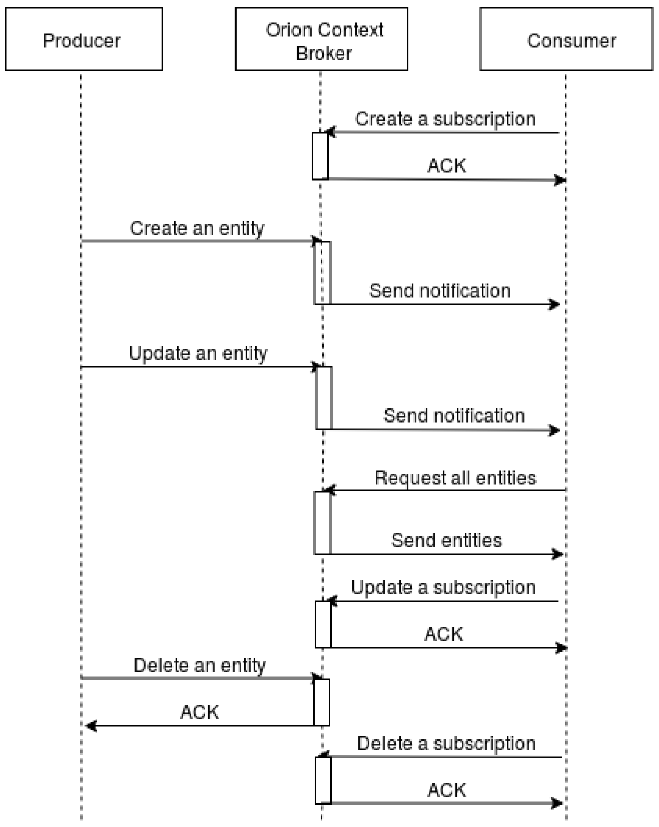
Orion Context Broker could be launched using Docker. For this, there are three possibilities:

- from a Docker Compose file declaring a MongoDB container,
- launching Orion Context Broker in a container connecting to a MongoDB instance independent of the container,
- and building a Docker image to have more control over the Orion Context Broker container.

For more information about the launch you can access the following link:

Once Orion Context Broker is launched, it is possible to make a set of requests to the REST API. Some of these requests are depicted in the figure above in this section. In this diagram, a consumer can create, update or delete a subscription to Orion Context Broker.

In case a consumer creates a subscription to Orion Context Broker to receive information about the creation and update of a specific type of entities, when a creation or update of an entity by a provider occurs, Orion Context Broker will send the consumer a notification with the changes. Two actors are commonly to illustrate the Orion Context Broker's flow: consumer and provider. The provider will be in charge of create, update or delete entities. On the other side, the consumer will receive notification if its subscripted to the entities modified or can request the entities in a synchronous way through the REST API.
In addition, Orion Context Broker offers the possibility of returning the response in different formats regarding the two version of OMA NGSI currently implemented.

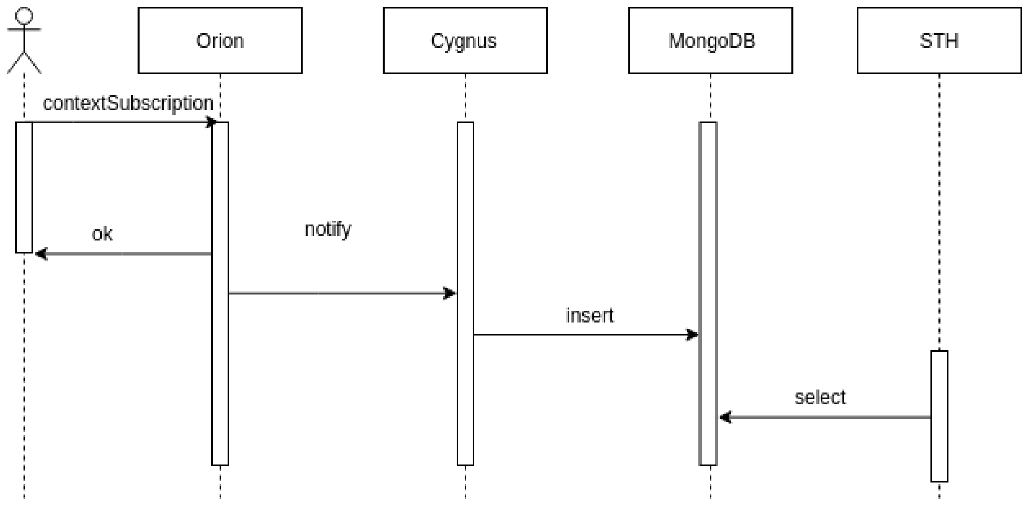


1.3.3.3 *Data at Rest processing*

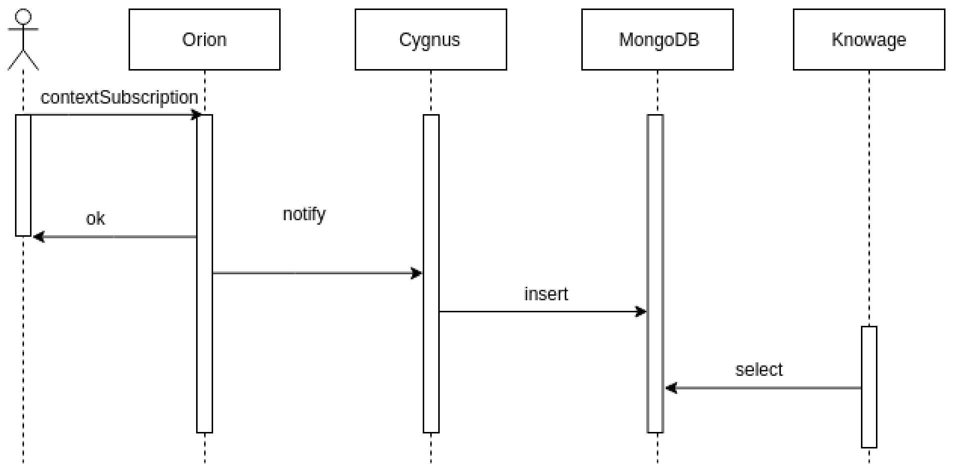
STH must be launched with other components like Cygnus and MongoDB in order to work properly. Using this configuration STH values generated can be retrieved directly from the MongoDB instance.
In order to better understand the scenario, a flow example is depicted in the figure below. Cygnus must be linked with STH and MongoDB generating raw and aggregated data that will be stored in the database. This generated data has a prefix that STH should be expecting in the database, in this way STH through it REST API using the values stored in the MongoDB instance.

The REST API makes possible to get raw data between two dates, before a date or after a date and the same with aggregated data. The API response with a JSON (NGSI). STH is compatible with both NGSI9 and NGSI10, the standard to RESTful API by FIWARE.

As can be seen in the next diagram, a client asks to Orion to establish a subscription, REST API of Orion, responds OK; every time that an entity of Orion changes, Orion will notify Cygnus and this will be added data to the MongoDB instance, then STH can access to information with a database query.

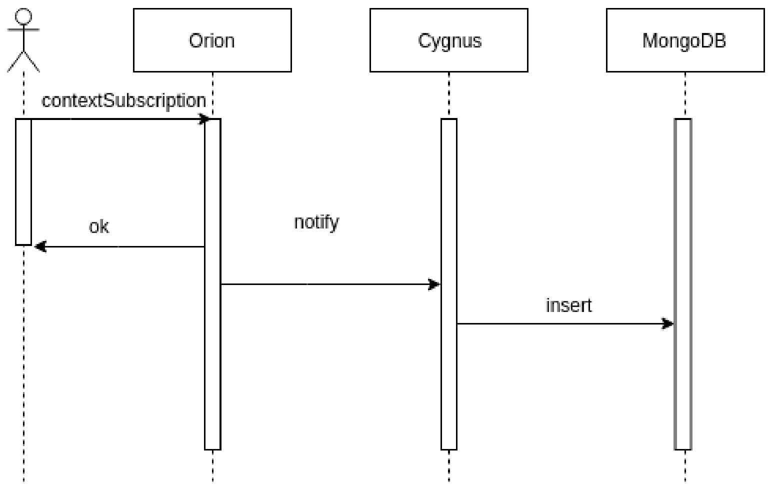


Just like as STH, Knowage have functions to show data stored on database, as can see on the diagram below. To launch knowage, it is necessary to run MySQL, so that all menus, users, documents or any changes made through the graphical interface will be stored in this database. Regardless of which database is used for data collection for subsequent sampling. Once this is done it is possible to launch the containers. More information about launching the container can be found here: (https://hub.docker.com/r/knowagelabs/knowage-server-docker/)Futhermore, the data received as an Orion's entity, is detected like NGSIv2 provided of course is activated the option from interface. All functions are performed through the graphical interface, once the services have started, Knowage is accessible with the next URL "<HOST-IP>:<PORT-INDICATED>/knowage". Users can view documents or dashboards generated by administrators. There are different roles that allow restricting or allowing access to the different parts of the interface; this interface is very powerful and complete. Knowage do not possess REST API. More information about how to use this tool can be found in the following community manual of use about graphical interface of Knowage:
(https://download.forge.ow2.org/knowage/Knowage_6.x_CE_Manual.pdf)



1.3.3.4 Data Persistence

The data persistence is composed by three components: Orion, Cygnus and a MongoDB instance. Cygnus requires a file as volume of docker, this is "*agent.conf*", containing the configuration of database, STH's sinks, and the port where is notified by Orion subscriptions. When launched a subscription must be created in Orion through its REST API. The request body will contain the URL where Orion will notify the changes of entities values. This URL must contain the IP and port where Cygnus is listening. Cygnus will be in charge of persist data received from Orion in configured databases.



1.3.4 Licensing

Most of the FIWARE components that form the platform are distributed under open source licenses such as GPL or AGPL.