D2.2 Extensions of FIWARE ecosystem with Big Data and AI frameworks

Author: Benoit Orihuela (EGM)

Co-Authors: Fernando López (FF), Lluis Echeverria (EUT), Alberto Abella (FF), Ahmed Abid (EGM), Franck Le Gall (EGM), Panagiotis Kossieris (NTUA), Gareth Lewis (EXE), Chris Pantazis (NTUA), Siddarth Seshan (KWR)

November 2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant agreement No. 821036.

Disclaimer

This document reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Intellectual Property Rights

Project Consortium

Executive Summary

Because FIWARE4Water project aims to develop a socially and business relevant system architecture for heterogeneous entities based in FIWARE technology, it is necessary to translate the identified Big Data and AI requirements identified in WP1 in order to provide the proper tools and frameworks for answering such requirements. The present deliverable provides an extended analysis of the FIWARE Generic Enablers devoted to Big Data and AI concerns and introduces the integrations of these Generic Enablers to answer business requirements that are currently worked on.

However, as current existing FIWARE Generic Enablers do not cover the full scope of the Demo Cases business requirements, this deliverable also introduces new architectural patterns and components that are needed to achieve these requirements and details the associated specifications and the on-going implementations. This work is going to be used by Demo Cases to extend their business capabilities by having the most efficient and adapted components in their toolbox. What's more, this work is expected to later enrich the FIWARE ecosystem with new innovative and state of the art components.

This deliverable then explains how these tools and frameworks are integrated and used inside the architecture of each Demo Case, more specifically emphasizing their relationship with the other components already deployed and with the data flowing into the platforms.

Last but not least, smart data models have been developed, and some more are actively being designed, to support the integration and use of Big Data and AI tools and frameworks. They are the necessary foundations to define a common ground for a full interoperability between all the components of the F4W-RA, but also with any external third-party willing to integrate with it.

Related Deliverables

This document is related to the requirements obtained in deliverables D1.1 - Requirements from Demo Cases and D1.4 - Gap analysis and final Requirements. It expands upon the deliverable D2.2 – System Architecture to focus on the implementation of Big Data and AI tools and frameworks and their integration inside the FIWARE4Water Reference Architecture.

Moreover, the tools, frameworks and implementations are being deployed on the Demo Cases sites and will be part of the base for the WP3 and WP4 and therefore to the corresponding demonstrators (D3.1 – D3.5, D4.1-D4.5) and lessons learned (D3.6, D4.6).

Document Information

Programme	H2020 – SC0511-2018
Project Acronym	**Fiware4Water**
Project full name	FIWARE for the Next Generation Internet Services for the WATER sector
Deliverable	**D2.2: Extensions of FIWARE ecosystem for Big Data and AI frameworks**
Work Package	WP2: Architecture/Data/Ontology/API/Legacy links/Standards
Task	Task 2.1: Big-Data Processing and AI FIWARE GEs for Water Management
Lead Beneficiary	2 - EGLOBALMARK
Author(s)	Benoit Orihuela (EGM)
Contributor(s)	Fernando López (FF), Lluis Echeverria (EUT), Alberto Abella (FF), Ahmed Abid (EGM), Franck Le Gall (EGM), Panagiotis Kossieris (NTUA), Gareth Lewis (EXE), Chris Pantazis (NTUA), Siddarth Seshan (KWR), Alex van der Helm (WATNL), Dirk Vries (WATNL)
Quality check	Elad Salomons (Optiwater)
Planned Delivery Date	M18 (30 November 2020)
Actual Delivery Date	M18 (30 November 2020)
Dissemination Level	Public (Information available in the Grant Agreement)

Revision history

Version	Date	Author(s)/Contributor(s)	Notes
Draft1	15/10/20	Fernando López (FF), Lluis Echeverria (EUT), Benoit Orihuela (EGM)	First version of the ToC
Draft2	20/11/20	Fernando López (FF), Lluis Echeverria (EUT), Alberto Abella (FF), Ahmed Abid (EGM), Franck Le Gall (EGM), Panagiotis Kossieris (NTUA), Gareth Lewis (EXE), Chris Pantazis (NTUA), Siddarth Seshan (KWR), Alex van der Helm (WATNL), Dirk Vries (WATNL), Benoit Orihuela (EGM)	Complete contribution to the version
Draft3	25/11/20	Fernando López (FF), Lluis Echeverria (EUT), Benoit Orihuela (EGM)	Improve sections II, III, IV
Draft4	27/11/20	Elad Salomons (Optiwater)	Review
Final	30/11/20	Benoit Orihuela (EGM)	Final version
FinalV2	15/03/21	Benoit Orihuela (EGM)	Explanation of the EU added value (following RP1 review)

Table of content

List of figures

List of tables

List of Acronyms/Glossary

AI	Artificial Intelligence
API	Application Programming Interface
CEP	Complex Event Processing
CI/CD	Continuous Integration/Continuous Deployment
CSV	Comma Separated Values
DT	Digital Twins
ETSI	European Telecommunications Standards Institute
F4W	Fiware4Water
F4W-RA	Fiware4Water Reference Architecture
FAQ	Frequently Asked Question(s)
GDPR	General Data Protection Regulation
GE	Generic Enabler
GIS	Geographic Information System
HDFS	Hadoop distributed file system
IoT	Internet of Things
ISG CIM	Industry Specification Group (ISG) for cross-cutting Context Information Management (CIM)
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation – Linked Data
LD	Linked Data
LoRaWAN	Low Range Wide Area Network
ML	Machine Learning
MLOps	DevOps for Machine Learning.
MQTT	Message Queuing Telemetry Transport
NGSI	Next Generation Services Interface
NGSI-LD	Next Generation Services Interface – Linked Data
SQL	Structured Query Language
URI	Uniform Resource Identifier
URN	Uniform Resource Name
VCS	Version control systems
VM	Virtual Machine
XML	eXtensible Markup Language
YAML	Ain't Markup Language

Introduction

This deliverable defines the Big Data and AI requirements, based on the F4W-RA previously defined in the deliverable “D2.1 System Architecture” that will be developed, integrated and deployed in the context of the FIWARE4Water project. In particular, it aims at defining the core architectural patterns and the appropriate technical components that will permit the Demo Cases to have a thorough insight into their data, to better anticipate any future need and to optimize their current uses.

The first section briefly reminds of the Big Data and AI requirements of the four Demo Cases, via a matrix table associating each use case to one or more Big Data or AI topics.

The second section presents in detail each Big Data and AI topic that will be addressed in the context of the FIWARE4Water project and aims at defining clear borders for each of them.

The third section then goes into the full details of the proposed Big Data & AI tools. After a SWOT analysis of each of the FIWARE Generic Enablers identified to fulfil the Big Data & AI requirements, it introduces and describes new architectural patterns and components that are needed to achieve the Demo Cases requirements.

The fourth section presents the updated architecture schema for each Demo Case, explaining how the Big Data & AI tools fit into the picture and what are their main interactions with the other components of the platform.

The fifth section focuses on the new smart data models that are being designed and deployed to support the deployment and usage of the AI components. It presents the new concepts introduced and how they are designed to be used by the Big Data and AI tools.

I. Demo cases business requirements

Based on the Demo Cases business requirements defined in "D1.1 - Requirements from Demo Cases" and "D1.4 - Gaps analysis and requirements", the following matrix table summarizes the requirements related to Big Data or Artificial Intelligence" for each of the Demo Case.

It has the objective to give a summarized and easily referenceable view of the technical challenges that have to be achieved for each of the Demo Case with respect to Big Data and AI. It is used as an implementation guideline to ensure the requirements are correctly handled and served by the appropriate tools.

The entries given in the table refer to the use cases defined in the afore mentioned deliverables.

	DC #1 (GR.EYDAP)	DC #2 (FR.3S)	DC #3 (NL.WNT)	DC #4 (UK.SWW)
Big Data real time ingestion	SA1.US01.UC01 SA2.US01.UC01	SA2.US01.UC01.Dfn003 SA2.US02.UC01.Dfn005	SA1.US01.UC01	SA1.US01.UC01
Stream processing (including data cleaning and validation)	SA1.US01.UC02 SA1.US02.UC01 SA1.US02.UC02 SA1.US02.UC03 SA1.US02.UC04 SA2.US01.UC02 SA2.US02.UC01 SA2.US02.UC02 SA2.US02.UC03	SA2.US02.UC02.Fn021 SA2.US01.UC02.Fn011	SA1.US01.UC01 SA1.US01.UC03	SA1.US02.UC01 SA1.US02.UC02
Big Data storage	SA1.US01.UC01 SA1.US01.UC02 SA1.US02.UC04 SA2.US01.UC01 SA2.US01.UC02	SA1.US01.UC01 SA1.US01.UC02 SA1.US02.UC01	SA1.US01.UC01	SA1.US01.UC01 SA2.US01.UC01
Batch processing (including data cleaning and validation)			SA1.US01.UC01	
Complex event processing			SA1.US01.UC01	SA1.US02.UC03
Analytics	SA1.US01.UC01 SA1.US01.UC02 SA2.US01.UC01 SA2.US01.UC02 SA2.US02.UC03			

Visualisation	SA1.US01.UC01 SA1.US01.UC02 SA1.US02.UC01 SA1.US02.UC02 SA1.US02.UC03 SA1.US02.UC04 SA1.US03.UC01 SA1.US03.UC02 SA2.US01.UC01 SA2.US01.UC02 SA2.US02.UC01 SA2.US02.UC02 SA2.US02.UC03	SA1.US01.UC02	SA1.US01.UC01 SA1.US01.UC04	
AI & ML	SA1.US02.UC02 SA1.US02.UC03 SA1.US02.UC04 SA1.US03.UC01 SA1.US03.UC02 SA2.US02.UC02 SA2.US02.UC03	SA1.US01.UC01 SA1.US02.UC01	SA1.US01.UC01 SA1.US01.UC04	SA1.US02.UC03 SA2.US02.UC01
Digital Twins	SA1.US02.UC02 SA1.US02.UC03 SA1.US02.UC04 SA1.US03.UC01 SA1.US03.UC02	SA1.US01.UC.02 SA1.US02.UC.01		

Table 1: Matrix table of Big Data and AI requirements for Demo Cases

II. Core Big Data & AI concepts

In order to share a common and understandable definition of the concepts behind the words generally used in the Big Data and Artificial Intelligence concepts, this section briefly reminds what are the technical processes and features typically associated with each of these concepts. They will later be used to qualify and identify the components that are to be integrated and developed in the scope of the project and to fulfil the requirements of the Demo Cases.

II.1. General introduction

Big Data and Artificial Intelligence (AI) concepts are a pair. In the Data Science domain, AI is used as a tool, together with other scientific methods, processes and resources from other fields (such as data mining, mathematics or statistics), to process data, extract valuable knowledge and insights, and finally improve our systems, outcomes or products in terms of better decisions and improved processes.

Machine learning (ML) is an AI subdomain where a set of data-driven algorithms allows the development of highly accurate intelligent models able to predict results without the need for explicit programming. ML is an essential part of AI, but AI is broader than ML, as it also covers the ability of a system to perceive data (e.g., natural language processing or voice/image recognition) or to control, move and manipulate objects based on learned information, whether it is a robot or other connected device.

Linked to the constant growth of the Internet of Things (IoT) data, the Big Data concept has emerged to identify and encompass the processes and technologies that allow, first, the storage of this large volume of digital data and, second, process and identify, through a series of techniques and mechanisms of ML and massive data visualization, patterns of relationship between value variables, for example, to be able to predict anomalies, optimize systems, control processes, etc. and thus generate appropriate and accurate decisions responding quickly and optimally to our particular problem.

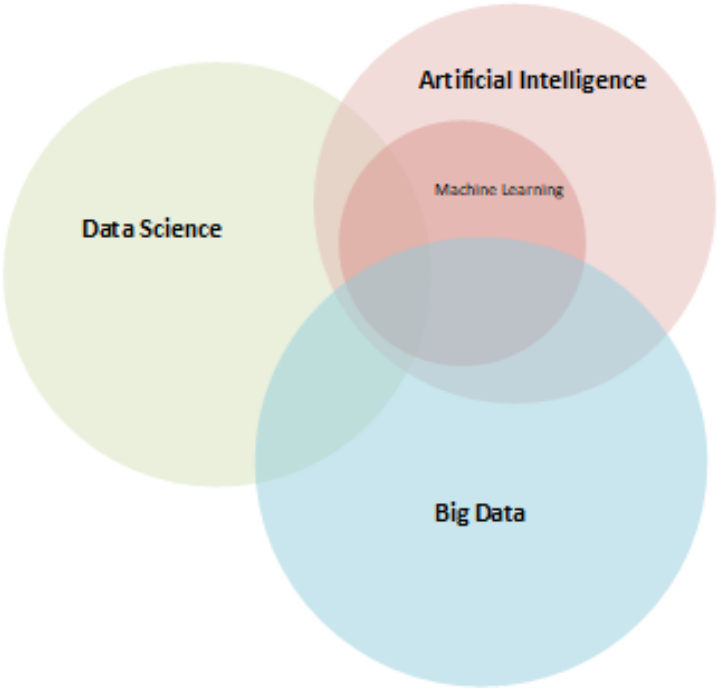


Figure 1: Visually linking AI, ML, Big Data and Data Science

AI and, more concretely, ML data-driven algorithms, usually perform better when more data are included in the models training phase, resulting in better models with improved generalization capacities. This fact boosts the link between Big Data and AI, since the first improves the second providing it with the required volumes of data to generate high-quality smart models. Then, in sync with recent advances in AI (e.g., Human-level control through deep reinforcement learning) and new computational capabilities (e.g., HPC engines), these technologies are leading the digital era, and the fourth industrial revolution.

Both concepts (Big Data and AI) are developing fast and both technologies are, and will be, directly related to the future development and growth of the global community, being both present and active at different scales (regional, national and global) and domains (healthcare, industry, resources management, agri-food system, climate change, etc). Due to this importance and criticality, EU is moving towards a new Data Strategy, including these concepts as a pillar of the new digital strategy of the Commission.

In the context of the FIWARE4Water project, Big Data is concerned mainly for the storage of the data coming from the sensors deployed on the sites of the Demo Cases. AI and ML are used to improve the business efficiency of the Demo Cases.

The following sections describe the most common Big Data tasks and procedures, such as ingestion and storage, stream processing or complex event processing, and the AI and ML lifecycle phases such as data collection & processing, or ML training & learning.

II.2. Big Data ingestion and storage

Data ingestion and storage are two key basic processes in a Big Data environment. Big Data solutions face the problem of dealing with a large amount of data, data types and data consumption rates, by distributing the processes (distributed architectures) and adding more hardware resources (scaling out nodes) in order to reduce the computational time, thus introducing complexity but also other advanced and useful capabilities such as scalability, fault tolerance, high availability and performance, among others.

First, data ingestion processes cover the ability of a system to consume both batch and stream data, coming from different sources, which may also have different format and protocol, requiring some types of transformations and conversions. Usually, stream, probably real-time, data ingestion is the most common process in IoT platforms, where IoT devices frequently send data to the cloud in order to, for example, be persisted and processed. In this case, data volume is an issue, but the capacity to integrate and consume from hundreds IoT data sources in real-time is another important one. To this end, it is required a powerful (highly) distributed front layer of consumer systems, able to establish communication (interoperability) with the IoT devices, ingesting and processing data at the appropriate frequency. Real-time consumption can be a critical part in several domains, where an immediate response is required to avoid, or reduce, potential issues.

Second, once data is consumed, usually it needs to be persisted. In this case, the main issue is the large amount of data that usually, in a Big Data case, needs to be stored, managed, maintained and ready to be queried thus satisfying the needs of applications that require access to the data. Depending on the case, the final use and the type of data, usually the ideal Big Data storage system would: i) allow storage of a virtually unlimited amount of data, ii) offer high rates of random write and read access, iii) flexibly and efficiently deal with a range of different data models, iv) support both structured and unstructured data, and v) for privacy reasons, only work on encrypted data.

Several solutions have been implemented during the last years to cover these Big Data tasks (ingestion and storage).

Major tools in the topic of Big Data storage are Apache Hadoop (HDFS)[1], PostgreSQL (via extensions like TimescaleDB[2]), CrateDB [3]or Cassandra[4].

Major tools in the topic of Big Data ingestion are Apache Kafka[5], Apache NiFi[6], Apache Storm[7] and Apache Flume[8].

II.3. Stream processing

Stream processing generally refers to the process of applying transformations, validations or else simple calculations on a stream of data flowing into a platform. It is closely related to the Big Data ingestion process as both usually work together: streamed data flows into the platform, along its way inside the platform, a stream processor receives the data (often via a publish / subscribe mechanism), processes it and then inject it back into the platform for other components to deal with it (apply other transformations, store it permanently, …).

It can be used to apply rules and procedures on the quality of incoming data, for instance:

- To qualify the accuracy of a measure
- To deal with duplicate observations and irrelevant observations
- To fix structural errors (arise during measurement, data transfer, or other types of "poor housekeeping")
- To deal with missing data (dropping or imputing? Or both? ...)

It can also be used to aggregate a bunch of measures (e.g., store the mean in a 1-minute range instead of storing a measure every second if such a precision is not needed) or else to make simple predictions based on a windowed range of values in the stream of data.

[1] http://hadoop.apache.org/

[2] https://timescale.com

[3] https://crate.io/

[4] https://cassandra.apache.org/

[5] https://kafka.apache.org

[6] https://nifi.apache.org

[7] https://storm.apache.org

[8] https://flume.apache.org

Major tools in this area are Kafka Streams[9], Spark Streaming[10] (and structured streaming), Akka Streams[11], and Flink[12].

II.4. Complex event processing

Similar to event stream processing, complex event processing (CEP) aims at aggregating, processing, and analysing large streams of data in order to gain real-time insights from events as they occur.

Today, users are flooded with facts that are difficult to understand and use appropriately. CEP transforms low-level data into high-level business information that users care about.

CEP applies domain knowledge across multiple sources of data to understand what is happening in terms of high-level concepts and complex events. It is designed to infer complex events from raw data using business patterns and concepts. The aim is to identify meaningful facts that can be used to make informed decisions.

Thus, CEP is a generalization of stream processing. Stream processing is concerned with finding low-level patterns in data, such as the number of mouse clicks within a fifteen-minute window. CEP promises much more. Using models of causality and conceptual hierarchies, CEP can make high-level inferences about complex events within the business domain.

It is often used for anomaly detection, predictive analysis, especially in the field of IoT.

In this area, the same tools as used for stream processing concerns can be used, mainly Apache Kafka[13], Apache Flink[14] and Apache Spark[15].

II.5. Visualisation and analytics

When a large volume of data is stored, and especially in the case of sensor data, it is particularly useful to be able to visualise it. Indeed, it is the first and easiest way to have a global view on the current state of the data and how it is evolving over time.

Typically, such visualisations make use of graphs, tables, heatmaps, or else maps. Arranged inside larger dashboards that allow to group different visualisations of related data, to define time filters or else to set real time data update, they provide the end user with a rich and powerful interface allowing discovery and visualisation of the data stored inside the platform.

---

9 https://kafka.apache.org/documentation/streams/

10 https://spark.apache.org/streaming/

11 https://doc.akka.io/docs/akka/current/stream/index.html

12 https://flink.apache.org/

13 https://kafka.apache.org/

14 https://flink.apache.org/

15 https://spark.apache.org/

Going a bit further, data analytics is the process by which a user can get insights and potentially detect patterns inside the data, and eventually make decisions based on these insights.

Data visualisation and analytics often work close together, as the former is the basis for the latter.

Major tools in this category are Apache Superset[16], Grafana [17]and Kibana[18].

II.6. AI and Machine Learning

Machine learning development lifecycle

The diagram below shows a typical Machine Learning development lifecycle: data collection & processing, training & learning, and deploying & using phases. It should be noted that this is a fairly iterative lifecycle.

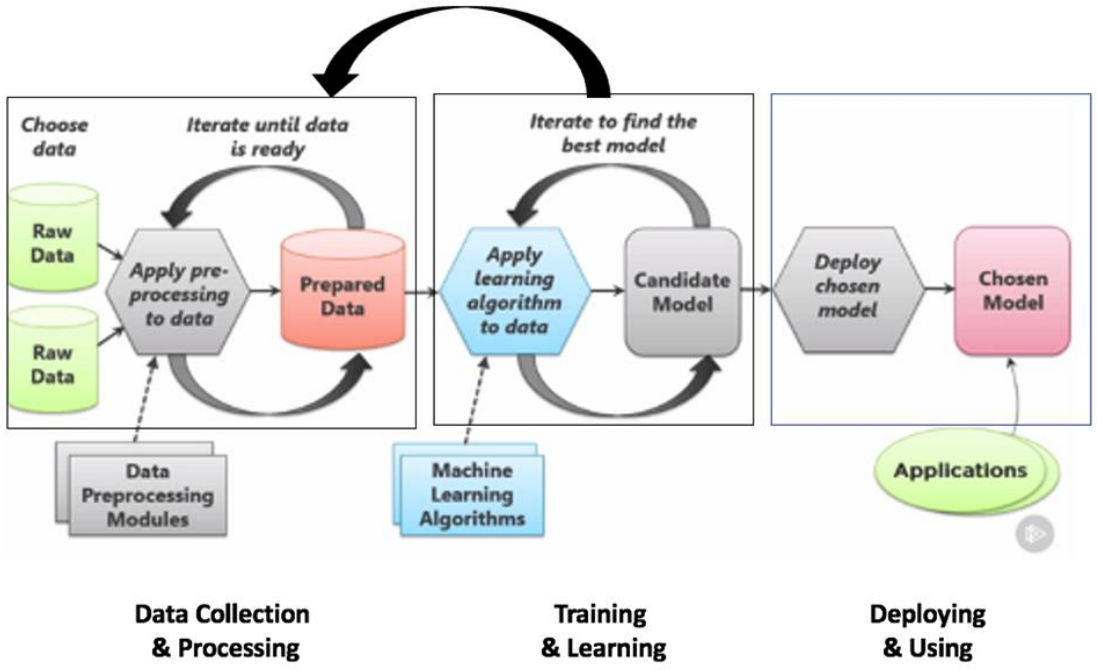


Figure 2: Machine learning development flow

Data Collection & Processing

Data collection and processing is about all the steps that are necessary to obtain enough structured, cleaned up and validated data to then be able to use it to train machine learning algorithms.

In the diagram above, the F4W-RA context information broker can be used as a provider of raw data (on the left) and it is expected to be combined and processed with raw data from other sources to create learning data for whatever machine learning algorithm is being used. Typically, this raw data is conveyed

[16] https://superset.apache.org/

[17] https://grafana.com/

[18] https://www.elastic.co/fr/kibana

in a spreadsheet row & column format and the F4W-RA should be able to export raw data in such a compatible format.

A good starting point for FIWARE-data science is to provide some functionality to ease this process for data scientists, a bit like the IoT agents in reverse. It's worth remembering that data scientists are not necessarily hard-core programmers (hence the use of tools like Python's pandas and Numpy in scikit), so functionality to ease this process, particularly if FIWARE data is being combined with other sources of data, will be welcomed.

Training & Learning

This phase of the lifecycle will train an ML algorithm with learning data and then test it to determine the quality of the model. This is a highly iterative phase of the lifecycle with data scientists typically adjusting algorithms' hyperparameters and training models in order to develop a robust solution. There is also scoped to iterate back into the data collection and processing phase to add and remove data features and to encode them in different ways prior to training.

Deploying & Using

Once the smart (ML) models are providing acceptable solutions to test data, it can be packaged into deployment. Generally, this is a case of pickling training weights from the ML algorithm and providing an application interface that will allow novel features to be encoded in line with existing training data.

Generally, the deployed solution is not a general case for 'all' instances of the problem addressed, given the type and format of features and the relationships encoded from the training data. Therefore, a solution for one water company may not provide a meaningful solution for another water company, given co-occurrence and causality.

Also, model deployment can imply more complex scenarios:

- Multiple models: sometimes, it is possible to have more than one model that perform the same task. In that case, it can be handy for consuming applications to access the different models with a single API call.
- Shadow models: when deploying a new version of a model, it can be useful to deploy it side by side with the currently running model, send him the same production data and validate his behaviour before promoting it into production.
- Competing models: in this scenario, multiple versions of the model in production are put in competition to find out which one is the better (similar to an A/B test). It implies an added complexity in infrastructure setup to ensure a correct redirection of the traffic and enough data to make significant decisions.
- Online learning models: unlike traditional ML models that are trained offline, online learning models use algorithms and techniques to continuously improve their performance, using new data arriving in the platform. It also implies an additional complexity as production data will also to be versioned (as it is used by the model to improve by itself).

Monitoring and observability

Once a model is deployed on a platform, it has to be monitored and observed like any piece of software. Fortunately, the same set of tools and infrastructure as described in Deliverable D2.1 (section IV.5 Operation support tools) can be used for this purpose.

Especially, the same tools for log aggregation and metrics collection can be used to gather data required to understand how the model in production is behaving:

- Model inputs and outputs: what data is being fed into the model and what calculations (predictions, recommendations, …) are made based on these inputs. And also, what decisions are taken based on the outputs of a model
- Model interpretability outputs: metrics such as model coefficients that allow later investigation to understand how the models are making predictions and to identify potential overfit or bias that was not found during training.

Scope of the F4W-RA

This section aims at defining, among the different stages of a ML development lifecycle, which ones are considered to be in the scope of the F4W-RA.

The first two stages (data collection & processing, training & learning) described above are typically performed by data scientists in specific environments (e.g., Jupyter [19] notebooks, TensorFlow [20] development environment, ...), similar to a software developer using its favourite IDE to develop an application. Thus, it is not considered as part of the F4W-RA which is focused on the runtime environment, with the exception of the consumption of data needed to train a model, which is neutral from the point of view of the F4W-RA (it can be seen in the same way as any consumer of data).

The third stage (developing & using) is definitely in the scope. Indeed, the F4W-RA aims to provide the necessary components to deploy and manage ML models and the necessary (NGSI-LD) APIs to interact with a deployed ML model. This interaction will typically consist of providing input data (preferably already stored in a context broker or other FIWARE component) and get in return one or more calculated values.

The fourth stage (monitoring & observability) also fits with what is expected from the F4W-RA, as it is crucial to assert and follow the behaviour of a ML model deployed in the platform.

II.7. Digital Twins

A digital twin is a virtual representation of a physical system, improving the interactions capabilities with the real artefact. Digital twins can be considered over the whole system lifecycle: design, (de)commissioning and operation. In the NGSI-LD context where we handle live change of context information, the focus is primarily on the operation phase where we can propose 2 levels of digital twins:

- Data twin: it provides a real time and queryable representation of observed properties of the physical twin. The representation is limited to the observed properties of that physical artifact. Such a data twin is a natively supported functionality at entity level in a NGSI-LD environment where the current and past states of the observed artefact can be queried. In addition, on-change notifications can be generated.
- Smart twin: In addition to the current state of observed properties, the digital twin includes a model of the physical twin. This model can be of any kind such as multiphysics model, machine learned model, etc. In most of the usages of NGSI-LD, it is not expected that the model provides an atomic level representation of the system but rather a representation of functional states within defined boundaries. Several usage patterns are foreseen and include predicting short-medium term behaviour of the system, detecting anomalies in operation or executing what-if

[19] https://jupyter.org/
[20] https://www.tensorflow.org/

scenarios based on the current state of the data twin. In addition, coupling to actuation can be envisioned to allow physical systems being driven by their twin.

Grieves [1] describes 3 level of complexity for digital twins:

- Simple: "The outside observer has no problem in discerning the operation of the system. The system is completely predictable. The inputs are highly visible. The actions performed on those inputs are obvious and transparent. The outputs are easily predictable.". This can be described as the thing level where the behavioural model and metrics of interest are well known, and the thing can be modelled as an entity within the NGSI-LD graph. Example is the individual water pump.
- Complicated: "Complicated systems are also completely predictable. The system follows well defined patterns. The difference between simple systems and complicated systems is the component count. Complicated systems have many more components. Complicated systems are often described as intricate. However, the inputs are well known, as are the resulting outputs. The connection between components is linear and straightforward." Within a NGSI-LD graph, it would include several entities connected through relationships. Example is a group of water pumps in parallel in a pumping station.
- Complex: "Complex systems have been characterized as being a large network of components, many-to-many communication channels, and sophisticated information processing that makes prediction of system states difficult". It implies a more complex graph structure within the NGSI-LD model which can even include graph composition. Example is a complete water distribution network.

III. Integration inside the F4W RA

III.1. Analysis of existing components in the FIWARE ecosystem

As initially identified in D2.1 'Specification of system architecture for water consumption and quality monitoring', section II 'Architecture layers', the FIWARE ecosystem provides a set of "Powered by FIWARE" components (a.k.a. FIWARE Generic Enablers), which are assembled with the FIWARE core Context Broker technologies, and facilitate the integration of third-party frameworks and services, thus boosting the smart solution functionalities and capabilities.

For the sake of clarity, the FIWARE ecosystem is reminded in the figure below.

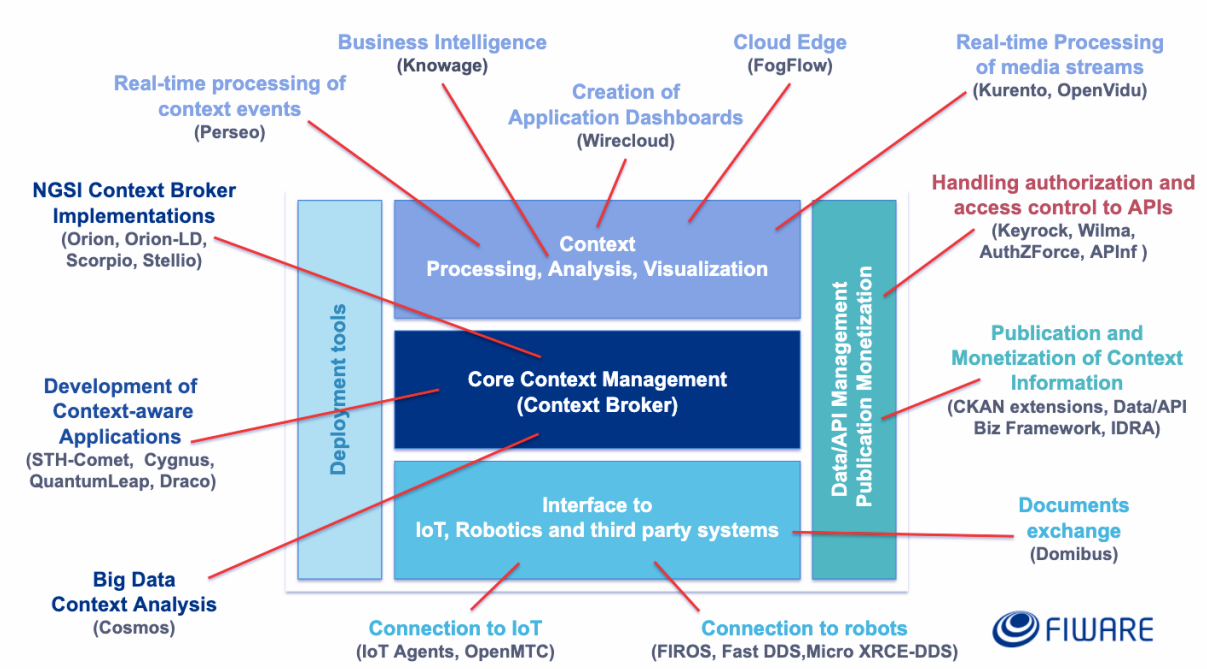


Figure 3: High level view of the FIWARE ecosystem of framework

Building around the FIWARE Context Broker, a rich suite of complementary FIWARE Generic Enablers is available, dealing with the following functionalities:

- Core Context Management manipulates and stores context data so it can be used for further processing.
- Interfacing with the Internet of Things (IoT), Robots and third-party systems, for capturing updates on context information and translating required actuations.
- Processing, analysis, and visualization of context information, implementing the expected smart behaviour of applications and/or assisting end users in making smart decisions.
- Context Data/API management, publication, and monetization, bringing support to usage control and the opportunity to publish and monetize part of managed context data.

This section aims to extend the initial analysis performed in D2.1, in terms of FIWARE Generic Enablers to integrate and support Big Data and Artificial Intelligence capabilities. It also presents the first on-going developments to integrate them into the F4W-RA. The following FIWARE Generic Enablers are considered:

- CYGNUS Generic Enabler
- DRACO Generic Enabler

- COSMOS Generic Enabler
- PERSEO Generic Enabler

It is important to note that these Big Data components rely on the core context broker to:

- Retrieve historical data on demand when it is needed (for instance, when a ML model needs to run a computation), thanks to the temporal API provided by NGSI-LD
- Store cleaned or processed data, results of computation, …

So, it alleviates the context broker from this memory and / or CPU consuming tasks and prevents it from becoming the bottleneck of the platform. What's more, the decoupling capabilities presented later on in this document allow to alleviate still more the load on the context broker, as a lot of data can be processed directly without going through the context broker (and even never be persisted in it if it has no business interest).

CYGNUS Generic Enabler

Cygnus Generic Enabler brings the means for managing the history of context that is created as a stream of data. It is a connector in charge of persisting certain sources of data in certain configured third-party storages, creating a historical view of such data. Internally, Cygnus is based on Apache Flume.

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.

The use of Apache Flume is not only restricted to log data aggregation. Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible. Flume allows a user to build multi-hop flows where events travel through multiple agents before reaching the final destination. It also allows fan-in and fan-out flows, contextual routing and backup routes (fail-over) for failed hops.

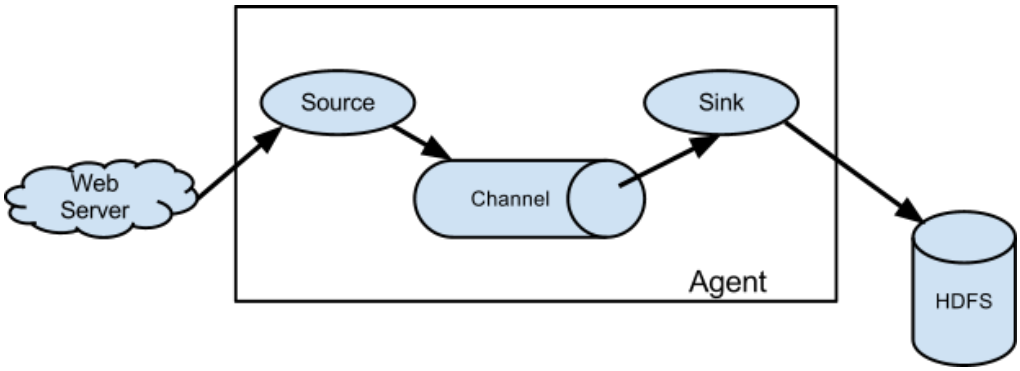


Figure 4: Agent component view of Apache Flume

Flume provides end-to-end reliability of the data flow thanks to its transactional single-hop message delivery semantics, where events (data) are removed from a channel only after they are stored in the channel of the next agent or in the terminal repository. Another important property of Flume is the recoverability, where the events are staged in the channel, enabling recovery from failure.

Cygnus is not only restricted to process data aggregation mainly due to data sources being customizable. This functionality allows using the component to transport massive quantities of event data which can include water network data, email messages, social media information and any other data source available.

F4W will use the Cygnus-LD agent, which is the connector in charge of persisting Context Information in NGSI-LD format into the different third-party storage system and creating a historical view of this data. As usual with the rest of FIWARE GEs, it makes use of the subscription/notification pattern. This subscription is made in the Context Broker on behalf of Cygnus-LD detailing which entities we want to be notified, how we want to receive those notifications and which entity attributes we want to receive.

Currently, the stable version of Cygnus-LD is able to persist the Linked Data from the Context Broker into the following databases:

- PostgreSQL, the well-known relational database manager.
- PostGIS, a spatial database extender for PostgreSQL object-relational database.
- CKAN, an Open Data platform. You may consider visiting Cygnus NGSI-LD Quick Start Guide before going deep into the details.

Additionally, to allow the proper execution of the FIWARE Generic Enabler, there is a minimum set of hardware requirements that can be summarized in:

- RAM: 1 GB, especially if abusing the batching mechanism.
- HDD: A few GB may be enough unless the channel types are configured as FileChannel type.

The following paragraph provides an example of use of this component integrated with a Context Broker and persisting the data into a database (in this case PostgreSQL). If we planned to create an instance, it is needed to understand what should be the architecture to be applied and following this approach, configure properly the component. For the purpose of the example, we can decide to have a HTTP Source that provides us information in NGSI, we process the information through only a Channel and finally we put in place a Sink to persist the information into the PostgreSQL instance. The configuration file should be the following:

```
cygnus-ngsi-ld.sources = http-source
cygnus-ngsi-ld.sinks = postgresql-sink
cygnus-ngsi-ld.channels = test-channel

cygnus-ngsi-ld.sources.http-source.channels = test-channel
cygnus-ngsi-ld.sources.http-source.type = org.apache.flume.source.http.HTTPSource
cygnus-ngsi-ld.sources.http-source.port = 5050
cygnus-ngsi-ld.sources.http-source.handler =
com.telefonica.iot.cygnus.handlers.NGSIRestHandler
cygnus-ngsi-ld.sources.http-source.handler.notification_target = /notify
cygnus-ngsi-ld.sources.http-source.handler.default_service = openiot
cygnus-ngsi-ld.sources.http-source.handler.ngsi_version = ld
cygnus-ngsi-ld.sources.http-source.handler.events_ttl = 2
cygnus-ngsi-ld.sources.http-source.interceptors = ts
cygnus-ngsi-ld.sources.http-source.interceptors.ts.type = timestamp



cygnus-ngsi-ld.channels.test-channel.type = memory
cygnus-ngsi-ld.channels.test-channel.capacity = 1000
cygnus-ngsi-ld.channels.test-channel.transactionCapacity = 100


cygnus-ngsi-ld.sinks.postgresql-sink.type =
com.telefonica.iot.cygnus.sinks.NGSIPostgreSQLSink
cygnus-ngsi-ld.sinks.postgresql-sink.channel = test-channel
cygnus-ngsi-ld.sinks.postgresql-sink.enable_encoding = false
```

```
cygnus-ngsi-ld.sinks.postgresql-sink.enable_grouping = false
cygnus-ngsi-ld.sinks.postgresql-sink.enable_lowercase = false
cygnus-ngsi-ld.sinks.postgresql-sink.enable_name_mappings = false
cygnus-ngsi-ld.sinks.postgresql-sink.data_model = dm-by-entity
cygnus-ngsi-ld.sinks.postgresql-sink.postgresql_host = localhost
cygnus-ngsi-ld.sinks.postgresql-sink.postgresql_port = 5432
cygnus-ngsi-ld.sinks.postgresql-sink.postgresql_database = postgres
cygnus-ngsi-ld.sinks.postgresql-sink.postgresql_username = postgres
cygnus-ngsi-ld.sinks.postgresql-sink.postgresql_password = example
cygnus-ngsi-ld.sinks.postgresql-sink.postgresql_options = sslmode=require
cygnus-ngsi-ld.sinks.postgresql-sink.attr_persistence = column
cygnus-ngsi-ld.sinks.postgresql-sink.attr_native_types = false
cygnus-ngsi-ld.sinks.postgresql-sink.batch_size = 1
cygnus-ngsi-ld.sinks.postgresql-sink.batch_timeout = 30
cygnus-ngsi-ld.sinks.postgresql-sink.batch_ttl = 10
cygnus-ngsi-ld.sinks.postgresql-sink.batch_retry_intervals = 5000
cygnus-ngsi-ld.sinks.postgresql.backend.enable_cache = false
```

Figure 5: Configuration file for the Cygnus-LD component

For more information about configuration parameters and different options, refer to the documentation (https://github.com/telefonicaid/fiware-cygnus/tree/master/cygnus-ngsi-ld)

Example the Subscription to Context Broker LD to recover the reservoirHead, location and initialQuality when the reservoirHead is less than 50 can be seen in the following code:

```
curl -L -X POST 'http://localhost:1026/ngsi-ld/v1/subscriptions/' \
-H 'Content-Type: application/ld+json' \
-H 'Link: <https://schema.lab.fiware.org/ld/context.jsonld>;
rel="http://www.w3.org/ns/json-ld#context"; type="application/ld+json"' \
--data-raw
'{
 "description": "Notify me when the reservoirHead is below 50 in Reservoir
001",
 "type": "Subscription",
 "entities": [{"type": "Reservoir"}],
 "watchedAttributes": ["reservoirHead"],
 "q": "reservoirHead<50;locatedIn==urn:ngsi-ld:Reservoir:reservoir001",
 "notification": {
   "attributes": ["reservoirHead"],
   "format": "keyValues",
   "endpoint": {
     "uri": "http://cygnus-ld:5050/notify",
     "accept": "application/json"
   }
 },
  "@context": "https://schema.lab.fiware.org/ld/context.jsonld"
}'
```

Figure 6: Example of subscription request to a Context Broker

We can check the created subscription in the Context Broker through the execution of the following command:

```
curl -L -X GET 'http://localhost:1026/ngsi-ld/v1/subscriptions/
```

Figure 7: Request subscription information to the Context Broker

And it returns the following information:

```
[
 {
  "id":"urn:ngsi-ld:Subscription:5fbd337ec13cd4f47afacebc",
  "type":"Subscription",
  "description":"Notify Cygnus of low stock in Store 001",
  "entities":[{"type":"Shelf"}],
  "watchedAttributes":["numberOfItems"],
"q":"https://schema.lab.fiware.org/ld/schema/reservoirHead<50;https://schema
.lab.fiware.org/ld/schema/urn:ngsi-ld:Reservoir:reservoir001",
  "notification": {
    "attributes":["reservoirHead"],
    "format":"keyValues",
    "endpoint": {
      "uri":"http://cygnus-ld:5050/notify",
      "accept":"application/json"
    }
  },
  "@context":"https://schema.lab.fiware.org/ld/context.jsonld"
 }
]
```

Figure 8: Response from the Context Broker with detailed subscription info

Once that the procedure is activated and the value of the property is changed and below the specified threshold, Cygnus-LD will receive the notifications and persist the information into the database. We can check for example the persisted information into PostgreSQL by executing the following content:

```
sudo docker exec -it postgres-db psql -U postgres

psql (12.2 (Debian 12.2-2.pgdg100+1))
Type "help" for help.


postgres=# \dn
  List of schemas
  Name   |  Owner
---------+----------
 openiot | postgres
 public  | postgres
(2 rows)

postgres=# \dt openiot.*
              List of relations
 Schema  |      Name                          | Type  |  Owner
---------+------------------------------------+-------+----------
 openiot | urn:ngsi-ld:Reservoir:reservoir001 | table | postgres
(1 row)

postgres=# select * from openiot.urn_entities_reservoir001;
```

```
         recvtime         |    entityid                | entitytype |
reservoirHead
--------------------------+----------------------------+------------+----
 2020-05-13T15:23:16.358Z | urn:entities:reservoir001 | Reservoir  | 49
 2020-05-13T15:25:16.358Z | urn:entities:reservoir001 | Reservoir  | 47
 2020-05-13T15:26:16.358Z | urn:entities:reservoir001 | Reservoir  | 48

(3 row)
```

Figure 9: PostgreSQL check of the persisted context information

SWOT Analysis

In this section, we planned to make a SWOT analysis of the FIWARE Cygnus component in order to study the identified requirements (Weakness and Threat) of the component and in case that they are relevant inside the Fiware4Water provide some new developments to cover them. Part of the purpose of SWOT analysis is also to identify those factors that influence the functioning of the FIWARE GE providing very useful information in the strategic roadmap definition of the component.

The result of this analysis is shown in the following figure.

Figure 10: SWOT Analysis of Cygnus

DRACO Generic Enabler

Draco Generic Enabler is an alternative data persistence mechanism for managing the history of context. It is an easy to use, powerful, and reliable system to process and distribute data. Internally, Draco is based on Apache NiFi and Apache MiNiFi (see below for an introduction to Apache NiFi) and is a dataflow system based on the concepts of flow-based programming.

Draco is composed by a set of processors in charge of persisting Orion context data in third-party storages, allowing to create a historical view of such data. Draco uses the subscription/notification feature of Orion. A subscription is made in Orion on behalf of Draco-NGSI, detailing which entities we want to be notified when an update occurs on any of those entities' attributes. Draco is based on Apache NiFi and Apache MiNiFi for cloud-edge or limited resources scenarios.

Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic. Some of the high-level capabilities of Apache NiFi include Web-based user interface, seamless experience between design, control, feedback and monitoring, data provenance, SSL, SSH, HTTPS, encrypted content and pluggable role-based authentication and authorization.

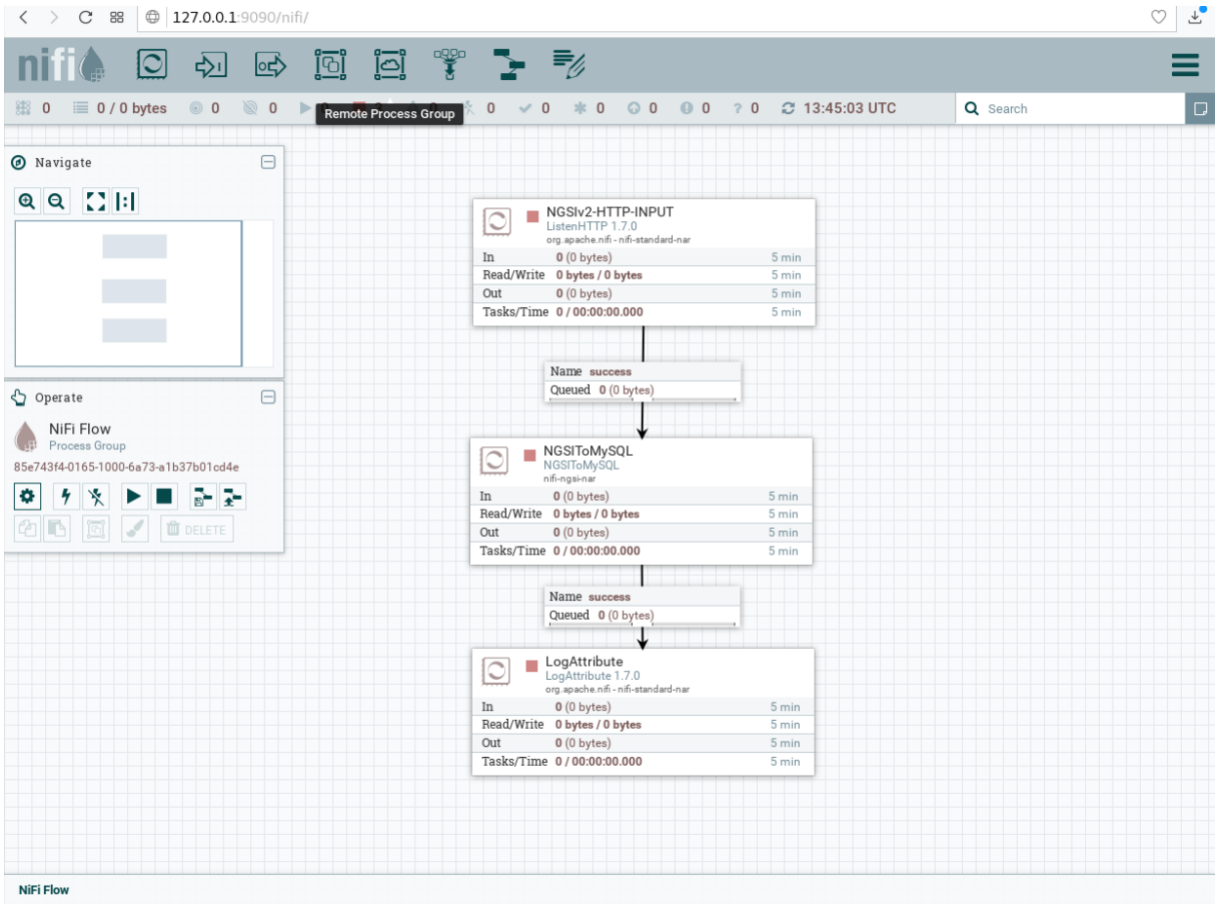


Figure 11: Screenshot of Apache NiFi configuration

Apache NiFi is highly configurable with loss tolerant vs guaranteed delivery, low latency vs high throughput, dynamic prioritization, and runtime flow and backpressure management.

Last but not least, Apache NiFi requests lots of resources, both computational, memory and disk capacity, to put up and run an instance fully operational. It produces some withdrawals in terms of scenarios in which you can apply this component as a Gateway in scenarios where the resources are limited. This is the reason behind the creation of Apache MiNiFi by the community as a subproject of Apache NiFi to collect data where it originates. with less resource's requirements. It is a complementary data collection approach focussed on the collection of data at the source of its creation and actuating immediately at or close to the data source (e.g., sensors, servers or even systems).

The functionalities offered by MiNiFi consist of:

- Small size and low resource consumption
- Central management of agents
- Generation of data provenance (full chain of custody of information)
- Integration with NiFi for follow-on dataflow management

Currently, Draco is based on NGSIv2 but it is expected that the following release includes the support to ETSI NGSI-LD provided by the FIWARE Community. Therefore, we do not introduce here the example code to proceed NGSI-LD messages and we delegate this operation for the next deliverable. Nevertheless, we provide the corresponding SWOT analysis of the component to be applied in the F4W RA.

Figure 12: SWOT Analysis of Draco

COSMOS Generic Enabler

FIWARE Cosmos[21] Big Data Analysis GE is a set of connectors that help to achieve stream and batch processing using well-known Open Source solutions Apache Spark and Apache Flink over Context Data. Both approaches allow the storage, acquire, analyse and process big data in different ways. Therefore, continuous data streams as well as data clusters or data lakes can be queried as well as new conditions can be detected quickly as soon as they are received. But what are the differences between Apache Spark and Apache Flink?

Apache Flink is used to provide stream processing and process data as much faster as possible with high accuracy, performance as well as stability on distributed systems. One of its main characteristics is that it provides high fault tolerance together with a low data latency. The architecture of Apache Flink was developed to process data in real-time.

On the other hand, Apache Spark is a cluster computing framework that processes data very fast and is used mainly for large scale data processing. The main characteristics of this architecture is that it was built around speed processing, the facility to use and improved with analytics functionalities.

It sounds very similar, they have similar APIs and components but from the architectural point of view they have several differences in terms of data processing that should be known in order to select the proper connector and processing engine.

Apache Spark	Apache Flink
The computational model used is based on the micro-batch model. This is an approach in which the incoming tasks are grouped into small batches with the purpose to achieve some performance improvement, without a major increase in the latency of the tasks. Therefore, it is important to see that this approach is not efficient in cases in which there is a need to process large streams of live data or to provide results in real-time.	The computational model used is based on the operator-based streaming model with the purpose to process streaming data in real time. Even batch processing is considered as a stream processing special case.
It can offer low latency responses even compared with Apache Storm but in general compared with Apache Flink, it has higher latency.	No minimum data latency. Apache Flink provides an optimizer totally independent of the programming interface.
Individual jobs are manually optimized and require longer time for processing.	The introduction of pipeline execution produces faster data processing compared with Apache Spark. Additionally, the use of closed-loop operators, specific machine learning libraries, as well as graph processing is faster than Apache Spark.
API is easier to call and use compared with Apache Flink.	The functionalities offered by the API are lesser compared with Apache Spark.
Provide connectivity to Java, Scala, Python and R.	Only available in Java and Scala.

[21] https://fiware-cosmos.readthedocs.io/en/latest/index.html

Data flows are represented in an acyclic graph although machine learning algorithms are cyclic data flow.	Support controlled cyclic dependency graphs in runtime.
Window criteria is time-based.	Window criteria is record-based or customer-defined.
Non-native iteration implemented as normal for-loop outside the system to support data iteration in batches. The inconvenience is that they have to be scheduled and executed separately.	Provide operation and delta iterations thanks to its streaming architecture.
Automated memory management but not yet enough matured.	Memory management system apart from Java Garbage Collector to identify and eliminate spikes.
Strong community support	

Table 2: Apache Spark vs. Apache Flink comparison

Once that we have a clear overview of which type of architecture we can adopt, we can differentiate the corresponding connectors and processing engine. We have to keep in mind that the current version of these components is based on Scala language and therefore any program should be developed using Scala language.

- FIWARE Cosmos Orion-Flink Connector which contains the **NGSILDSource** for receiving NGSI-LD events notifications based on the subscription information provided to the Context Broker through HTTP and the **OrionSink** for sending the processed data back to Context Broker.
- FIWARE Cosmos Orion-Spark Connector, the same approach, it contains the corresponding **NGSILDSource** and **OrionSink** classes to manage the information.

NOTE: Keep in mind that it is needed to have a Spark or Flink cluster to develop solutions based on these connectors.

Once we have introduced the different scenarios and the classes to be used, we introduce the normal way of working on them with a specific example step by step. Imagine that we are subscribed to the Context Broker (see the previous section) in order to receive the notifications about all the Tanks in our Water Network in order to calculate the maximum capacity that we can manage in our network.

The first step is to download the corresponding classes (.jar file) from the latest release:

```
mvn install:install-file -Dfile=$(PATH_DOWNLOAD)/orion.flink.connector-1.2.3.jar -
DgroupId=org.fiware.cosmos -DartifactId=orion.flink.connector -Dversion=1.2.3 -Dpackaging=jar
```

Figure 13: Maven installation of the Apache Flink connector (v.1.2.3)

Additionally, if we want to compile properly the code, we have to include the following lines into our pom.xml

```
<dependency>
  <groupId>org.fiware.cosmos</groupId>
```

```
  <artifactId>orion.flink.connector</artifactId>
  <version>1.2.3</version>
</dependency>
```

Figure 14: Update of your pom.xml file to detail Flink Connector dependency

Next step, in our Scala code we have to import the corresponding new classes with the following command:

```
import org.fiware.cosmos.orion.flink.connector.NGSILDSource
```

Figure 15: Import the NGSILDSource in your Scala program

and of course, we have to assign the data source to the Flink environment, indicating which port will be used to listen the Context Broker notifications:

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
val eventStream = env.addSource(new NGSILDSource(9001))
```

Figure 16: Adding your data source (NGSILDSource)

Finally, we have to parse the received data:

```
val processedDataStream = eventStream.
        .flatMap(event => event.entities)
        // ...processing
```

Figure 17: Process your data stream

The received data is a DataStream of objects of the class NgsiEventLD. This class has the following attributes:

- **creationTime**: Timestamp of arrival.
- **service**: FIWARE service extracted from the HTTP headers.
- **servicePath**: FIWARE service path extracted from the HTTP headers.
- **entities**: Sequence of entities included in the message. Each entity has the following attributes:
  - **id**: Identifier of the entity.
  - **type**: Node type.
  - **attrs**: Map of attributes in which the key is the attribute name, and the value is an object with the following properties:
    - **type**: Type of value (Float, Int, …).
    - **value**: Value of the attribute.
  - **@context**: Map of terms to URIs providing an unambiguous definition.

The Scala method flatMap takes a predicate function, applies it to every entity in the collection. It then returns a new collection by using the entities returned by the predicate function. The flatMap method is essentially a combination of the map method being run first followed by the flatten method which allows afterwards a direct data management over the different attributes. For example, in our example about maximum capacity of an element in the Water Network, the Scala code should be the following:

```
package org.fiware.cosmos.orion.flink.connector.watermanagement.tankexample

import org.apache.flink.streaming.api.scala.{StreamExecutionEnvironment, _}
import org.apache.flink.streaming.api.windowing.time.Time
import org.fiware.cosmos.orion.flink.connector.NGSILDSource

object TankExample {

  def main(args: Array[String]): Unit = {
    val env = StreamExecutionEnvironment.getExecutionEnvironment

    // Create Orion Source and receive notifications on port 9001
    val eventStream = env.addSource(new NGSILDSource(9001))

    // Process event stream
    val processedDataStream = eventStream
      .flatMap(event => event.entities)


      .map(entity => {
        val maxLevel =
entity.attrs("maxLevel").value.asInstanceOf[Number].floatValue()
        new MaxLevel_Node(entity.id, maxLevel)
      })

      .keyBy("id")
      .timeWindow(Time.seconds(5), Time.seconds(2))
      .sum("maxLevel")

    // Print the results with a single thread, rather than in parallel
    processedDataStream.print().setParallelism(1)
    env.execute("F4W Tank Example")
  }

  case class MaxLevel_Node(id: String, maxLevel: Float)
}
```

Figure 18: Full Scala code to get the sum of all maxLevel of the different Tanks

Once you have all the entities with the flatMap function, we can iterate over them, with the map one, and extract the desired attributes; in this case the maxLevel of the Tank. Additionally, on each iteration we create a custom object (MaxLevel_Node) with the purpose to keep only the needed attributes (entity id and maxLevel).

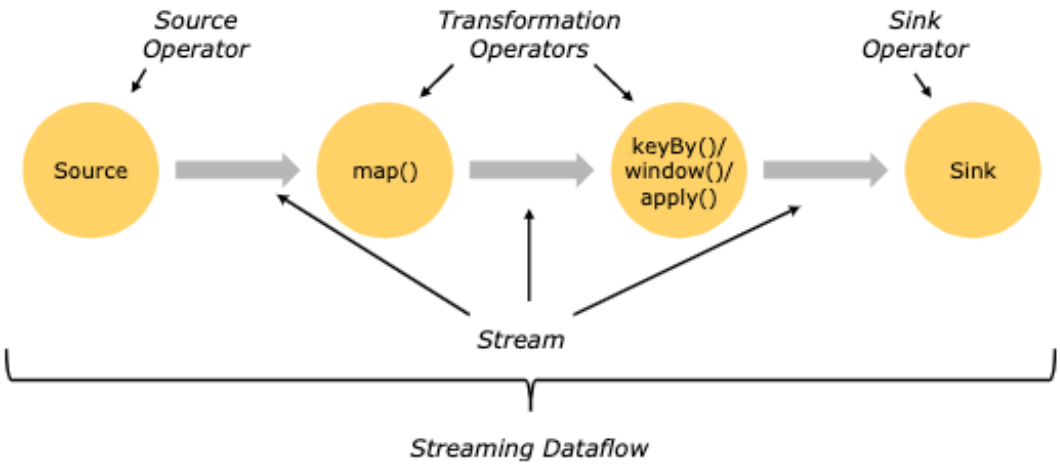


Figure 19: Example of an "entity create" event

Afterward, we group the objects by their entity id (***.keyBy("id")***) and define a custom processing window that represents a time interval between the start (inclusive) and the end (exclusive) (***.timeWindow(Time.seconds(5), Time.seconds(2)***). And finally, obtain the sum of the values for the maxLevel property (***.sum("maxLevel")***).

The same procedure we have to follow for the Spark connector, just changing the reference to the classes and download:

```
mvn install:install-file -Dfile=$(PATH_DOWNLOAD)/orion.spark.connector-1.2.1.jar -
DgroupId=org.fiware.cosmos -DartifactId=orion.flink.connector -Dversion=1.2.1 -Dpackaging=jar
```

Figure 20: Maven installation of the Apache Spark connector (v.1.2.1)

and

```
<dependency>
  <groupId>org.fiware.cosmos</groupId>
  <artifactId>orion.spark.connector</artifactId>
  <version>1.2.1</version>
</dependency>
```

Figure 21: Update of your pom.xml file to detail Spark Connector dependency

If we plan to send back new data to the Context Broker, we have to use the **OrionSink and OrionSinkObject** together with the corresponding classes of the HTTP Request (ContentType and HTTPMethod).

```
import
org.fiware.cosmos.orion.flink.connector.{OrionSink,OrionSinkObject,ContentType,HTTPMethod}
```

Figure 22: Import the OrionSink and OrionSinkObject and dependencies

The next step will be the definition of the parameters, content type and headers needed to send a notification to the Context Broker.

```
final val CONTENT_TYPE = ContentType.Plain
final val METHOD = HTTPMethod.POST
final val URL_CB = "http://localhost:1026/ngsi-ld/v1/entities/urn:ngsi-
ld:Tank:tank001/attrs"
final val HEADERS = Map("fiware-service" -> "openiot",
                        "fiware-servicepath" -> "/",
                        "Accept" -> "*/*",
                        "Link" -> "<https://fiware.github.io/tutorials.Step-by-
Step/tutorials-context.jsonld>; rel=\"http://www.w3.org/ns/json-ld#context\";
type=\"application/ld+json\"")
```

Figure 23: Definition of HTTPCLient variables to be used

The header's attributes have to be passed inside a **Map** class in the key -> value format. Finally, we have to call the Sink stream for each of the data

```
val sinkStream = processedDataStream.map(tempNode => {
      new OrionSinkObject(MaxLevel_Node.toString, URL_CB, CONTENT_TYPE, METHOD,
HEADERS)
    })

// Add Orion Sink
OrionSink.addSink(sinkStream)
```

Figure 24: Processing the Context Broker Sink

The arguments of the OrionSinkObject are as follows:

- **Message**: { "{\"maxLevel\": { \"value\":" + maxLevel + ", \"type\": \"Float\"}}" }. We update the value of the Tank 001 with the maxLevel equal to the sum of calculated maxLevels just for the purpose of the example. We are using a specific function inside the class `MaxLevel_Node.`
- **URL**: "http://localhost:1026/ngsi-ld/v1/entities/urn:ngsi-ld:Tank:tank001/attrs". URL of the Tank entity in the Context Broker to update a specific attribute (in this case maxLevel).
- **Content Type**: ContentType.Plain.
- **HTTP Method**: HTTPMethod.POST.
- **Headers**: Map(key -> value). Optional parameter. We add the headers we need in the HTTP Request (Link, Accept, fiware-service, fiware-servicepath).

```
case class MaxLevel_Node(id: String, temperature: Float) extends  Serializable {
    override def toString :String = { "{\"maxLevel\": { \"value\":" + maxLevel + ",
\"type\": \"Float\"}}" }
}
```

Figure 25: Extended Temp_Node class to serialize the Content Information to update in the Context Broker

The SWOT analysis of the Cosmos connectors can be summarized in the following table.

Figure 26: SWOT Analysis of Cosmos

III.2. Analysis and qualification of new components

Based on the SWOT analysis of the existing FIWARE Generic Enablers performed in the previous section, and on the identification of the missing components, this section presents in more detail these components, how they technically integrate into the F4W-RA, and the on-going developments and integrations for each of them.

During the second half of the project, the DCs' requirements will be monitored in order to provide them all the tools needed to implement and deploy the smart applications. During this period, DCs will probably identify new tools that will have to be included in the F4W-RA.

Streamed ingestion of data

In a typical FIWARE architecture, data is uniquely ingested by issuing requests on the NGSI-LD HTTP API exposed by a context broker (e.g., as it is done by IoT Agents).

This raises some scalability problems if there is a high flow of data coming into the platform, which is expected when Demo Cases will scale up deploying a lot of sensors of their sites. In this case, HTTP is not the right protocol to handle such a flow. So, the need for an intermediate middleware that can handle important concerns like backpressure and guaranteed delivery of messages arises strongly.

What's more, such an intermediate allows to perform some of the data cleaning, validation and streaming tasks before the data gets into the context broker.

Then, a typical flow of data from outside to inside the platform will look like:

- An external component (e.g., sensor, CSV connector, …) sends data to the platform (e.g., a new measure for a sensor, export of some CSV data, …)
- Data is received by a gateway component that is in charge of transforming the data from an external format to a common, NGSI-LD based, format explained below (as is typically done by an IoT Agent)
- Instead of sending the data directly to a context broker, as it is currently done, the gateway component pushes the data to a message broker, on an intermediate topic
- If needed, data cleaning and validation engines deployed inside the platform can listen to this predefined topic, perform any cleaning or validation tasks they have to do on the data, then republish it on a final topic. If there is no such task to perform, a simple component can simply forward the data on the final topic without any additional modification on it.
- A context broker listening on the final topic can finally integrate the data inside the information context
- Additionally, a stream processing engine (like the Cosmos GE embedding Flink or Spark) can subscribe to some topics of interest and run specific algorithms on the flow of data (e.g., anomaly detection)

On the implementation side, schemas have been defined to represent each type of message that can be sent into the platform. The types of the messages map to the different endpoints and operations exposed by an NGSI-LD API:

- ENTITY_CREATE
- ENTITY_REPLACE
- ENTITY_UPDATE
- ENTITY_DELETE
- ATTRIBUTE_APPEND
- ATTRIBUTE_REPLACE
- ATTRIBUTE_UPDATE
- ATTRIBUTE_DELETE

Depending on the type of the message, additional information is conveyed into the message:

- *entityId*: identifier of the entity
- *operationPayload*: the payload of the operation, as it appears in the corresponding NGSI-LD operation
- *updatedEntity*: a snapshot of the entity, in compacted form, after the operation has been applied
- *attributeName*: the name of the attribute subject of the operation, when applicable
- *datasetId*: the dataset id of the multi-attribute subject of the operation, when applicable
- *contexts*: the JSON-LD contexts applicable to the message

To illustrate these new messages formats, the figures below show some concrete examples of events messages.

This first example illustrates an event in the case an attribute replace has occurred:

```
{
   "entityId":"urn:ngsi-ld:Bus:A4567",
   "attributeName":"color",
   "datasetId":"urn:ngsi-ld:Dataset:color:1",
   "operationPayload":"{ \"type\": \"Property\", \"value\": \"red\" }",
   "updatedEntity":"updatedEntity",
   "contexts":[
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
   ],
   "operationType":"ATTRIBUTE_REPLACE"
}
```

Figure 27: Example of an "attribute replace" event

This second example illustrated an event in the case an entity creation has occurred:

```
{
    "entityId":"urn:ngsi-ld:Vehicle:A4567",
    "operationPayload":"{\n \"id\": \"urn:ngsi-ld:Vehicle:A4567\",\n\"type\":
\"Vehicle\",\n\"brandName\": {\n\"type\": \"Property\",\n\"value\":
\"Mercedes\"\n},\n\"@context\": [\n\"http://example.org/ngsi-
ld/latest/vehicle.jsonld\",\n\"https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-
context.jsonld\"\n]\n}",
    "operationType":"ENTITY_CREATE"
}
```

Figure 28: Example of an "entity create" event

Finally, a topic topology has been set up to better orchestrate the flow of messages and define the formats expectations for each topic.

This is how the topology is orchestrated:

- Messages sent by the southbound components (IoT Agents, …) are sent to a topic name *ngsild.raw.entity.<entity-type>*
- Typically, stream processors listen to these topics, apply some transformations and send new entity events in a topic named *ngsild.event.entity.<entity-type>*
- If there is no specific stream processor deployed inside the platform or if there is no specific process to apply for an incoming event, a simple passthrough Kafka Streams component is deployed and republishes events as is in the *ngsild.event.entity.<entity-type>* topic
- Other components of the platform (e.g., ML models) can listen to the raw topic or the transformed one, according to their specific business requirements.

As this streamed ingestion is based on the Kafka (see below for an introduction to Apache Kafka) message broker, the next planned step is to formalize these message formats using Avro. Indeed, Avro is "an Open Source data serialization system that helps with data exchange between systems, programming languages, and processing frameworks. Avro helps define a binary format for your data, as well as map it to the programming language of your choice".

Avro provides the support and tooling to define a common data model for the messages that can then be shared by all the consumers without much effort. It also has bindings for a lot of different programming languages, thus allowing it to integrate consumers (and publishers) developed in (almost) any programming language.

Introduction of Apache Kafka

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Kafka combines three key capabilities in order to support different use cases for end-to-end event streaming with a single battle-tested solution:

- To publish (write) and subscribe to (read) streams of events, including continuous import/export of your data from other systems.
- To store streams of events durably and reliably for as long as it is required.
- To process streams of events as they occur or retrospectively.

All these functionalities are provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner. Kafka can be deployed on bare-metal hardware, virtual machines, and containers, and on-premises as well as in the cloud.

Kafka's main core capabilities are:

- High throughput: Deliver messages at network limited throughput using a cluster of machines with latencies as low as 2ms.
- Scalability: Scale production clusters up to a thousand brokers, trillions of messages per day, petabytes of data, hundreds of thousands of partitions. Elastically expand and contract storage and processing.
- Permanent storage: Store streams of data safely in a distributed, durable, fault-tolerant cluster.
- High availability: Stretch clusters efficiently over availability zones or connect separate clusters across geographic regions.

Kafka clusters can be self-managed, but there also exist several fully managed services offered by a variety of vendors.

Model serving

There currently exists one FIWARE Generic Enabler that provides ML capabilities: Cosmos, which allows the use of the ML features of Spark. Spark is a great and powerful stack, but it can also be a complex one as it not only covers ML but also streaming, batching, graph processing, … and may require some complex configurations and setups.

Some of the AI requirements from the Demo Cases only require a service to perform predictions and calculations based on some historical and real time data.

Thus, this has triggered the opportunity to identify and integrate a third party, more lightweight and ML focused tool.

One objective is to provide a component that is easy to deploy on a platform (i.e., it does not require a complex setup of infrastructure and is ideally self-contained as a Docker image). The added value in such a behaviour is to give the possibility, for a Demo Case, to deploy a new ML model in a few seconds without any hassle and to almost immediately get first results from it. Of course, it also allows to easily stop any deployed ML model that would have been deprecated by a newer and more efficient one.

So far, some tools have been identified and are being analysed:

- BentoML: Open source framework that makes it possible to go from trained ML models to production-grade prediction services with just a few lines of code. It supports all major ML frameworks (PyTorch, TensorFlow, Scikit, Keras, …) and is designed to work natively in a DevOps friendly architecture.
- ML Flow: Open source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry. It has a wider spectrum of features than this current requirement of model serving, but it also supports all major ML frameworks. However, its DevOps capabilities will have to be checked carefully.
- OpenDataHub: Open source platform qualified as a blueprint for building an AI as a service platform (based on Open Stack). It offers great features, but it is a large platform and does not fulfill the requirement of having a simple self-contained deployable micro-service.

We conducted some first experiments with the BentoML solution. They have already showed promising results:

- An already trained ML model can be embedded and deployed quite easily
- Interactions (e.g., asking for a prediction based on input data) are easy to perform, thanks to the natively provided HTTP API

The next actions to be conducted in the coming months are the following:

- Wrap the component into an NGSI-LD aware layer, in order to obtain an interoperable, NGSI-LD based, FIWARE-compliant, module
- Integrate the component with the necessary tooling and mechanisms to allow on-demand and immediate deployment of a new embedded ML model (along the principles of "Machine Learning as as Service")
- Experiment and validate the observability features of the component
- Document and apply the FIWARE recommendations so that it joins the FIWARE catalogue

Mechanisms and life cycle of a ML model

Related to the model serving topic discussed above, a standardized way to interact with a ML model has to be defined, in the objective to cover the following user requirements:

- Register a ML model in the context broker, providing the necessary information and metadata, and optionally giving the rights to use it to an user or group of users. A first directory in the incubator repository has been created to work at the definition of the corresponding data model, and it is expected during the project execution that a generic data model describing machine learning algorithms will be completed. The registration of a ML model will use the standard NGSI-LD to create an entity, as a ML model will at the end be represented as a standard entity.
- Ask to deploy a ML model triggered by a subscription based on certain conditions to be fulfilled. A first way to do this will consist in creating an "ML model instance" entity, that will have a relationship with a previously registered ML model entity. Then, a newly developed component, subscribed to entities of type "ML model instance", will be notified of this creation. Upon this notification, it will deploy a new component embedding the associated ML model, set the correct access rights, and send back to the context broker the necessary information, attached to the ML model instance entity, so that a consumer can later interact with the deployed ML model (for instance, the URL where it is accessible).
- The consumer who has created the ML model instance can now interact with the deployed model. To do this, the following minimal information is needed:
    - Input data, which can be communicated either directly in the request body (not the recommended way as it implies large payloads and it also implies that the consumer has to store the input data locally), or preferably via a link to a temporal query that returns the set of expected input data. In the latter case, the temporal query looks like "*/ngsi-*

ld/v1/temporal/entities/urn:ngsi-ld:MlModelInstance:ID?timerel=after&time=2020-01-01T14:30:00Z&attrs=attributesOfInterest&options=temporalValues" and returns an entity containing a property populated with its temporal evolution in the given time range (e.g. "*[[12.0, 2020-01-01T15:30:00Z],[12.1, 020-01-01T15:30:00Z],...]*"). The deployed model can then use these values to perform the required computation.

- An optional endpoint to send the results to. In any case, the results are stored inside the ML model instance entity, inside a multi-instance property, so that a given instance can store the results of many computations

The major change in the process described above is to develop and deploy a new component in charge of performing the automatic and continuous deployment of ML models. In this respect, a focused, DevOps ready and preferably lightweight is preferred, as it has to allow quick and performant delivery, using only a minimum number of resources. As explained in the previous section, as of today, BentoML is the component that is being experimented.

Real time analytics and visualisation

All the Demo Cases expressed a requirement to visualize and analyse the real time data that is flowing inside the platform. For this, it is needed a tool that can not only easily offer a way to build complete dashboards, updating in real time, with a rich set of visualisation options, but also a tool can be easily adopted and customised by non-technical people.

Currently, there exists the Wirecloud Generic Enabler that offers visualisation capabilities, but it is more oriented at providing a web mashup platform to make it easier to develop operational and easily customizable dashboards. Thus, it is less suited to requirements related to analytics and data visualisation updating in real time.

That's why, to support in this task, the Grafana solution has been integrated as part of the on-going work in the F4W project:

- It is an Open Source software
- It offers a rich set of visualisation options (called "panels" in Grafana world): graphs, maps, tables, gauges, …
- It offers a rich set of data sources, among which the main data sources already used in the F4W-RA: PostgreSQL / TimescaleDB, MySQL, ...
- It can easily be extended to add new visualisations or new data sources
- It provides a catalogue of plugins where you can find a lot of contributions for visualisations, data sources, or else dashboards
- It allows to define alerts on missing, incomplete or erroneous data

The solution is already integrated in the UK Demo Case platform to allow displaying business dashboards related to water consumption and alerts raised by the water smart meters deployed in the Great Torrington area.

To benefit from a smooth and optimized integration within the FIWARE ecosystem, an NGSI-LD data source plugin is under development: https://github.com/easy-global-market/grafana-ngsild-plugin.

As show in the figures below, it currently allows to:

- Display a synopsis table of an NGSI-LD entity
- Display a selection of entities on a map

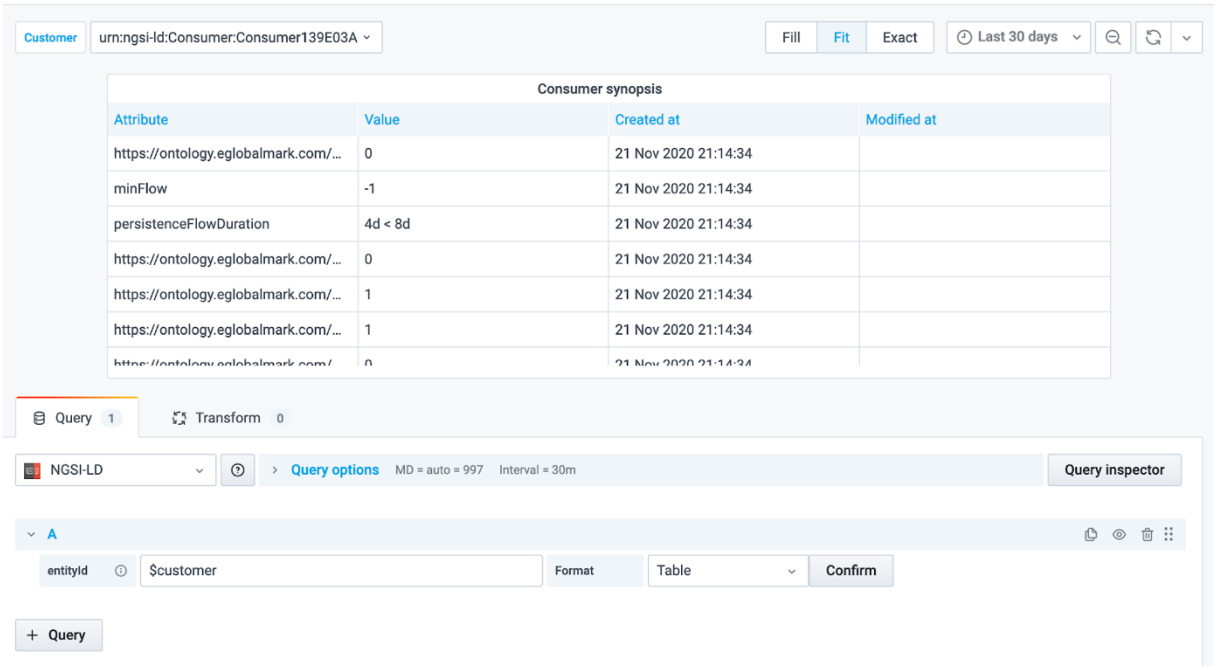


Figure 29: Configuration of the synopsis table of an NGSI-LD entity

Consumer synopsis

Attribute	Value	Created at
https://ontology.eglobalmark.com/WaterSmartMeter#alarmMetrology	1	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#alarmSystem	1	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#alarmWaterQuality	0	2020-11-23 02:44:17
minFlow	1	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#alarmFlowPersisten...	Nothing to report	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#alarmInProgress	1	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#alarmTamper	0	2020-11-23 02:44:17
maxFlow	880	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#moduleTampered	1	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#acquisitionStageFai...	0	2020-11-23 02:44:17
https://ontology.eglobalmark.com/WaterSmartMeter#alarmStopsLeaks	1	2020-11-23 02:44:17
persistenceFlowDuration	15m < 60m	2020-11-23 02:44:17
waterConsumption	187111	2020-11-23 02:44:17

Figure 30: Display of the synopsis of an NGSI-LD entity

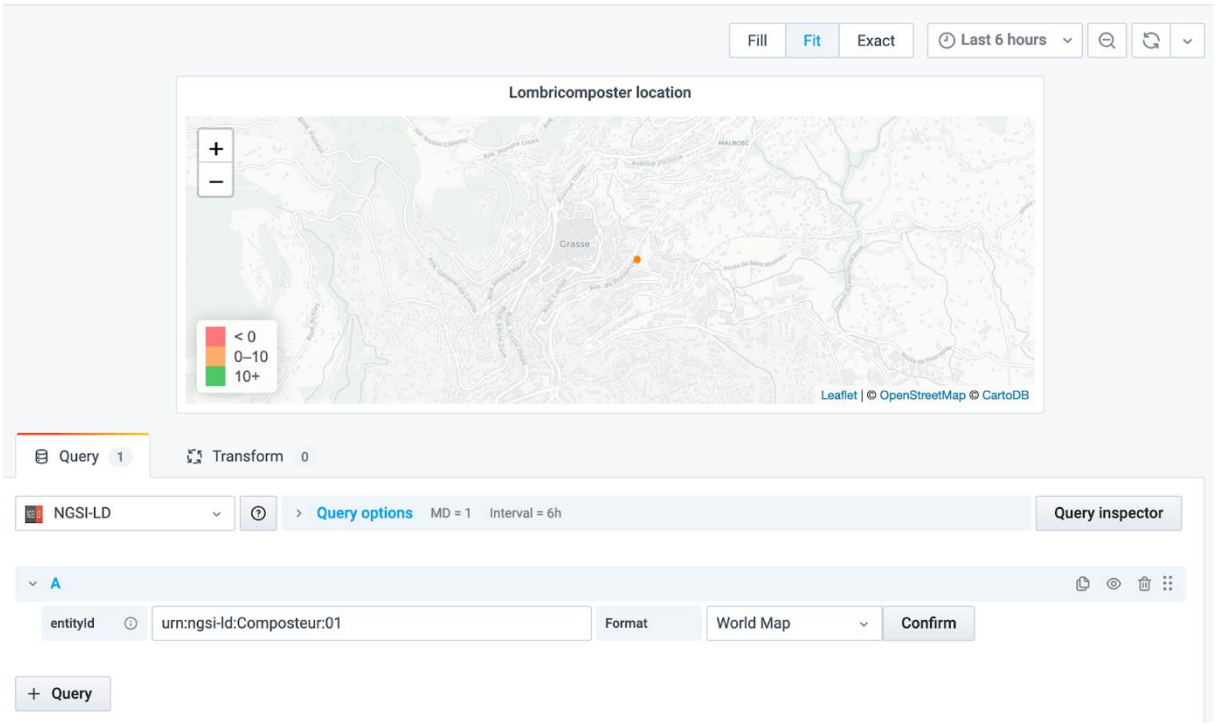


Figure 31: Configuration of the map display of an NGSI-LD entity

IV. Updated architecture for the Demo Cases

Based on the existing and new Big Data and AI tools introduced in the previous sections, this section elaborates updated architectures for each Demo Case.

It aims at demonstrating how these tools will be integrated and how they will communicate with the other components of the F4W-RA. Also, for each Demo Case, the main flows of data and responsibilities for each introduced component will be detailed.

IV.1. Demo case #1. Athens (Greece)

The system reference architecture for Athens Demo Case is graphically presented in Figure 32, providing a schematic overview of the underlying mechanisms enabling data transport and manipulation along with the essential FIWARE modules.

Legacy systems of EYDAP are depicted on the right of the image (yellow area). To support and facilitate the integration of existing sensors, the *Data Warehouse* of EYDAP (as one-stop service) was employed. The data from both the existing flow and quality sensors, stored up to now into different information systems within EYDAP, are redirected into *Data Warehouse*. A scheduler was developed and configured into the different subsystems to update *Data Warehouse* with new data, when are available. The new flowmeter and water level sensors will be also integrated with *Data Warehouse*, after the completion of their installation. The data from the *Data Warehouse* are scheduled to be exported to the File Storage/Main File System area where they become available to the other modules (see upper middle box). Then, the "CSV IoT Agent" retrieves the files from the File Server in a timely fashion and converts them into NGSI-LD requests for the Orion-LD Context Broker. Orion-LD uses a subscription-based system to notify external real-time applications, depicted on the left side of the architecture diagram (magenta area). To certify data persistence, Cygnus-LD module is notified when new data are available, inserting them using SQL queries to the PostgreSQL/PostGIS Database Server (see second middle box). NESSIE (NTUA in-house developed Web Server and Data Analysis & Archiving Engine) retrieves on real-time basis timestamped data from the Context Broker after subscribing, while in cases where historical data are required, NESSIE retrieves data from the PostgreSQL database server (see third middle box). NESSIE also provides the necessary tools to analyse historical and real-time data, supporting their display into user defined dashboards using a modern asynchronous web interface.

To support fast and robust Big Data processing in real-time, the "Apache Spark" module is also included in the platform architecture. This module provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing. Apache Spark module executes the necessary Big Data processing operations and pushes the results back to Orion-LD Context Broker (via *Cosmos* Big Data Generic Enabler) so as for registered 3rd party applications to be notified.

The FIWARE components (i.e., the *IoT Agent*, the *Orion Context Broker* with its MongoDB counterpart, and the *Cosmos* Big Data Generic Enabler) incorporated in the reference system architecture for Athens demo case are presented in Figure 32 in a blue box.

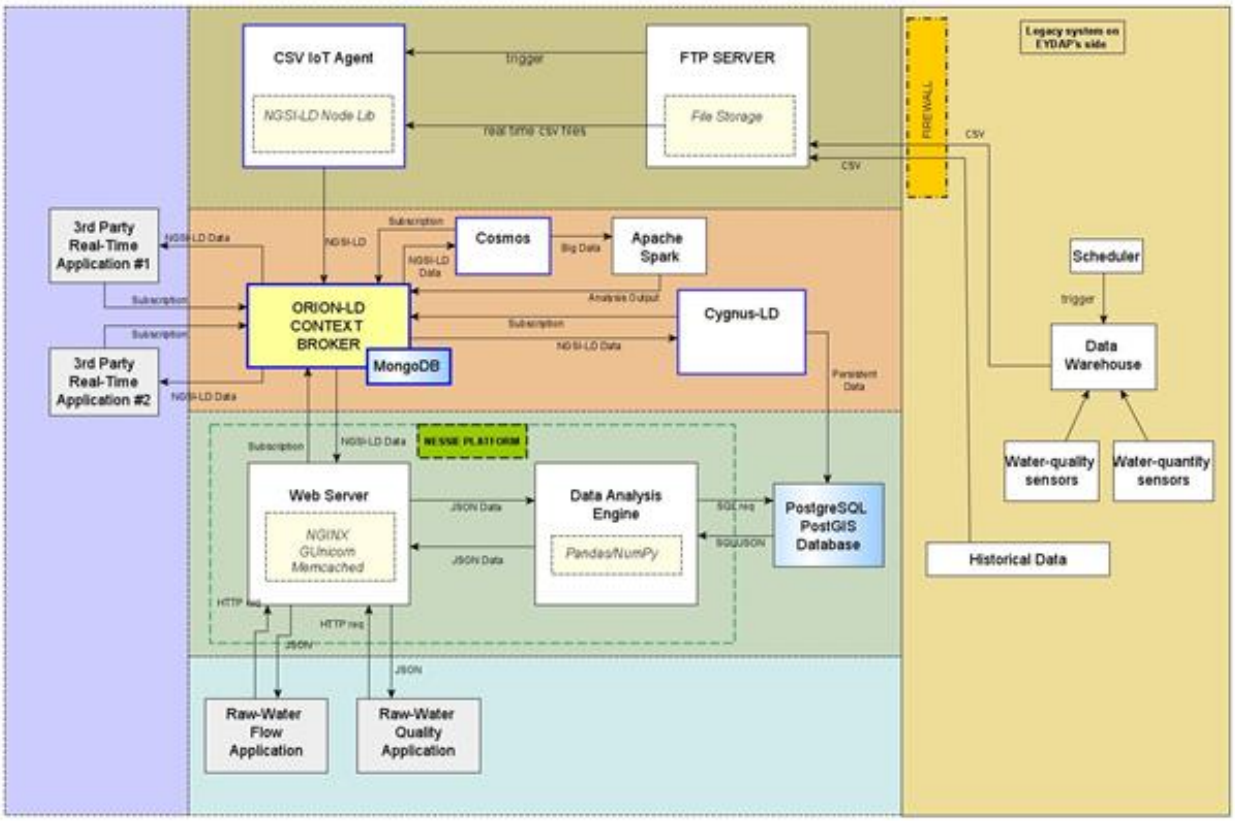


Figure 32: Overview of the FIWARE architecture that communicates with the legacy system and the smart application layer (DC1)

IV.2. Demo case #2. Cannes (France)

In the figure of the DC2 architecture scheme, the updated version of the reference architecture for DC2 is given. It includes:

- The systems managed by 3S in the blue blocks,
- The FIWARE components in the red blocks

The communication principles to be put in place are the following:

- Aquadvanced Water Networks (AWN) is the legacy system that receives data from sensors already deployed on Demo Case site
- Upon an action of a 3S operator, AWN sends an export of sensor data to the NGSI-LD context broker.
- The scientific models (SM) legacy system, that has previously subscribed to such data, is notified (via an HTTP POST request) of this new data (that is conveyed inside the notification)
- The SM component performs evaluations of the scientific models it embeds and sends back the results to the NGSI-LD context broker when they are ready
- AWN, that has previously subscribed to the results, is automatically notified (via an HTTP POST request) of the availability of the results (that are conveyed inside the notification)
- Complementary to this, newly deployed sensors (known as “nanosensor”) will communicate directly with the FIWARE platform (i.e. without transiting into a legacy system). More precisely, real time data from the nanosensors is sent (in LoRaWAN or Modbus) to an IoT Agent that handles the sensors provisioning, and the transformation of the received measures into valid NGSI-LD payloads

- Finally, the Cosmos Generic Enabler is deployed inside the platform. It is used by other partners of the project to run their own machine learning models on the sensor data that flows into the platform.

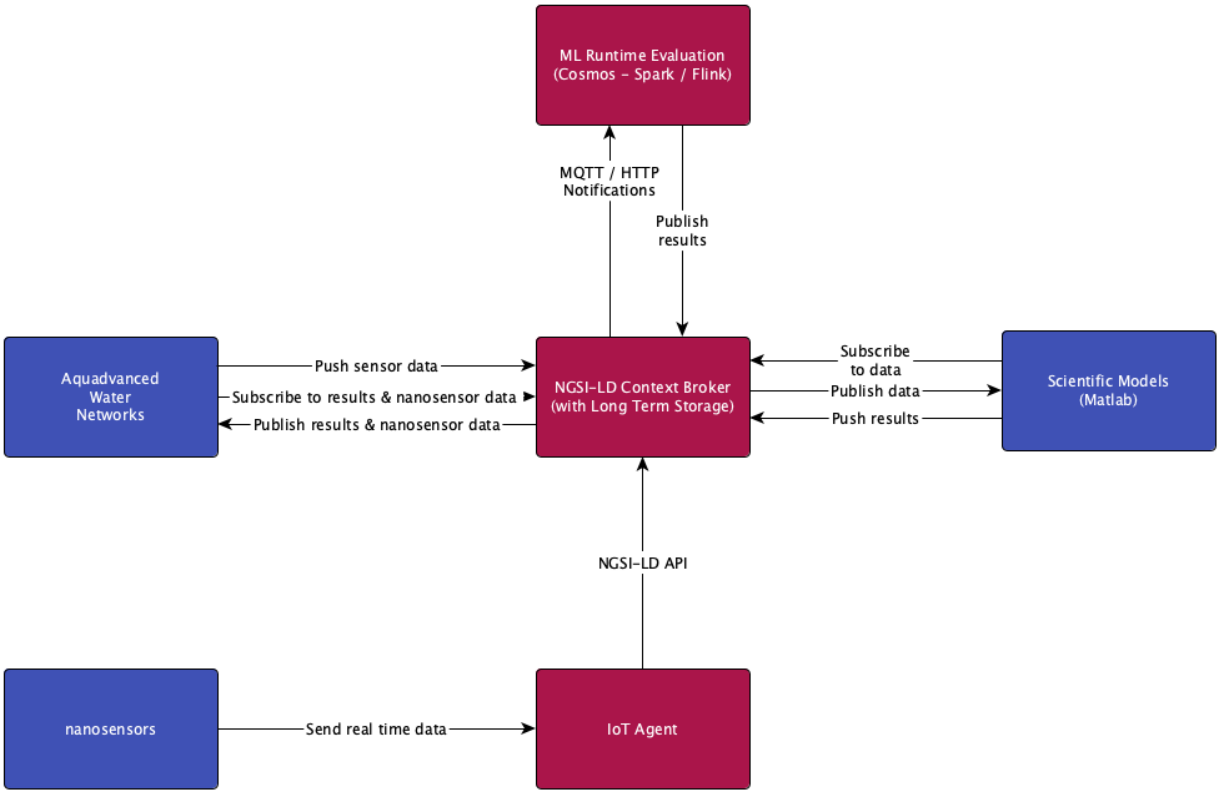


Figure 33: Overview of the FIWARE architecture that communicates with the legacy system (DC2)

IV.3. Demo case #3. Amsterdam (the Netherlands)

In the figure of the DC3 architecture scheme, the updated version of the reference architecture for DC3 is given. It includes:

- The legacy system in orange block,
- The FIWARE components in the blue block
- The smart applications in the green block

Measured data from the new and existing sensors will flow through the Orion Context Broker to the smart application layer where data will be validated by using statistical and Spark AI methods and sent to CrateDB via the context broker. From CrateDB, validated data will flow to the control model in the SMART application layer and subsequently through the Orion Context Broker back to the legacy system for setting the control setpoints of the research lane by the DCS system.

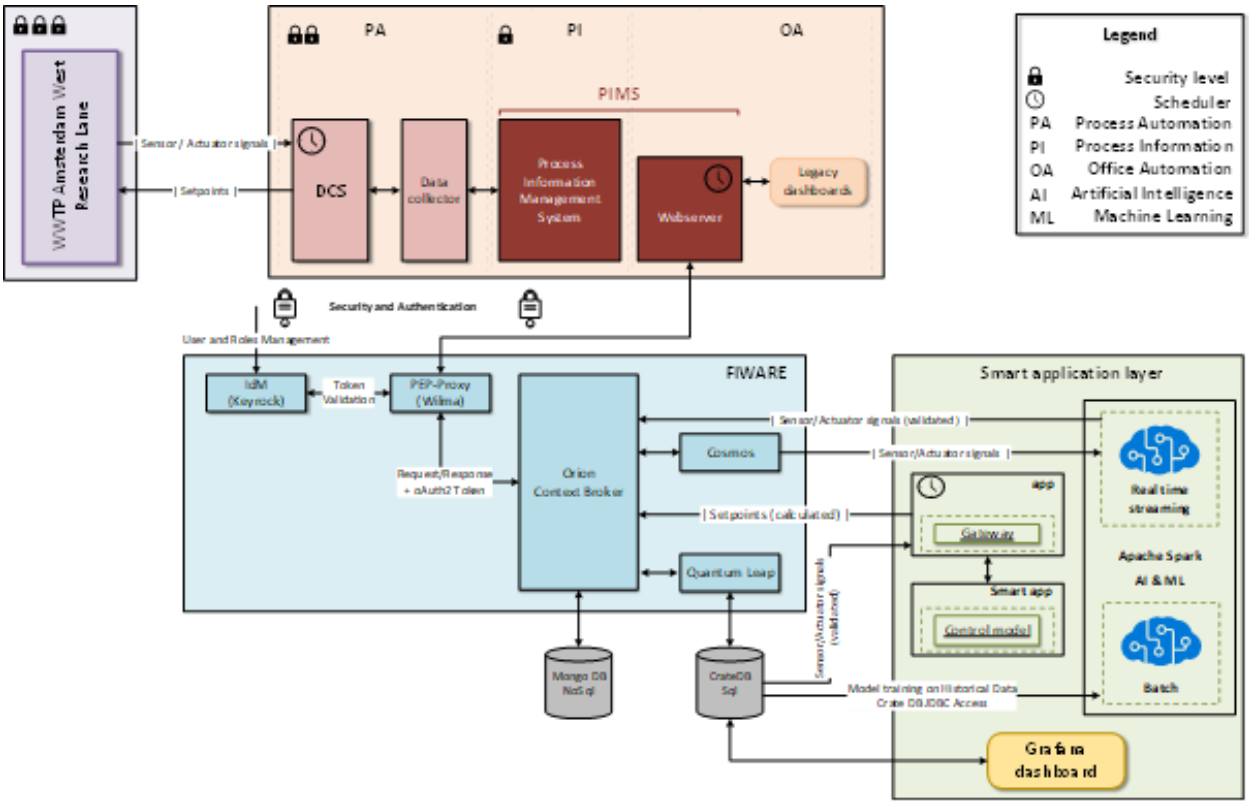
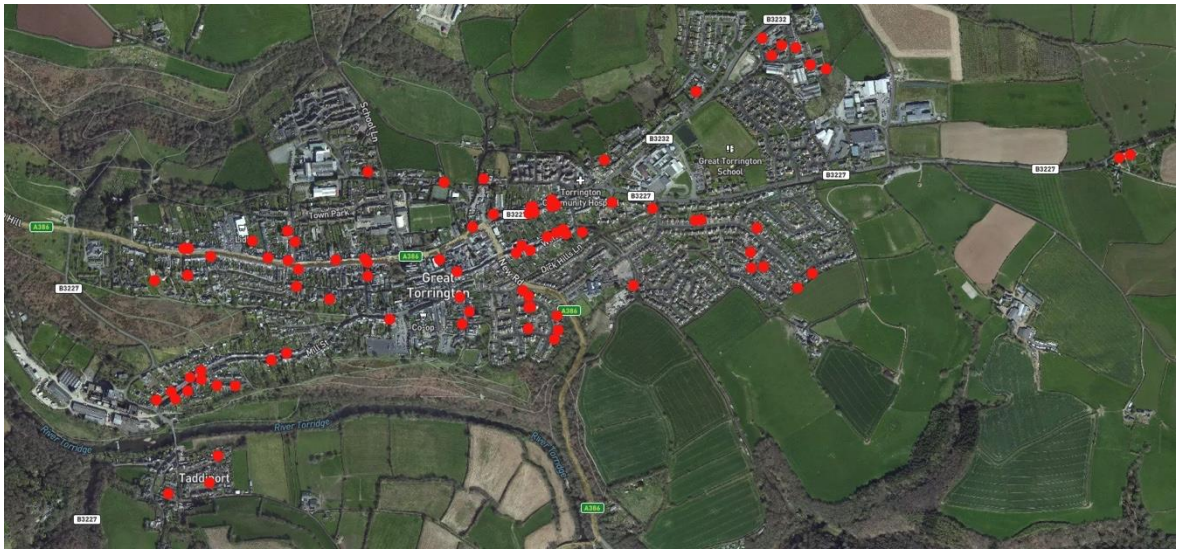


Figure 34: Overview of the FIWARE architecture that communicates with the legacy system (DC3)

IV.4. Demo case #4. Great Torrington (United Kingdom)

Currently, SWW has FIWARE access to historical smart meter data for the Gt.Torrington pilot through Stellio Context Broker. Data is currently collected on a daily basis (one datapoint per meter/day) consisting of cumulative consumption and various alarms. Meters have geographic locations.



Based on the requirements expressed for the UK Demo Case, the following schema provides a proposal for an updated architecture deployment taking account of Big Data and AI processing.

The main points are:

- The Sigfox IoT Agent publishes telemetry data received from the water smart meters to a Big Data ingestion component.
- This Big Data ingestion component, based on the Kafka message broker, notifies the components that have subscribed to this telemetry data.
- The stream processing module, based on the Cosmos Generic Enabler, which has subscribed to receive every new data coming from the smart meters, applies any processing it needs of the messages (for instance, standard predictions based on linear regression or anomaly detection) that it needs. To help in this task, the module can, if needed, query the context broker for historical data. Finally, it uses the context broker to permanently store all the processed results.
- Similarly, a ML module is also subscribed to the data coming from the smart meters. It can perform real time evaluation of machine learning models. As for the stream processing module, it can query the context broker for historical data and it also uses the context broker to permanently store all the processed results.
- Complementary to this, the ML module can be invoked “on demand”, via a ML extension to the NGSI-LD API, by the user app in order to perform on demand predictions for users’ needs.

For each of these new modules, different technical components are being evaluated in real situation:

- For the stream processing module, the Cosmos Generic Enabler has been identified (internally using either Spark or Flink).
- For the ML runtime evaluation, the Cosmos Generic Enabler is also evaluated. Complementary to this, Bento ML is also evaluated to serve as a pure ML serving component. If the experiment shows positive results, BentoML will then be integrated into the FIWARE ecosystem as a new Generic Enabler.

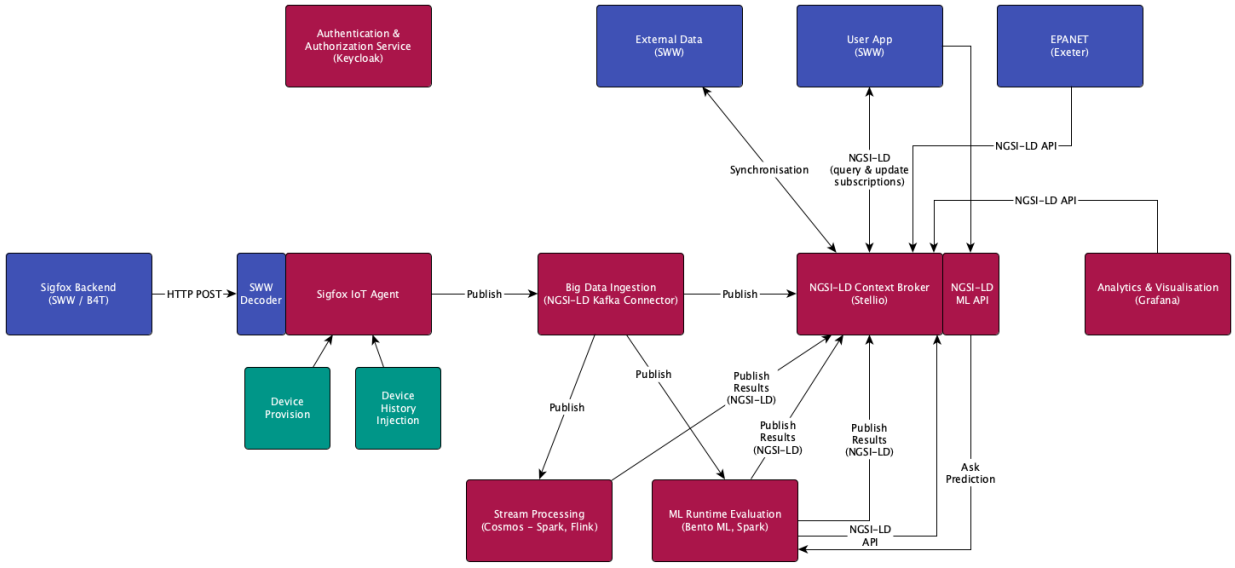


Figure 35: Overview of the FIWARE architecture that communicates with the legacy system and the smart application layer (DC4)

Finally, analytics and visualisation dashboards have been setup to allow SWW to have a better insight into their data:

Figure 36: Summary view of a single water smart meter

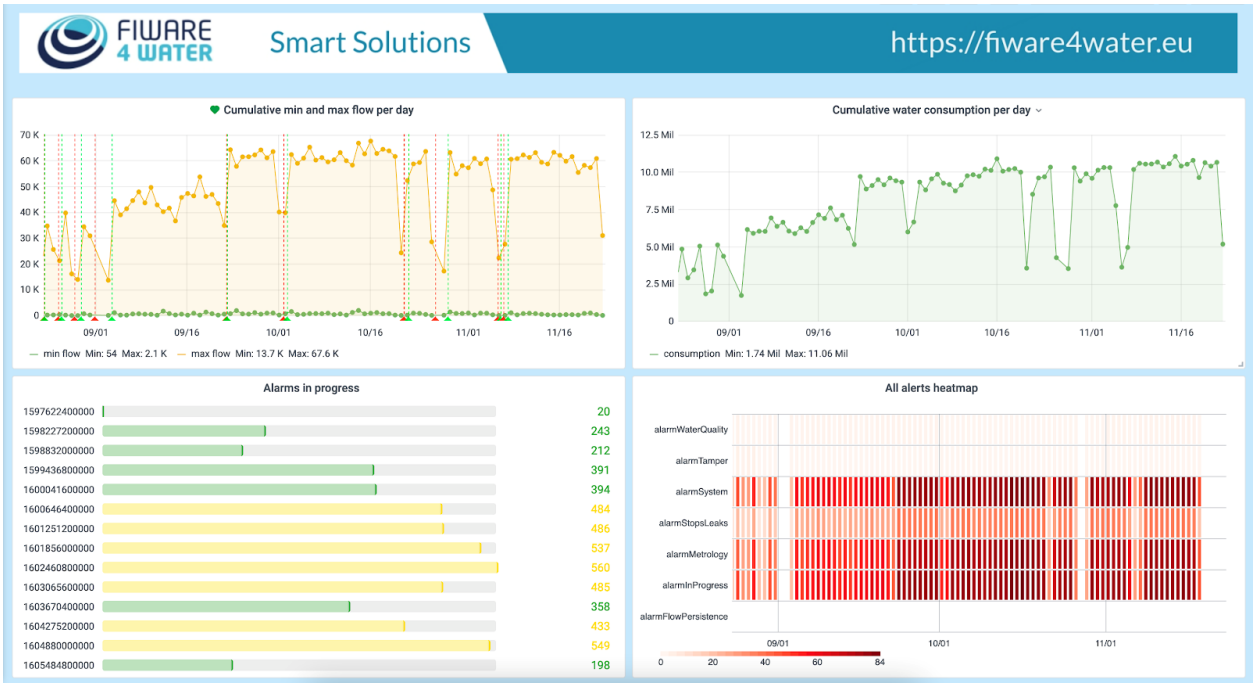


Figure 37: Cumulative view of all the installed water smart meters

V. Smart Data models for Big Data and AI support

This section introduces and presents the first data models designed to support the Big Data and AI tools introduced inside the F4W-RA. They will be completed and enhanced as the project moves forward and tools are exercised.

As explained in Deliverable D2.1, data models are groups of relationships and properties defining a real or conceptual entity required for the management of a system. This case refers to water networks and other water infrastructure elements. These data models are compatible with the reference standard NGSI-LD allowing the digital representation of the assets of our system.

The water network management data model is intended to model the main physical entities of a water network. The chosen approach derives from the model used by the EPANET simulation platform with the final purpose to execute these simulations based on the actual water network state so as to execute advanced scenarios such as real-time alerting, predictive evaluation and what-if analysis. This implies the importance of capturing real system behaviour in the following an efficient (virtual) data model that serves as the basis for experimentations and intelligent processes.

This leads to focus more on the processes of data modelling which have to be realised through three main layers:

1. Physical Water Deployment

In this layer, physical entities of the network (Curve, Junction, Pattern, Pipe, Pump, Reservoir, Tank and Valve) are described through their NGSI-LD properties. Interactions and links between these components are captured by NGSI-LD relationships. This process is based on translating existing EPANET models into the requisite NGSI-LD format. Main NGSI-LD entities definitions and specifications are depicted in the appropriate water network management data model from the FIWARE smart data models initiative located in https://github.com/smart-data-models/dataModel.WaterNetworkManagement

The physical water deployment layer also has to connect observations coming from the IoT environment to their appropriate placement in the water components. The observed values are then used by simulation algorithms when capturing the real-time behaviour of the water network.

This layer will aliment the NGSI-LD context broker with all the evolutions of the behaviour of the real physical water network that can be queried and exploited.

2. Water Network Topologies

The existing water networks are not only defined through physical entities, they are also often organised as several sets of connected sub-networks where the physical entities may be a (sub) part of them. In a water distribution network, this is generally called sectorisation or district metered area (DMA). It is thus important to represent these topologies within the model to ease its management and to be able to run simulations of different parts (sub-network) of the network. As an example, a what-if simulation can be executed on some parameters of a Pump and results are expected to be visualized on a Tank related to this Pump with Pipes and Junctions, thus all these components have to belong to the same network (or sub-network) in order to capture their behaviour.

To model the Water Network topologies in NGSI-LD, these rules are applied:

- All Water Networks and Water Sub-Networks are defined as “WaterNetwork” NGSI-LD entities.
- To relate a sub-network to its components in both directions we introduced the relationship “isComposedOf”.
- To relate a Network to its component sub-networks we introduce an additional reciprocal relationship ‘hasSubNetwork”.

- In order to support the multi-sub-Network belonging, the NGSI-LD multi-attribute aspect is supported.

The water network called "*SophiaWaterNetwork*" has two sub-networks called "*TreatmentPlant*" and "*DM1*". Each sub-network is composed of a set of physical water components. The NGSI-LD data model of this example is illustrated in the figure below:

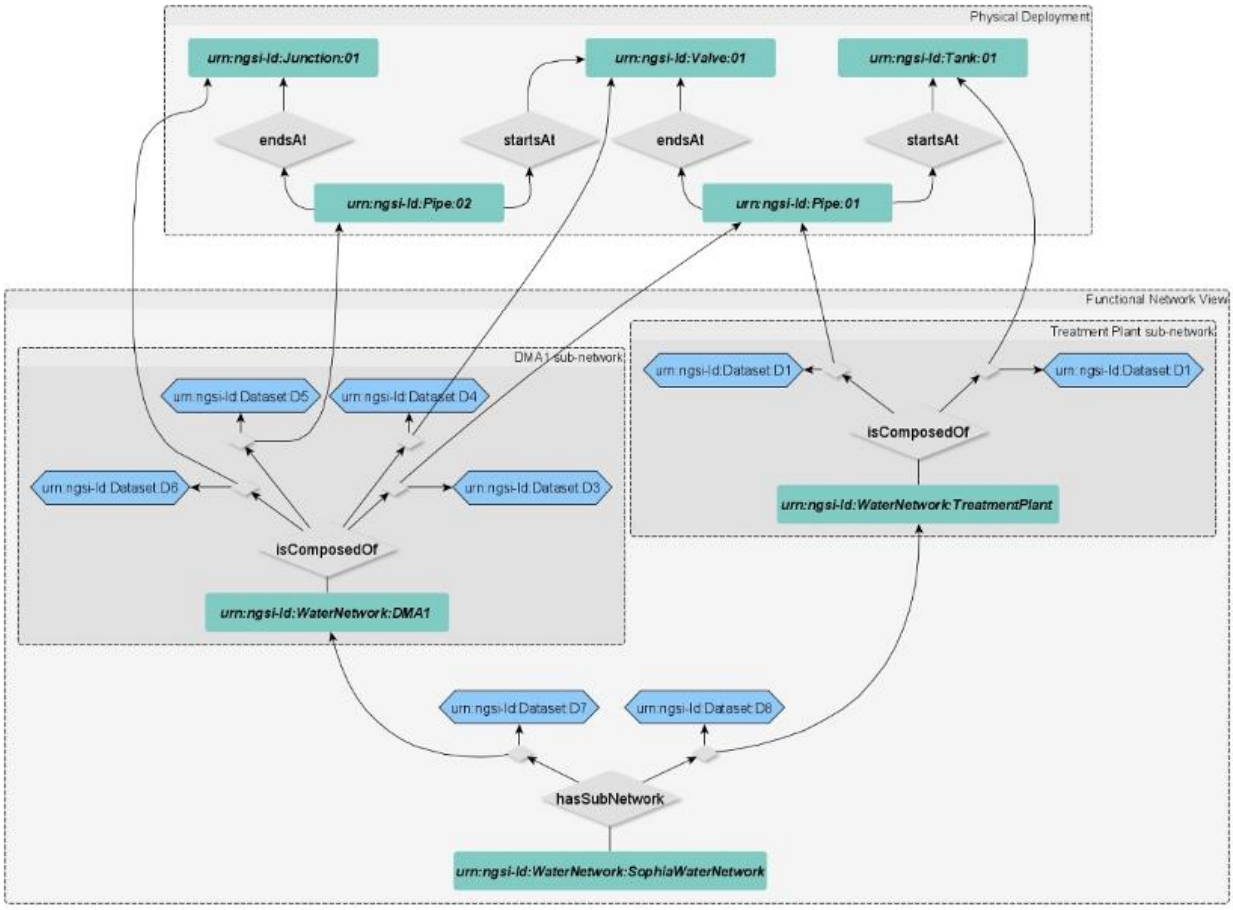


Figure 38: Water network data model

An example of NGSI-LD serialization of the Sophia Water Network Entity is depicted below:

```
{
     "id": "urn:ngsi-ld:WaterNetwork:SophiaWaterNetwork",
     "type": "WaterNetwork",
     " hasSubGraph": [
       {
            "type": "Relationship",
            "object": "urn:ngsi-ld:WaterNetwork:DMA1",
            "datasetId": "urn:ngsi-ld:Dataset:07"
       },
       {
```

```
            "type": "Relationship",
            "object": "urn:ngsi-ld:WaterNetwork:TreatmentPlant"
            "datasetId": "urn:ngsi-ld:Dataset:D8"
        }
      ],
     "@context": [ "https://schema.lab.fiware.org/ld/context" ]
}
```

Figure 39: Short NGSI-LD representation of a water network

3. Simulations scenarios

We distinguish three main types of scenarios:

- Real-Time scenario: which reflects the current real behaviour of the water network. In such cases intelligent applications or end-users need to query (real) data. This is mainly ensured by the physical water deployment layer via querying the context broker following the NGSI-LD API specifications.
- Predictive Scenarios: which defines some rules to be applied on the current water network. These rules could be:
    - Alerting Rules: this is ensured by the NGSI-LD API subscription services. In fact, the NGSI-LD context broker offers several tools for making subscriptions on available data on the broker. For instance, it is possible to define a subscription on a property when its value exceeds a threshold. The context broker will then send notifications when this threshold is exceeded.
    - Operational Rules: this is based on descriptions of controls applied to the current network. This implies that there are some parameters to be applied to the current network following the defined controls.
      A simulation Scenario Entity is defined for this purpose. The NGSI-LD property “operationalControl” will model data of this type of Predictive Scenarios.

      The “operationalControl” property contains:
        - § “triggerLevel”: level at which control is activated.
        - § “setting”: setting applied to the controlled link when trigger level is reached.
        - § “controlledLink”: link controlled.
        - § “monitoredNode”: node which is monitored for control trigger level

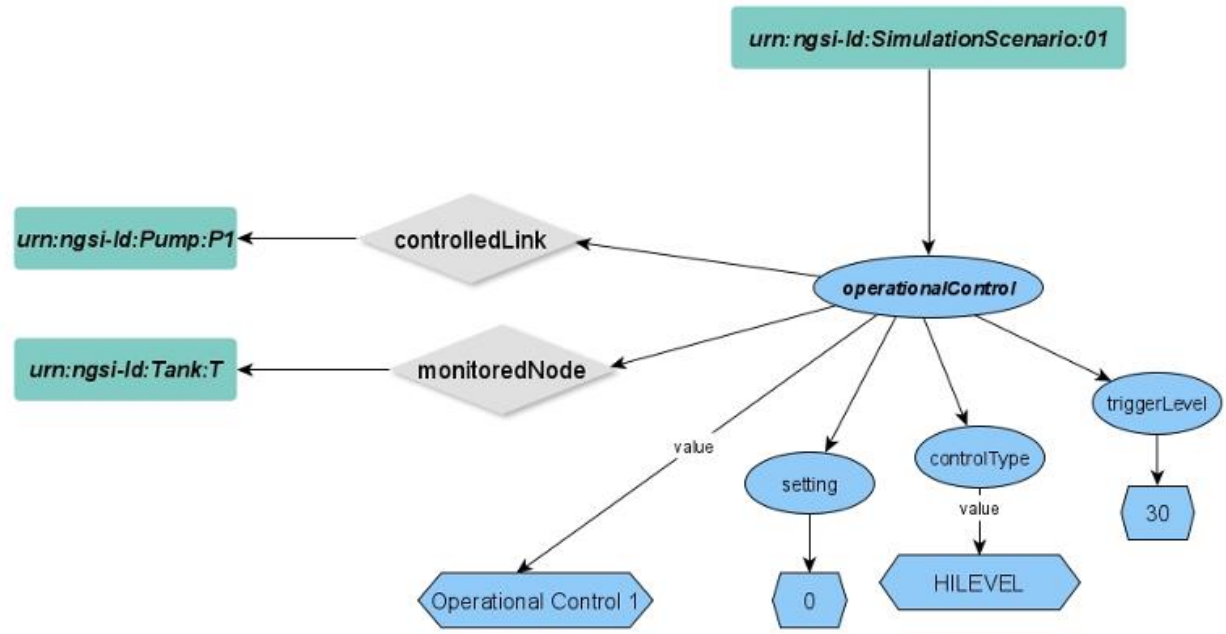


Figure 40: Operational Control Property

- What-if scenarios: which would modify some parameters of the current water network and then check the results of these modifications. This example of scenario needs to store the scenarios actions, results and the state of the network when this scenario was executed.

For these purposes, a simulation Scenario NGSI-LD Entity is defined in order to store the input parameters in the scenarios. A simulation Result NGSI-LD Entity is also defined to capture the result of running the simulation. Both of these entities are connected to the water Network Entity (defined in the second layer) using the NGSI-LD relationship "hasInputNetwork". When running a simulation, the actions of this simulation are stored in the simulation Scenario Entity, Results are stored in the simulation Result Entity and all parameters of the current network are stored (archived) in their physical entities (ensured by the first layer) and linked to the scenario entities via the Water network entity (ensured by the second layer).

The input parameters in the Simulation scenario entity are ensured via its "inputParameter" NGSI-LD properties. The principle of this property is as follows; any modified parameter is modelled as a sub-property of the input parameter. The concerned physical entity is also modelled as a sub-relationship "targetURI" of the current modified property.

For instance, the NGSI-LD data model of a Simulation scenario that contains one action which is what if we modify the setting property of the Tank of water Network will be modelled as in the figure below:

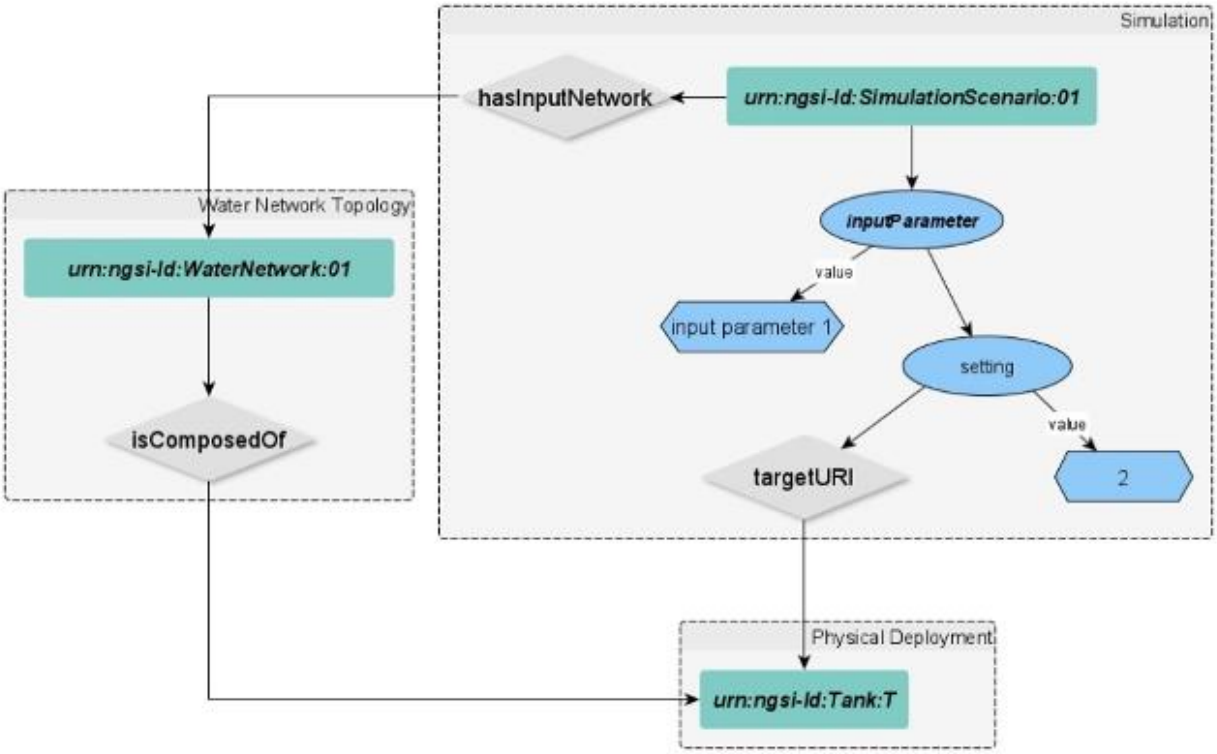


Figure 41: Simulation Scenario with one parameter

The Scenario Entity contains the property “inputParameter”, the “setting” property is a sub-property of “inputParameter”. The value of setting will be assigned to the tank as an action of this scenario. A possible NGSI-LD serialization of the previous entity is depicted below:

```
{
     "id": "urn:ngsi-ld:SimulationScenario:01",
     "type": "SimulationScenario ",
     "inputParameter": {
              "type": "Property",
              "value": " input parameter 1",
              "setting": {
                     "type": "Property",
                     "value": " input parameter 1",
                     "targetURI": {
                             "type": "Relationship",
                             "object": "urn:ngsi-ld:Tank:T",
                            }
                     }
       } ,
     "@context": [
```

```
        "https://schema.lab.fiware.org/ld/context"
        ]
}
```

Figure 42: Short representation of simulation entity

As a simulation scenario may have more than one action, the “inputParameter” supports the multi-attribute property introduced in version 1.3 of the NGS-LDI specification.

The example depicted in the figure below shows a Simulation scenario with 3 actions: (1) putting the value of setting the Tank at 2. (2) Changing the “efficCurve” of Pump to another curve. (3) Changing the demandCategory of the Junction.

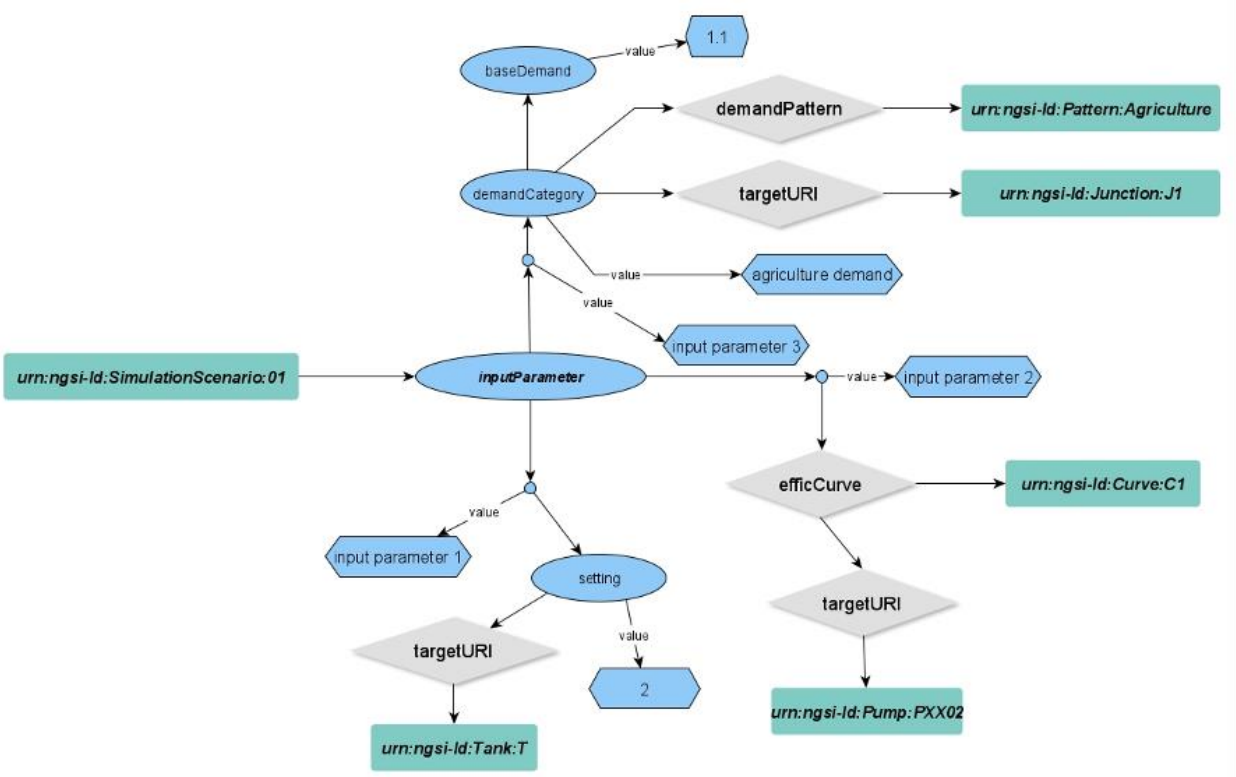


Figure 43: Simulation Scenarios with multiple input parameters

The simulation result entity will contain all captured results on the current network following the modifications brought by the Simulation Scenario. The results are under the NGSI-LD property “outputParameter”. The results are attached to the concerned water component entity via the “targetURI” relationship.

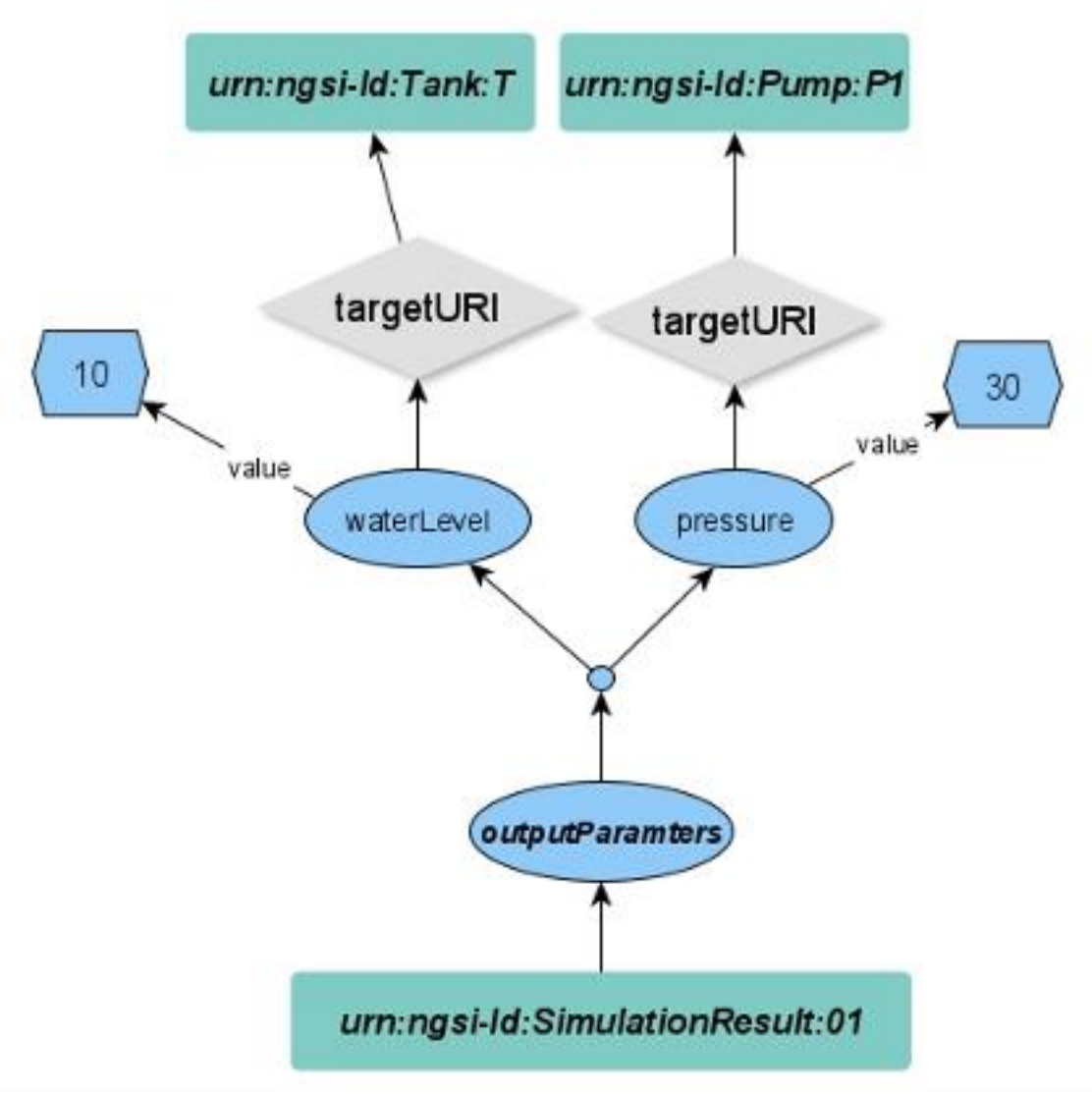


Figure 44: Simulation Result NGSI-LD Entity

Global approach for the data models of the infrastructure. It depicts the different entities created (apart from the Network and simulation previously described). All of them are primarily associated with the water network management vertical and related IoT.

It compiles these entities:

- Curve. This entity contains a harmonised description of a generic curve made for the Water Network Management domain.
- Junction. This entity contains a harmonised description of a generic junction made for the Water Network Management domain.
- Pattern. This entity contains a harmonised description of a generic pattern made for the Water Network Management domain.
- Pipe. This entity contains a harmonised description of a generic pipe made for the Water Network Management domain.
- Pump. 'This entity contains a harmonised description of a generic pump made for the Water Network Management domain.
- Reservoir. This entity contains a harmonised description of a generic Reservoir made for the Water Network Management domain.

- Tank. This entity contains a harmonised description of a generic tank made for the Water Network Management domain.
- Valve. This entity contains a harmonised description of a generic Valve made for the Water Network Management domain.

The links provided include the different elements included into the description of the data models. A complete example of the specification can be found in the Annex. A specification describing the different properties of the data models, a json schema for the validation of the payloads adopting this data model and examples for the different versions of the FIWARE platform, NGSI-LD, the version used in this project and NGSI-V2 a previous version already available for its use. These examples are provided in two formats, normalized, including all the verbosity of the NGSI-LD standard and a key-values format for a simplified and more portable approach.

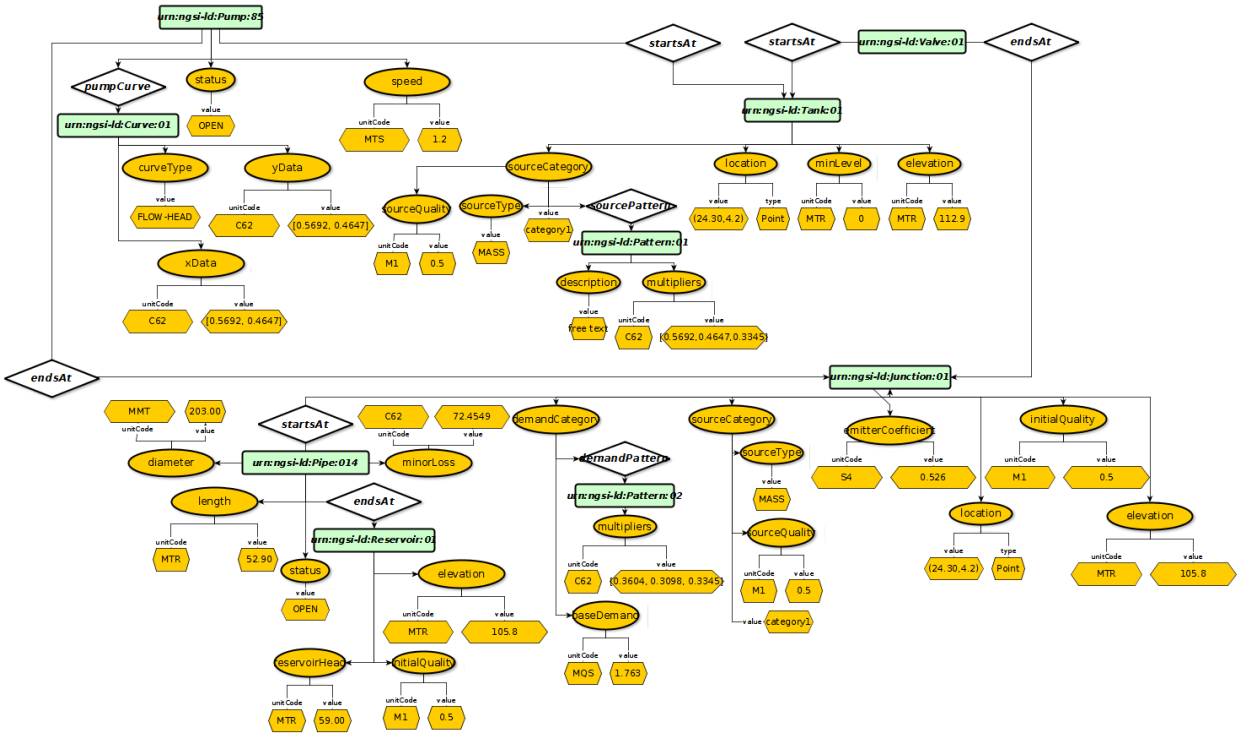


Figure 45: Relation between the different data models created for the WaterNetworkManagement subject

VI. EU Added value

We can understand the EU added value for the project as the value resulting from an EU project which is additional to the value that would have been created by individual states members alone. This means that there are several areas of interest to cover this added value:

- The creaction of Smart Data Models related to Machine Learning and Big Data topics helps in driving the adoption and the development of new applications making use of Machine Learning and Big Data algorithms and mechanisms, thus increasing the efficiency and the time to market of the European Industry and SMEs.
- Similarly, the development of Machine Learning models related to water management can benefit to the European Industry and SMEs, these models being designed as reusable and interoperable runtime units. They could later be offered on an AI marketplace under the umbrella of the FIWARE foundation.
- One step above the previous one, the integration of generic and interoperable Machine Learning as a Service modules, as well as new Generic Enablers in Machine Learning and Big Data topics, that can be used inside any FIWARE compliant architecture provides a great benefit to the whole FIWARE ecosystem in EU countries (and beyond). Indeed, the integration and democratization of such technically advanced paradigms demonstrates a strong position of the EU on this topic.
- New Machine Learning and Big Data patterns and concepts integrated in the scope of the FIWARE4Water project help to improve the ETSI NGSI-LD specification as a whole by using it in complex scenarios involving real-time flow of data and processing of Machine Learning models. As a consequence, it raises the confidence of the whole community and helps driving the adoption of the NGSI-LD specification in EU.
- Finally, the validation of the integration of legacy systems in the water management can drive the adoption of other actors in the Water Industry. Indeed, every actor already has (a lot of) applications in place inside its information system and the adoption of a FIWARE based architecture must take place without requiring changing all the applications already used, which would of course be a blocking point.

Conclusions and perspectives

In this report, it is presented the FIWARE Generic Enablers, the new architectural patterns and tools, and the supporting smart data models, that are currently being worked on to fulfil the Big Data and AI requirements that have been expressed by the Demo Cases. It is also presented how they fit into the architecture of each Demo Case.

For each Demo Case, an architecture able to handle Big Data and AI requirements has been defined, whether by integrating existing FIWARE Generic Enablers or by introducing new patterns and components.

However, these architectures have now to be fully integrated and deployed, and, more important, to be exercised and validated with real and live data. That will typically imply to improve the integrations on some important aspects:

- Global performance of the system
- Improvements in the deployed ML algorithms, and possibly design of new ones
- Adjustments in the already developed smart data models and design of new ones
- Improvements and industrialisation in the tools and methodology used for the operation of the platform.

Another important aspect to be conducted in the coming months is the sustainability of the on-going developments which will benefit to the FIWARE4Water project but are also expected to be standardized and industrialized. That means that new components integrated in the scope of the project will be prepared to join officially the FIWARE ecosystem of Generic Enablers. That also means new architectural patterns will be documented and disseminated inside the FIWARE community to gain feedback and achieve a larger adoption.

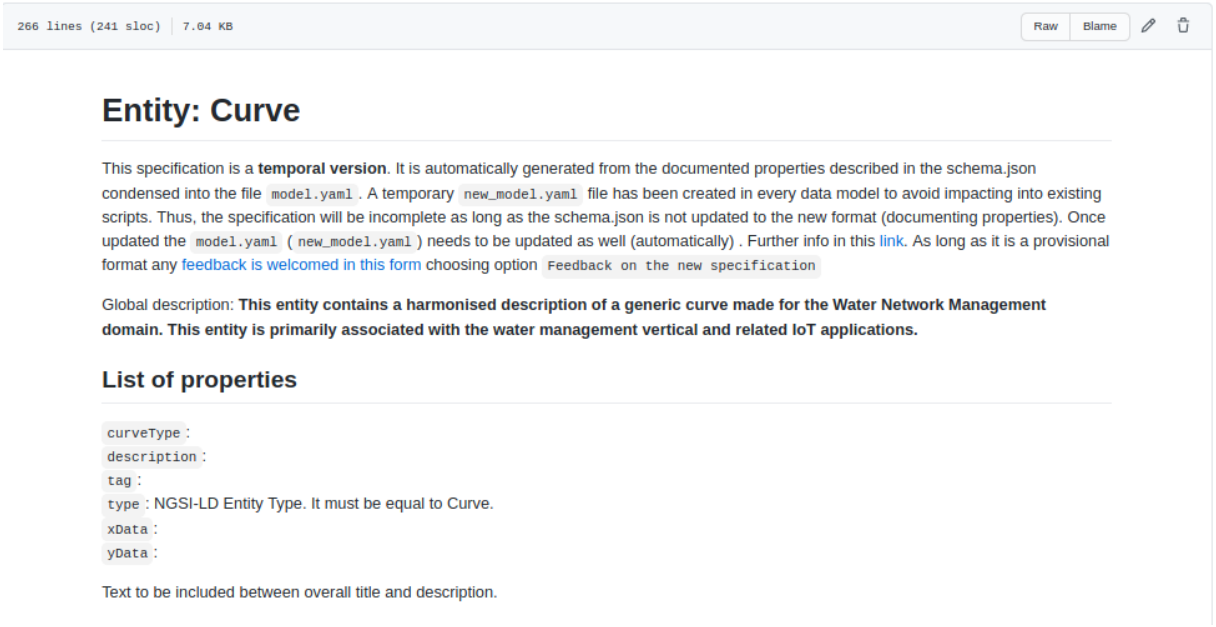
Finally, the innovative uses of the NGSI-LD API will be shortly introduced to the Industry Specialist Group in charge of the NGSI-LD API specification inside ETSI for a thorough discussion about further standardisation of the newly introduced principles.

References

[1] Grieves, Michael, et John Vickers. « Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems ». In Transdisciplinary Perspectives on Complex Systems, edited by Franz-Josef Kahlen, Shannon Flumerfelt, et Anabela Alves, 85-113. Cham: Springer International Publishing, 2017. https://doi.org/10.1007/978-3-319-38756-7_4.

Annex

Example of a specification



266 lines (241 sloc) | 7.04 KB

Raw | Blame

Entity: Curve

This specification is a **temporal version**. It is automatically generated from the documented properties described in the schema.json condensed into the file `model.yaml`. A temporary `new_model.yaml` file has been created in every data model to avoid impacting into existing scripts. Thus, the specification will be incomplete as long as the schema.json is not updated to the new format (documenting properties). Once updated the `model.yaml` (`new_model.yaml`) needs to be updated as well (automatically) . Further info in this link. As long as it is a provisional format any feedback is welcomed in this form choosing option `Feedback on the new specification`

Global description: **This entity contains a harmonised description of a generic curve made for the Water Network Management domain. This entity is primarily associated with the water management vertical and related IoT applications.**

List of properties

`curveType` :
`description` :
`tag` :
`type` : NGSI-LD Entity Type. It must be equal to Curve.
`xData` :
`yData` :

Text to be included between overall title and description.

Data Model description of properties

Sorted alphabetically


```
Curve:
  description: 'This entity contains a harmonised description of a generic curve made for the Water Network Management dom
  properties:
    curveType:
      properties:
        createdAt:
          format: date-time
          type: string
        modifiedAt:
          format: date-time
          type: string
        observedAt:
          format: date-time
          type: string
        type:
          enum:
            - Property
          type: string
        unitCode:
          type: string
        value:
          enum:
            - FLOW-HEAD
            - FLOW-EFFICIENCY
            - FLOW-HEADLOSS
            - LEVEL-VOLUME
          type:
            - number
            - string
            - array
      required:
        - type
        - value
      type: object
    description:
      properties: &curve_-_properties_-_tag_-_properties
        createdAt:
          format: date-time
          type: string
        modifiedAt:
          format: date-time
          type: string
```

```
        observedAt:
          format: date-time
          type: string
        type:
          enum:
            - Property
          type: string
        unitCode:
          type: string
        value:
          type:
            - number
            - string
            - array
      required: &curve_-_properties_-_tag_-_required
        - type
        - value
      type: object
    tag:
      properties: *curve_-_properties_-_tag_-_properties
      required: *curve_-_properties_-_tag_-_required
      type: object
    type:
      description: 'NGSI-LD Entity Type. It must be equal to Curve.'
      enum:
        - Curve
      type: Property
    xData:
      properties: *curve_-_properties_-_tag_-_properties
      required: *curve_-_properties_-_tag_-_required
      type: object
    yData:
      properties: *curve_-_properties_-_tag_-_properties
      required: *curve_-_properties_-_tag_-_required
      type: object
  required:
    - id
    - type
    - curveType
    - xData
    - yData
  type: object
```

Text to be included after list of properties

Here is an example of a Curve in JSON format as key-values. This is compatible with NGSI V2 when using `options=keyValues` and returns the context data of an individual entity.

```
{
  "id": "fAM-8ca3-4533-a2eb-12015",
  "type": "Curve",
  "curveType": {
    "type": "Property",
    "value": "FLOW-HEAD"
  },
  "xData": {
    "type": "Property",
    "value": [
      0.5692,
      0.4647
    ]
  },
  "yData": {
    "type": "Property",
    "value": [
      0.5692,
      0.4647
    ]
  },
  "description": {
    "type": "Property",
    "value": "Free Text"
  },
  "tag": {
    "type": "Property",
    "value": "DMA1"
  }
}
```

Here is an example of a Curve in JSON format as normalized. This is compatible with NGSI V2 when using `options=keyValues` and returns the context data of an individual entity.

Here is an example of a Curve in JSON format as normalized. This is compatible with NGSI V2 when using `options=keyValues` and returns the context data of an individual entity.


```
{
    "id": "fAM-8ca3-4533-a2eb-12015",
    "type": "Curve",
    "dateCreated": {
        "type": "DateTime",
        "value": "2020-02-16T17:43:00Z"
    },
    "dateModified": {
        "type": "DateTime",
        "value": "2020-02-16T17:43:00Z"
    },
    "curveType": {
        "value": "FLOW-HEAD"
    },
    "xData": {
        "value": [
            0.5692,
            0.4647
        ]
    },
    "yData": {
        "value": [
            0.5692,
            0.4647
        ]
    },
    "description": {
        "value": "Free Text"
    },
    "tag": {
        "value": "DMA1"
    }
}
```


Here is an example of a Curve in JSON-LD format as key-values. This is compatible with NGSI-LD when not using options and returns the context data of an individual entity.


```
{
  "@context": [
    "https://schema.lab.fiware.org/ld/context"
  ],
  "id": "fAM-8ca3-4533-a2eb-12015",
  "type": "Curve",
  "curveType": {
    "type": "Property",
    "value": "FLOW-HEAD"
  },
  "xData": {
    "type": "Property",
    "value": [
      0.5692,
      0.4647
    ]
  },
  "yData": {
    "type": "Property",
    "value": [
      0.5692,
      0.4647
    ]
  },
  "description": {
    "type": "Property",
    "value": "Free Text"
  },
  "tag": {
    "type": "Property",
    "value": "DMA1"
  }
}
```

Here is an example of a Curve in JSON-LD format as normalized. This is compatible with NGSI-LD when not using options and returns the context data of an individual entity.


```
{
    "id": "urn:ngsi-ld:Curve:fAM-8ca3-4533-a2eb-12015",
    "type": "Curve",
    "createdAt": "2020-02-16T17:43:00Z",
    "modifiedAt": "2020-02-16T17:43:00Z",
    "curveType":{
        "type": "Property",
        "value": "FLOW-HEAD"
    },
    "xData": {
        "type": "Property",
        "value": [0.5692, 0.4647],
        "unitCode":"C62"
    },
    "yData": {
        "type": "Property",
        "value": [0.5692, 0.4647],
        "unitCode":"C62"
    },
    "description": {
        "type": "Property",
        "value": "Free Text"
    },
    "tag": {
        "type": "Property",
        "value": "DMA1"
    },
    "@context": [
        "https://schema.lab.fiware.org/ld/context"
    ]
}
```


Text after all