Data Platform for the Connection of Cognitive Ports

Title:	Document Version:
D3.1 Data Access Interfaces M18	1.0

Project Number:	Project Acronym:	Project Title:
H2020-871493	DataPorts	A Data Platform for the Cognitive Ports of the Future

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security*:
M18 (June 2021)	M18 (June 2021)	O- PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other; ORDP: Open Research Data Pilot; E: Ethics.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
José Antonio Clemente	PRO	WP3

Authors (organisation):	
José Antonio Clemente (PRO) Héctor Iturria (PRO)	

Abstract:

This deliverable describes the implementation of the Data Access Component, which is the entry point of data to the DataPorts platform. This document provides a description of the component taking into account:

- Position in the architecture
- Functionality
- Relation with other components

This description also includes the current implementation status, information about the different elements needed for their deployment as well as an illustrative example.

Keywords:

Data Access, NGSI Agents, Fiware, Semantic Interoperability, Data Sources

Revision History

Revision	Date	Description	Author (Organisation)
V0.1	31.05.2021	First version of the document.	José Antonio Clemente (PRO)
V0.2	04/06/2021	Version for internal review.	José Antonio Clemente (PRO)
V0.3	10/06/2021	Version for second internal review.	José Antonio Clemente (PRO)
V0.4	14/06/2021	Version ready for WP and PC review	José Antonio Clemente (PRO)
V0.5	25/06/2021	Version ready for TC review	José Antonio Clemente (PRO)
V1.0	30/06/2021	Final Version	Santiago Cáceres (ITI)

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement № 871493.

More information available at https://DataPorts-project.eu

Copyright Statement

INDEX

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 DATAPORTS PROJECT OVERVIEW

DataPorts is a project funded by the European Commission as part of the H2020 Big Data Value PPP programme, and coordinated by the ITI - Technological Institute of Informatics. DataPorts rely on the participation of 13 partners from five different nationalities. The project involves the design and implementation of a data platform, its deployment in two relevant European seaports connecting to their existing digital infrastructures and addressing specific local constraints. Furthermore, a global use case involving these two ports and other actors and targeting inter-port objectives, and all the actions to foster the adoption of the platform at European level.

Hundreds of different European seaports collaborate with each other, exchanging different digital data from several data sources. However, to achieve efficient collaboration and benefit from AI-based technology, a new integrating environment is needed. To this end, DataPorts project is designing and implementing an Industrial Data Platform.

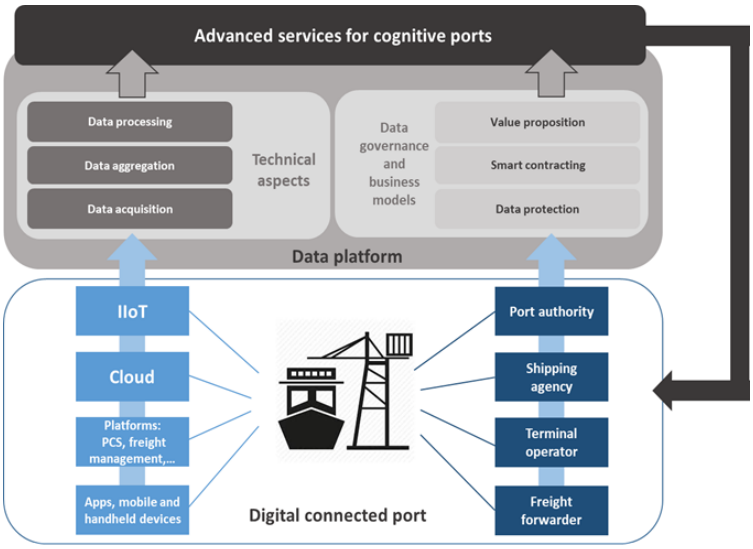

Figure 1 - DataPorts project overview

The DataPorts Platform aim is to connect to the different digital infrastructures currently existing in digital seaports, enabling the interconnection of a wide variety of systems into a tightly integrated ecosystem. In addition, to set the policies for a trusted and reliable data sharing and trading based on data owners' rules and offering a clear value proposition. Finally, to leverage on the data collected to provide advanced Data Analytic services based on which the different actors in the port value chain could develop novel AI and cognitive applications.

DataPorts will allow establish a future Data Space unique for all maritime ports of Europe and contribute to the EC global objective of creating a Common European Data Space.

1.2 DELIVERABLE PURPOSE AND SCOPE

Specifically, the DOA states the following regarding this Deliverable:

This deliverable will include the set of interfaces to provide a homogeneous set of access tools, and the adaptation needed to deal with logistics and freight transport sector and to be used in the use cases and pilots.

The purpose of this deliverable is to describe the mechanisms for accessing the different data sources (Agents) and send data to the upper layers of DataPorts. As well as the mechanisms for managing these tools (Data Access Manager).

1.3 DELIVERABLE CONTEXT

D3.1 relationship to other documents is as follows:

Primary Preceding documents:

- None.

Primary Dependant documents:

- D3.2 Data processing services (M18). The Data Access Component has a very closed relationship with the Semantic Interoperability Component. They have been designed to work together because of the flow of the data and their interactions.
- D5.3 Use Case oriented pilots initial version (M24). The agents to develop for the pilot implementation and the integration of the current data sources depend on the Data Access Component architecture and the provided SDK.

1.4 DOCUMENT STRUCTURE

This deliverable is broken down in the following sections:

- **Section 1 Introduction**: Includes an overview of the project, a short description of the purpose and scope of this document and the dependencies with other deliverables.
- **Section 2 Technical objectives:** Details the technical objectives that each of the components described in this deliverable fulfils according to the project proposal. It also links the objectives of the component with the technical objectives of the WP to which it belongs (WP3 for Data Access Component).
- **Section 3 Position in the DataPorts Architecture:** Briefly describes the position of the component in the architecture and the relationships with the other components.
- **Section 4 Data Access Component**: Describes in detail the Data Access Component. The description includes an overview of the component, its technological description and the status and roadmap the component, as well as an example of usage
- **Section 5 Release summary:** It provides information related to the code release of the sub-components detailed in the deliverable.
- **Section 6 Conclusions:** Main conclusions that can be drawn from the component described in this document.
- **Section 7 References and Acronyms.** It contains the references used in the document as well as the list of acronyms used.

Annexes:

- **Annex A:** *Development of templates for the SDK*. It details how this functionality has been developed and how it is possible to create new templates.
- **Annex B:** *Development with pyngsi*. It details the main features of this framework. The information is accompanied by a couple of examples.
- **Annex C:** *Installation guide*. It details the prerequisites for the deployment of the component. It provides a step-by-step guide to deploying the two services that make up the component.

2 TECHNICAL OBJECTIVES

This deliverable is named Data Access Interfaces and describes an implementation of these interfaces for accessing to data: **Data Access Component (DAC)**. The large number of available data sources, not only in a port ecosystem but in other systems and services that may be interested in management and the activities carried out at them, makes indispensable the development of a component capable of integrating these data, connecting through different ways to different technologies, accessing to the sources and acquiring the data. This integration of data sources of different origins is one of the technical challenges of the DataPorts project and is one of the enablers for the evolution of seaports from digital and connected to smart and cognitive. They are summarized in the following technical objectives of the project:

- O2. *"To design and validate next generation set of advanced interoperable data related and AI based services"*: The component described in this document provides the necessary functionality to acquire data from different data sources or platforms.
- O4. *"To define, design and incorporate a novel, scalable, resilient, semantic approach for data sharing"*: The component presented in this deliverable offers a set of agents that transform the data from the sources in its original format into the DataPorts common data models.

This deliverable presents the features of the Data Access Component. These features ease the access to any data source and integrate it into the DataPorts platform. This facilitates the integration of many data sources and systems into a single platform.

In addition, the Data Access Component fulfils the following technical objectives of WP3:

- O3.1. *"To identify different data sources to be integrated in the DataPorts data platform, including the mechanisms to store and facilitate data management"*. The component provides an SDK that helps the developers to integrate the data sources and the sharing of the information with the other components of the platform.
- O3.2. *"To define ontologies, mechanisms and enablers to provide semantic interoperability with data platforms, IoT devices, robots and other data sources, and develop the semantic-based tools needed to facilitate generation of interfaces for sensing and actuation required in the DataPorts data platform"*. The agents presented in the deliverable are the mechanisms and enablers provided to facilitate the transformation of the data and the access to their metadata. The component also provides a framework to manage the agents.

3 POSITION IN THE DATAPORTS ARCHITECTURE

The position of the Data Access Component in the DataPorts architecture [1] is shown in Figure 2.

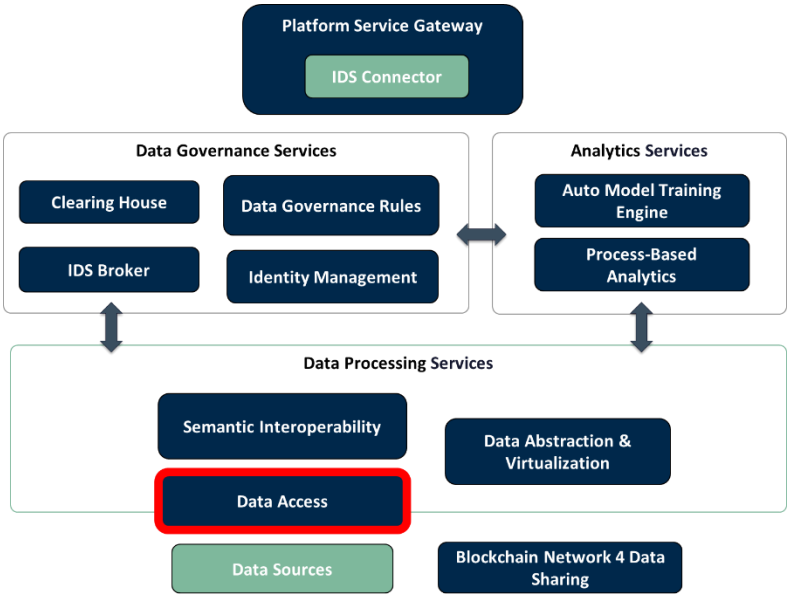

Figure 2 – Data Access position in the architecture building blocks

The Data Access Component is the lowest layer of DataPorts platform. It accesses and connects to the data and feeds the Semantic Interoperability component. It is the only component of the DataPorts platform capable to interact with the data sources and provide access to their raw data

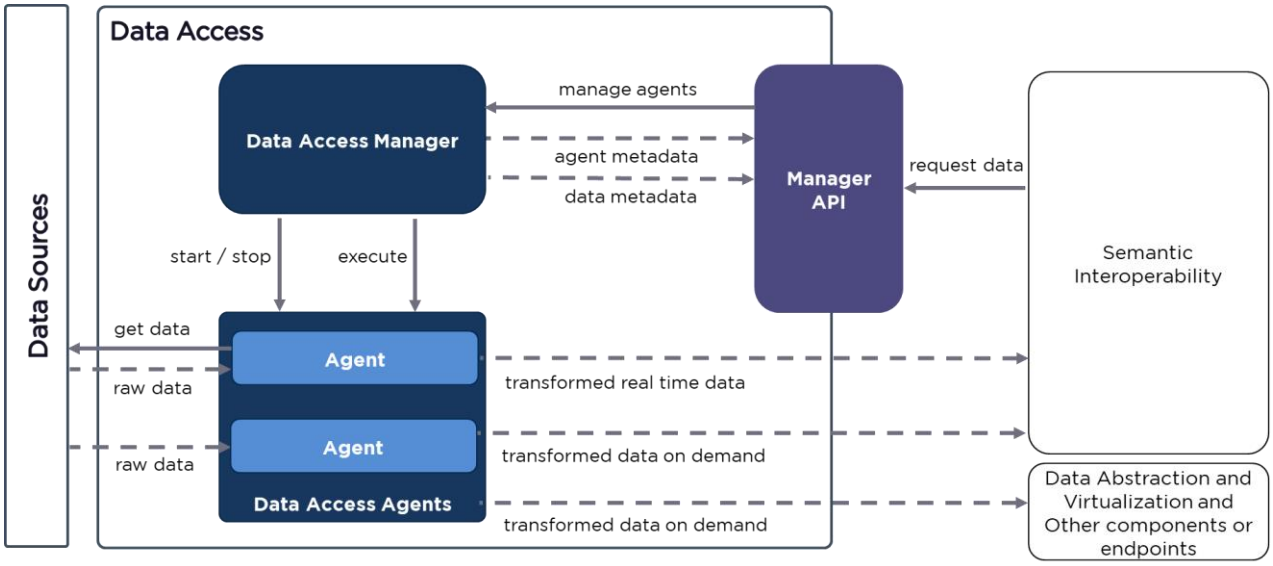

Figure 3 - Interactions of Data Access Component

The Data Access Component is the component of DataPorts closest to the data sources. It is the data input component to the platform, and it is responsible for acquiring the data and sending it to the upper layer (Semantic Interoperability Component). This task is the responsibility of the agents, who are in charge of accessing the different data sources, converting this data into data models understood by DataPorts and sending these data to higher layers (the functionality of the agents will be detailed in later sections).

DAC has a Representational State Transfer Application Programming Interface (REST API) that manages the different CRUD operations over the agents. This component also has a User Interface (UI) responsible for

showing information related to the different images (Docker images of the agents) and agents working in the platform. This UI also has a Software Development Kit (SDK) to facilitate the creation of agents for developers.

In addition, this component is also connected to the Data Virtualization component (Figure 3), which receives the transformed data on demand through the callback Uniform Resource Locator (URL) declared at the time to run the agents (on-demand type). This will be explained in more detail, as well as the different types of agents that the platform will be able to manage, in Section 4.2.1.

4 DATA ACCESS COMPONENT

This document addresses the implementation of the Data Access Component. Since T3.1 (to which this report is directly connected) ends in M18, this deliverable covers the complete implementation of the component.

This section of the document focuses on:

- Overview of the implementation of the component.
- Technological stack used for the implementation of the component.

4.1 OVERVIEW

The component is basically a web-based tool to manage and run agents that integrate data into the DataPorts platform. The component is divided into two sub-components:

- **Agents**. Set of sub-components developed to access the different data sources and send the data to the upper layers of the platform (*Semantic Interoperability* or *Data Abstract and Virtualization Component* depending on the type of agent).
- **Data Access Manager**. Manages the agents available in the platform. To do so, it has at its disposal an API that interacts directly with the agents deployed into the DataPorts platform and a UI which displays to the end-user the results of the actions executed over the agents via the API.

DAC manages the following:

- Distributes data to upper layers:
  - Semantic Interoperability for publish-subscribe agents.
  - Data Abstraction and Virtualization for on-demand agents.
- Offers an SDK to facilitate the creation of agents.
- Validates that the data sent to Semantic Interoperability Component (implemented by ORION[1], more information on deliverable D3.2 Data processing services) is in the correct format. ORION is the broker that has the Semantic Interoperability Component. Its function will be to manage the context information (entities, subscriptions). It works internally with a MongoDB database and has a REST API that allows it to perform the following operations: register new context providers (sensors), update the information of these sensors, and receive notifications of that information.
- Provides an UI for managing all the actions related with the agents available into the platform.
- Provides a REST API to interact with agents and their access to the different data sources.

As mentioned above, DAC is the data entry point to the platform. Therefore, one of its main objectives is to integrate the pilots' involved data sources into DataPorts. This fact implies that, although the development of T3.1 ends in M18, the development of the component depends on the availability of the data sources. If a new data source will be included for which no agent exists, then there will be a development of a new agent to be able to access the dataset. This development will take place along the WP5 activities.

Also, the agents need to transform incoming data into the platform data model, which is being defined in the scope of task T3.2, so that part of the agents' development will be done under T3.2, following its planning and delivery dates.

On the other hand, the relationship between the *Data Access Component* and the *Semantic Interoperability Component* deserves special mention. The following characteristics make it possible to deploy both components together:

[1] https://fiware-orion.readthedocs.io/en/master/

- Common Database (MongoDB) will be used to store the templates that are distributed by default when deploying DataPorts. The same applies to the data models that are supported by the platform by default.
- Common graphical interface. An option has been included in the DAM UI to manage the data sources and subscriptions available in ORION.
- Use of the same internal data model. The entity data model developed from the Semantic Interoperability Component to register agents, data sources, data models available in ORION are shared with the DAC. So, every time an agent is started, it generates a record of the defined entity.

4.2 TECHNOLOGICAL DESCRIPTION

The different blocks that comprise the implementation of the Data Access Component are shown in Figure 4. The mapping between the sub-components defined in the DataPorts architecture (Figure 3) and the technologies used for the development of each one is the following:

- **Data Access Agents**. Comprises the agents developed to access the different data sources. The agents have been developed using pyngsi[2] (NGSI Python framework intended to build a Fiware[3] NGSI Agent). This framework has been developed within the context of H2020 Project: PIXEL[4].
- **Data Access Manager**. The sub-components defined for this block are:
  - **Data Access Manager UI**. Developed using Element[5] or vue-element-admin[6]. It's a component library for developers based on Vue[7] 2.0.
  - **Data Access Manager API**. REST API developed in Node.js[8] following the MVC pattern[9]. Internally it makes use of the Docker[10] API to manage agents as containers. The Specification of the API is done in Swagger[11]. The API makes use of MongoDB [12]to store the templates used to build the agents with the SDK functionality.

[2] https://pypi.org/project/pyngsi/

[3] https://www.fiware.org/developers/

[4] https://pixel-ports.eu/

[5] https://element.eleme.io/#/en-US

[6] https://panjiachen.github.io/vue-element-admin-site/

[7] https://vuejs.org/

[8] https://nodejs.org/en/

[9] https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1

[10] https://docs.docker.com/get-started/overview/

[11] https://swagger.io/

[12] https://www.mongodb.com/

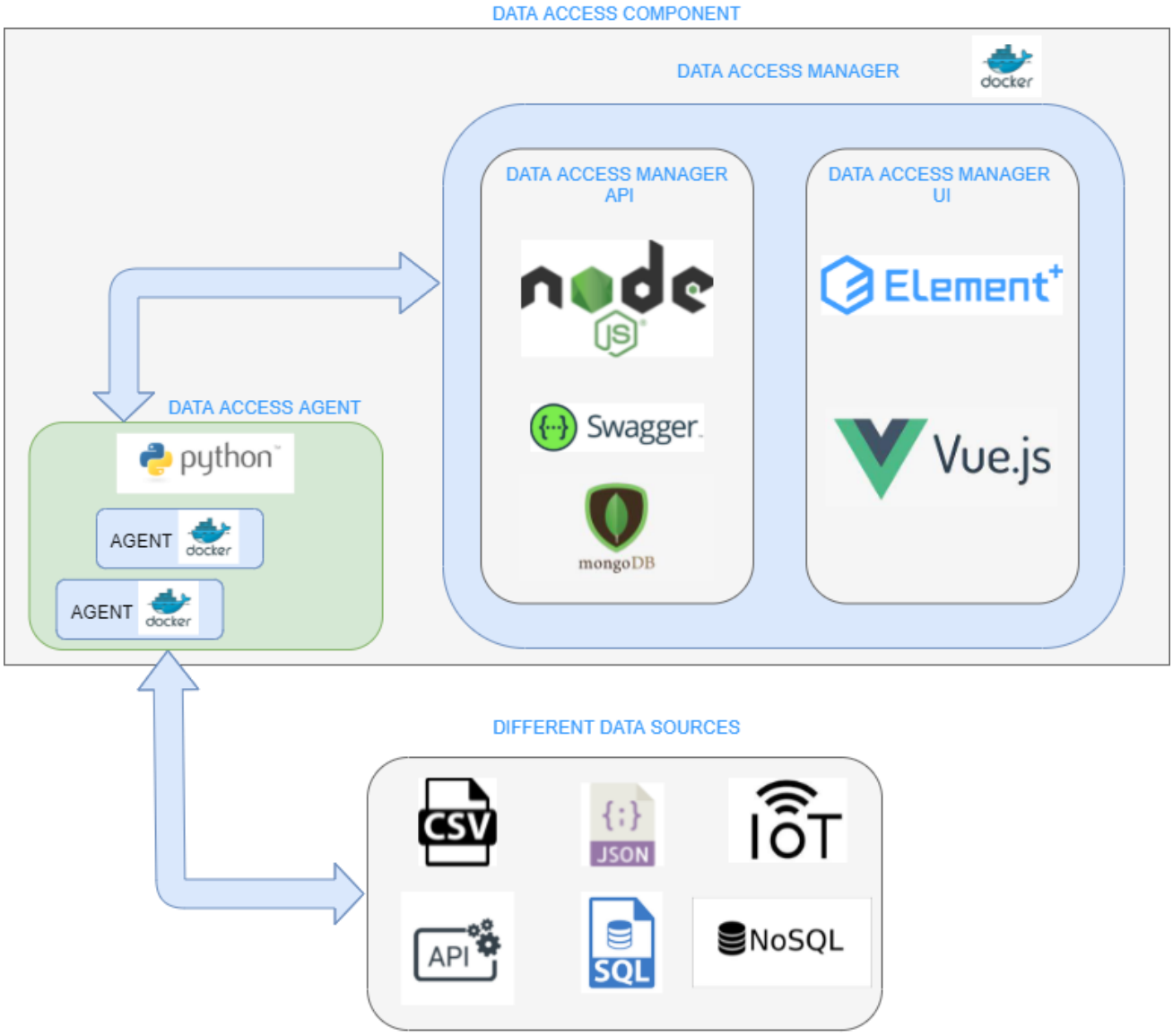

Figure 4 - Data Access Component implementation

The following subsections provide more specific details of the technology stack.

4.2.1 Data Access Agent

Data Access Agent is the term used to name all the agents available on the platform. It is not a component per se. The key element within this term is the agents.

An agent represents the smallest entity with the necessary logic to be able to access different data sources. Each agent shall have the ability to access the data sources for which it has been developed. Figures 4 and 5 show how the agents work. Basically, their functionality is as follows:

- Access to the data source. Access to the specific data source and recover the data.
- Transform data to a valid data model. Convert the retrieved data into a data model that can understand the platform.
- Send data to the upper layer. Depending on the agent type the data is sent to one component or another. For publish-subscribe type agents the data is sent to ORION. In the case of on-demand type agents, the data is sent to a callback url that is indicated when the agent is started.

The agent creation process (section 4.3.1.2) has these parameters or steps to be defined:

- **Type.** Depending on the type, the agents can be:
  - Publish-subscribe. This agent is used to subscribe to a data source to obtain information periodically. As Figure 5 shows, these agents manage the data from the data sources to ORION Context Broker.

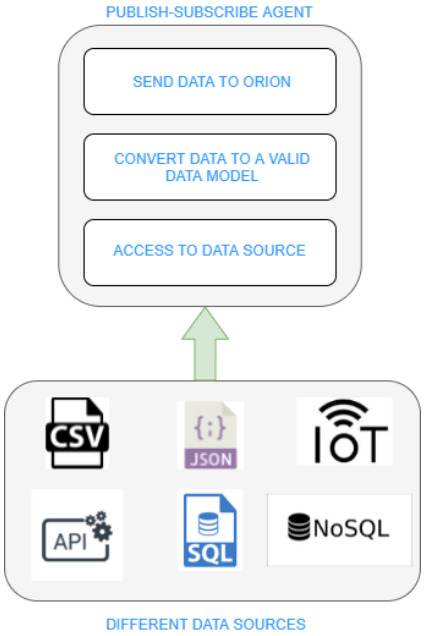

Figure 5 - Publish-subscribe Agent flow

  - On-demand. This type of agent is used when access to a data source is required on a one-off basis and for large volumes of data. For example, access to historical data. In this case, the agent is not a publish-subscribe mechanism. Therefore, the data are not returned to ORION. It is returned to a callback URL that must be indicated at the time of creating the agent. Figure 6 - On-demand Agent flow, depicts this type of agents.

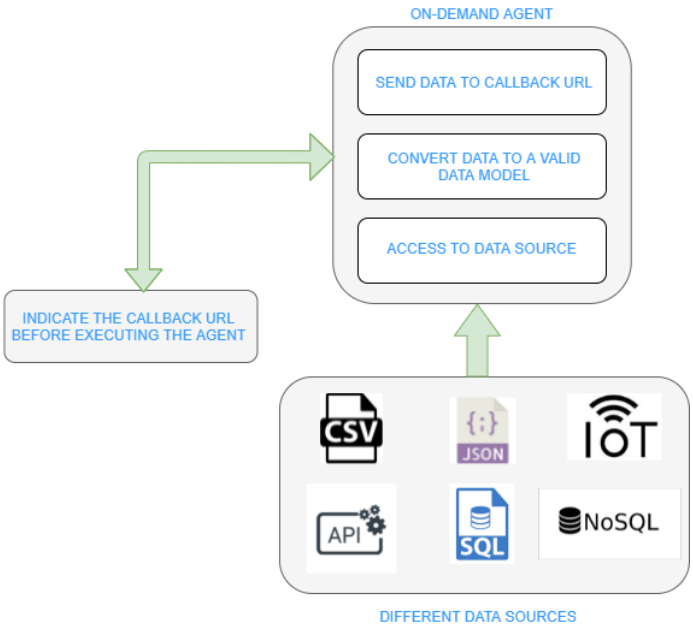

Figure 6 - On-demand Agent flow

- **Data Source.** Here the subdivision is made according to the origin of the data source. Therefore, there will be agents accessing:
    - FTP
    - File (CSV file, TXT file)
    - API
    - SQL Database
    - Non-SQL Database (e.g., Elasticsearch13, MongoDB, …)

Apart from the previous parameters, another aspect to consider is the Data Model in which the agents will return the data to ORION or to the callback URL. With the use of these Data Models, the data is intended to reach the next component of the platform in a standardised format. The Data Models to be used for this purpose are the Fiware Data Models[14]. In some cases, they can be used directly and in other cases these Data Models can be extended. The agents to be developed for the agents will also be integrated with the DataPorts Data Model currently under development.

4.2.2 Data Access Manager

Data Access Manager (DAM) is the component responsible for interacting with the agents as well as displaying their status (running, stopped, etc.) to the end user. As shown in Figure 4, this component is divided in two sub-components:

- **DAM UI**. UI where the user can manage the creation of agents and their subsequent execution quickly and easily. It also offers the possibility of consulting certain ORION entities. It offers the following menu options:
    - Agents. This section shows a grid divided into two tabs:
        - Images. This tab manages the images of agents available in the platform.
        - Instances. This tab is used to manage the agents when they are in execution (containers).
    - ORION. This section allows the user to list existing subscriptions and data sources in ORION. The content of this section is explained in more detail in deliverable D3.2 Data Processing Services. More details about this sub-component and how it works is provided in Section 4.3.
- **DAM API**. REST API that exposes the functions needed for the DAM UI to manage the agents. This API has been implemented in Node.js and exposes an Open API description. The interfaces covered by the API are the following (an example is depicted in Figure 7):
    - Related with the containers of the agents (ORION AgentContainer entity). It is possible to create, start, stop and delete an agent. It is also possible to list all the agents available on the platform and the obtain the log or description of an agent that is in execution.
    - Related with the images of the agents (ORION AgentImage entity). The API gives the option to list all the image available on the platform, to delete one of them. It is possible to obtain the template of the docker image needed to execute the agent.
    - Related with the On-demand functionality, that is, it will be executed from the On-demand module of the Semantic Interoperability Component. API has the option to get historical data.
    - Related with the ORION Context Broker. The API provides methods to create and delete entities. It also returns the list of suscriptions and entities registered in ORION.
    - Related with the SDK functionality (for these methods a MongoDB is used which is shared with the Semantic Interoperability Component). The SDK offers the possibility to generate the agents

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

[14] https://www.fiware.org/developers/data-models/

quickly and easily through a wizard. To do this, it has been necessary to generate a series of templates in order to generate the forms with the necessary fields to fill in for the generation of the agents. Within these interfaces, the API gives the option of: creating, updating and deleting a template (Annex A for more information), listing the available templates, obtaining a specific template and even downloading the zip file with all the necessary files to generate the docker image of an agent.

- Related with the SDK functionality for managing the Data Models that will be available in the SDK. The API has the option to list all the data models available on the platform for creating an agent and offers the option to create, update and delete a data model.

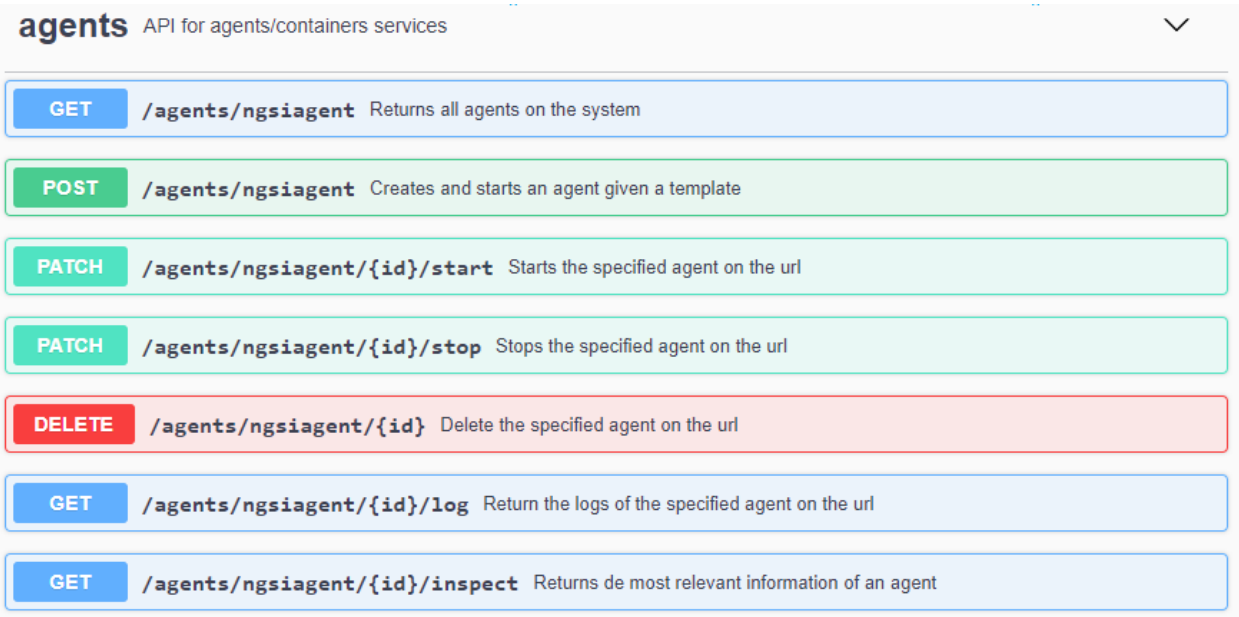

Figure 7 - Example of Swagger description

4.3 EXAMPLE OF USE: DEMONSTRATION

The main purpose of this section is to illustrate how the Data Access Component works. In particular, the DAM UI, which is the core DAC element. All the management of images and agents is done from this UI.

Our example includes a description of all the steps necessary to get an agent up and running in the platform. The section is divided into the following subsections as shown henceforth:

1. Creating an agent
2. How to run an agent
3. Actions available on a running agent

4.3.1 Creating an agent

Figure 8 depicts the UI once deployed.

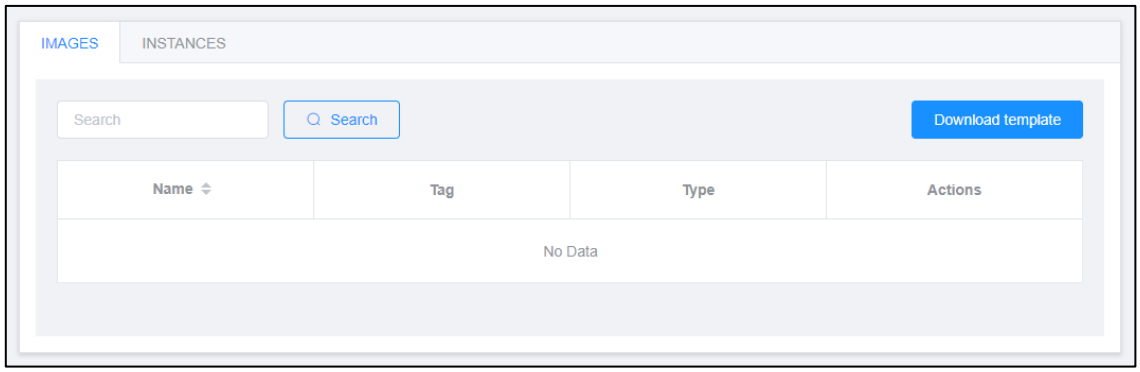

Figure 8 - DAM UI appearance

As can be seen, by default there is no agent image available in the platform. Therefore, the first step is to upload, generate, or import the image of an agent into the platform. There are three different options for this (explained below):

1. Importing the image from an external repository.
2. Using the wizard.
3. Developing the image.

4.3.1.1 Import image from external repository

This option will be developed in the context of WP5. The DAM UI will offer the option to import an image from an external repository. To do so, there is a button in the top right corner of the Images tab (Figure 9).

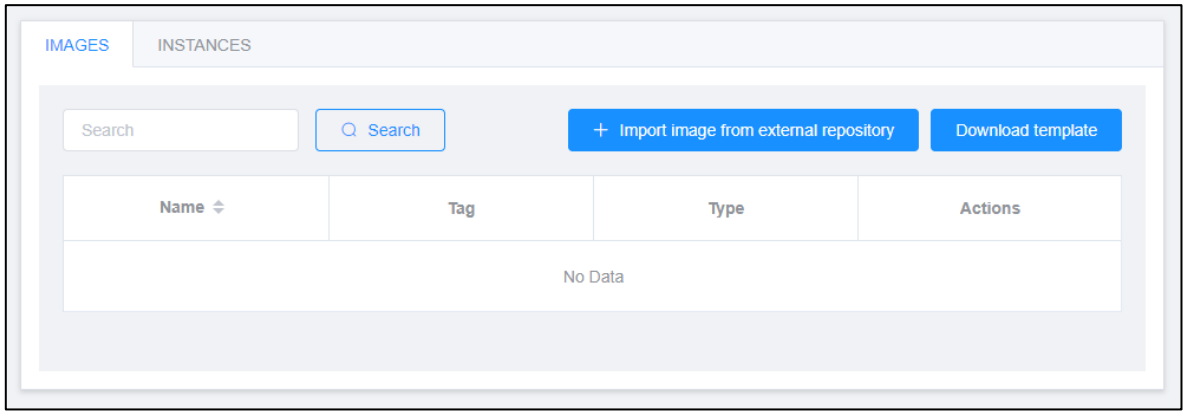

Figure 9 - Draft for Import image from external repository

When this option is executed, the dialog box shown in the following figure appears.

Figure 10 - Import image from external repository mockup

This form gives the option to download an image (this will make a 'pull'[15]) from the internal repository of the project (DataPorts GIT[16]). The UI will connect to the repository and will list all the images on the dropdown. So, the end-user only must select the image to download from the available ones.

In case a user would like to import images from any other repository, the best way to do it will be through a console and by executing the corresponding commands.

4.3.1.2 Using the wizard

DAM UI includes an SDK to generate all the necessary files to create an agent image. This SDK is presented in the form of a Wizard. The wizard has 4 steps:

1. **Select the data source** where the agent will access. Next figure shows data sources as for example: API, FTP or CSV file.

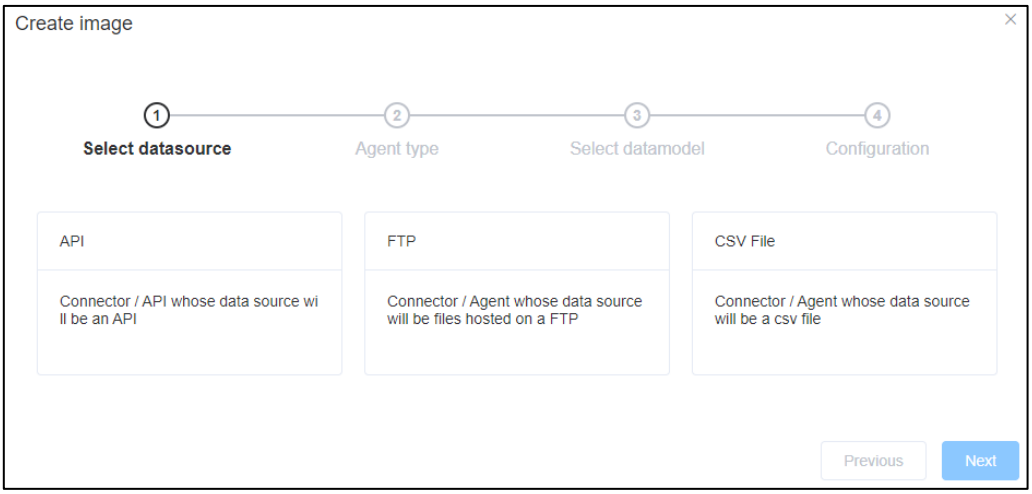

Figure 11 - Wizard: Select data source

2. **Select the type of agent**. Publish-subscribe or on-demand. This was explained in section Data Access Agent.

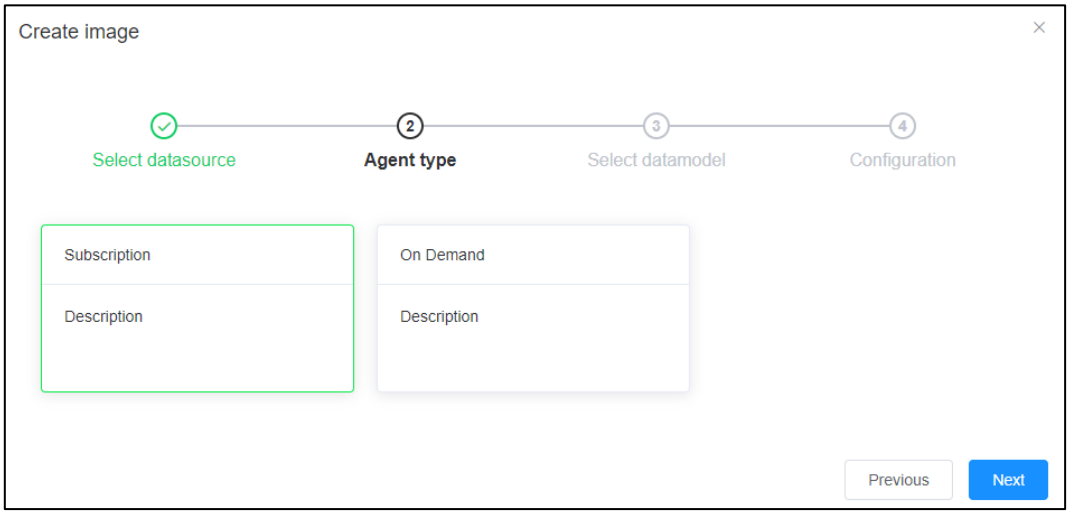

Figure 12 - Wizard: Agent type

[15] https://docs.docker.com/engine/reference/commandline/image_pull/

[16] https://en.wikipedia.org/wiki/Git

3. **Select the Data Model** used for sending the data to ORION, as shown in next figure.

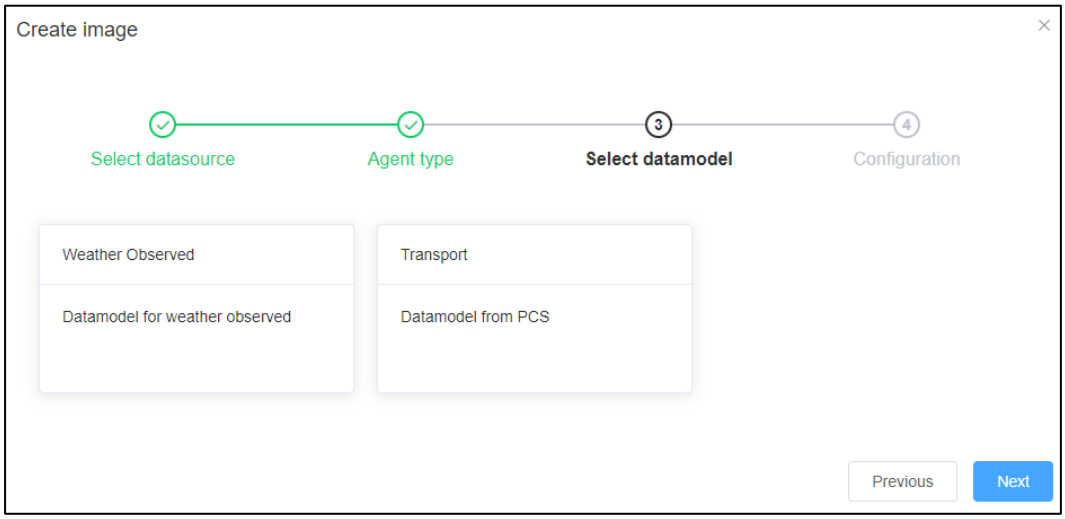

Figure 13 - Wizard: Select Data Model

These data models will be hosted in a repository (public or private). Currently only the project's GIT is supported as private repository. In the case of public repositories, as it is enough to indicate the URL, there is no problem to use any public repository. The API has the necessary methods to manage the Data Models available in the platform. This includes these properties:

- **Name.** Name of the data model.
- **Description.** Indicates the description of the data model.
- **Link.** URL where the data model is hosted. This property must point to the JavaScript Object Notation (JSON) schema of the Data Model.
- **PrivateRepository.** Property to indicate if the repository where the data model is hosted is public or private.
- **ProjectName.** Name of the project in the repository where the data models are stored.

Figure 14, depicts an example of a data model hosted in a public repository and available in the platform (SDK functionality).


```
{
  "_id": "60c1cbd16738452a08c09a8e",
  "name": "WeatherObserved",
  "description": "An observation of weather conditions at a certain place and time",
  "link": "https://fiware.github.io/data-models/specs/Weather/WeatherObserved/schema.json",
  "privateRepository": false,
  "createdAt": "2021-06-10T08:22:41.816Z",
  "updatedAt": "2021-06-10T08:22:41.816Z",
  "__v": 0
}
```


Figure 14 - Example of request for a data model available on the platform

4. **Configuration**. In this step, all the fields necessary to generate the agent must be filled in. These fields are the union of the previously selected properties: data source, agent type and data model. Regarding the selected data model step, two columns appear:

- **Data Model properties.** The list of properties of the data model.
- **Selected properties.** The properties that are selected to include in the agent that is going to be generated.

The code region of the script where the properties of the data model are added will have to be reviewed to complete with the properties of the data source.

The following Figure depicts an example of fields to be filled in for a publish-subscribe agent that reads data from an API and uses the **WeatherObserved** [17] data model. WeatherObserved is one of the consensus data models within Fiware for use in different applications such as: Smart Cities, Smart Water, Smart Energy and other domains. One of the aims of the project is to create a Smart Ports data model.

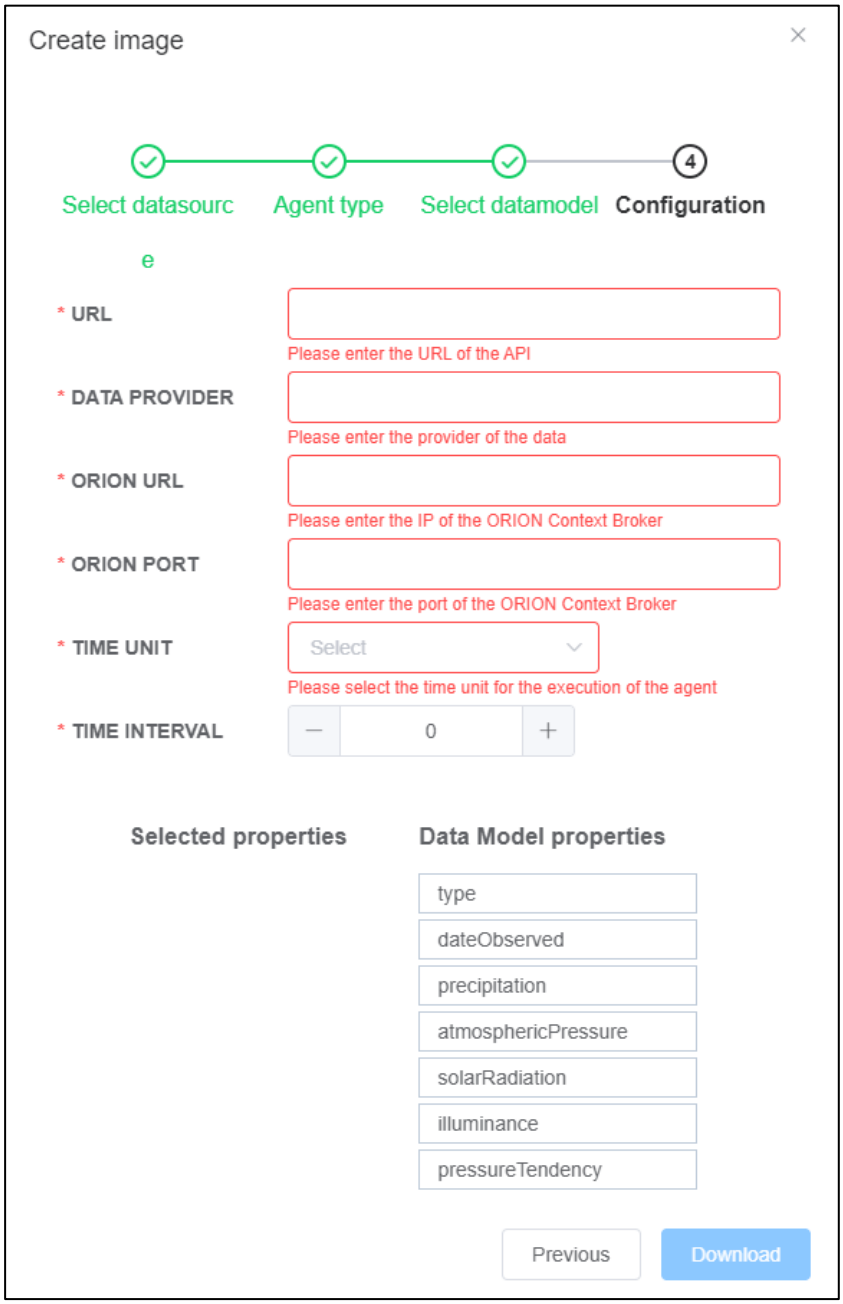

Figure 15 - Wizard: Configuration

[17] https://fiware.github.io/data-models/specs/Weather/WeatherObserved/schema.json

Once all the fields have been filled in, clicking on Download button downloads a zip file containing all the necessary files to build the agent image. The content of the zip file will vary depending on the type of agent and data source. But, at least, it must contain the following files:

- **script.py**. Python script (pyngsi) with all the agent logic.
- **Dockerfile**. File necessary to dockerise the agent.
- **Readme.txt**. File with indications for developers.

Prior to the generation of the image, the developer must review the script to validate the content of the script and the properties used.

The agent image will be created with the following instruction:

```
Get-Content Dockerfile | docker build . -t [imageName]:[version]
#imageName represents the name given to the image
#version represents the version assigned to the image
```

Figure 16 - Instructions to generate an agent image

This instruction must be executed in the same directory as the contents of the zip file.

4.3.1.3 Developing the image

This option involves developing the agent from scratch.

For the development of the agents, the chosen option is pyngsi (ANNEX B: Development with pyngsi gives some recommendations and examples on how to develop with pyngsi). Therefore, as a minimum, it will be necessary to develop a file with extension 'py'. Also, a Dockerfile to package as a docker image the agent.

This option will be appropriate when there is no template that fits the agent we are going to carry out.

4.3.2 How to run an agent

Figure 17 depicts the appearance of the UI after an image is available.

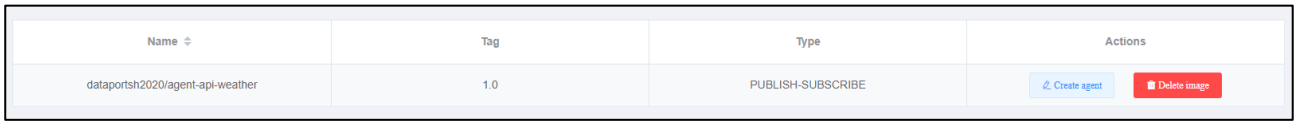

Figure 17 - DAM UI with an image available

The actions that can be executed on the images available on the platform are:

- **Create agent**. This button triggers the action of creating the agent. Before that, the following parameters must be filled in, which are read from the Dockerfile of the image.

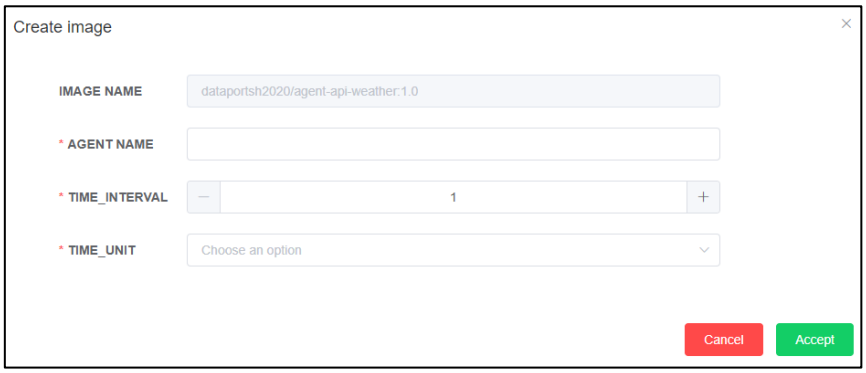

Figure 18 - Form to Create Agents

Once the previous form has been accepted the new agent will appear in the Instances' tab.

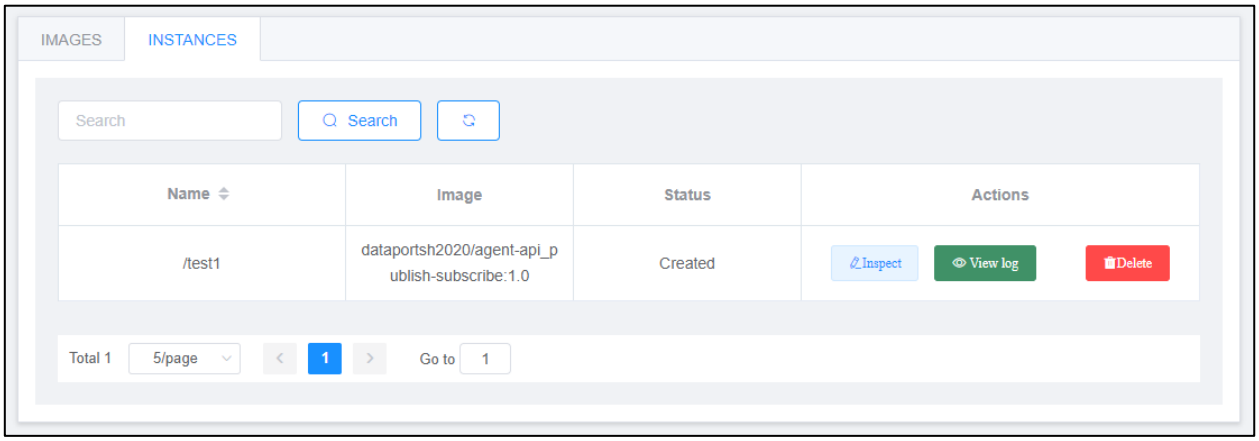

Figure 19 - Agent up & running

It should be noted that it is possible to run more than one agent from the same image.

- **Delete image**. This option removes the image from the platform. Before deleting the image, a dialogue box appears to confirm the action (Figure 20 - Dialogue box to confirm action: Delete Image).

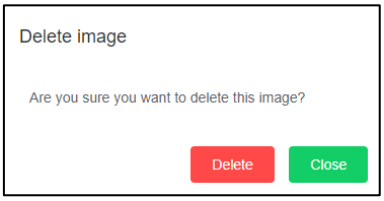

Figure 20 - Dialogue box to confirm action: Delete Image

4.3.3 Actions available in a running agent

Actions available in a running agent (Figure 19 - Agent up & running) are:

- **Inspect**. It allows inspecting the agent's content. Retrieves the agent's configuration parameters and the data model it is using.

Figure 21 - Inspecting an Agent

- **View Log**. Download the agent execution log. This allows checking if the agent is running as planned.
- **Stop**. Stops the execution of the container. When the agent stops, the **Start** action appears to restart the container execution. Figure 22 - Agent exited, shows an agent with Status exited and showing the Start button to restart the execution of the agent.

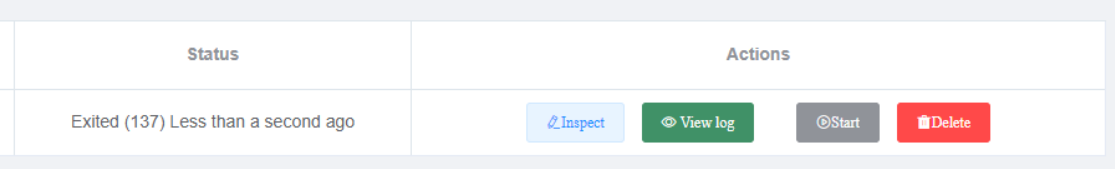

Figure 22 - Agent exited

- **Delete**. Remove the container. Prior to deletion, a confirmation is required from the user, just as when deleting an image.

4.4 DEVELOPMENT STATUS (AT M18)

This section describes the status of the implementation of the component and how it covers the expected features. The component implementation has been finished and will be improved to support the pilot specific requirements along the WP5 activities. D2.1 Industrial Data Platforms and seaport community requirements and challenges contains the list of requirements per WP and task. The specific requirements to be fulfilled by T3.1 are:

ID	3.1
Description	The DataPorts Platform must homogenize the input data so that it can be retrieved in the same format and data model regardless of the data source.
Status	Done. The agents convert the data into data models that can be understood by the platform. Some changes will be made in T3.2

ID	3.2
Description	The Data Access component should be able to make use of the interface of each data source connected to the DataPorts Platform in order to retrieve the data in its original format. Supported data interchange formats should be at least: XML and JSON. The Data Access component should also support the security mechanisms implemented in each data source.
Status	Done. Data are retrieved from data sources in their original format. It is then converted into the data models understood by the platform.

ID	3.3
Description	Each data source that is going to be connected to the DataPorts infrastructure through an API could provide version management for its API, in order to allow a proper identification of the API and the data formats.
Status	When the agents are developed and the data models are available, the necessary modifications will be made.

ID	3.4
Description	Each data source that is going to be connected to the DataPorts infrastructure through an API should provide documentation about it (such as an OpenAPI/Swagger definition).
Status	Necessary changes will be mad in the context of WP5.

ID	3.6
Description	The DataPorts Platform must be able to obtain data from platform/sensors/sources with and without explicitly defined ontology in order to support as many potential data sources as possible.
Status	Done. Agents have been developed that are to access different data sources (with and without ontology).

ID	3.8
Description	The Data Access component must provide the proper connectors to communicate with the available data sources.
Status	Done. This functionality will be completed in the context of WP5.

ID	3.9
Description	The Data Access component should support the most common communication protocols in order to be able to communicate with as many potential data sources as possible.
Status	Done. A SDK for creating agents have been developed.

ID	3.23
Description	The components of the DataPorts Platform could be virtualized, in order to ease its deployment and portability.
Status	Done. All subcomponents of the DAC have been package as docker images with the aim to ease its deployment and portability.

ID	3.24
Description	The DataPorts Platform could offer mechanisms for the automatic deployment, maintenance and scaling of the developed components.
Status	Done. All the components have been packaged as docker images for easing these tasks.

ID	3.31
Description	As a developer, I want the platform to deal with the original data sources heterogeneity, real-time (streaming) or persistent data, relational or non-relational databases (NoSQL), so that I can use the data without taking into account the underlying storage system.
Status	Done. Within WP5 the number of templates available on the platform for the creation of agents will be extended. Currently there are already agents that can access data sources with these characteristics.

ID	3.32
Description	As a data provider, I want to publish data in the DataPorts Platform, so that it is available for the platform users with the access rights.
Status	Done. All the data sent by agents are published in the DataPorts platform.

ID	3.33
Description	As a data consumer, I want to get the list of the available data sources and all the methods provided by the platform to subscribe or request data on demand.
Status	Done. From the ORION menu option of the DAM UI it is possible to manage the available data sources.

ID	3.34
Description	As a data consumer, I want to subscribe to an available subscription provided by the DataPorts Platform.
Status	Done. This can be managed from the ORION menu option of the DAM UI.

ID	3.35
Description	As a data consumer, I want to be able to cancel a current subscription, so that I stop receiving data modifications.

Status	Done. This can be managed from the Semantic Interoperability Component or by stopping / deleting the agent container.

ID	3.36
Description	As a data consumer, I want to request data on demand from the data sources using the methods provided by the DataPorts Platform.
Status	Done. DAM API has methods for such requests.

ID	3.43
Description	Integration of legacy sources. The DataPorts Platform must be able to integrate with relevant port platforms.
Status	This integration will take place in WP5. DAM API is prepared to integrate with external data sources regardless of the origin.

Among the modifications that will be developed in the upcoming months are the following:

- Development of a data validation module prior to sending the data to ORION. This development is carried out within T3.2.
- Support for NGSI-LD. This modification shall be performed within the task T3.2.
- Agents' development will take place in WP5.
- Enhancement of the SDK, adding templates for more agent types. This improvement will be done in the scope of WP5.

5 RELEASE SUMMARY

The Table 1 shows the release information summary for the DAC component and its subcomponents at M18, including component name, source code and dependencies.

Component	Subcomponent (if needed)	Release Information	Details and Links
Data Access Component	Data Access Manager UI	Source Code	https://egitlab.iti.es/dataports/data_access/data-access-manager-ui
		Dependencies	Data Access Manager API
	Data Access Manager API	Source Code	https://egitlab.iti.es/dataports/data_access/data-access-manager-api
		Dependencies	Mongo (service deployed with the docker-compose)
	docker-compose	Source Code	https://egitlab.iti.es/dataports/data_access/data-access-deployment
		Dependencies	Docker and Docker-compose
	mongo-seed-templates	Source Code	https://egitlab.iti.es/dataports/data_access/mongo-seed-templates
		Dependencies	Mongo (service deployed with the docker-compose)
	mongo-seed-datamodels	Source Code	https://egitlab.iti.es/dataports/data_access/mongo-seed-datamodels
		Dependencies	Mongo (service deployed with the docker-compose)

Table 1 – Release Information of the Data Access Component

6 CONCLUSIONS

This document contains the final version of the Data Access Component. This component is responsible for integrating all data sources into the platform and presents a number of advantages / facilities for the user of the platform:

- It allows direct management of the data sources available on the platform.
- It is possible to create instances of the different data sources or to delete those that are no longer of interest.
- It facilitates the creation of agents. It is quick and visual the creation of a base of an agent that accesses a specific data source.
- It allows managing the information in a centralised way. DAC allows to control the data sources available on the platform, how often the data will be retrieved, as well as to see the data and the subscriptions available in ORION.

The document reviews the position of the component within the overall DataPorts architecture. It also explains the technological stack for the development.

The design of the component and its development has been done within T3.1. As seen in Section 4.4 the design of the component fulfils all the requirements defined in deliverable D2.1.

The Annexes of the deliverable (ANNEX A: Development of templates for the , ANNEX B: Development with pyngsi and ANNEX C: Installation guide) complement the information described in the deliverable.

Although this document is the closing result of the task T3.1, the development of the component and sub-components will be extended. This extension will take place in the following tasks:

- WP5: The agents of the different pilots will be implemented.
- T3.2: The transformations of the data models to the ontology of the platform will take place in this task.

7 REFERENCES AND ACRONYMS

7.1 REFERENCES

[1] DataPorts, D2.4 Platform architecture and specifications, December 2021.

7.2 ACRONYMS

Acronym List	
API	Application Programming Interface
CRUD	Create, Read, Update and Delete
CSV	Comma-Separated Values
DAC	Data Access Component
DAM	Data Access Manager
DoA	Description of Action
FTP	File Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notification
MVC	Model-View-Controller
NGSI	Next Generation Sensors Initiative
PC	Project Coordinator
REST	Representational State Transfer
SDK	Software Development Kit
SQL	Structured Query Language
TC	Technical Coordinator
UI	User Interface
URL	Universal Resource Locator
WP	Work Package

Table 2 – Acronyms

8 ANNEX A: DEVELOPMENT OF TEMPLATES FOR THE SDK

Currently the SDK offers support for some type of agents. This functionality will be extended as agents are developed. It will take place within WP5.

In the last step of the wizard (Configuration), the end-user must fill in the form of the different fields generated depending on the data source and type of agent. The combinations of fields for the form can be infinite. It was impossible to develop for every combination. This implied making a deployment every time a change was made to both the DAM UI and DAM API.

Therefore, it was decided to combine templates with a library that would allow the automatic generation of forms. With the use of templates, it was intended to create a model that could support all possible field combinations as well as all necessary properties to generate the forms. These templates would be defined in JSON format. Figure 23 - Template's schema, depicts the schema of the model used for the generation of the SDK forms.

```
import { model, Schema } from 'mongoose';

const mySchema = new Schema(
  {
    source: {
      type: 'String'
    },
    type: {
      type: 'String'
    },
    datamodel: {
      type: 'String'
    },
    constants: {
      type: 'String'
    },
    dockerFile: {
      type: 'String'
    },
    script: {
      type: 'String'
    },
    template: { type: 'Mixed' }
  },
  {
    autoCreate: true,
    timestamps: true
  }
);

mySchema.index({ '$**': 'text' });

export default model('PythonTemplate', mySchema);
```

Figure 23 - Template's schema

The properties are:

- **source**. This property indicates the data source.
- **type**. Indicates the type of the agent. Could be: 'publish-subscribe' or 'on-demand'.
- **datamodel**. It will contain the data model used for the combination of the above properties (source and type).
- **Constants**. Indicates the location of the 'constants.py' script used for the creation of the agent.
- **Dockerfile**. Indicates the location of the 'Dockerfile' used for the creation of the agent.
- **Script**. Indicates the location of the 'script.py' file used for the creation of the agent.
- **Template**. This property is the most important of all. It contains the JSON for the automatic creation of the form (SDK). This property is divided into two sub-properties:
  - *Rules*. Defines the validation rules of the form (which fields are mandatory or not).

Figure 24 - Example of validation rules definition

  - *Elements*. This section defines the type of control associated with each of the form fields that will be generated.

Figure 25 - Definition of a property in the elements section

No tool has been developed for the creation of these templates. They must be written in JSON format. These templates are stored in a MongoDB database, which can be shared with the one of the Semantic Interoperability Component.

Within the DAM API there are the necessary methods for template management. Among the actions are:

1. List templates available
2. Get specific template
3. Create, update and delete a template

Once the template has been developed and stored in MongoDB, it's time to talk about the element responsible for understanding this format and create the forms automatically: **element-form-builder**[18]. This library builds element-ui forms with JSON schema. It is, therefore, able to understand the defined JSON and create forms like the one show in Figure 18.

[18] https://www.npmjs.com/package/element-form-builder/v/1.0.0

9 ANNEX B: DEVELOPMENT WITH PYNGSI

pyngsi is a Python framework developed to build Fiware NGSI Agents. pyngsi was developed in the context of H2020 project PIXEL.

One of its main features is that it makes it easier for developers to write pipelines for their FIWARE data flows. Other features are:

- NGSI v2 support
- Map Python –native data to NGSI entities
- Write NGSI entities to Fiware ORION
- Handle incoming data through a common interface
- Allow visualization/debugging facilities

These options made it the choice for the development of the agents.

The following examples are intended to give a basic notion of how to develop with pyngsi. In case you want more information, check out the links about the framework at chapter 7 REFERENCES AND ACRONYMS.

Figure 26 depicts how to create a basic data model using pyngsi.

```
from pyngsi.ngsi import DataModel

m = DataModel(id="Room1", type="Room")
m.add("temperature", 23.3)
m.add("pressure", 36)
m.add("location", (44.83,-0.57))

m.pprint()
```

Figure 26 - How to create a basic data model

It will be necessary to import the DataModel namespace. In this example we are creating a data model of type ‘Room’ with the identifier ‘Room1’ and we assign it the properties: temperature, pressure and location with their respective values.

Figure 27 illustrates the full structure of a basic agent.

```
from pyngsi.ngsi import DataModel
from pyngsi.source import Row
from pyngsi.sink import SinkOrion, SinkStdout
from pyngsi.agent import NgsiAgent

def build_entity(row:Row)->DataModel:
    pass # Implementation is up to you (a row is approx. an incoming data record)

src = SourceSample() # datasource of your choice

# default Orion Server listens on 127.0.0.1:1026
# help(SinkOrion) for information on availabl arguments
sink = SinkOrion()

#SinkStdout just for visualization
# sink = SinkStdout()

agent = NgsiAgent.create_agent(src,sink,process=build_entity)

# the agents iterate over the source and do the job
agent.run()

# close the agent and update statistics
agent.close()
```

Figure 27 - Creation of a basic NGSI Agent

An example of a typical structure could be:

1. Define the function responsible for iterating the registers and create the data model.
2. Create the source object. It depends on the data source.
3. Create the sink object. Could be: SinkOrion or SinkStdout.
4. Create the agent with its necessary arguments.
5. Run the agent.
6. Close the agent.

10 ANNEX C: INSTALLATION GUIDE

All components are dockerised. So, the first step will be the installation of docker[19] and docker-compose[20].

10.1 INSTALLATION OF DOCKER & DOCKER-COMPOSE

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

Table 3 - Installation of docker & docker-compose

10.2 INSTALLATION OF DATA ACCESS COMPONENT

DAC will be deployed using docker-compose. For the installation it will be necessary to have a directory with the following elements:

- Docker-compose file. File that contains all the services needed to raise an instance of the DAC.

The services included in the docker-compose are:

- **dataports-ui**. Service that deploys the UI.
- **dataports-api**. Service that deploys the REST API.
- **mongo**. Service that deploys an instance of MongoDB.
- **mongo-seed-templates**. Service that insert in the database the templates that are distributed by default with the platform.
- **mongo-seed-datamodels**. Service that store in the database the data models used in the platform to send data from the agents to DataPorts.

[19] https://en.wikipedia.org/wiki/Docker_(software)

[20] https://docs.docker.com/compose/

Therefore, the contents of the directory where the DAC will be deployed should be as shown in the following figure:

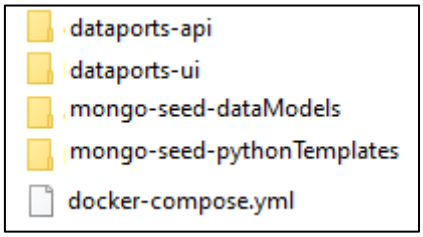

Figure 28 - Directory where deploys DAC

Each of the folders contains:

- **dataports-api.** The source code of the API.
- **Dataports-ui.** Source code of the UI.
- **mongo-seed-dataModels.** Contains a Dockerfile to import the dataModelsObject data collection into MongoDB.
- **mongo-seed-pythonTemplates**. Contains a Dockerfile to import the templates data collection into MongoDB.

The instructions to be executed to deploy the services are as follows:

- **dataports-ui**. *sudo docker-compose up --build -d dataports-ui*

  The above instruction deploys not only the UI. Due to the dependencies among services it installs the following services in this order:

  1. *mongo*
  2. *dataports-api*
  3. *dataports-ui*

- **mongo-seed-templates**. *sudo docker-compose up --build -d mongo-seed-templates*
- **mongo-seed-datamodels**. *sudo docker-compose up --build -d mongo-seed-datamodels*

In this way, we would have all services deployed. The only services that may require an update are the API and the UI. The instructions to update them are:

- **dataports-ui**. *sudo docker-compose up --build -d dataports-ui*
- **dataports-api**. *sudo docker-compose up --build -d dataports-api*

Next table depicts hoy to verify that the services have been correctly deployed.

Table 4 illustrates how to verify the services are correctly deployed.

dataports-ui
It is possible to check with the instruction "*docker-compose ps*" if the service is in "*Up*" status, and with the command "telnet IP 8080" that the TCP port is listening
dataports-api
It is possible to check with the instruction "*docker-compose ps*" if the service is in "*Up*" status, and with the command "telnet IP 3000" that the TCP port is listening
mongo
It is possible to check with the instruction "*docker-compose ps*" if the service is in "*Up*" status
mongo-seed-templates

This service will only be deployed once. It's possible to check with the instruction "*docker logs –f mongo-seed-templates*". The log will indicate that 2 documents have been successfully imported
mongo-seed-datamodels
This service will only be deployed once. It's possible to check with the instruction "*docker logs –f mongo-seed-datamodels*". The log will indicate that 2 documents have been successfully imported

Table 4 - How do you verify the service has been correctly deployed?