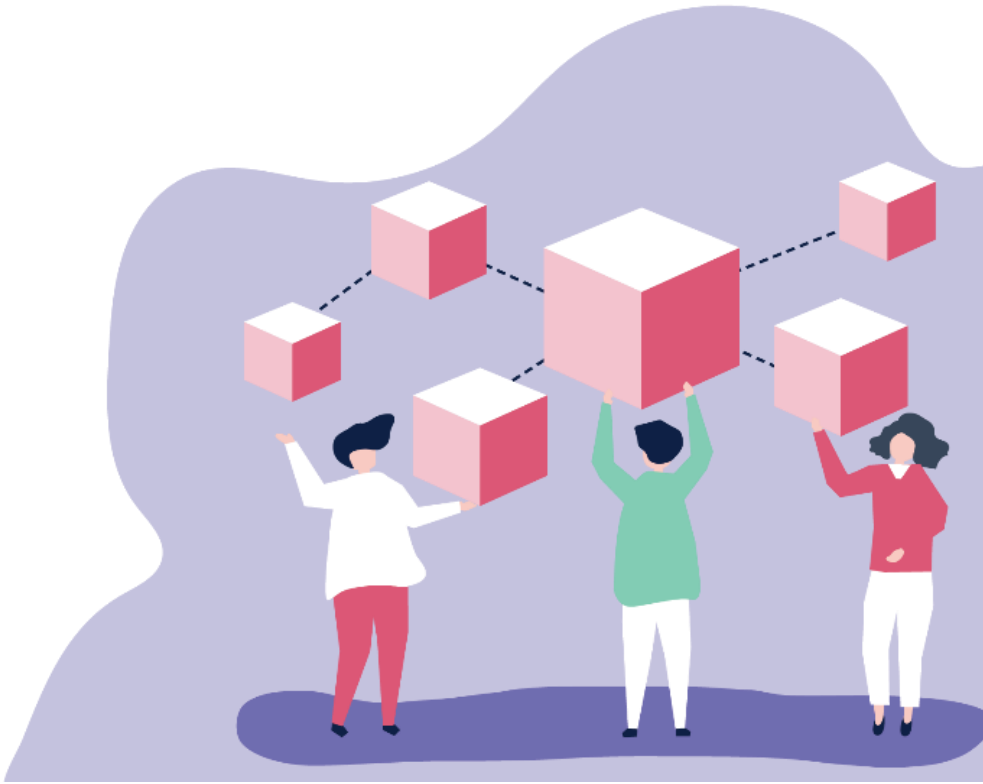
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 857223

D4.12 Data Federation and Integration and Health Semantic Data Lake

Deliverable No.	D4.12	**Due Date**	31/12/2021
Description	Report describing the Data Federation & Integration framework allowing the integration and harmonization of data coming from heterogeneous data sources.		
Type	Report	**Dissemination Level**	PU
Work Package No.	WP4	**Work Package Title**	GATEKEEPER Things Management Infrastructure & Development
Version	1.0	**Status**	Final



Authors

Name and surname	Partner name	e-mail
Domenico Martino	ENG	domenico.martino@eng.it
Paolo Zampognaro	ENG	paolo.zampognaro@eng.it
Vincenzo Falanga	ENG	vincenzo.falanga@eng.it
Federica Saccà	ENG	federica.sacca@eng.it
Eugenio Gaeta	UPM	eugenio.gaeta@lst.tfo.upm.es

History

Date	Version	Change
05/11/2021	0.1	Updated Initial ToC from the first version
09/11/2021	0.1	Update section within the document
12/11/2021	0.2	Updated and added new section
18/11/2021	0.3	Added some description on component interaction
26/11/2021	0.4	Added information on GKIE and pilot needs
01/12/2021	0.4	Integrated UPM contribution and added Appendix section
03/12/2021	0.5	Requirements section has been improved
06/12/2021	0.6	Added information on OKD and some updates
15/12/2021	0.7	Improved section on component interaction
21/12/2021	0.8	Document improvements and conclusions
29/12/2021	0.9	Internal review
30/12/2021	0.9	Addressed W3C comments
20/01/2022	0.9	Quality check review
10/02/2022	0.9	Addressed technical quality check comments
14/02/2022	1.0	Final version

Key data

Keywords	Data Federation, FHIR, RDF
Lead Editor	ENG
Internal Reviewer(s)	Dave Raggett (W3C)

Abstract

This deliverable is the final version of the Data Federation & Integration framework and represents an update on the progress of T4.4 aiming to design a framework, named Data Federation & Integration (DFI), conceived (i) to collect data coming from heterogeneous data sources (EHR and IOT), (ii) to harmonize the data against specific semantic models and, finally, (iii) to persist the data in pilot specific cloud nodes. The harmonization steps will proceed exploiting the GK-FHIR profile as for guidelines and indications provided by T3.5.

In particular, the updates regard (i) the design and deployment of the communication function towards the big data platform using kafka; (ii) the integration of a modular web console named Hawtio. In addition, it has been implemented a set of conversion rules from custom data model to GK-FHIR-IE based on T3.4 and T3.5 (i.e. Profiles within the FHIR Implementation Guide). Furthermore, Section I presents a technical feature developed to integrate/communicate with SAMSUNG devices; Section II is updated with additional information about implementation details on GK-IE, GK-FHIR Server, a new component called GK-RDF Watcher and a paragraph on the data conversion. Section III is updated with the description of the Openshift platform and some technical information on the already presented component and their relative deployment procedure on the HPE platform. Besides, other pilots' needs are added to justify few changes into DFI such as the use of the OKD procedure.

Finally, in the Appendix are reported Pilots' data model and the conversion into GK-FHIR Profile.

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of contents

List of tables

List of figures

Introduction [UPDATED]

This deliverable aims to document the second version specification of the "Data Federation and Integration and Health Semantic Data Lake" by extending and replacing the previous deliverable (T4.4, D4.4). In particular, it provides design details of the ***connectivity layer*** bridging IT systems and the GK WoT data space on a pilot-level, thus enabling data access to final applications or to further predictive analytics and data mining core services carried out in WP5. It offers the needed mechanisms to harmonize data coming from **heterogeneous data sources** registered in the platform, including personal clinical data source (EHR/EMR), social care data sources, wearable device data sources, home-based sensor data and activity sensor data, thus producing a Health Semantic Data Lake (HDSL).

The document provides some updates (marked as [UPDATED]) and new sections (marked as [NEW]) in order to describe the changes which were needed mainly related:

(i) to enable a better logging\monitoring

(ii) to improve the performance and integrability with the other platform components and within the HPE deployment environment and

(iii) to design and implement the data converter expected by the project pilots

(iv) to support new conversion flow requested by open callers (Call 1).

According to the development approach, adopted in the whole project, the design of the **Data Federation & Integration** framework is defined using an incremental approach. Therefore, the main goal of the design activity reported in the first version of the document was to satisfy the requirements scheduled for the first period of the project (M15) while this second version (M27) was updated taking into account mainly feedback reports coming from initial pilots' roll out experiences, issues arising from integration step with the other GK platform components, open caller requests.

In order to improve the readability of the document some sections of the previous deliverable (T4.4, D4.4) not affected by any changes (e.g. involved standards) have been dropped. Details of the document structure are the following:

Section I (ex-Section II in the first version) describes the position of Data Federation & Integration into Gatekeeper architecture showing the several components with which it interacts and some updates made in this last version of the deliverable. It offers an update (compared to the previous one) of the easy modality to enable external heterogeneous data sources to send their data by harmonizing such data against common semantic models selected by the project (e.g. HL7 FHIR) and it allows other Things to access such data. It also provides an update of the general overview of the problems with data integration process together with several approaches available in the literature and the perspectives. It lists also all requirements collected during the phone calls made with the leader of each pilot and updated for this second version; such requirements have been the starting point for designing the architecture and the interfaces of the Data Federation & Integration. Moreover, it supports two types of approaches (that are unchanged) to convert data coming from pilots' application into the adopted semantic models; the first one is a declarative approach where a declarative language is used to specify the conversion rules that are executed by a specific internal engine, while the second one is a programmatic approach where some JAVA interfaces are provided in order to load a specific converter. Finally, regarding the interactions managed by the GK platform, in this version of the deliverable (i) it is added the description of the interaction between Data

Federation and Kafka; (ii) some changes/addition in the interactions and integrations compared to the previous release are reported. For instance, Medisantè IoT Connecter, Samsung Activage gateway, CSS, some pilots' applications.

Section II describes the internal architecture of the Data Federation & Integration. It consists of four main components: gk-integration-engine, gk-fhir-server, gk-rdf-watcher, gk-rdf4j. gk-integration-engine provides the southbound APIs that can be invoked by pilots' applications to send their data by harmonizing such data against common semantic models selected by the project. gk-integration-engine and gk-rdf4j provide the north bounds APIs that allows the external applications to retrieve persisted data in FHIR and RDF format. The keycloak component is used to implement the security level of the application in order to perform integration tests with the other components. These interactions have been exploited for testing purpose only. In pilots' environments all calls to southbound and northbound APIs are expected to be trusted since the interaction with the Data Federation & Integration is mediated by GTA. In addition, TMS mitigates access to the Data Federation after authorization provided by the Keycloak service offered by the GTA. For all components consisting of DFI a docker image has been created and added in a docker-compose file so that all the containers can be started with one single command.

Section III provides the migration of DFI to OKD that will be deployed on the HPE (task 4.1) infrastructure. On the HPE Centre have been also deployed the four main components (gk-integration-engine, gk-fhir-server, gk-rdf-watcher and gk-rdf4j). The different possible deployment scenarios are described. The cluster consists of several PODs that interaction with Service. The contact point among OKD and external applications are Routes.

Section IV provides the conclusion the document.

Appendix A reports information about pilot data models, FHIR data type conversion and describes the instruction to build a new Java converter.

Change log [NEW]

In this paragraph, are listed modified/updated/unchanged chapters compared to the first version of the deliverable

Reason for change	Issue	Revision	Date
Updated version for gatekeeper architecture	2	0.1	05/11/2021
Updated 'Pilots and applications requirements', 'Architecture' and 'Adopted semantic models' of the 'Data Federation & Design' paragraph	2	0.1	09/11/2021
Updated paragraphs 'Apache Camel' and 'Data Converter' related to the 'GK-Integration Engine'	2	0.2	12/11/2021

Updated information on 'GK-FHIR Server' and on 'Data Federation & Integration Dockerization'			
Added new paragraphs about the deployment environments related to Data Federation (i.e. 'Platform description: OpenShift', 'GK FHIR Server and GK RDF Watcher deployed on HPE Data Centre') **Updated 'Deployment Scenarios'**	2	0.2	15/11/2021
Added description of the 'Interaction between Data Federation and Kafka' **Added information about new 'Component Interaction and Integrations' such as interactions and integrations with HCP, CSS, Medisantè IoT Connector, and so on.**	2	0.3	18/11/2021
Added information about 'Web Console' related to the GK-IE, GK-RDF Watcher, **Added information about the 'Data Converter'** **Added information about the 'Source Code'**	2	0.3	23/11/2021
Added description of the 'GK Integration Engine deployed on HPE Data Centre' **Added description of the 'RDF4J Workbench deployed on HPE Data Centre'** **Added information about the 'Pilot needs' and the description of the 'CI and CD for Data Federation: Jenkins'**	2	0.4	26/11/2021
New Appendix sections are added in order to complete information within the deliverable	2	0.4	1/12/2021
'Requirements' completed with the Open Callers information **Updated figures according to the overall changes**	2	0.5	3/12/2021
Added information on OKD	2	0.6	6/12/2021
Updated information on pilots' needs	2	0.6	9/12/2021
Update the whole Section I	2	0.6	15/12/2021

Added paragraphs 'Interaction and integration with Medisantè IoT Connector' and 'Interaction and integration with Activage gateway'	2	0.7	15/12/2021
Added paragraphs 'Interaction and integration with HealthCloudProxy (HCP)', 'Interaction and integration with Aragon (SALUD) Application', 'Interaction and integration with Poland Application'	2	0.7	17/12/2021
Completed Section I, Section II and Section III	2	0.8	20/12/2021
Document improvements	2	0.8	29/12/2021
Version ready for internal review	2	0.9	29/12/2021
Integration of comments provided by W3C	2	0.9	30/12/2021
Version ready for quality check review	2	0.9	20/01/2022
Integration of technical comments provided by quality check	2	0.9	10/02/2022
Final version	2	1.0	14/02/2022

1 Data Federation and Integration v2: Overview, requirements, design

This chapter presents the updates implemented for the Data Federation and Integration.

Such updates are the results of a synthesis of the outputs coming from D3.1.2 from one side and of the conclusions arisen during the bi-weekly phone calls in WP4.

1.1 Position of Data Federation & Integration into Gatekeeper Architecture [UPDATED]

The Data Federation and Integration (DFI) is one of the core components described in deliverable D3.2. Its purpose is twice: (i) to offer an easy modality to enable external heterogeneous data sources to send their data by harmonizing such data against common semantic models selected by the project (e.g. HL7 FHIR) and (ii) to allow the other "Thing" (e.g. the Integrated Dynamic Intervention Services of WP5 or even external applications) to access such data. A representation of this flow is shown in Figure 1.

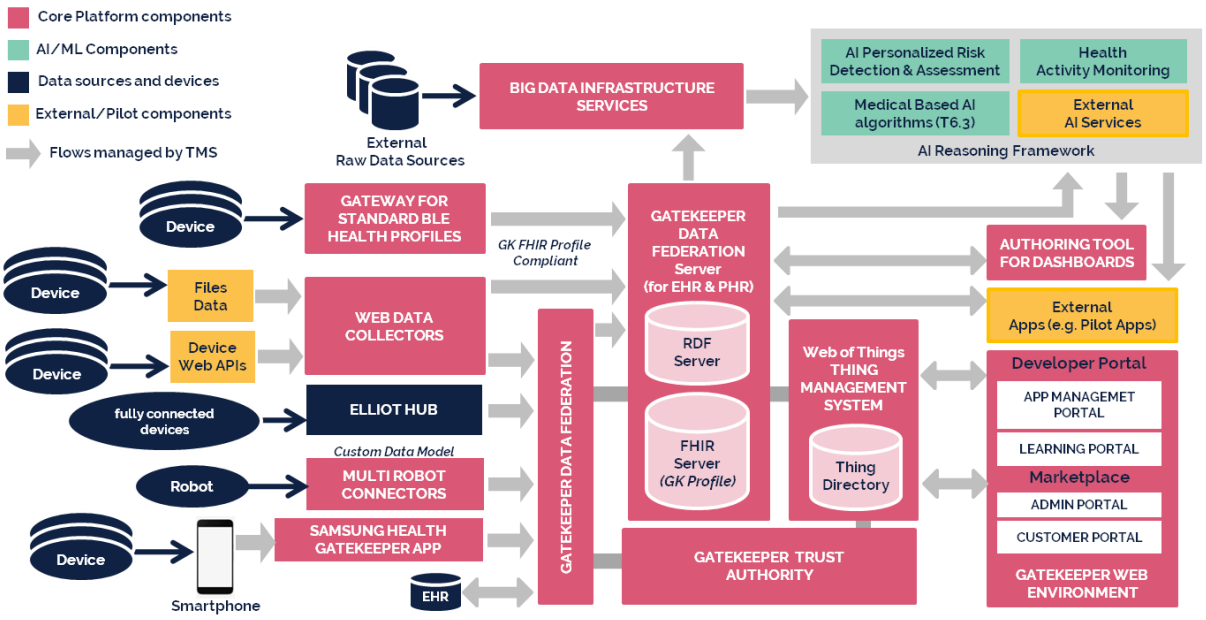


Figure 1 GATEKEEPER Architecture

It is worth to mention that the DFI is itself an aggregation of WoT, as it will be clarified in the next sections. Consequently, "Thing(s)" will be exposed and accessible, by the other components, exclusively through the interaction with the Thing Management System (TMS). Such mediated access also guarantees the respect of the authentication and authorization policies since the TMS performs security check (interacting with the Gatekeeper Trust Authority – GTA – component) each time an access to a GK Thing is requested.

1.2 Pilots and applications requirements [UPDATED]

The goal of this section is to provide the updated requirements affecting the GK Integration Engine. Most of the requirements have already been collected and described in the first version of this document but new ones have been discovered and requested by the pilots, involved in the project, during the second year.

The next section describes the requirements collected in these two years of the project detailing the ones listed in the first version of the document and the ones discovered in this second year.

1.2.1 Requirements [UPDATED]

In this section will be briefly described the requirements affecting the whole GK integration process. They have been collected by analysing the DOA description and considering the new requirements arisen from pilots' specific requests. Such requirements represented the baseline for the Data Federation & Integration design activity outlined in the next sections. Pilots' requirements, including the new ones, have been collected during several dedicated phone calls with the involved pilots and by analysing documents produced in other work packages (e.g. WP3). Before resuming the requirements in Table 9, a brief overview of the analysis pilot per pilot is reported here below.

- *Puglia*

In the period following the first deliverable version, and taking the pilot's feedback into account, new requirements arisen from the Puglia pilot where several external systems are involved as described during dedicated phone calls and also reported in the pilot specific architecture (deliverable D1.3). More in detail it is expected the involvement of three intermediary collecting services, linked to technologies provided by other GATEKEEPER Partners (namely, Medisantè ELIOT Hub, Samsung Health) as well as by the Health Cloud Proxy, developed by ENG in the WP7, that integrates market available data collection platforms (i.e. Google Fit [26], Fitbit [27], iHealth [28] and Biobeat [29]) in order to gather data from a wider set of **IoT** sensors, either provided to patients by healthcare providers or directly acquired by the patients themselves on the consumer market, with the ultimate goal of consistently presenting such data to the clinicians (mainly GPs, in the course of the Pilot experiment, but also specialists or hospital clinicians, in perspective), for them to obtain a richer but uniform view on patients' health status, meeting the monitoring needs of various health profiles of elderly citizens in the Puglia Region.

Moreover, other data are expected to be received externally from the **HIS** (Hospital Information System) of "Casa Sollievo della Sofferenza" Hospital as outlined in Figure 2, in order to conduct research on predictive models for diabetes control, that includes both features available in the HIS and features coming from consumer devices, such as smartwatches equipped with HR/HRV, physical activity, sleep quality and stress detection sensors.

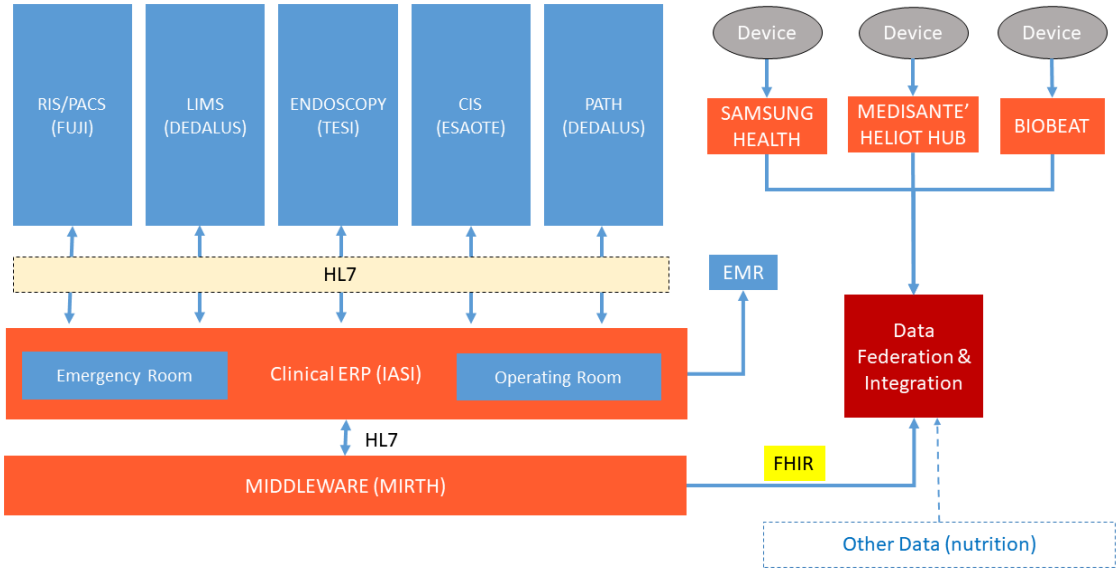


Figure 2 Puglia pilot scenario

The Medisantè ELIOT Hub is a Cloud service able to collect and forward (**PUSH**) data to other systems. It can be configured in order to register the third part APIs to call for data forwarding. The Samsung Health based client, is an Android based mobile app able to retrieve data from sensors (that need to be paired to the app through Bluetooth) store such data on a local PHR on the smartphone (Samsung Health Store) and synchronize such data with the Samsung Health Server in the Cloud and send (**PUSH**) such data to other systems.

Moving to the CSS's EHR data sources it is worth to point out as several internal (HIS) systems could be involved in principles (e.g. RIS/PACS, UMS, ENDOSCOPY etc.). As shown in the figure an intermediary middleware (Mirth) will be exploited to collect and send (**PUSH**) the data to the Data Federation. The input data format is expected to be a custom model, JSON format, so that the Data Federation is mainly involved (i) to convert the structure to the specific GK-FHIR profile (ii) and to redirect (**ROUTE**) the data to the pilot specific FHIR Server.

Health Cloud Proxy (HCP) is service able to collect data coming from different platform such as Google Fit, Fitbit, IHealth and Biobeat. The interaction between Data Federation and HCP is expected to be in **PULL** modality, i.e., it is needed that Data Federation invokes the APIs provided by HCP in order to retrieve the different type of data. Every 24 hours, Data Federation invokes a specific HCP API in the middle of the night to retrieve all data of all registered patients related the whole day. HCP returns data in its custom data model so that Data Federation has (i) convert this structure in the specific GK-FHIR profile (ii) and to redirect (**ROUTE**) them to the Puglia specific FHIR Server.

About the output semantic model, the pilot aims at building final applications (e.g. DMCoach for type II diabetes management) relying on state of the art HL7 standard (i.e. **FHIR**). It is expected the input data (both IoT and EHR) to be "harmonized" (i.e. converted) to such semantic model. The data will be also available in a graph DB to fully exploit the semantic reasoning capabilities that is something useful for the pilot and the intelligent GK components (i.e. WP5).-Finally, about the data location the pilot team has expressed

the preference that acquired data, through the Data Federation, would be held in a **dedicated cloud cluster**.

In the following table, details are resumed regarding the pilot specific requirements arising from the analysis above.

Table 1 Puglia pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
IOT	BP800 (BP, Glucose)	Medisantè ELIOT Hub	PUSH	GK-FHIR Profile	RDF4J	Dedicated cluster
	BC800 (body weight and composition)					
	Biobeat wrist device (HR, BP, SpO2)					
	Samsung smartwatch (HR, physical activity, sleep, stress level) and Activage	Samsung Health and Bixby capsules	PUSH			
	iHealth, Biobeat	Health Cloud Proxy	PULL			
EHR	-	Mirth	PUSH	GK-FHIR Profile	RDF4J	Dedicated cluster

- *Saxony*

Considering the new requirements arisen during dedicated phone calls it is expected the involvement of the intermediary collecting of Samsung Health in order to gather data from several IoT sensors and data coming from the proprietary Saxony application containing data collected by the response of several questionnaires. The interaction with the Data Federation is shown in the Figure 3.

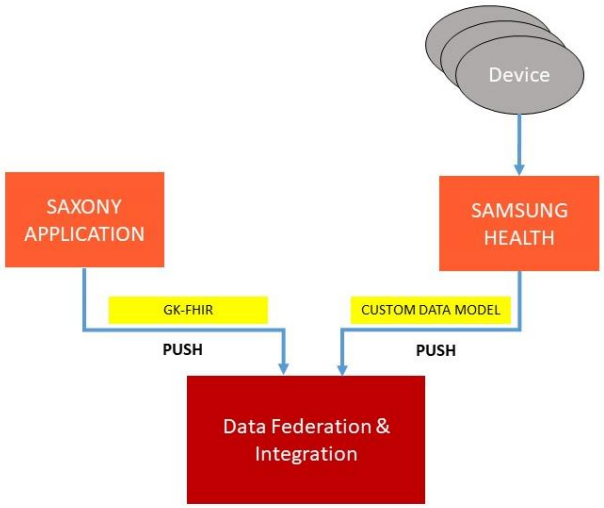


Figure 3 Saxony pilot scenario

More in details, data coming from Saxony application are sent to Data Federation & Integration component by means PUSH modality invoke specific REST API provided by this component already in GK-FHIR format. Such data can be stored in the FHIR Server component without any transformation process.

Data coming from Samsung devices are collected by Samsung Health and forwarded (**PUSH**) to Data Federation & Integration that converts them into GK-FHIR format based on the specific FHIR-Profile developed in the task 3.5.

The data need to be also available in a graph DB to fully exploit the semantic reasoning capabilities that is something useful for the pilot and the intelligent GK components (i.e. WP5).

In the following table, details are reported about the device name, intermediate gateway involved and main expected interaction modality.

Table 2 Saxony pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
IOT	Samsung Smartphone	Samsung Health	PUSH	GK-FHIR Profile	RDF4J	Dedicated cluster
	Samsung Tablet					
	Samsung smartwatch (HR, physical activity, sleep, stress level)					

EHR	N/A	SAXONY Application	PUSH	GK-FHIR Profile	RDF4J	Dedicated cluster

About the output model, the work package 5 aims at building a final Web-based platform for clinicians relying on the state-of-the-art HL7 standard (i.e. **FHIR**) to communicate, receive notifications and for remote monitoring. It is expected the input data to be "harmonized" (i.e. converted) to such semantic model. The data will be also available in a graph DB and we will use RFD4J to fully exploit the semantic reasoning capabilities that is something useful for the pilot and the intelligent GK components (i.e. WP5).

Finally, about the data location the pilot team has expressed the preference that acquired data, through the Data Federation, would be held in a **dedicated cloud cluster**.

- *Aragon*

In the Aragon pilot, several external systems are involved as described during dedicated phone calls and also reported in the pilot specific architecture (deliverable D3.1). More in detail, it is expected the involvement of only one intermediary collecting service called **Data Extraction** that is a module that will be implemented inside the SALUD Application as outlined in Figure 4.

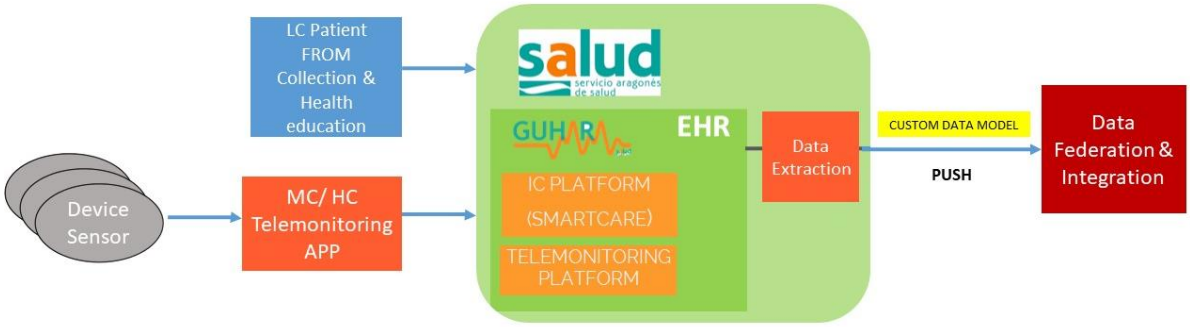


Figure 4 Aragon pilot scenario

SALUD is an EHR data source that collects and groups data coming from two components: "*LC Patient FROM Collection & Health education*" and "*MC/HC Telemonitoring APP*". The first one is used by patients aiming to manage information about their health education while the second one is a gateway, running on smartphone, that retrieves some data coming from sensors and devices and forwards such data to **SALUD** web-app.

SALUD application manages data and information about:

- **Patient / participant** including the basic personal, demographic and recruitment data of the citizen.
- **Social assessment** with basic information regarding social status.
- **Habits** with information on daily routines.
- Clinical Activity **(Hospitalisation)** with information regarding admissions to the Hospital.
- Clinical Activity **(Consultations)** including information related to consultations in primary and specialized care.

- **Prescribed Medication** with information about the drugs prescribed to the patient.
- **Clinical variables** values, including information on vital signs capture values
- **Symptoms** representing information about the existence and/or the intensity of symptoms
- **Forms and questionnaires** (e.g. PROMS)
- **Comorbidities** additional pathologies that belong to episodes active in the patient EHR that are different from the main disease.

As shown in the Figure 4, a direct interaction with the GK platform is not expected, instead a **Data Extractor** engine, deployed inside **SALUD** application, is used to send data to the Data Federation & Integration. Such engine extracts specific data from the **SALUD** EHR and sends to DFI by mean **PUSH** modality. The input format is expected to be in custom JSON or XML representation, in this pilot, so that the DFI will be mainly involved in (i) adapting the structure to the specific GK-FHIR profile and (ii) redirecting (**ROUTE**) the data to the specific FHIR Server pilot.

About the output semantic model, the pilot aims at building final applications (e.g. machine learning algorithms) relying on the state of the art HL7 standard (i.e. **FHIR**). It is expected the input data to be "harmonized" (i.e. converted) to such semantic model. The data need to be also available in a graph DB to fully exploit the semantic reasoning capabilities that is something useful for the pilot and the intelligent GK components (i.e. WP5).

Finally, in respect to the data location, the pilot team has expressed the intention of evaluating the opportunity to send out their data from their owner premise in order to feed GK systems. In the case acquired data, through the Data Federation, would be held in a **dedicated cloud cluster** unless there will be a strong justification to have data in a shared cloud cluster.

In the following table, details summarise the pilot specific requirements arising from the analysis above.

Table 3 Aragon pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
EHR	N/A	ARAGON APPLICATION	PUSH	GK-FHIR Profile	N/A	Dedicated cluster

- *Greece*

About the Greece pilot, the only interaction with Gatekeeper ecosystem is between Greece Application and Data Federation & Integration. The Greece application will collect data from several sensors and EHR applications and will forward them to DFI using the PUSH modality in FHIR version 3 model as shown in Figure 5. This pilot required, for the Gatekeeper project, to use FHIR version 3 and not FHIR version 4.

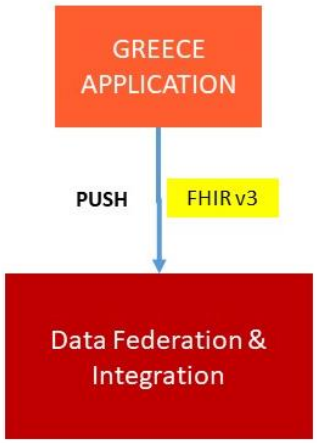


Figure 5 Greece pilot scenario

About the output semantic model, Greek pilot aims at building final applications relying on state of the art HL7 standard (i.e. **FHIR v3**). Finally, about the data location the pilot team has expressed the preference that acquired data, through the Data Federation, would be held in a **dedicated cloud cluster**.

In the following table, details summarise the pilot specific requirements arising from the analysis above.

Table 4 Greece pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
EHR	N/A	GREECE EHR APPLICATION	PUSH	FHIR v3	N/A	Dedicated cluster

Greece will send its data already compliant with the GK-FHIR Profile in form of FHIR dstu3.

- *Basque country*

For the Basque country scenario is expected the involvement of several applications that will send data to Data Federation component by means PUSH modality. Apart from the data coming from Samsung devices that will be converted into FHIR resources by DF components, the other data will be sent already compliant with the GK-FHIR Profile so no transformation/adaption is needed for such type of data as shown in Figure 6. In particular, Sense4Care sends Holter STAT-ON data, IBERMATICA pushes data collected from Abbott devices and Biobeat provides its own devices data. Furthermore, the MAHA application collects and shares data related to questionnaires for user self-assessment, such as the mHealth Application Usability Questionnaire (MAUQ), as well as his/her physical activity data (steps, sleep, etc.) while MYSPHERA integrates directly data from GK-Gateway.

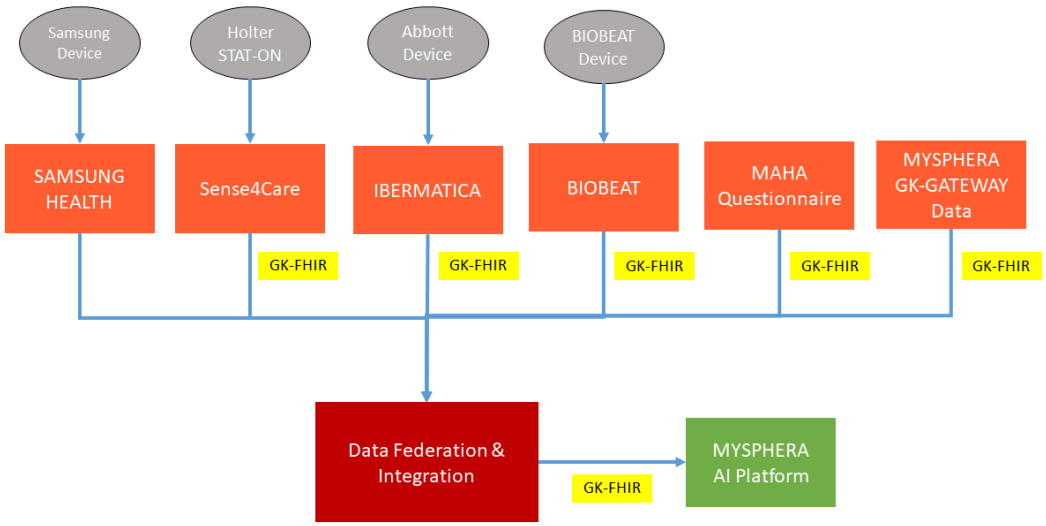


Figure 6 Basque Country pilot scenario

Then, data coming from Basque Country application are forwarded (**PUSH**) to Data Federation and persisted in the FHIR Server. Finally, in the case of acquired data, through the Data Federation, these will be held in a **dedicated cloud cluster** unless there is a strong justification for having data in a shared cloud cluster. In addition, a subset of these data are used by the MYSPHERA AI platform.

Table 5 Basque Country pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
IOT	N/A	Samsung Health	PUSH	GK-FHIR Profile	N/A	Dedicated cluster
	Holter STAT-ON	Sense4Care				
	Abbott devices	IBERMATICA				
	Biobeat devices	Biobeat				
External Health Applications	N/A	UPM Questionnaire	PUSH	GK-FHIR Profile	N/A	Dedicated cluster
		MYSPHERA GK-Gateway				

- *Cyprus*

In the Cyprus pilot, the only interaction with Gatekeeper ecosystem is between Cyprus Application and Data Federation & Integration. The Cyprus application will collect data from several sensors and EHR applications and will forward them to DFI using the PUSH modality in FHIR version 3 model as shown in Figure 7. This pilot required, for Gatekeeper project, to use FHIR version 3 and not FHIR version 4.

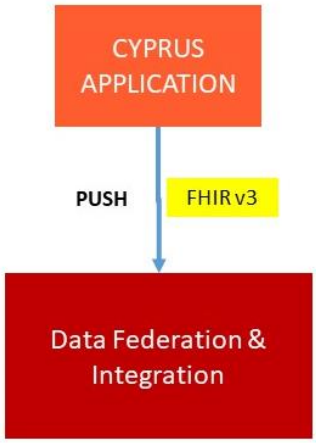


Figure 7 Cyprus pilot scenario

About the output semantic model, the pilot aims at building final applications relying on state of the art HL7 standard (i.e. **FHIR v3**). Finally, about the data location the pilot team has expressed the preference that acquired data, through the Data Federation, would be held in a **dedicated cloud cluster**.

In the following table, details are resumed regarding the pilot specific requirements arising from the analysis above.

Table 6 Cyprus pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
EHR	N/A	Cyprus Application	PUSH	Fhir v3	NO	Dedicated cluster

Cyprus will send its data already compliant with the GK-FHIR Profile in form of FHIR dstu3.

- *Poland*

Figure 8 shows the scenario of the Poland pilot based on the pipeline defined in the deliverable 3.1.2 and the phone call had with the pilot. It will use a custom EHR Poland application that will send data by means PUSH modality to Data Federation in its custom model representation. Data Federation has to convert such raw data to GK-FHIR format based on the profile defined in task 3.5. Once these data are transformed to FHIR, they are persisted in FHIR Server so they are available to the task involved in work package 5.

About the output semantic model, the pilot aims at building final applications (e.g. machine learning algorithms) relying on the state of the art HL7 standard (i.e. **FHIR**). It is expected the input data to be "harmonized" (i.e. converted) to such semantic model.

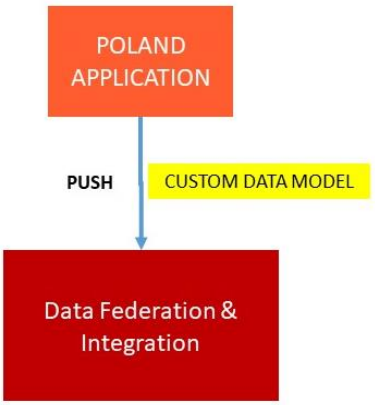


Figure 8 Poland pilot scenario

In the following table, details are resumed regarding the pilot specific requirements arising from the analysis above.

Table 7 Poland pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
EHR	N/A	Poland Application	PUSH	GK-FHIR	NO	Dedicated cluster

- *Milton Keynes*

In the Milton Keynes pilot, several external systems are involved as described during dedicated phone calls and also reported in the pilot specific architecture (deliverable D3.1.2). More in detail, it is expected the involvement of the intermediary collecting services Samsung Activage in order to gather data from several **IoT** devices, smartphones and the Robotic platform Human activities.

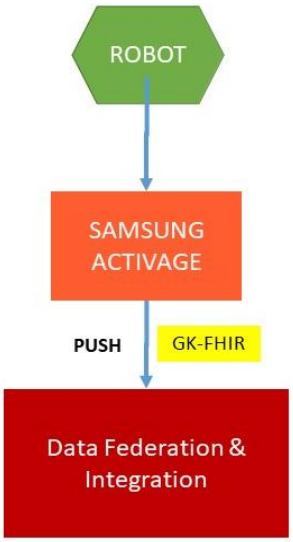


Figure 9 Milton Keynes pilot scenario

The Samsung Activage is a cloud based application able to retrieve the sensors data from the Samsung Health Cloud and send (**PUSH**) such data to other systems. Samsung Activage is also used to retrieve data coming from the robot platform, by means a robot event logger module, and forward (**PUSH**) such data to other systems (in the case of GATEKEEPER to DataFederation & Integration module). Data will be sent already compliant with GK-FHIR Profile.

Milton Keynes pilot is very interested in adopting HL7 **FHIR** standard, that is the final model that will be used. Data need to be provided in a graph DB, to fully exploit the semantic reasoning capabilities that is something useful for the pilot and the intelligent GK components (i.e. WP5).

Finally, about the data location, the pilot team has not expressed any preference that acquired data, through the Data Federation, are held or not in a dedicated cloud cluster.

In Table 8, details summarise the pilot specific requirements arising from the analysis above.

Table 8 Milton Keynes pilot sources

Source type	Device Type	Gateway	DFI Interaction Modality	Output Semantic Model	Graph DB	Data Location
IOT	Samsung smartwatch (HR, physical activity, sleep, stress level)	Samsung Activage	PUSH	GK-FHIR Profile	YES	No preference about the use of a Dedicated cluster
	Robot platform Human					

	activities Environment data remote control					

It is worth to mention that for some pilots (e.g. Hong Kong, Singapore) the requirements needs will be tracked exclusively by Trello board (instantiated within the context of WP7). The information available (at time of writing this document) from such tool, are summarized, along with the other pilots, in the tables below while details about data flow and data sources are not reported here since under finalization

Table 9 Pilots' requirements-1

	Puglia	Saxony	Greece	Aragon	Milton Keynes	Cyprus	Poland	Basque Country
Data acquisition modality: PUSH	✓	✓	✓	✓	✓	✓	✓	✓
Data acquisition modality: PULL	✓	N/A	N/A	N/A	N/A	N/A	N/A	N/A
External system: IoT	✓	✓	✓	N/A	✓	N/A	N/A	✓
External system: EHR	✓	✓	✓	✓	✓	✓	✓	N/A
Output semantic model: FHIR	✓	✓	✓ Dstu3	✓	✓	✓ Dstu3	✓	✓
Output semantic model: SAREF/OTHER	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Data availability in a graph DB	✓	✓	N/A	✓	✓	N/A	N/A	N/A
Dedicated data repository	✓	✓	N/A	N/A	✓	N/A	N/A	N/A

Table 10 Pilots' requirements-2

	Bangor	COVID-19	Singapore	Hong Kong	Taiwan
Data acquisition modality: PUSH	✓	✓	✓	✓	✓
Data acquisition modality: PULL	N/A	N/A	N/A	N/A	N/A
External system: IoT	N/A	N/A	N/A	N/A	N/A
External system: EHR	N/A	N/A	N/A	N/A	N/A
Output semantic model: FHIR	✓	✓	✓	✓	✓
Output semantic model: SAREF/OTHER	N/A	N/A	N/A	N/A	N/A
Data availability in a graph DB	✓	N/A	N/A	N/A	N/A
Dedicated data repository	✓	✓	✓	✓	✓

1.3 Data Federation & Design v2 [UPDATED]

1.3.1 Architecture [UPDATED]

Data Federation and Integration aims to integrate data coming from a different and heterogeneous data source in a common selected data model harmonizing their representation in order to create a single view of such data that can be accessed from external applications, i.e. from "Thing" developed in the scope of work package 5.

Figure 10 shows an update of the general overview of such component, it is able to accept data coming from a different source (i.e. devices, sensors, electronic health records, and so on) in a different format (json, xml, etc.) and store them in a common repository. It provides some APIs to allow this integration. The selected ontologies are HL7-FHIR v4 [10] and SAREF. Data can be retrieved in FHIR and RDF format [15], but in this updated version it also supports the sending of stored data on a Kafka channel in order to support Big Data Analytics environments.

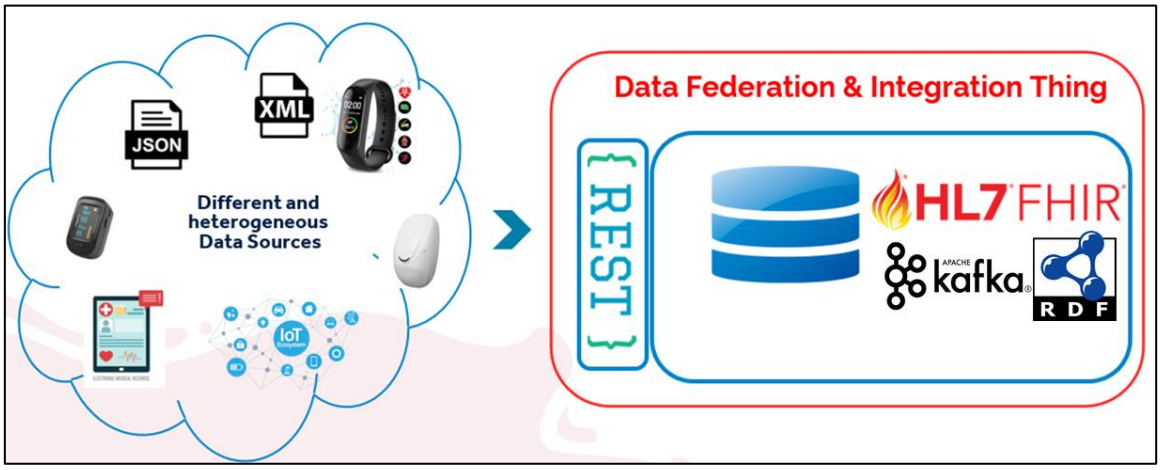


Figure 10 Data Federation & Integration Thing: Overview

DFI offers a utility to harmonize data against the GateKeeper defined FHIR Profile coming from task 3.5. In details:

- It offers REST APIs (southbound) to acquire data from IOT/EHR data source to GK-FHIR Profile compliant data.
- It offers REST API (northbound) to access the converted data for immediate integration in external component or application.

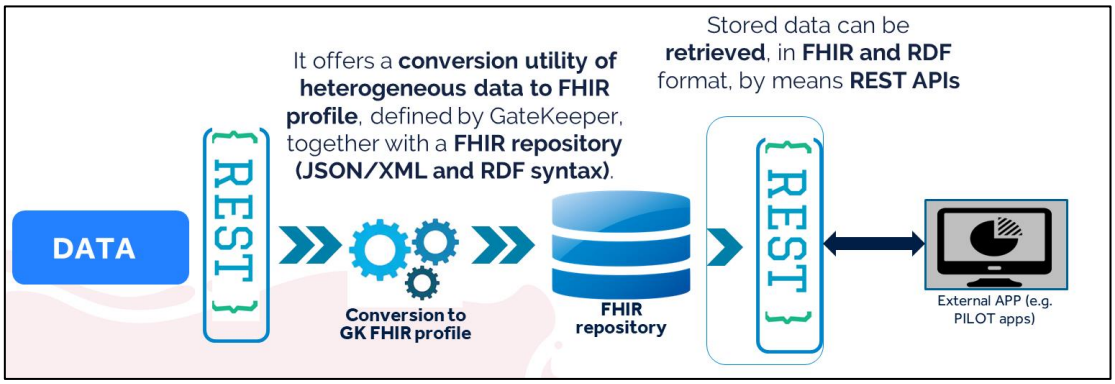


Figure 11 Data Federation & Integration pipeline

As shown in Figure 11 DFI provides some REST APIs that are able to accept data coming from different sources. Collected data are converted in FHIR and RDF representation, by a set of conversion routines, and persisted in a common repository. Stored data can be retrieved, in FHIR and RDF format, by means REST APIs.

The architecture of Data Federation & Integration in this version of the deliverable is updated and currently it consists of four main components gk-integration-engine, gk-fhir-server, gk-rdf-watcher and gk-rdf4j. Logically, it is a composition of three "Things" each one providing its Thing Descriptor (TD) as shown in Figure 12 and a component that supervises and facilitates the RDF conversion. Such TDs describe the three distinct APIs exposed by this component.

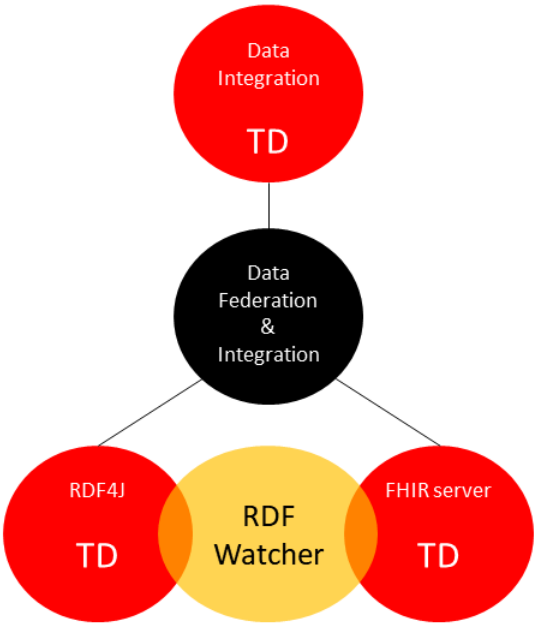


Figure 12 Data Federation & Integration Thing

The internal components, updated in this version with RDF-watcher, are:

- gk-integration-engine
- gk-fhir-server
- gk-rdf-watcher
- gk-rdf4j

each one with specific features and responsibilities.

In Figure 13 an updated schema of how to use DFI Thing is shown. In detail:

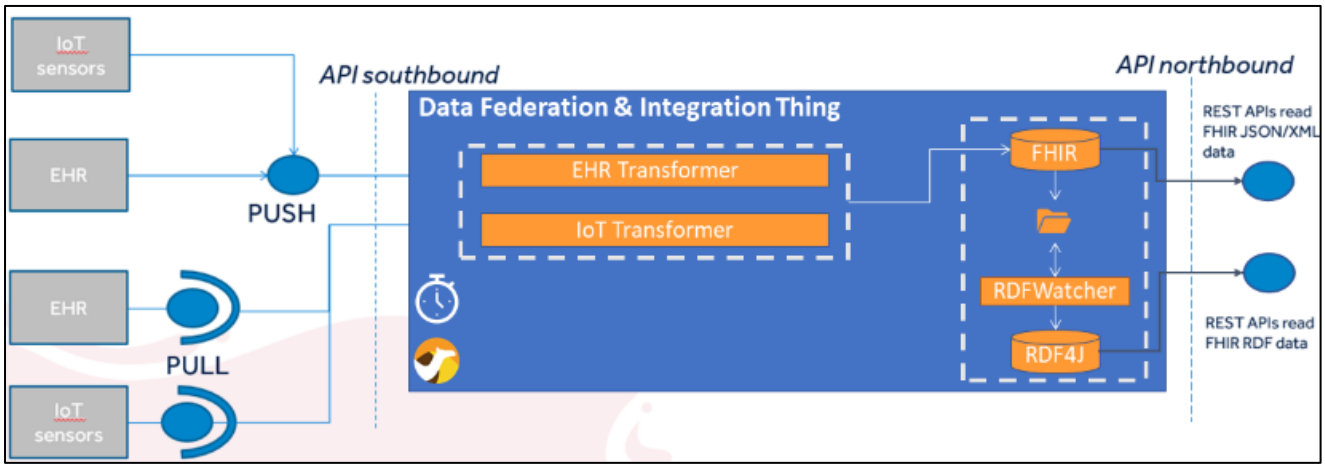


Figure 13 How to use Data Federation & Integration Thing

gk-integration-engine provides the southbound APIs to receive raw data from external data sources, acquired data are converted to FHIR/RDF representation according to preload conversion rules, finally converted data are sent to gk-fhir-server and rdf4j through of the APIs that they provide.

This component offers two kinds of modalities for the interaction called PUSH and PULL. With the modality PUSH external applications have to invoke the gk-integration-engine APIs in order to send data to DFI instead with PULL modality the DFI invoke APIs offered by external applications in order to share data with DFI. With the latter modality it is clear the external application should provide the APIs that can be called by the DFI.

gk-fhir-server is a web server compliant to FHIR standard that provides the set of operations to retrieve, store, update and delete FHIR Resources. Data are persisted in a dedicated repository. It offers a set of northbound APIs that can be invoked by external application to retrieve persisted information according to FHIR specification in JSON and XML format. How we will discuss in this document, through the FHIR interceptor, we are able to store the received resources in a series of JSON files to enable their conversion by the RDF Watcher. At the same time, always through interceptor, we can set a Kafka channel and publish all the received resources also on a pre-setup channel in order to able big data analytics processes.

gk-rdf-watcher, the new component described in this version of the deliverable, is a Linux-based process that listens to every change that occurs against a folder shared with the gk-fhir-server; more in detail, it takes all the JSON files stored by the FHIR Server and manages the conversion and sending to the RDF Server.

gk-rdf4j provides a set of APIs to store, update and retrieve data in RDF format offering a set of utilities to execute SPARKQL queries. It has a dedicated repository where data are stored in RDF representation.

All these modules are described in detail in a dedicate section.

Data Federation is integrated within the TMS that implements an API gateway microservice and load balancer pattern. Figure 14 shows the sequence diagram that involves the TMS, GTA and Data Federation. Following the figure, the TMS offers an API management service that sits between a client (pilot application) and a collection of Data Federation back-end services. It mitigates access to the Data Federation after authorization provided by the Keycloak service [11] offered by the GTA.

In relation to the Data Federation, the TMS acts as a reverse proxy to accept all API calls, aggregate the various services required to manage them and return the appropriate results.

The APIs associated with Data Federation services, such as GK Integration Engine, GK-FHIR Server and GK FHIR RDF, are deployed through the TMS which acts as an API gateway. Typically, API gateways handle common tasks used on an API service system, such as user authentication, replication for high availability, and statistics.

In Gatekeeper, authentication and statistics are offered by the GTA through the keycloak component and the Audit blockchain service, while replication for high availability is offered by the Open Shift platform.

Basically, an API service accepts a remote request and returns a response. But when hosting APIs on a large scale, the scenarios that can occur are varied and managing them can be complex.

To prevent the API from being over-exploited or used without proper permissions, it is necessary to implement an authentication mechanism and multiple request management through load balancers. Also, to understand how developers use APIs, analytics and monitoring tools are needed.

In a microservices-based architecture, this means that a single request could require calls to several different services before providing the result to the client. Another important aspect is the dynamicity of the APIs, some new API services can be added and others removed, so it is needed to ensure that clients can always find them in the same place.

Providing clients with a simple and reliable experience, regardless of this complexity, is an important factor to ensure in a microservices-based architecture. The TMS offers a way to decouple the client interface from the Data Federation back-end implementation. When a client sends a request, the TMS splits it into multiple requests, routes then where needed, processes responses, and tracks each operation. It is the heart of the API management system and offers secure access to the Data Federation by intercepting all incoming requests and sending them through the gateway, which processes a series of necessary functions.

In Gatekeeper, the TMS provides interfaces to authentication, routing, high availability, analytics, policies, alerts, and security.

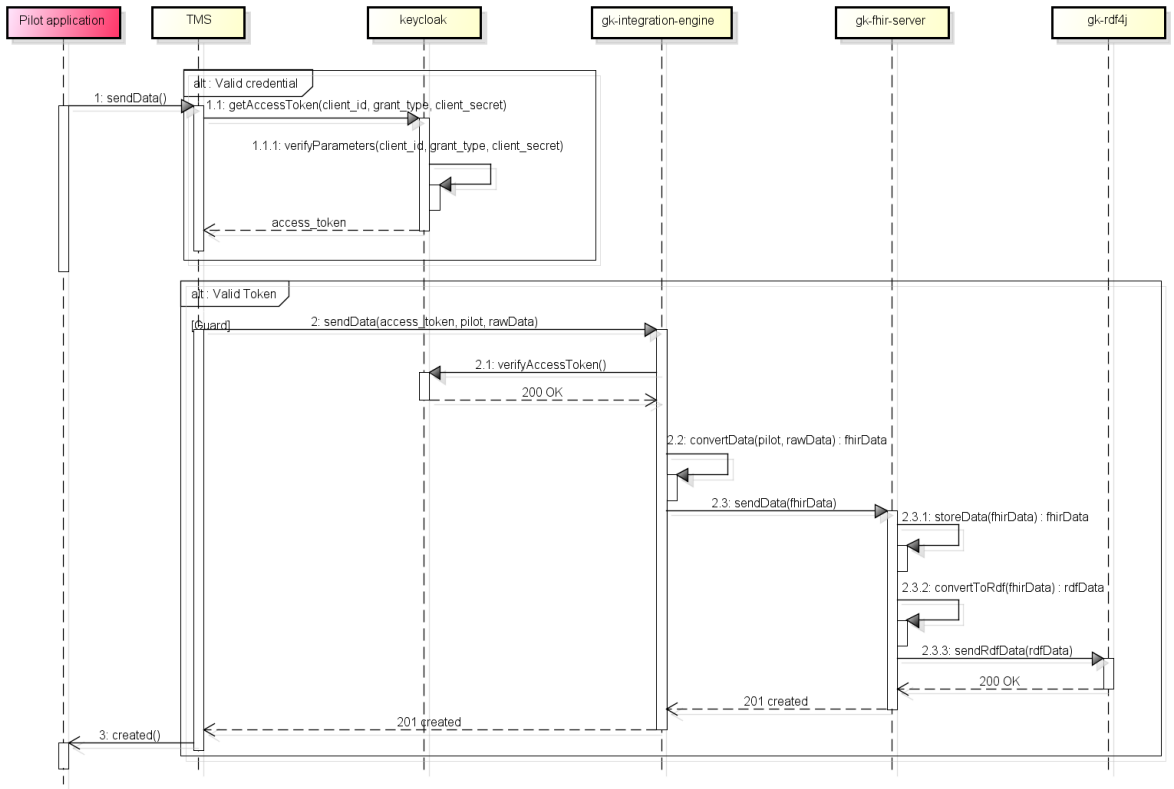


Figure 14 Data Federation & Integration flow

Figure 14 is updated with the new interactions with the TMS component and it shows the steps followed by Data Federation & Integration to persist data coming from an external application using the PUSH modality. The pilot application that wants to send data to DFI asks the permission token to the TMS module. TMS module, then, forwards the client_id, grant_type and client_secret related to such pilot to the keycloak to verify if the passed values are correct and returns an access token to TMS. Now, the TMS with valid token is able to make a request towards the gk-integration-engine passing the received token (by keycloak) and raw data that wants to persist into DFI repository. gk-integration-engine, based on pilot name, selects the right routine to convert custom raw data to FHIR standard according to the GK-FHIR Profile. Transformed FHIR data are sent to gk-fhir-server

invoking the APIs that it provides. gk-fhir-server persists received data in dedicated FHIR repository, convert them to RDF format and sent such data to gk-rdf4j that persists them into its repository. A response message is returned to the pilot.

1.3.2 Adopted Semantic models [UPDATED]

As already described above, Data Federation & Integration aims to integrate and harmonize data coming from heterogeneous data source, registered in GateKeeper platform, including EHR, wearable device data sources, home-based sensor data and sensor activity sensor data, thus, to producing a Health (semantic) interoperability repository enabling the development of advanced services to focus on scenarios and requirements provided by the pilots involved in the project.

Based on the analysis performed during the remote calls scheduled with pilots (also highlighted in Table 9) the main semantic model adopted is HL7-FHIR. Moreover, to ensure semantic interoperability, in the current version of this deliverable a controlled and shared vocabulary has been applied, also based on the use of appropriate terminologies (s. A.1). Such terminology models are built to meet the specific needs of a specific domain, where their nature is structured by vocabularies. Several terminological sources should be available to a community, in order to foster and ensure consistency between the data and information exchanged. Furthermore, the ability to provide coherent representations and the possibility of having access to a wide range of terminologies allows accelerating the interoperability process. Within clinical processes, medical terminology plays a very important role. In fact, it represents a central service for the provision of semantic interoperability between different systems and applications. In particular, appropriate terminology can be used to represent the information contained in clinical databases, data resulting from observations produced by qualified personnel in a specific domain, observations deriving from meetings with patients, as well as health guidelines, expert systems, and medical knowledge. In fact, terminologies provide a means to organize information and serve to define the semantics of the latter, using coherent mechanisms that can be computed by a machine. In addition, they are extensible, meaning that the data described by a particular collection of terms can, in turn, incrementally collect additional terms, which will then be reclassified and re-indexed. Summarizing, therefore, the main purposes for which it is necessary to use standard terminologies concern the ability to provide consistent meaning, the need to promote shared understanding, the ability to facilitate communication with humans, the need to enable comparisons and data integration and the possibility of guaranteeing the portability and sharing of Electronic Health Records (EHR).

1.3.2.1 Process to define GK HL7 FHIR Implementation Guide [UPDATED]

In order to build a common semantic and integrated GK repository, the DFI framework has to know which resources of FHIR standard must be used together with the selected vocabularies to represent information coming from the several pilots' applications. To reach this goal inside the scope of the Gatekeeper project has been designed and applied a specific integration and interaction process that has involved tasks 3.4, 3.5 and 4.4. During the period the deliverable refers to, such joined work proceeded in order to gather the new data models (T4.4) to acquire them within the GK-FHIR Profile (T3.5) and finally, to produce new custom transformation rules.

Task 3.4, as documented by the relative deliverable, has prepared a template to collect data models and vocabularies used by pilots in their applications. Collected and filled

templates are used by the task 3.5 to build a set of FHIR logical models continuously harmonized for considering the input progressively received by task 3.4**.** The output of task 3.5 is the GK-FHIR Profile and more, in general, the **GK HL7 FHIR implementation guide (IG)** [2] that is used by task 4.4 to convert heterogeneous data coming from pilot application to the GK-FHIR data model. An HL7 FHIR implementation guide (IG) is "*a set of rules about how FHIR resources are used (or should be used) to solve a particular problem, with associated documentation to support and clarify the usage*[1]." A FHIR IG, at the moment of writing the second version of the deliverable, includes very different kinds of artefacts (Figure 15), as FHIR logical models (Figure 16), FHIR API conformance resource; FHIR profiles (Figure 17), and many other FHIR and non-FHIR artefacts.

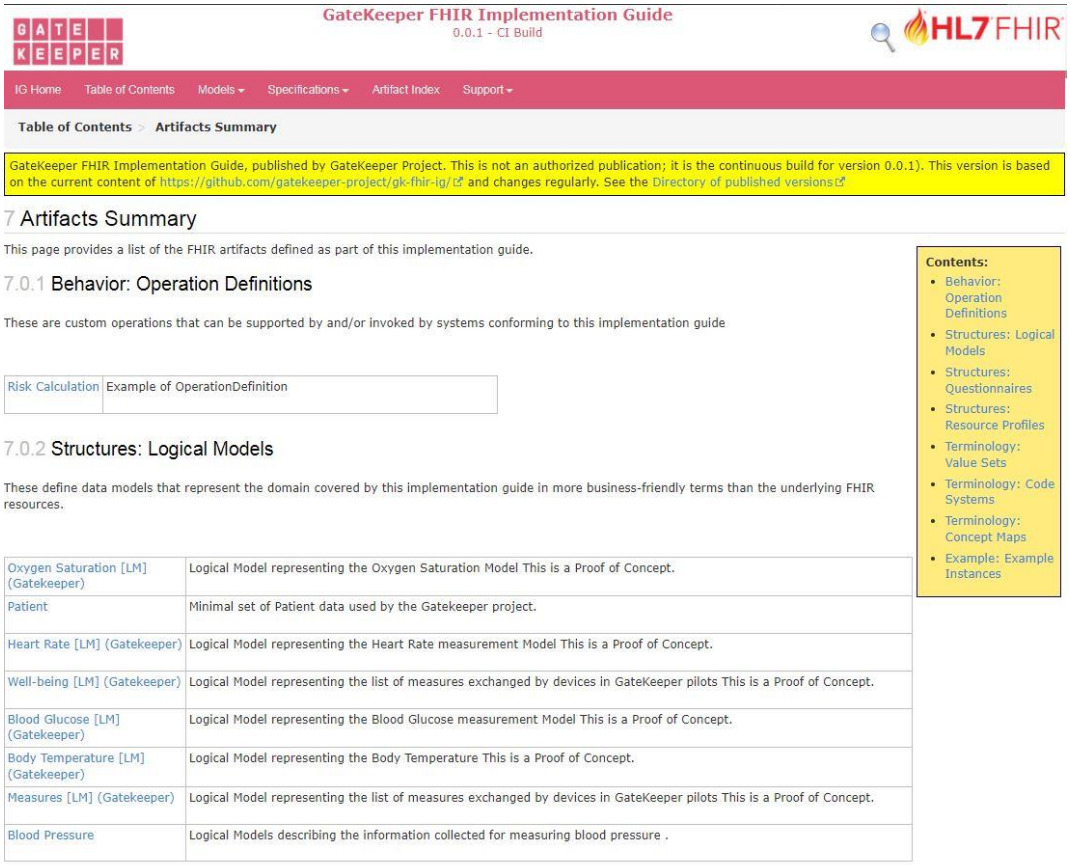


Figure 15 FHIR IG: an example of the Artifact summary

[1] https://www.hl7.org/fhir/implementationguide.html

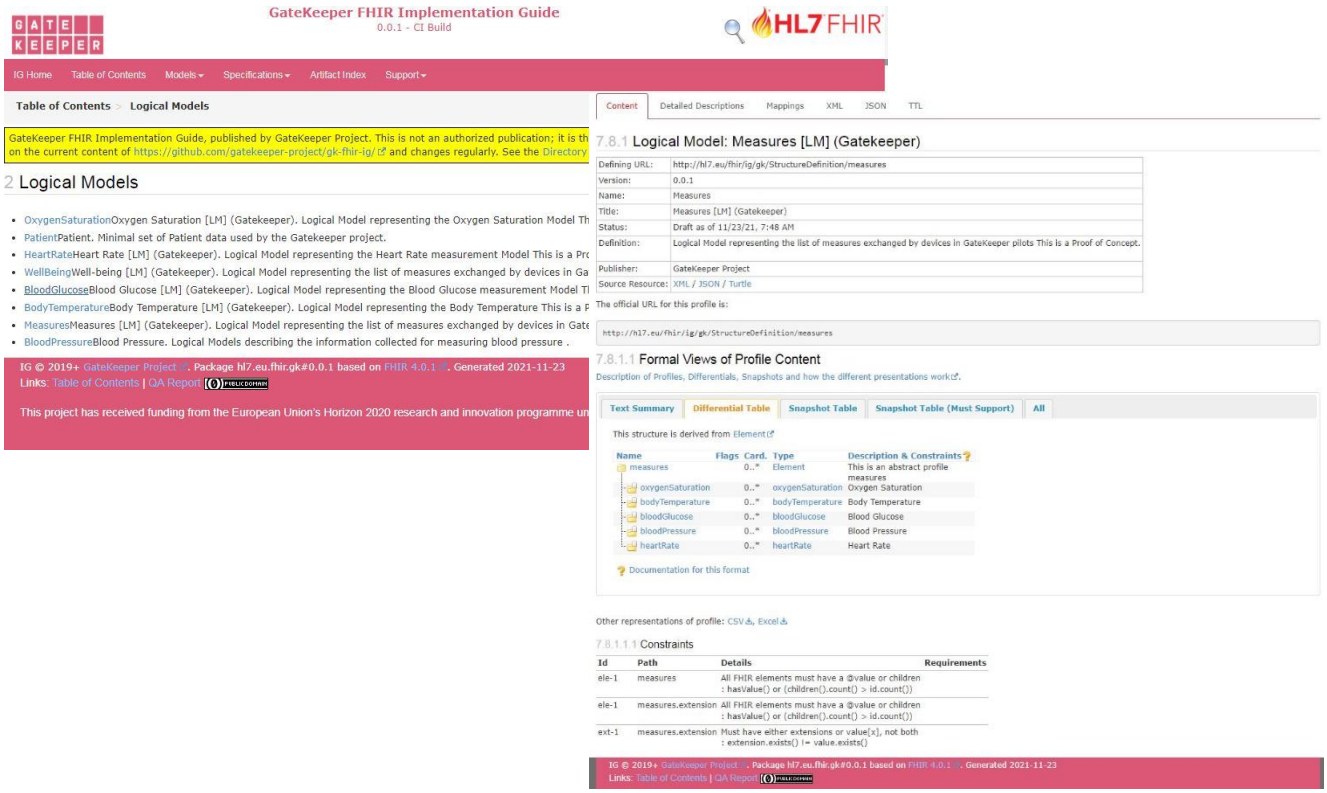


Figure 16 FHIR IG: an example of Logical Models

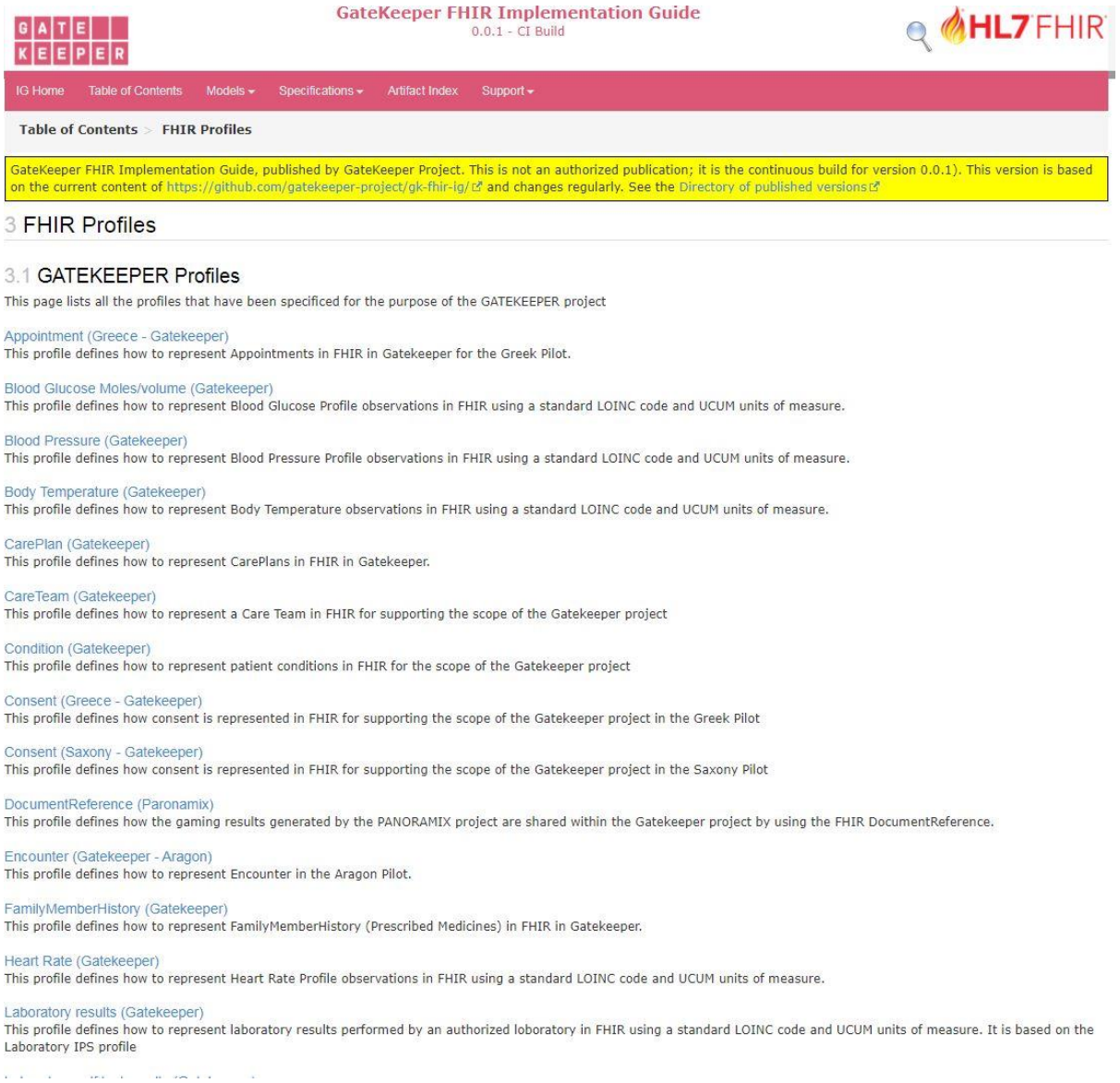


Figure 17 FHIR IG: an example of Profiles

By the way, the focus of the Gatekeeper FHIR IG (Figure 18) is unchanged and is on the data space, thus logical models, profiles, terminologies, and their relationships are specified for GateKeeper project.

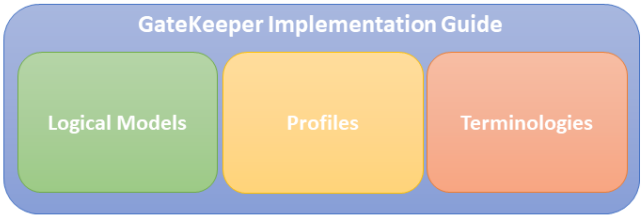


Figure 18 Gatekeeper Implementation Guide (from task 3.5)

Figure 19 shows the interaction process updated with RDFWatcher component to define the Gatekeeper data models, based on FHIR and the relative selected terminologies, that is used to persist and retrieve data from Data Federation & Integration module.

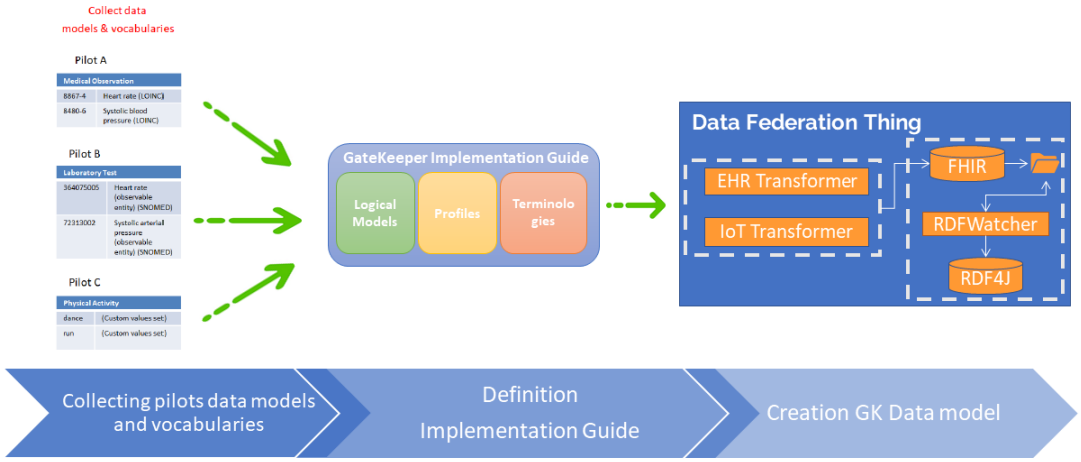


Figure 19 Gatekeeper Data Models definition process

GK-FHIR Implementation guide, more in detail profiles and vocabularies, is a very important input for Data Federation & Integration since it is used to build the conversion rules that are applied by transformers, when external applications invoke the Southbound APIs, to convert pilot data to GK-FHIR profile. Moreover, even if some pilot sends their data already in FHIR format, such data must be adapted and converted to GK-FHIR profile in order to be harmonized with data coming from other pilots as discussed below (s.2.1.4).

1.3.3 Declarative approach

One of the core key components of Data Federation & Integration is gk-integration-engine. This module exposes APIs that are able to acquire data from external heterogenous data sources and "harmonizing" such data to be compliant to GK-FHIR-Profile and the other IoT ontologies selected by the GK project. The heterogeneous data source can be both electronic health record system and IoT devices. Collected data by gk-integration-data must be converted, according specific rules, to GK-FHIR-Profile or some ontology and then sent to the FHIR or RDF repository. In order to perform these conversions, the gk-integration-engine contains a conversion utility that can work in two different approaches: declarative approach and programmatic approach. This section is focused on the declarative approach.

Figure 20 shows the design data model for the Data Source and for the Converter. Data Source is an abstract entity representing a generic source that can be specialized in two subclasses representing the concrete sources, EHR and IoT. EHR data source represents the electronic health records containing data, for example about the clinical status of a person, generated by hospital or health care system while IoT data source represents data that are generated by IoT devices such smartwatch, sensors and so on. Such data differs from the one generated by health electronic health records due to the nature of the information that they manage. IoT devices are used to perform some measurement, with a certain frequency, on a subject and forward result to an application or gateway via Bluetooth.

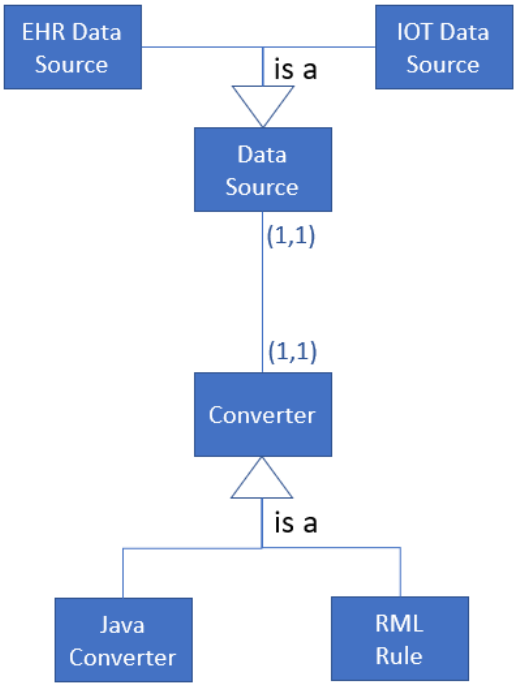


Figure 20 Data Source and Converter model

Each Data source is associated with a specific converter with a relation one to one, this design enforces to have a converter for each instance of data source. Converter, in similar way of data source, is an abstract entity that can be specialized to a Java Convert o RML Rule. The RDF Mapping language (RML) [16] is a generic scalable mapping language defined to express rules that map data in heterogeneous structures and serializations to the RDF data model. RML deals with the mapping definitions in a uniform, modular, interoperable and extensible fashion. RML is defined as a superset of the W3C-recommended [21] mapping language, R2RML, that maps data in relational databases to RDF. In RML, the mapping of data to the RDF data model is based on one or more Triples Maps that defines how the triples will be generated. A Triples Map defines rules to generate zero or more RDF triples sharing the same subject. A Triples Map consists of a Logical Source, a Subject Map and zero or more Predicate-Object Maps:

- A *Logical Source* consists of (i) a reference to input source(s), (ii) the Reference Formulation to specify how to refer to the data and (iii) the iterator that specifies how to iterate over the data. The following reference formulations are predefined but not limited: ql:CSV, ql:CSS3, ql:JSONPath, rr:SQL2008 and ql:XPath.
- The *Subject Map* consists of the URI pattern [20] that defines how each triple's subject is generated and optionally its type. The references to the input data occurs using valid references according to reference formulation specified at the Logical Source.
- *Triples* are generated using Predicate Object Maps. A Predicate Object Map consists of Predicate and an Object Map(s). A Predicate Map specifies how the triple's predicate is generated. An Object Map specifies how the triple's object(s) are generated.

The output of RML is a sematic knowledge that can be persisted in sematic repository. An example in Figure 21 is provided to show how RML rules can be written to produce a sematic representation raw data.

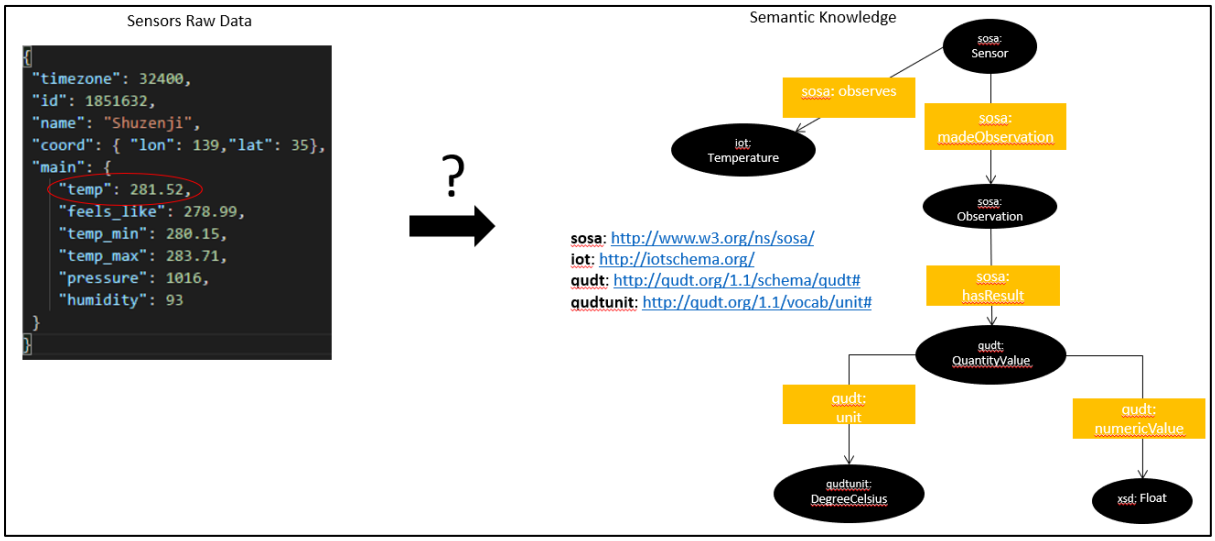


Figure 21 Sensor raw data to Semantic knowledge

On the left of the Figure 21 there is an example of raw representation of weather temperature in JSON format containing information about time zone, sensorID, name of the city where data are related on (Shuzenjii), coordinates (longitude and latitude), external temperature, minimum and maximum temperature, pressure, and humidity. The goal is to convert the json representation of temperature in a semantic model by mean RML declarative rules language. A preliminary activity to perform this task is to select the ontologies that should be used for semantic representation (as already described, an ontology is a formal representation model of the reality and knowledge). It is a data structure that allows the description of the entities (objects, concepts, etc.) and their relationship in a specific knowledge domain. An ontology is the explicit formal description of the concepts of a domain, that is, a model that allows to represent reality (being) in the domain in question, in the form of a set of objects and relations (class of objects).

The ontologies selected to represent the temperature data of the example are sosa[2] and iot[3] for the iot schema, qudt[4] for the representation of a quantity and qutunit[5] to represent of the unit of measure as shown in Figure 22.

[2] http://www.w3.org/ns/sosa/

[3] http://iotschema.org/

[4] http://qudt.org/1.1/schema/qudt#

[5] http://qudt.org/1.1/vocab/unit#

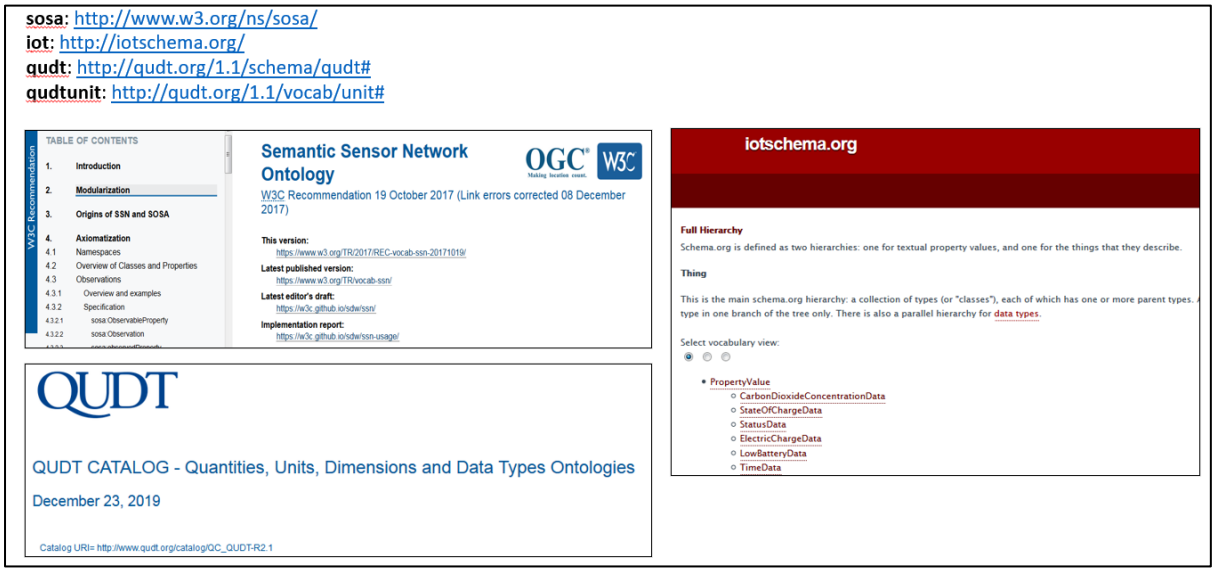


Figure 22 Sensors ontologies

To continue, the right side of the Figure 21 shows the desired target semantic knowledge in a graph representation of the temperature, by means the selected terminologies, represented in the left side of the same figure in json format.
Black ovals represent concepts of the selected terminologies while yellow rectangles represent relations among the different concepts. Arrows are the navigability directions. In detail the device is a sensor (*sosa: Sensor*) that observes (*sosa:observes*) a temperature (*iot:Temperature*); it makes an observation (*sosa:madeObservation*) and the result is an observation (*sosa:Observation*) that has a result (*sosa:hasResult*) of type quantity (*qudt:QuantityValue*). Quantity consists of a numeric value (*qudt:numericValue*) of type float (xsd:Float) and a unit of measure (*qudt:unit*) of type degree Celsius (*qutunit:DegreeCelsius*). By means RML, it is possible to write RML rules that analyses the JSON raw data and provides as output the sematic model described above. Figure 23 shows the rules in RML syntax to produce the semantic model representation.

```
<#WeatherSensor>
    rml:logicalSource [
        rml:source "src/test/resources/examplewebinar/wheater_sensor_info_raw.json";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$";
    ];
    rr:subjectMap [
        rr:template "http://www.gk.namespace/weathersensor/{id}";
        rr:class sosa:Sensor;
    ];
    rr:predicateObjectMap [
        rr:predicate sosa:observes;
        rr:objectMap [ rr:constant iot:Temperature; ];
    ];
    rr:predicateObjectMap [
        rr:predicate sosa:madeObservation;
        rr:objectMap [ rr:parentTriplesMap <#WeatherSensor/Temp/Obs>; ];
    ].

<#WeatherSensor/Temp/Obs>
    rml:logicalSource [
        rml:source "src/test/resources/examplewebinar/wheater_sensor_info_raw.json";
        rml:referenceFormulation ql:JSONPath;
        rml:iterator "$";
    ];
    rr:subjectMap [
        rr:termType rr:BlankNode;
        rr:class sosa:Observation;
    ];
    rr:predicateObjectMap [
        rr:predicate sosa:hasResult;
        rr:objectMap [ rr:parentTriplesMap <#QuantityValue>; ];
    ].
```

Figure 23 Example of RML rule specification for a sensor raw data

Data Federation & Integration also includes a conversion utility that takes in input a raw format (in the case of the example JSON) and the relative RML rules and provides as output the semantic representation, in RDF format, of the source raw data. Rules must be written according to the source raw input and the selected terminologies. Output of the application of the RML rules for the JSON representing temperature is shown in the Figure 24.

```
@prefix iot: <http://iotschema.org/> .
@prefix qudt: <http://qudt.org/1.1/schema/qudt#> .
@prefix qudtunit: <http://qudt.org/1.1/vocab/unit#> .
@prefix sosa: <http://www.w3.org/ns/sosa/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.gk.namespace/weathersensor/1851632> a sosa:Sensor;
  sosa:madeObservation _:0 .

_:0 a sosa:Observation;
  sosa:hasResult _:1 .

_:1 a qudt:QuantityValue;
  qudt:numericValue "281.52"^^xsd:float;
  qudt:unit qudtunit:DegreeCelsius .

<http://www.gk.namespace/weathersensor/1851632> sosa:observes iot:Temperature .
```

Figure 24 Temperature semantic representation in RDF format

As it is possible to see from the Figure 24, it is the same representation of the graph shown in Figure 21. It is defined a prefix for each selected URL terminology, this means that each terminology can be referred by the prefix without using the whole URl, this syntax improves the readability of the code. Device with id 1851632 is a sosa Sensor that made a sosa Observation with id *_:0*. Observation with id _:0 is a sosa:Observation that has a quantity value result referred by id *_:1.* QuantityValue consists of a numeric value and a unit. The value of numericValue is 281.52 of type float while the quantity unit is degree Celsius. Last line of code says that the sensor having id 1851632 observes a temperature. From the view of the model represented by Figure 20 an instance of Converter class is a file containing RML rules that are able to convert data in a semantic representation associated to a specific source that can be an EHR data source or IoT data source.

1.3.4 Programmatic approach v2

Previous section describes the specialization of Converter class in RML rules while this section describes the converter when it is specialized in a Java routine. As already said to each data source can be associated one Converter that can be a Java Converter or a RML Rule. Figure 25 shows data model of the Java Converter offering the possibility to add a new transformation Java class to a specific data resource. New class will be added by hand in the actual release of the Data Federation & Integration.

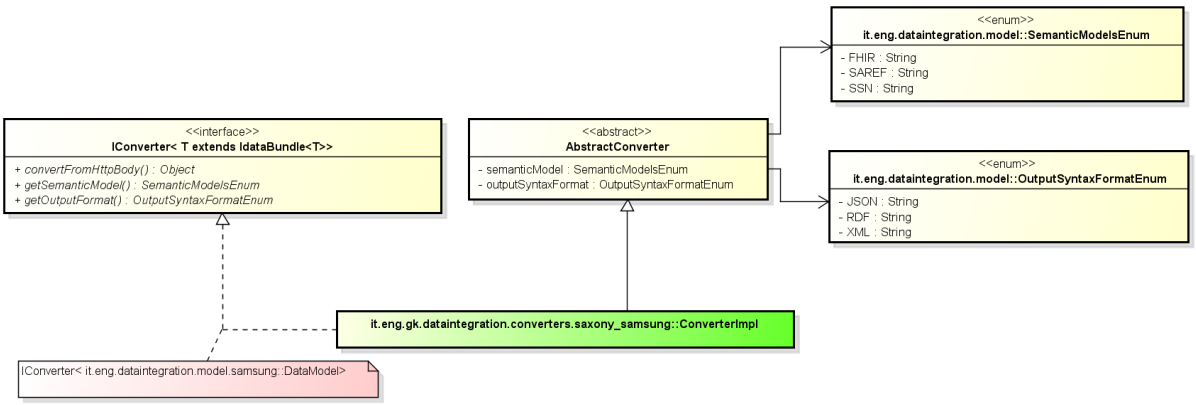


Figure 25 Java Converter Model

Java Converter model consists of an abstract class named *AbstractConverter* that contains two attributes *semanticModel* of type *SematicModelsEnum* and *outputSyntaxFormat* of type *OutputSyntaxFormatEnum*. Attribute *semanticModel* represents the output format ontology that can be FHIR, SAREF and SSN while attribute *outputSyntaxFormat* represents the format of the output data that can be JSON, RDF or XML. When a new converter for a specific data source, belonging to a specific pilot, the concrete implementation of the class *AbstractConverter* must be provided inside new package containing the name of the pilot followed by the name of the application that is used. Figure 25 shows an instance of implemented converter for data collected by Samsung gateway (class in green color) that is used by Saxony pilot.
Each *ConverterImpl* class must selected both the output data format (JSON, XML, RDF) and adopted ontology (FHIR, SAREF and SSN) and it has to implement methods of the interface *IConverter<T extends IDataBundle<T>>* together with the DataModel representing the Java beans of the incoming data. *IConverter<T extends IDataBundle<T>>* offers three methods:

- *convertFromHttpBody()* this method contains the logic to unmarshal data from string to Java Object.
- *getSemanticModel()* that returns the selected semantic model.
- *getOutputFormat()* that returns the format of output model.

Each *DataModel* must implement the interface *IDataBundle<T>*.
To facilitate the implementation of a new converter, an ECLIPSE sample project has been provided together with a guide containing the instruction to implement a new Java converter; reading such guide and modifying the ECLIPSE sample project it is possible to develop and plug a new Java Converter in easy way.
The complete guide is reported in A.4 and A.5.

1.4 Interaction with Big Data Platform Design (T4.3) [NEW]

During the period this deliverable refers to an architectural pattern has been discussed and agreed with HPE in order to forward the data properly converted to the GK-FHIR Profile, to the Big Data Infrastructure in order to make the data available to the AI development environment built on top of the Big Data Infrastructure itself. Such pattern is based on a publish\subscribe mechanism relaying on Apache Kafka integration layer.

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. Its core capabilities can be summarised as high throughput, scalability, permanent storage and high availability. Furthermore, Kafka is offering a set of functionalities in order to manage process streams and integrations of different sources such as the Kafka's out-of-the-box Connect interface. In fact, it is able to integrate with hundreds of event sources and event sinks including third-party products such as Maria DB, JMS, Elasticsearch, AWS S3, and more; or the built-in stream processing that provides process streams of events with joins, aggregations, filters, transformations, and more, using event-time and exactly-once processing.

Finally, Kafka is trusted by thousands of organisations also for its ease of use.

1.4.1 Interaction between Data Federation and Kafka [NEW]

The Data Federation, after every resource's storage stage, if the Kafka channel is enabled, sends the received bundle or resource to the corresponding Kafka channel.

We have one channel for bundle transfer and one channel for single resource transportation. These channels can be used by the big data analytics process listening for new data sent by the FHIR Server.

For the Kafka integration process completion into the FHIR Server lifecycle, we designed and developed a FHIR interceptor. The FHIR interceptors are the mechanism useful for FHIR event acquiring; they enable the registration of some additional execution logic fired when a particular server event occurs. There are many events captured during the FHIR "resource" lifecycle like the one we used: SERVER_PROCESSING_COMPLETED_NORMALLY; this event is called after all processing is completed for a request, but only if the request completes normally without any problems.

When a bundle or resource is received by the FHIR Server, after their storage stage, their JSON representation is published over the correct Kafka channel. After that, all the subscribers can receive the sent information and use them for their own needs.

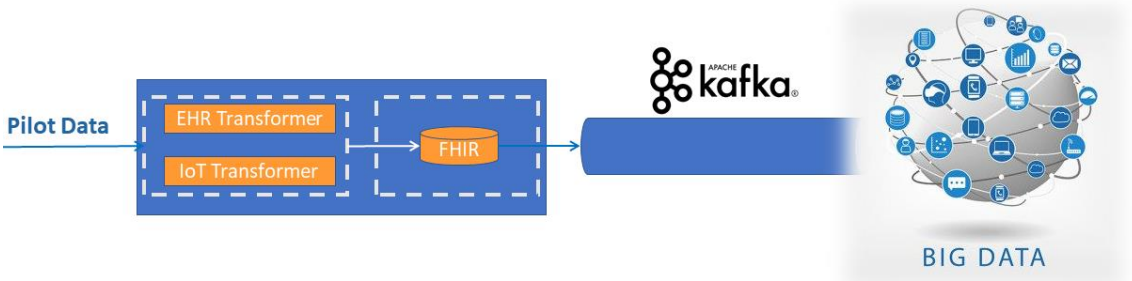


Figure 26 Schema of interaction between Data Federation and Kafka

1.5 Component Interaction and Integrations [NEW]

Data Federation & Integration is naturally linked with other GK Thing. In the second year of the project, HPE has provided the whole infrastructure where all applications involved in Gatekeeper platform must be executed, so ENG has moved the deployment of Data Federation & Integration from ENG Server to HPE platform. This new scenario has required new configuration since applications hosted on HPE ecosystem run behind a proxy and

external system can interact only with a site-to-site connection. The authentication process is not yet managed by ENG but by the GTA task for this reason this feature is deleted by Data Federation and managed by task 4.5.

1.5.1 Interaction and integration with Medisantè IoT Connector [UPDATED]

Eliot Hub is a IoT device platform to enable connectivity of medical devices with any clinical system. It simplifies deployment of telemonitoring at scale on the pilot side providing remote management capabilities. The platform only hosts non-identifiable patient data based on device number (e.g. IEMI) and pull data based on a direct-to-cloud approach into any target system used by the physicians – and patients. The platform relies currently on a limited set of medical devices (CE mark, class I, class II), collecting periodically the most common vital signs (blood pressure, blood glucose level, weight, arrhythmia, …). Over time, the number of vital sign and medical devices will increase to address clinical needs[6].

This section describes how the interactions between DFI and Eliot Hub is performed and tested once the two applications are deployed on HPE platform (and not on ENG Server) with a security site-to-site proxy connection. Eliot Hub includes a test environment where it is possible to register an application where to send fake data representing information coming from the devices that it supports. This test environment has been used to perform some tests of integration to check if Eliot Hub is able to invoke DFI IOT API hosted on HPE Server to send data representing measures made with the devices that it supports by means a site-to-site connection.

On the other side it has been tested if DFI receives correctly data coming from Eliot Hub framework and if such data can be converted and stored in FHIR and RDF repository according the GK-FHIR-profile. The hypothesis is that these devices are used by Puglia pilot.

Since DFI is deployed on the HPE Server behind a proxy, it is reached by external applications only with a site-to-site connection. All service has been deployed on OKD platform as Docker containers running in a POD. In order to perform this integration, following tasks have been performed:

1. Register an organization on Eliot Hub cloud application
2. Add a new user for the organization registered at point 1.
3. Register DFI as a new data source in the section “target systems” of Eliot Hub.
4. Added in DFI a new Java Converter for Puglia pilot and Eliot Hub application.
5. From Eliot Hub application run test executions aiming to send test measurements to DFI.

[6] The portfolio of cellular-based devices will continue to increase according to clinical needs, volume, connectivity attributes, data security attributes and internal validation.

6. Check if data sent by Eliot Hub reach DFI and they are converted and persisted in right way both in FHIR and RDF representation according GK-FHIR profile.

Also, in this updated version of the deliverable, Eliot Hub added a new container on its side in order to integrate a VPN proxy.

Eliot Hub offers a set APIs allowing to register a user, an organization, one or devices and a target system to which sends collected data by the registered devices. The swagger of the application is available on this URL https://api-docs.medisante.net/#/.

The first step is the registration of the organization (e.g. Puglia) and the creation of a new user for such organization. Successful it needs to register the DFI platform as target system. Following there are information required to register the new DFI system:

- Name of the application, "*DataFederation*".
- URL where send data, "*http://gk-integration-engine-gatekeeper-dev.apps.okd.seclab.local/gkie/IOT/data/puglia/medisante*"[7]. Interface enabling IOT Medisantè devices, used for Puglia pilot, this API means that data coming from devices used by Puglia pilot registered into Eliot HUB collector
- Authentication type, "*OAuth 2.0*".
- Grant type, "Client credentials".
- Username.
- Password.

Figure 27 shows the screenshot of the Eliot Hub target system form.

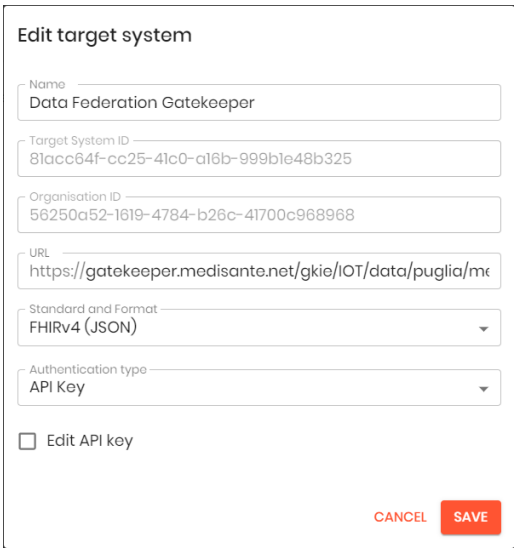


Figure 27 Eliot Hub Target System

[7] this URL will be customised according to the pilot's namespace provided by HPE.

After clicked on save button the new system is registered in the environment test, as shown in Figure 28.

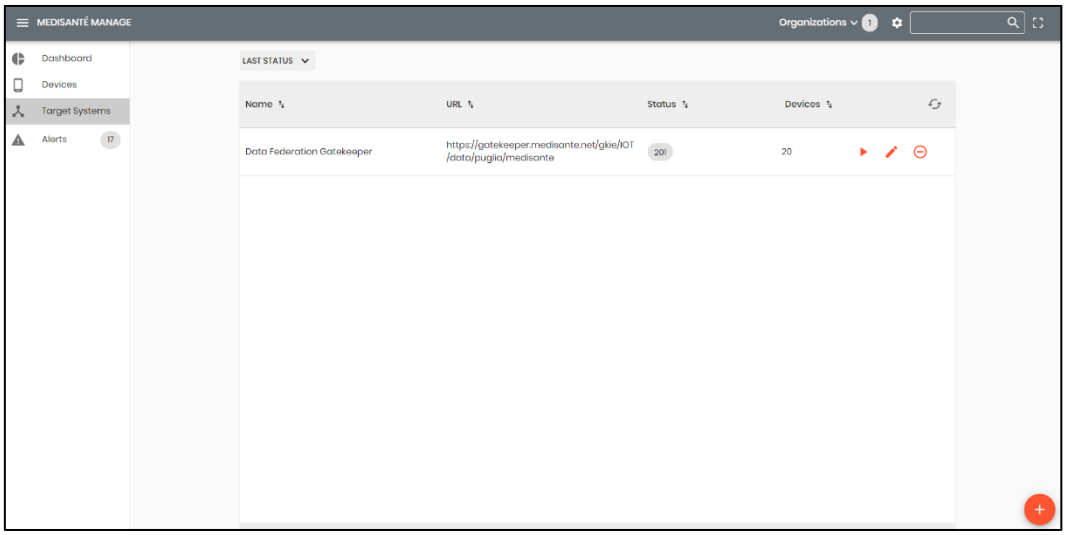


Figure 28 Eliot Manage page

Once the system is registered it is possible to send fake data to DFI. The supported devices[8] from the test environment are:

- BG800, to measure the level of haemoglobin glucometer.
- BP800, to measure the arm blood pressure and blood glucose level.
- BC800, to measure the body weight.
- BT105, to measure heart rate together with the blood pressure systolic and diastolic

Each device is assigned a unique identifier, named IMEI, as shown in Figure 29.

[8] https://medisante-group.com/devices

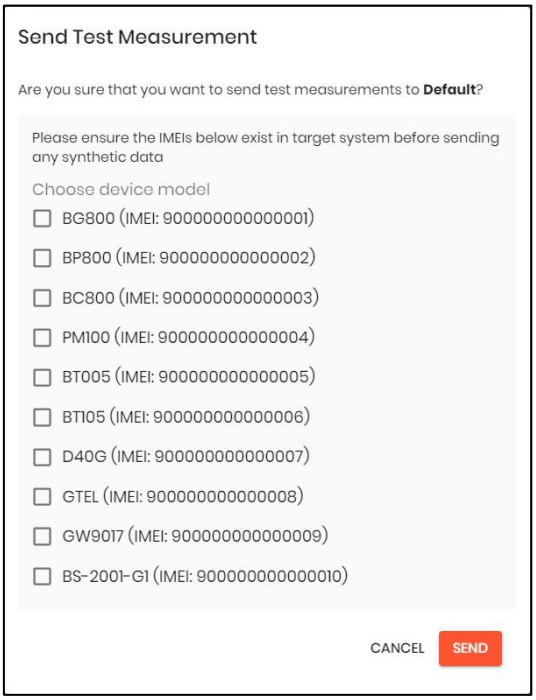


Figure 29 Eliot Hub test devices

Eliot Hub framework produces test data in FHIR v4 JSON representation, such JSON consists of a Bundle resource containing one or more entry, one for each type of performed measure represented as FHIR Observation measure. Each Observation contains the type of measure, the date when it has executed, the value of the measure together with the unit of measure and finally the identifier of the device that has generated the measure represented as contained FHIR Device resource. In addition, in this version of the deliverable, Eliot Hub updated the HL7 data format adding a 'suggestions' field.

When Eliot Hub application sends data to Data Federation & Integration, such data are transformed to GK-FHIR-Profile and stored in the repository in FHIR and RDF format.

Following an example of a piece of information generated by Eliot Hub application for the device BT105.

```
{
  "resource": {
    "resourceType": "Observation",
    "id": "randomized",
    "contained": [
      {
        "resourceType": "Device",
        "id": "1",
        "identifier": [
          {
            "value": "900000000000006"
          }
        ]
      }
    ],
    "status": "final",
    "category": [
      {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/observation-category",
            "code": "vital-signs",
            "display": "Vital Signs"
          }
        ]
      }
    ],
    "code": {
      "coding": [
        {
          "system": "http://loinc.org",
          "code": "8867-4",
          "display": "Heart rate"
        }
      ],
      "text": "Heart rate"
    },
    "effectiveDateTime": "2020-09-04T09:36:48.904234877Z",
    "device": {
      "reference": "#1"
    },
    "component": [
      {
        "code": {
          "coding": [
            {
              "system": "http://loinc.org",
              "code": "8867-4",
              "display": "Heart rate"
            }
          ],
          "text": "Heart rate"
        },
        "valueQuantity": {
          "value": 66,
          "unit": "bpm",
          "system": "http://unitsofmeasure.org",
          "code": "{Beats}/min"
        }
      }
    ]
  },
```


Figure 30 Example of data generated by Medisantè device BT105

Figure 31 shows the steps of the integration between the Eliot Hub application and Data Federation and Integration. In order to perform this integration, it is needed that the DFI is registered to Eliot Hub collect. After the registration whenever a new measure is generated by a Medisantè device, used from a patient of a specific pilot, such measure is shared with Eliot Hub intelligent connector that forwards it (PUSH modality) to DFI as FHIR Bundle in JSON format. DFI loads the implemented transformer for Medisantè application,

it transforms data to GK-FHIR Profile and it invokes the API of gk-fhir-server, belonging to specific pilot, to store arrived data and FHIR and RDF format.

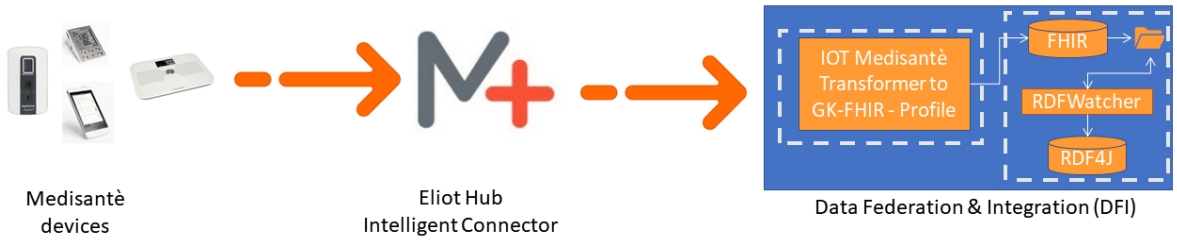


Figure 31 Eliot Hub and DFI integration

In order to stay updated on the operation outcome described above, Eliot Hub added status response codes for seeing server responses on its frontend application.

1.5.2 Interaction and integration with Activage gateway [UPDATED]

This section describes how Activage gateway and Data Federation & Integration interact in order to share data collected by the Samsung Health app, Smarthings, Home Monitoring (Robot), Health and Home Monitoring (IOT) and UK Questionnaire.

ACTIVAGE is a European Multi Centric Large Scale Pilot on Smart Living Environments [33]. The main objective is to build the first European IoT ecosystem across 9 Deployment Sites (DS) in seven European countries, reusing and scaling up underlying open and proprietary IoT platforms, technologies and standards, and integrating new interfaces needed to provide interoperability across these heterogeneous platforms, that will enable the deployment and operation at large scale of Active & Healthy Ageing IoT based solutions and services, supporting and extending the independent living of older adults in their living environments, and responding to real needs of caregivers, service providers and public authorities.

The project will deliver the ACTIVAGE IoT Ecosystem Suite (AIOTES), a set of Techniques, Tools and Methodologies for interoperability at different layers between heterogeneous IoT Platforms and an Open Framework for providing Semantic Interoperability of IoT Platforms for AHA, addressing trustworthiness, privacy, data protection and security. User-demand driven interoperable IoT-enabled Active & Healthy Ageing solutions will be deployed on top of the AIOTES in every DS, enhancing and scaling up existing services, for the promotion of independent living, the mitigation of frailty, and preservation of quality of life and autonomy. ACTIVAGE will assess the socio-economic impact, the benefits of IoT-based smart living environments in the quality of life and autonomy, and in the sustainability of the health and social care systems, demonstrating the seamless capacity of integration and interoperability of the IoT ecosystem, and validating new business, financial and organizational models for care delivery, ensuring the sustainability after the project end, and disseminating these results to a worldwide audience. The consortium comprises industries, research centres, SMEs, service providers, public authorities encompassing the whole value chain in every Deployment Site.

Samsung Health (originally S Health) is a free application developed by Samsung that serves to track various aspects of daily life contributing to wellbeing such as physical activity, diet, and sleep. Launched on 2 July 2012, the application was installed by default

only on some smartphones of the brand. It could also be downloaded from the Samsung Galaxy Store.

Since mid-September 2015, the application is available to all Android users. From 2 October 2017, the app is available for iPhones from iOS 9.0. The application is installed by default on some Samsung smartphone models and cannot be removed without root. It is possible to disable this application. The app changed its name from S Health to Samsung Health on 4 April 2017, when it released version 5.7.1.

The dashboard is the main display of the application. This is the main novelty introduced during the redesign of the application in April 2015 in version 4.1.0. The table shows on one page, a general overview of the most recent data saved. In addition, it provides direct access to each feature. Its composition and layout are customizable.

Some features are tracked by testing with phone sensors or phone accessories (Fitbit, Galaxy Active, Galaxy Fit, etc.) and some features are tracked by user input. (food/calories, weight, water amount, etc.).

Even if the Samsung Health App is able to collect a wide range of data, for GateKeeper projects only a subset of data type are collected and share with the platform. In detail the acquired data are:

- Blood Glucose
- Blood Pressure
- Caffeine Intake
- Floors Climbed
- Heart Rate
- Oxygen Saturation
- Sleep
- Sleep Stage
- Step Count
- Exercise
- Water Intake
- Weight
- Height
- Step Daily Trend

Some of these data are tracked by Galaxy Wearable App running on phone accessories (e.g. Samsung Watch) and other ones are tracked by user input. Data tracked by phone accessories are blood glucose, blood pressure, floors climbed, heart rate, oxygen saturation, sleep, sleep stage, step count, exercise, and step. Data tracked user input are caffeine intake, water intake, weight, and height.

Figure 32 shows in which way data acquired by Activage cloud platform are send to Data Federation & Integration invoking the provided southbound API. All interactions are performed using the PUSH modality, i.e. Activage will invoke the southbound APIs provided by DFI.

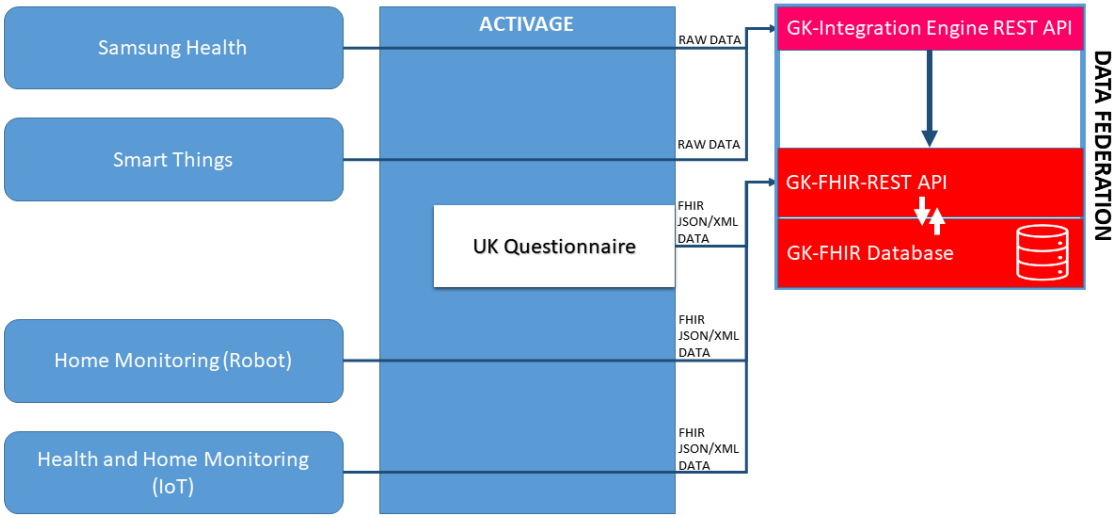


Figure 32 Interaction flows between Activage and Data Federation & Integration

Data coming from Samsung Health and SmartThings are acquired by Activage web could platform which forwards them to DFI framework invoking the southbound IOT (module integration engine) API in PUSH modality with an interval of 1 hour. Activage sends such data using its proprietary data model format. In order to allow the integration and the processing of such data in WP5 according to GK-FHIR format DFI includes a routine that transforms these raw data, coming from the Activage, to HL7 FHIR and RDF format (according the defined GK-FHIR-Profile) and stores them to the relative repositories. For this kind of transformation from Activage raw data to GK-FHIR format several rules have been define and described in the next chapter.

Activage gathers also data coming from several robots that monitor home and other IOT devices for the health. Such data are sent to Data Federation & Integration using directly in FHIR Server using the northbound APIs since data are already compliant with GK-FHIR profile so no conversion is needed. Of course, acquired data are both persisted in FHIR Server and at the same time transformed in RDF format and persisted in the internal graph data DFI.

Finally, Activage contains an internal module, called UK Questionnaire, that is able to make several questions to the users and collect their responses. This information are sent to DFI using the northbound FHIR API since such data will be shared already in GK-FHIR representation and the only conversion made by DFI is from GK-FHIR json to GK-FHIR RDF format and persisted in the internal graph data DFI.

All data coming from Activage can be retrieved from the work package 5 by means the northbound APIs in GK-FHIR and FHIR format.

1.5.3 Interaction and integration with Casa Sollievo della Sofferenza (CSS) Puglia [UPDATED]

CSS's EHR data source it is worth to point out as several internal (HIS) systems could be involved in principles (e.g. RIS/PACS, UMS, ENDOSCOPY etc.). It includes an intermediary middleware (Mirth) that collects and sends (PUSH) the data to the Data Federation. The

input data format is expected to be a custom model (JSON format) so that the Data Federation is mainly involved to convert the structure to the specific GK-FHIR profile and to redirect (ROUTE) the data to the Puglia FHIR Server for the JSON FHIR representation and RDF4J for RDF format.

Every day, at midnight, CSS invokes the southbound Puglia DFI API provided by Gk-Integration-Engine to send patient's data gathered through the whole day, The interaction modality is PUSH because CCS invokes the API provide by DFI.

Since DFI is hosted on OKD platform that works behind a proxy, the interaction between CSS and DFI happens with a vpn site-to-site connection. Figure 33 shows the sequence diagram of the integration.

When CSS has to share data with GateKeeper platform and more in details with DFI framework, it makes a to GTA for the login to the platform (GTA is the Gatekeeper component that manage the authentication/authorization processes); if the authentication is successful, CSS can invoke the southbound API provides by DFI system and more in detail by the GK-Integration-Engine component, of course, using the PUSH modality.

Once GK-Integration-Engine received data from CSS, it can invoke an internal very complex routine that takes input (i) data sent by CSS, (ii) the list of GK-FHIR profiles defined by task 3.5 and (iii) the set of custom conversion rules for each type of data processed for the CSS Puglia pilot in order to convert incoming data to GK-FHIR based on defined GK-FHIR Profiles running the defined conversion rules. The output is a FHIR Bundle of type transaction that contains the list of created FHIR Resource to be sent to FHIR Server. As soon as this bundle is ready it can be sent to FHIR Server component, by GK-Integration-Engine, invoking a POST operation and passing in the body of the request the created Bundle in JSON format.

When the POST request with the Bundle, containing the list of FHIR resources, arrives to FHIR Server, there is a specific internal function that scrolls the list of resources and save them into database. Then FHIR Server returns to gk-Integration-engine a OutocomeResponse containing the output of all operation performed for the request. Finally, the gk-integration-engine returns to the CSS an '*http ok*' message if the operation is successful otherwise an '*http error*' message if some issues happened in this process.

FHIR Server contains an interceptor that allows to the RDF Watcher to retrieve the persisted resource in JSON formal. As soon as the resource is retrieved, RDF Watcher invokes an internal routine that convert it to RDF format and invoke the RDF post API in order to persist it into RDF4J Server. This process is built to run in background avoiding blocking the APIs in the FHIR Server to retrieve and save resources.

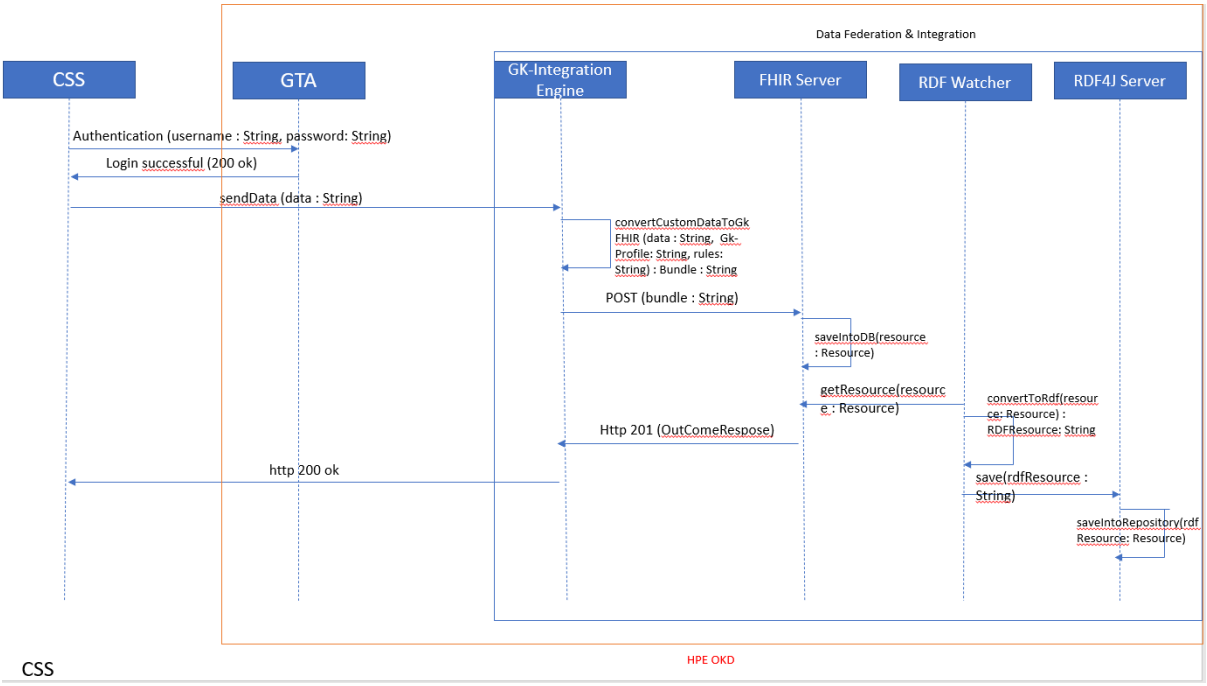


Figure 33 Sequence Diagram interaction between CSS and Data Federation

Great effort has been spent to realize this integration by several technical phone calls to address all the technical problems/bugs raised during the definition of the data format and the type of interaction above all when the deployment of DFI has been moved from the ENG Server to the OKD platform.

1.5.4 Interaction and integration with HealthCloudProxy (HCP) [NEW]

HealthCloudProxy (HCP) is a web application and also a series of REST APIs able to retrieve and gather data from different health cloud base platforms such as Google Fit, Fitbit, iHealth and Biobeat. With this component it is possible to obtain different kind of data, on the fly, belonging to one or more patients registered into the system. Obviously, the involved patients in the data retrieving process have to provide their authorisation to the data retrieval operations. In addition to provide data in a proprietary format based on the health cloud origin, the HCP components can also provide the data in a format compliant to the Open mHealth standard [34]. In this second scenario, all the data will be transported on the common data model and this is a great advantage for every service/application that needs these data.

The interaction between HCP and DFI is performed by means the southbound APIs provided by the internal component gk-integration-engine using the PULL modality.

Using the PULL modality it is not required a site-to-site connection between DFI and HCP since the request starts from a component hosted inside OKD platform.

Figure 34 shows the sequence diagram of the interaction and integration between HCP and DFI.

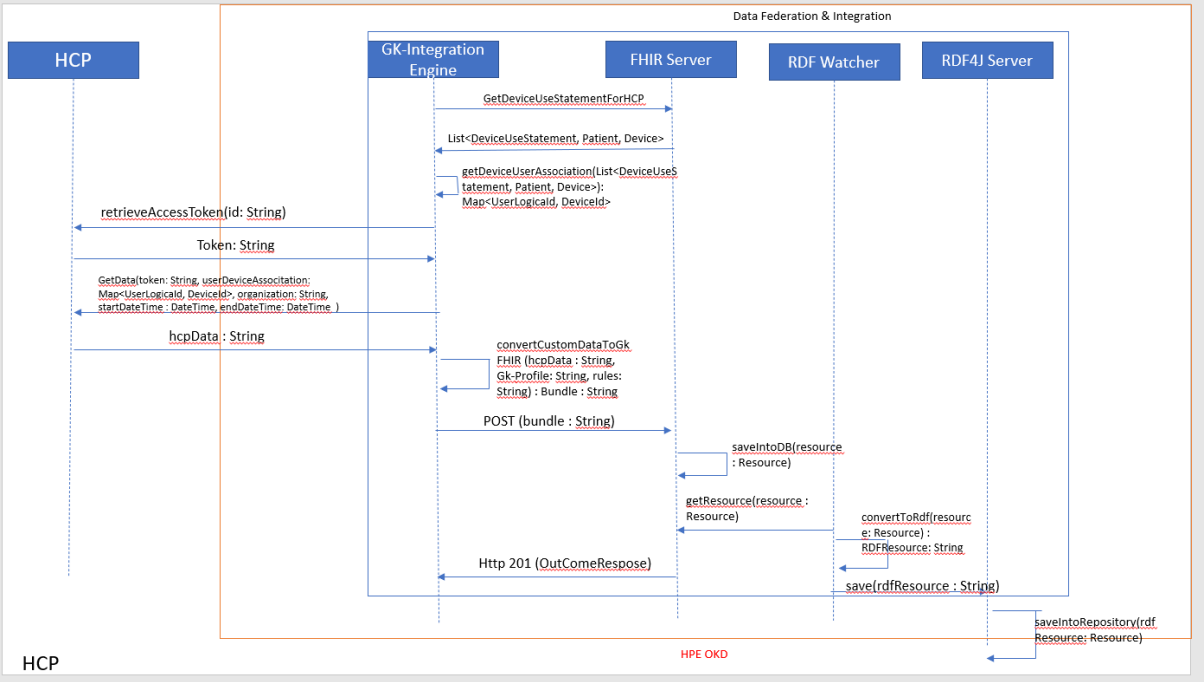


Figure 34 Sequence Diagram interaction between HCP and Data Federation

DFI contains a cron job that starts every day at 3:00 AM to retrieve data from HCP and persist them into FHIR and RDF Server. Such data are those collected the day before (from 00:01 AM to 00:00 PM) and belonging to all patients who use HCP and who are enlisted within the Gatekeeper project.

At 3:00 DFI makes a request to FHIR Server to retrieve all FHIR resources DeviceUseStatement, Patient and Device associated to the HCP. Then, FHIR Server searches the Device resources relative to the HCP and returns to DFI the list of Device resources and the DeviceUseStatement and Patient resources linked to the first ones. This information is needed due to the lack of a direct link between the Device and Patient resources within the FHIR Server as the association is maintained by DeviceUseStament resource as shown in Figure 35.

Figure 35 Link between Patient and Device by means DeviceUseStament according to FHIR model

Thanks to this information DFI can build a Map containing the association between the logical id of the device and the logical id of the patient. Subsequently, the gk-integration-engine makes a request to HCP to retrieve the authentication token passing a specific id that represents the id associated to DFI when this one is registered to HCP. Now, gk-integration-engine has all information to make a request to HCP to retrieve HCP data, in fact it invokes the POST method passing as input:

- Token returned by HCP;
- Map containing the association between user logical id and device logical id;
- Start datetime;

- Name of the pilot/organization;
- End date time;

HCP will return all data based on the received input in a json format using a custom model representation.

Once GK-Integration-Engine received data from HCP, it can invoke an internal very complex routine that takes input (i) data sent by HCP, (ii) the list of GK-FHIR profiles defined by task 3.5 and (iii) the set of conversion rules written ad hoc for each type of data processed for the HCP gateway in order to convert incoming data to GK-FHIR based on defined GK-FHIR profiles running the defined conversion rules. The output is a FHIR Bundle of type transaction that contains the list of created FHIR Resource to be sent to FHR Server. As soon as this bundle is ready it can be sent to FHIR Server component, by GK-Integration-Engine, invoking a POST operation and passing in the body of the request the created Bundle in JSON format.

When the POST request with the Bundle, containing the list of FHIR resources, arrives to FHIR Server, there is a specific internal function that scrolls the list of resources and save them into database. Then, FHIR Server returns to gk-Integration-engine a OutocomeResponse containing the output of all operations performed for the request. Finally, the gk-integration-engine returns to the HCP an http ok message if the operation is successful otherwise an http error message if some problems happened in this process.

FHIR Server contains an interceptor that allows to the RDF Watcher to retrieve the persisted resource in JSON formal. As soon as the resource is retrieved, RDF Watcher invokes an internal routine that convert it to RDF format and invoke the RDF post API in order to persist it into RDF4J Server. This process is built to work in background avoiding blocking the APIs in FHIR Server to retrieve and save resources.

Great effort was spent to realize this integration by means several technical phone calls to address all technical problems/bugs raised during the definition of the data model for the association between device and patient.

1.5.5 Interaction and integration with Aragon (SALUD) Application [NEW]

SALUD is an EHR data source that collects and groups data coming from two components: “LC Patient FROM Collection & Health education” and “MC/HC Telemonitoring APP”. The first one is used by patients aiming to manage information about their health education while the second one is a gateway running on smartphone, that retrieves some data coming from sensors and devices and forwards them to SALUD web-app.

Once data have been persisted in SALUD web-app they can be sent to DFI invoking the southbound APIs with the PUSH modality as shown in Figure 36.

DFI is hosted on OKD platform that works behind a proxy so the interaction between SALUD and DFI happens with a vpn site-to-site connection. When SALUD has to share data with GateKeeper platform and more in details with DFI framework, it makes a login request to the GTA (it is the Gatekeeper component that manage the authentication/authorization processes); if the authentication is successful, SALUD can invoke the southbound API provides by DFI system and more in detail by the GK-Integration-Engine component, of course, using the PUSH modality.

Once GK-Integration-Engine received data, it can invoke an internal very complex routine that takes input (i) data sent by SALUD, (ii) the list of GK-FHIR profiles defined by task 3.5

and (iii) the set of conversion rules written ad hoc for each type of data processed for the SALUD in order to convert incoming data to GK-FHIR based on defined GK-FHIR profiles running the defined conversion rules. Rules are based on the Aragon data model collected in the task 3.5 and the strong collaboration with task 3.5 for the definition of FHIR profile. The output is a FHIR Bundle of type transaction that contains the list of created FHIR Resources to be sent to FHR Server. As soon as this bundle is ready, it can be sent to FHIR Server component, by GK-Integration-Engine, invoking a POST operation and passing in the body of the request the generated Bundle in JSON format.

When the POST request with the Bundle, containing the list of FHIR resources, arrives to FHIR Server, there is a specific internal function that scrolls the list of resources and saves them into database compliant to FHIR standard. Then FHIR Server returns to gk-Integration-engine a OutocomeResponse containing the output of all operations performed in the request. Finally, the gk-integration-engine returns back to SALUD an http ok message if the operation is successful otherwise an http error message if some problems happened in this process.

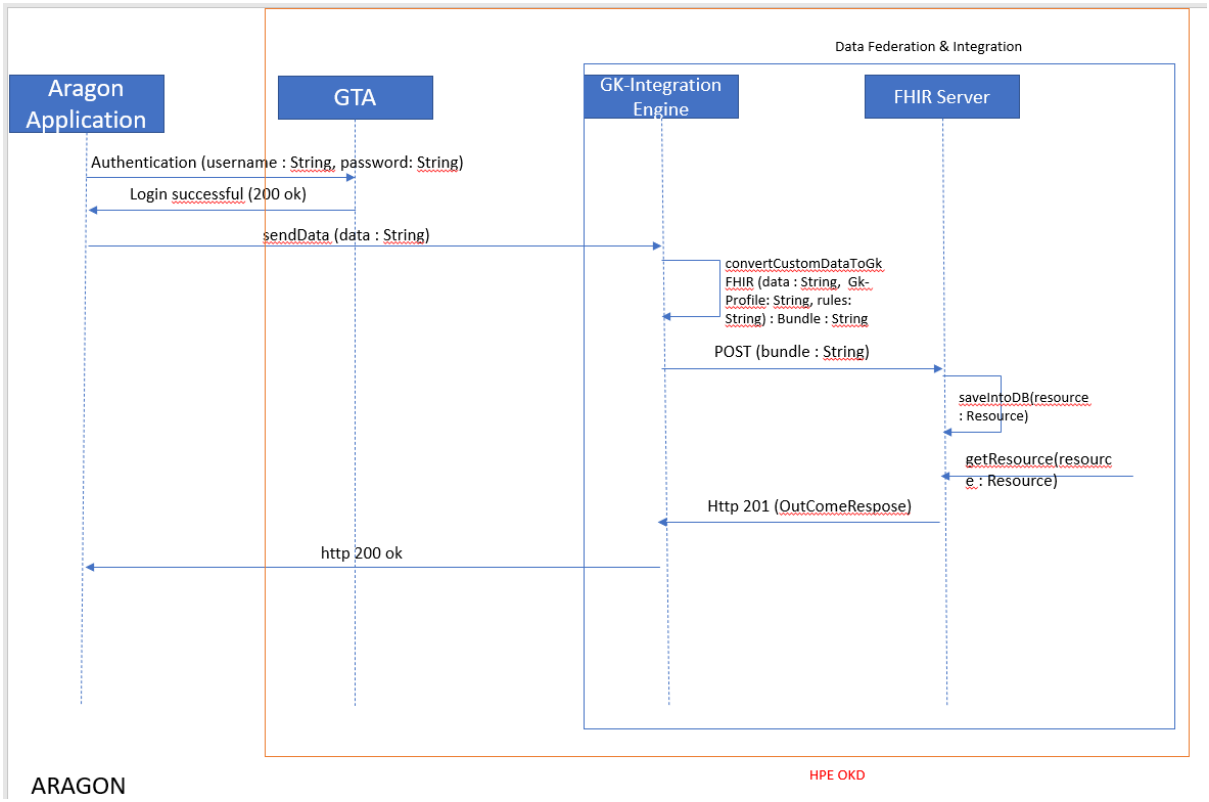


Figure 36 Sequence Diagram interaction between Aragon EHR Application and Data Federation

1.5.6 Interaction and integration with Poland Application [NEW]

Data gathered by Poland application can be sent to DFI invoking the southbound APIs with the PUSH modality as shown in Figure 37.

DFI is hosted on OKD platform that works behind a proxy so that the interaction between Poland application and DFI happens with a vpn site-to-site connection. When Poland has

to share data with GateKeeper platform, and in particular with DFI framework, it makes a login request to the GTA (it is the Gatekeeper component that manage the authentication/authorization processes); if the authentication is successful, Poland application can invoke the southbound API provided by the DFI system and more in detail by the GK-Integration-Engine component using the PUSH modality.

Once GK-Integration-Engine received data, it can invoke an internal very complex routine that takes input (i) data sent by Poland app, (ii) the list of GK-FHIR profiles defined by task 3.5 and (iii) the set of conversion rules written ad hoc for each type of data processed for the Poland app in order to convert incoming data to GK-FHIR based on defined GK-FHIR profiles running the defined conversion rules. Rules are based on the Poland data model collected in the task 3.5 and the strong collaboration with task 3.5 for the definition of FHIR profile (s. A.1.3). The output is a FHIR Bundle of type transaction that contains the list of the created FHIR Resources to be sent to FHIR Server. As soon as this bundle is ready, it can be sent to the FHIR Server component, by GK-Integration-Engine, invoking a POST operation and passing in the body of the request the generated Bundle in JSON format.

When the POST request with the Bundle, containing the list of FHIR resources, arrives to FHIR Server, there is a specific internal function that scrolls the list of resources and saves them into database compliant to FHIR standard. Then FHIR Server returns to gk-Integration-engine an OutocomeResponse containing the output of all operations performed in the request. Finally, the gk-integration-engine returns back to Poland application an http ok message if the operation is successful otherwise an http error message if some problems happened in this process.

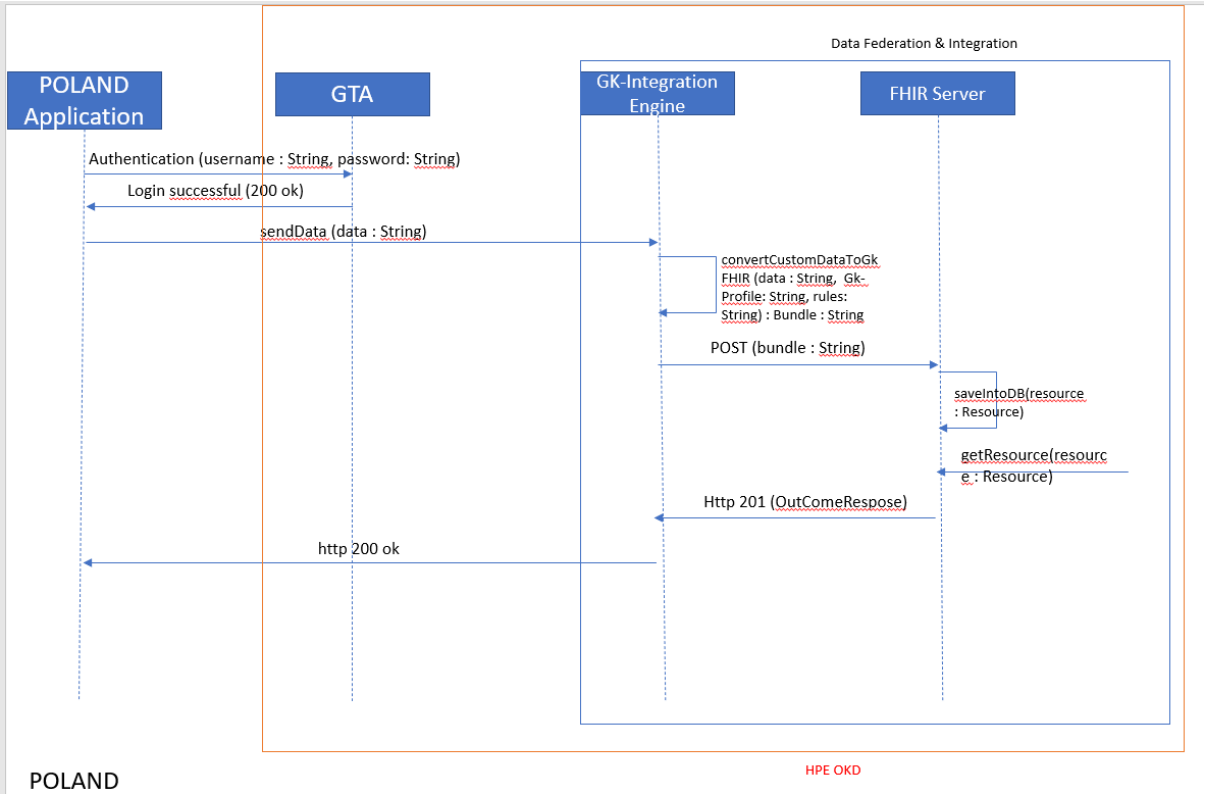


Figure 37 Sequence Diagram interaction between Poland EHR Application and Data Federation

1.5.7 Interaction with the OpenCaller [NEW]

During the last period of the GK project, it has been realised an Open Call for new stakeholders in order to:

- develop/implement technology to complement the current one available in GATEKEEPER
- develop innovative services based on AI available in GATEKEEPER
- evaluate and validate the functionalities offered in GATEKEEPER through new and existing use cases

Thus, the Open Caller (OC) is an integration project on the top of the platform driven by open caller developers. So, an OC consumes platform assets in an harmonized way for a specific scope. In some cases it should also provide new assets that further populate the platform (D2.6)

This paragraph presents an example of interaction with the OpenCaller ENVIRA.

First of all, ENVIRA has sent to ENG its data model that is related to environmental monitoring measures collected through Nanoenvi IAQs (environmental devices). Such devices work using a push model gathering measures from every sensor and composing a JSON message (an extract in Figure 38) which is sent using MQTT protocol. In the Appendix A.6 is reported the overall JSON message.


```
{
    "device_info": {
        "uuid": "30afdf79-b363-40d4-ac99-70db778c744b",
        "fw_ver": "V1.4.2"
    },
    "measures": [
        {
            "n": "co2",
            "u": "ppm",
            "v": 673.000
        },
        {
            "n": "voc",
            "u": "ppm",
            "v": 275.000e-3
        },
        {
            "n": "co",
            "u": "ppm",
            "v": 0.206
        },
```


Figure 38 ENVIRA MODEL: an extract of the message

Measurements inside the Json message (Figure 38) follow SenML (Sensor Measurement Lists) format [35] because of the nature of such data that are relative to simple sensor measurements and device parameters. Thus,

- "device_info", represents the UUID of the device and its firmware version
- "measures", represents an array that collects all the environmental data with its name ("n"), its unit of measure ("u") and its value ("v").

To continue, messages from Nanoenvi IAQs are received by a MQTT broker and then redirected to a Nanoenvi IAQ Hub that exposes REST API to request values from sensors' devices. Then, such data need a transformation in order to adapt them to DFI format (as reported in A.1.6) and to be pushed to DFI. For this reason, a new converter has been implemented, as described in A.5.
Currently, this OC is not assigned to a specific final user (i.e. clinics or similar) so it has been codified as 'genericpilot'.

When ENVIRA has to share its data with the DFI framework, it makes a login request to the GTA; if the authentication is successful, ENVIRA application can invoke the southbound API provided by the GK-Integration-Engine component using the PUSH modality.

Once the GK-Integration-Engine received data, it can invoke an internal routine that takes input (i) data sent by ENVIRA, (ii) the list of GK-FHIR profiles defined by task 3.5 and (iii) the set of conversion rules written ad hoc for each type of data processed for the ENVIRA in order to convert incoming data to GK-FHIR based on defined GK-FHIR profiles running the defined conversion rules. Rules are based on the ENVIRA data model collected when the Open Caller has been involved in the project and the strong collaboration between ENG and HL7 for the definition of FHIR profile (for further detail see A.1.3). The output is a FHIR Bundle of type transaction that contains the list of the created FHIR Resources to be sent to FHIR Server. As soon as this bundle is ready, it can be sent to the FHIR Server component, by GK-Integration-Engine, invoking a POST operation and passing in the body of the request the generated Bundle in JSON format.

When the POST request with the Bundle, containing the list of FHIR resources, arrives to FHIR Server, there is a specific internal function that scrolls the list of resources and saves them into database compliant to FHIR standard. Then FHIR Server returns to gk-Integration-engine an OutocomeResponse containing the output of all operations performed in the request. Finally, the gk-integration-engine returns back to ENVIRA an http ok message if the operation is successful otherwise an http error message if some issues have been detected in this process. Several input tests have been performed by ENG to verify this conversion procedure.
In Table 11 and Table 12 are reported two examples of the conversion procedure related to ENVIRA device and the Observation of the CO measure. Further details on codes and information carried by the ENVIRA JSON are reported in the A.1.6.

Regarding to ENVIRA device, only its uuid has been mapped into GK-FHIR as showed here below:

Table 11 ENVIRA Device FHIR conversion: an example

ENVIRA Device	FHIR Device
uuid	identifier

ENVIRA Observation has been mapped as reported in the following example:

Table 12 ENVIRA Observation FHIR conversion: an example

ENVIRA Observation (CO)	FHIR Observation
	category
n	code
u	value.unit
v	value.value

About the 'category', it is not present within the ENVIRA JSON data model, but it is included into FHIR mapping rule in order to keep the information within the GK-FHIR Server that such data is related to the environmental monitoring (coded as liv-environment).

2 DATA FEDERATION AND INTEGRATION V2: IMPLEMENTATION DETAILS

2.1 GK-Integration Engine [UPDATED]

2.1.1 Apache Camel [UPDATED]

As described in the section above, the Data Federation takes care "to route" the data (properly converted to the GK-FHIR Profile) to the *pilot specific* "data node" (i.e. dedicated FHIR Server and RDF4J Server) hosted in dedicated cluster - see Section 3 for details. Moreover, it is expected to also route such data toward other external systems (e.g. Big Data infrastructure). These requirements convinced us to adopt the Apache Camel framework that aims to make systems integration easier relying on message routing features. The description of the Camel framework architecture and core components, have been already described in D4.4(M15) and, then, not reported here for convenience. Here below are briefly re-called, instead, the concrete technological stack adopted in GK (relying on Camel) and the main logic implemented within the **DataProcessor.**

The Apache Camel framework has been used as development framework of the GK integration engine along with the spring framework. In the figure below the real technological stack adopted.

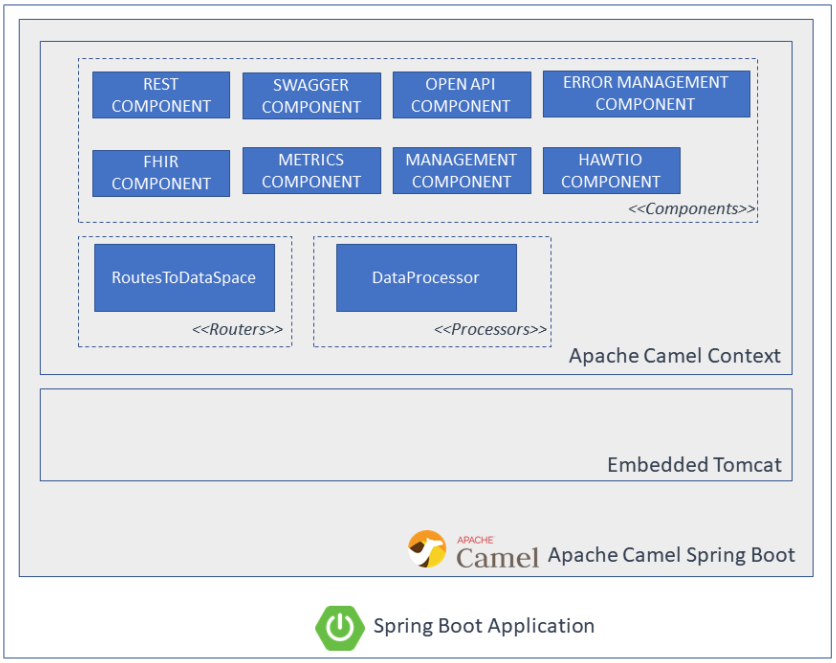


Figure 39 GK Integration Engine: implementation stack

For this purpose, a specific component for exposing REST API has been integrated (see **REST COMPONENT** in the figure). It offers a REST styled DSL which can be used with Java or XML. The intention is to allow end users to define REST services using a REST style with verbs such as get, post, delete etc. To use the Rest DSL in Java then it was sufficient to do as with regular Camel routes by extending the RouteBuilder (see **RoutesToDataSpace** java class in the figure) and define the routes in the configure method.

```
onException(InvalidRequestException.class)
    .handled(true)
    // use HTTP status 400 when we had a FHIR Compliance error
    .setHeader(Exchange.HTTP_RESPONSE_CODE, constant(400))
    .setBody(simple("Bad Request. Probably your Payload is not compliant with the expected structure.\n"));

restConfiguration()
  .apiComponent("openapi")
  .port(env.getProperty("server.port"))
  .skipBindingOnErrorCode(false)
  .contextPath("/gkie")
  .enableCORS(true)
  .apiContextPath("/api-doc")
  .apiProperty("api.title", "Data Sources Integration - API")
  .apiProperty("api.version", "v1")
  .apiProperty("openapi.version", "3.0")
  .apiProperty("cors", "true")
  .apiContextRouteId("doc-api")
  .component("servlet")
  .dataFormatProperty("prettyPrint", "true")
  //.scheme("https")
  .bindingMode(RestBindingMode.off);

// EHR
 rest("/EHR")
  .description("Interface enabling remote pilot EHR to send data. If a FHIR processor has been preliminary registered for that pilot, data will be converted
  .post("/data/{pilot}")
  .id("ehr")
  .security("bearerAuth")
  //.param().name("Authorization").type(RestParamType.header).defaultValue("Bearer ").description("Bearer token (Add the JWT after Bearer)").endParam()
  .produces("text/plain")
  .consumes("application/json, application/xml")
  .type(IDataBundle.class)
  .responseMessage().code(200).message("Operation Successful").endResponseMessage()
  .to("direct:remoteService");

// IOT
 rest("/IOT")
  .description("Interface enabling remote IOT devices (or connector services) to send data. If a FHIR processor has been preliminary registered for that devi
  .post("/data/{pilot}/{sensorID}")
  .id("iot")
  .security("bearerAuth")
  //.param().name("Authorization").type(RestParamType.header).defaultValue("Bearer ").description("Bearer token (Add the JWT after Bearer)").endParam()
  .param().name("body").type(RestParamType.body).dataType("string").required(false).endParam()
  .param().name("datafile").type(RestParamType.formData).dataType("file").required(false).endParam()
  .produces("text/plain")
  .consumes("multipart/form-data, application/xml, application/json")
  .type(IDataBundle.class)
  .responseMessage().code(200).message("Operation Successful").endResponseMessage()
  .to("direct:remoteService");
```

Figure 40 REST interface using Camel

In order to expose such REST interface trough swagger interface, the appropriate Camel component (see **SWAGGER COMPONENT** in the figure) has been integrated.

One of the core Camel components is **DataProcessor** that implements the **Processor**[9] interface used to implement consumers of message exchanges or to implement a **Message Translator**[10].

The Processor interface requires to implement a process method that accepts an **Exchange** class parameter containing all the information needed for the route. Once a Processor is developed then it can be easily used inside a route by the declaring of the bean in Spring or suing the **DSL** syntax.

Figure 41 shows the operating logic inside the **DataProcessor,** represented with a flow chart, for selecting the converter to be used, the output of the semantic model and the server where to send and store converted data (FHIR Server o RDF Server). The **DataProcessor** is used by the **RouteBuilder** class that is derived from to create routing rules using the DSL. Instances of **RouteBuilder** are then added to the **CamelContext**.

[9] https://camel.apache.org/manual/latest/processor.html

[10] https://camel.apache.org/components/latest/eips/message-translator.html

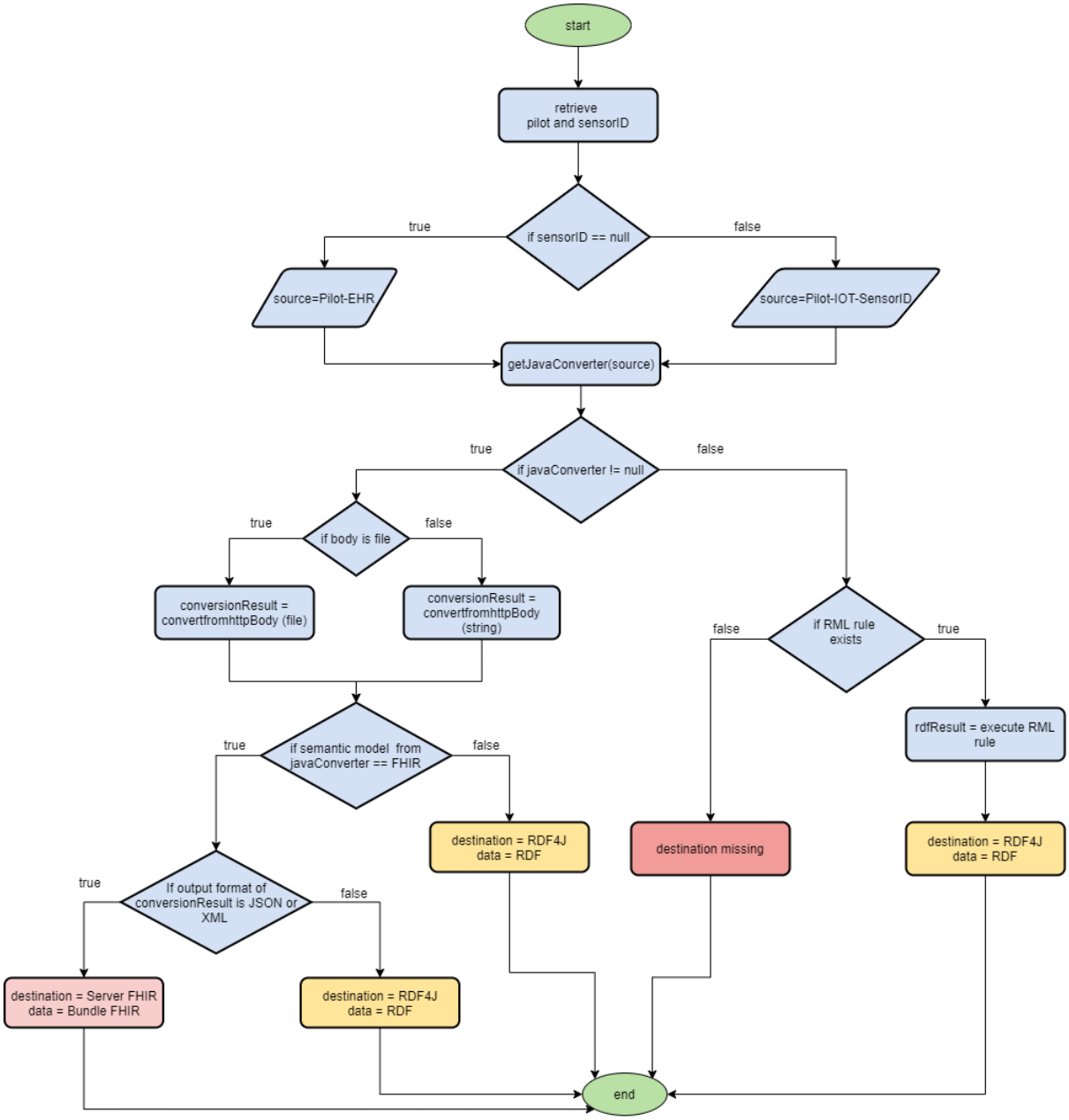


Figure 41 DataProcessor flowchart

When the processor is invoked, it retrieves the name of the pilot and the sensorID passed as path parameter in the REST request; this information is needed to understand which of the two RESP APIs has been invoked, if EHR or IOT. To perform this operation, it is checked if the sensorID is null (or empty), if true, the source consists of the pilot name and the EHR otherwise the source is an IOT with pilot name and the id of the sensor. Information added inside the source are needed to select the relative converter from raw data to FHIR/RDF. Afterwards it is invoked a method, named getJavaConverter that takes in input the source built in previous step that returns the corresponding Java converter. If the returned javaConverter is not null, then there is a java converter associated to that source otherwise it is checked if there exists e RML processor for the selected source.

If a Java converter exists, it is verified if the body of the request is a file or a string (with a JSON/XML representation). If the body is a file, it is invoked a method (convertfromhttpBody) that reads the content from the file otherwise the same method is invoke but it reads the content from a string variable. After it is verified if the output sematic model retrieved by JavaConverter is FHIR, if false the destination of the converted data is set to RDF4J and data are RDF, this mean that the converted data are send to RDF4J Server with a RDF format. If the semantic model is FHIR then it is verified if the output format is XML or JSON, if true the destination is set to FHIR Server and data is the FHIR Bundle, this means that converted data are in the FHIR Bundle and sent to the FHIR Server. If the output format is neither JSON nor XML then the destination is RDF4J Server and data are in RDF format. If the semantic model is not FHIR then the destination is RDF4J and data are in RDF format.

Returning to the step where it is checked if the javaconverter is not null (second rhombus), if false this mean that the conversion has been developed using the RML rule languages. For this reason, it is checked if exists a RML processor associated to the built source, if true the RML engine is executed with the associated rules and the destination is set to RDF4J with data in RDF format otherwise if it is not existing the RML processor associated with that source then the destination is missing, and data are not sent to any server.

Summarizing, the goal of the flow chart (described above) is to retrieve the type of the source for the specific pilot (EHR or IoT), the type of converter (JavaConverter or RML), the output format (FHIR or RDF) and the server to send converted data (FHIR Server of RDF4J Server).

Another component used in Apache Camel context is the **FHIR component** [4]: it integrates with the HAPI-FHIR library which is an open-source implementation of the FHIR (Fast Healthcare Interoperability Resources) specification in Java. It uses the URL format *fhir://endpoint-prefix/endpoint?[options], endpoint* prefix can be one of capabilities, create, delete, history, load-page, meta, operation, patch, read, search, transaction, update and validate. It is used in the **RouteBuilder** to invoke FHIR Server to store FHIR data.

2.1.2 Interface [UPDATED]

During the initial tests and interaction with pilot teams we identified the need to improve the management (and logging) of errors during data conversion steps. For this reason compared to the first version, the programmatic interface has been updated with the design and the development of the error management feature performed by the GK-Integration-Engine towards not supported or malformed data with respect to the data model declared by the pilot/OC. In case of a malformed or not supported received data from the caller, the GKIE responds with an HTTP 400 Bad Request message. Otherwise, if the server falls in error the caller receives an HTTP 500 Server error.

Data Federation & Integration exposes two southbound APIs to accept data coming from heterogeneous data sources registered in the platform, including personal clinical data source (EHR), social care data sources, wearable data sources, thus producing a HL7 FHIR and semantic repository.

These APIs aiming to accept heterogeneous data, coming from the several pilots' applications registered into the platform, to produce a repository where data can be retrieved from the northbound APIs described in the next section. Figure 42 shows the swagger of the two interfaces while Table 13 and

Table 14 provide their descriptions.

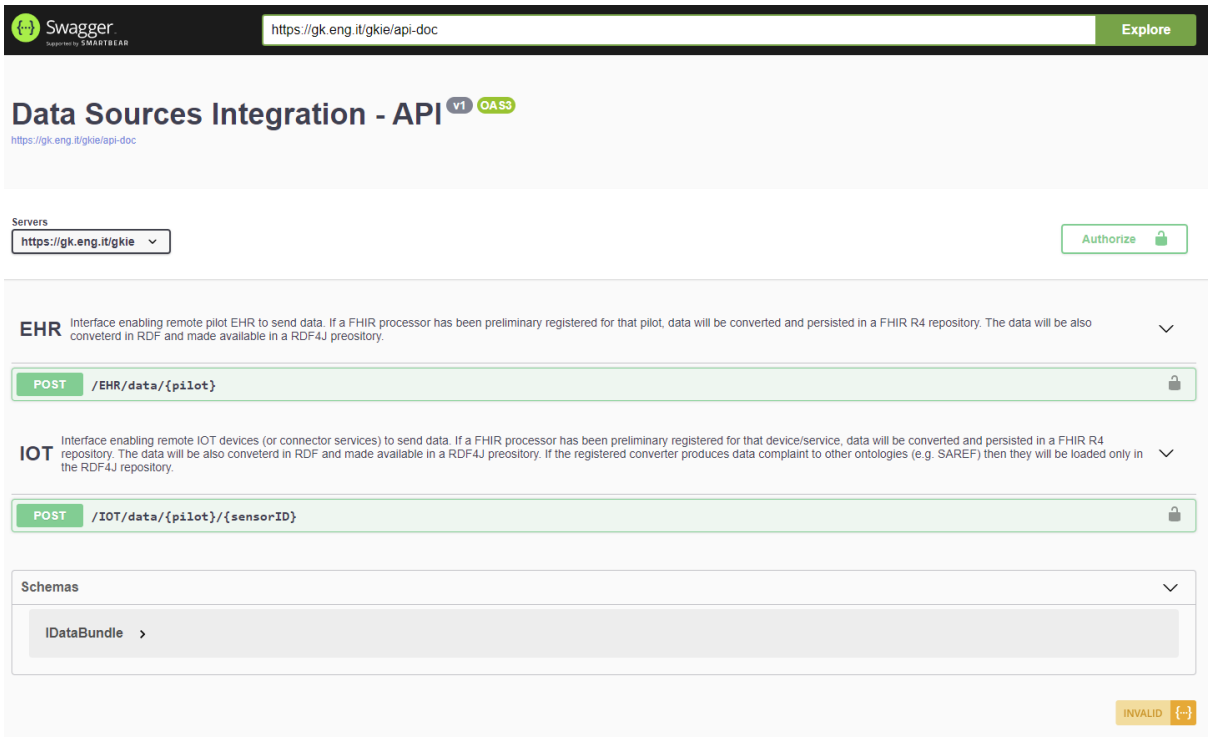


Figure 42 Swagger of gk-integration-engine

EHR Interface enabling remote pilot EHR to send data. If a FHIR processor has been preliminary registered for that pilot, data are converted and persisted in a FHIR R4 repository. These data are also converted in RDF and made available in a RDF4J repository.

The interface accepts two inputs: the name of the pilot and the JSON/XML representation of data to be stored in GK platform. The Pilot's name is passed in the URI pattern of the request and in order to understand to which pilot data belongs to.

The second input is data in JSON/XML format to be stored. The structure of this data must be the same of the FHIR processor that has been preliminary registered for the specific pilot. gk-integration-engine, based on the name of the pilot, select the corresponding FHIR processor that is applied on the data passed in the body of the request. The output of the interface is an HTTP 201 message if data are successful converted and persisted in FHIR repository or HTTP 500 if an error has occurred, this could happen for example when there is no FHIR processor registered to the pilot declared in the URI pattern of the FHIR Server is not available.

Table 13 EHR southbound API

ID Operazione	3.1.2.1
Signature	void saveEHRdata (String pilot, String data)

URI pattern	POST https://{host}:{port}/gkie/EHR/data/{pilot}
Input	• pilot (pilot name) • Json/XML representation of EHR data to be stored in Data Federation & Integration
Output	• HTTP 201 if data are successful persisted • HTTP 400 if the sent payload is not compliant with the expected structure • HTTP 500 if a server error has occurred

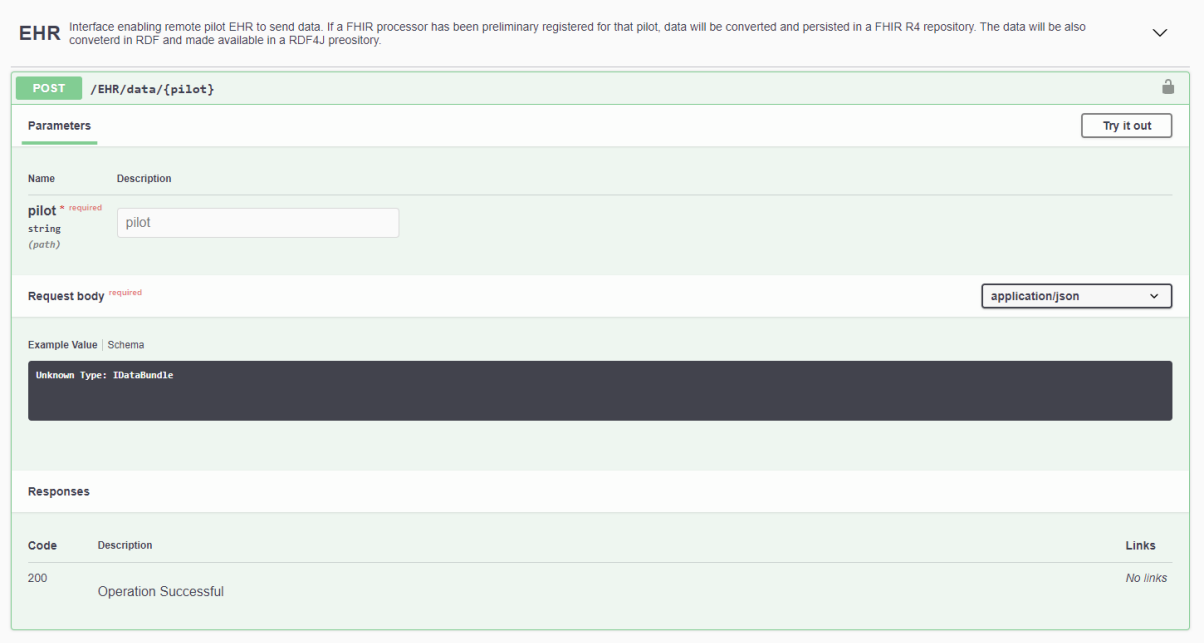


Figure 43 Swagger EHR southbound API

The IoT Interface enables remote IoT devices (or intelligent connector services) to send data. If a FHIR processor has been preliminary registered for that device/service, data will be converted and persisted in a FHIR R4 repository. The data will be also converted in RDF and made available in a RDF4J repository. If the registered converter produces data compliant to other ontologies (e.g. SAREF) then they will be loaded only in the RDF4J repository.

The interface accepts in input three parameters: name of the pilot, sensorID (or the name of the intelligent connector) and the JSON/XML representation of raw data as string or in a file.

The association between pilot and sensorID allows to select the FHIR processor, if preliminary registered, to convert data to FHIR format and RDF4j repository. If it is registered only a RML converter are compliant to other ontology and they will be stored only in RDF repository and not in FHIR.

The output of the interface is an HTTP 201 if data are successful converted and persisted in FHIR repository and HTTP 500 if an error has occurred, this could happen for example when there is no FHIR processor registered to the pilot declared in the URI pattern of the FHIR Server is not available.

Table 14 IoT southbound API

ID Operazione	3.1.2.1
Signature	void saveIOTdata (String pilot, String sensorID)
URI pattern	POST https://{host}:{port}/gkie/IOT/data/{pilot}/{sensorID}
Input	• pilot (pilot name) • sensorID (sensorID or name of the IOT collector that is sending data) • Json/XML representation of raw data to be stored in Data Federation & Integration • Or file containing JSON/XML representation of raw data to be store in Data Federation & Integration
Output	• HTTP 201 if data are successful persisted • HTTP 400 if the sent payload is not compliant with the expected structure • HTTP 500 if a server error has occurred

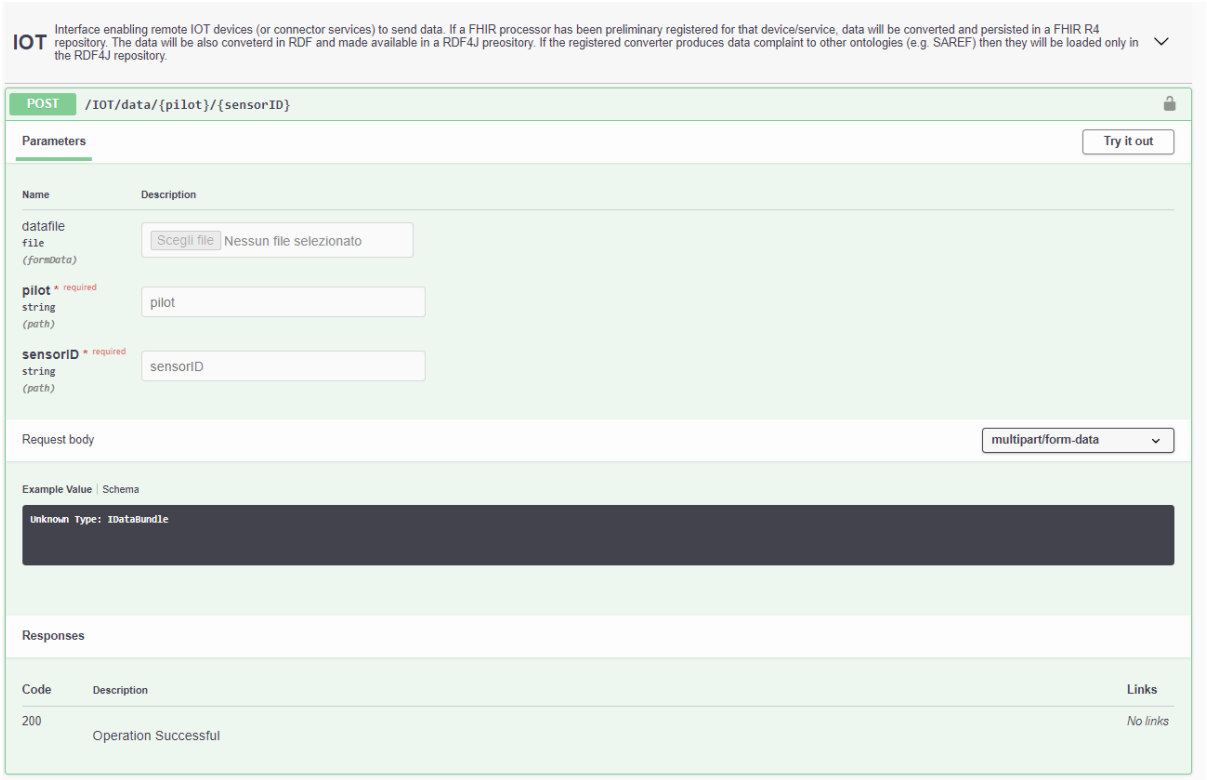


Figure 44 Swagger IOT southbound API

2.1.3 Monitoring Console [NEW]

As already documented in the first deliverable (D4.4) we adopted Apache Camel "to route" the data (properly converted to the GK-FHIR Profile) to the *pilot specific* "data node" (i.e. dedicated FHIR Server and RDF4J Server) hosted in dedicated namespace- see Section 3 for details and, moreover, to also route such data toward other external systems (e.g. Big Data infrastructure). The Camel architecture based on enterprise integration pattern simplified the work so that we confirmed our choice also for the next version of the DFI. However, one of the things we've noticed is the lack of visibility into what Camel is doing. In principle it would be possible to monitor things by looking at the incoming/processing/done queues but getting numbers on performance etc is hard and looking at JMX output is not very easy.

For this reason, we decided to integrate a dedicated management console that was conceived as a web application offering a dedicated monitoring view over Camel routes. It relies on an open-source layer [1] which is a lightweight and modular HTML5 web console with lots of plugins for managing Java applications. Once installed it can connect to any Java process which has a Jolokia agent installed and view its JMX MBeans and work with its various services such as Apache Camel, Apache ActiveMQ or fabric8.

The management console consists of 2 parts, the backend which is running in a Java Web Container the Jolokia gateway (JMX to JSON) and the front end containing the Angular, D3 and Javascript to do the rendering of the JSON responses in a very nice way. Depending on how it is expected to use Hawtio it is possible to run the backend using a java standalone jar, a servlet engine, application server or an OSGI container. A general overview of the management console now integrated along with the DF is shown in **Errore. L'origine riferimento non è stata trovata.**.

Some of its main features are listed below:

- Dynamic agent/service search. Discover function used to detect services which can be integrated with Hawtio, in our case the service is Camel (s. Figure 49).
- JMX Management. Powerful JMX management using Jolokia and on the fly refreshes when a new application is deployed.
- It provides system information, statistics, analysis and management about the Routes (Figure 47)
- How many times has the route worked (live reload)?
- Show Jvm parameters
- Heap objects and count
- How many errors were received?
- How many transactions were successful?

Here below we have reported some screenshots about these features.

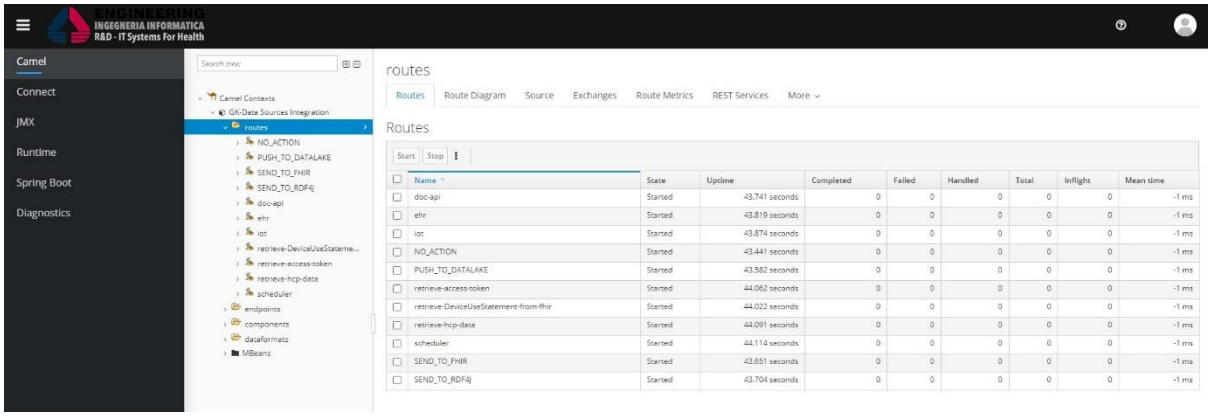


Figure 45 Camel existing Routes

All the routes are listed along with some meaningful metrics (e.g. *uptime* which is the overall time the route was up and running, *completed* which is the number of completed execution of the route, mean time which is the mean execution time for the specific route etc.)

We can analyse more in details every Route in Figure 45 with the route diagram panel Figure 46 that shows the details about the routes and also the dynamic evolution about the path of the information through the route components.

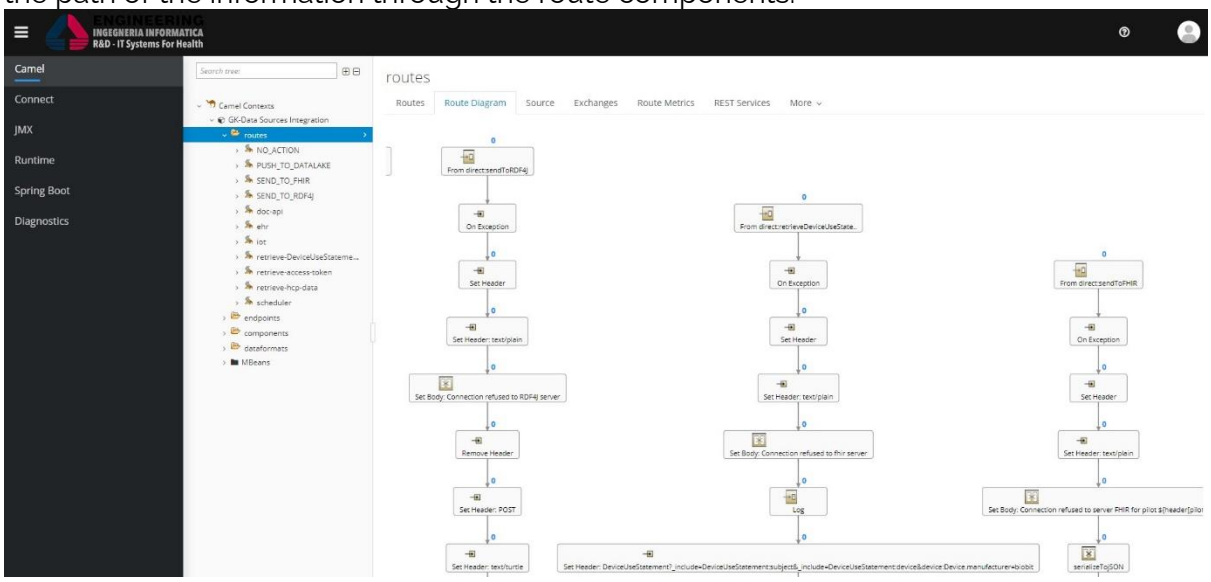


Figure 46 Route Diagram

The numbers within each box are the messages processed, and the numbers on the arrows (0 in the figure) are the number of messages that are in flight (currently being processed). The numbers update in real time.

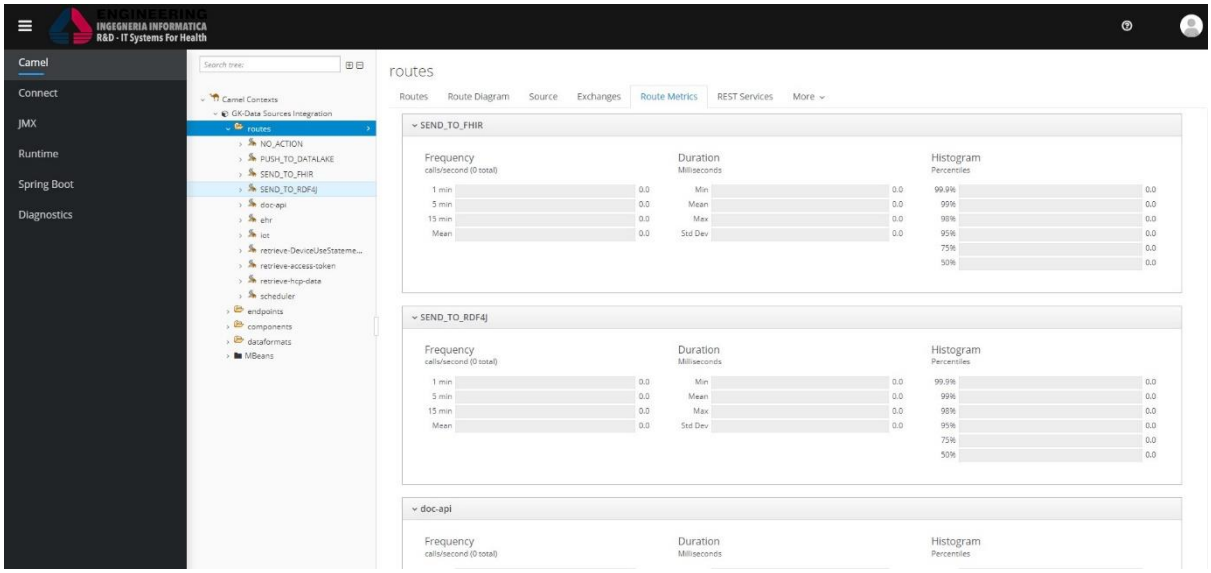


Figure 47 Route Metrics

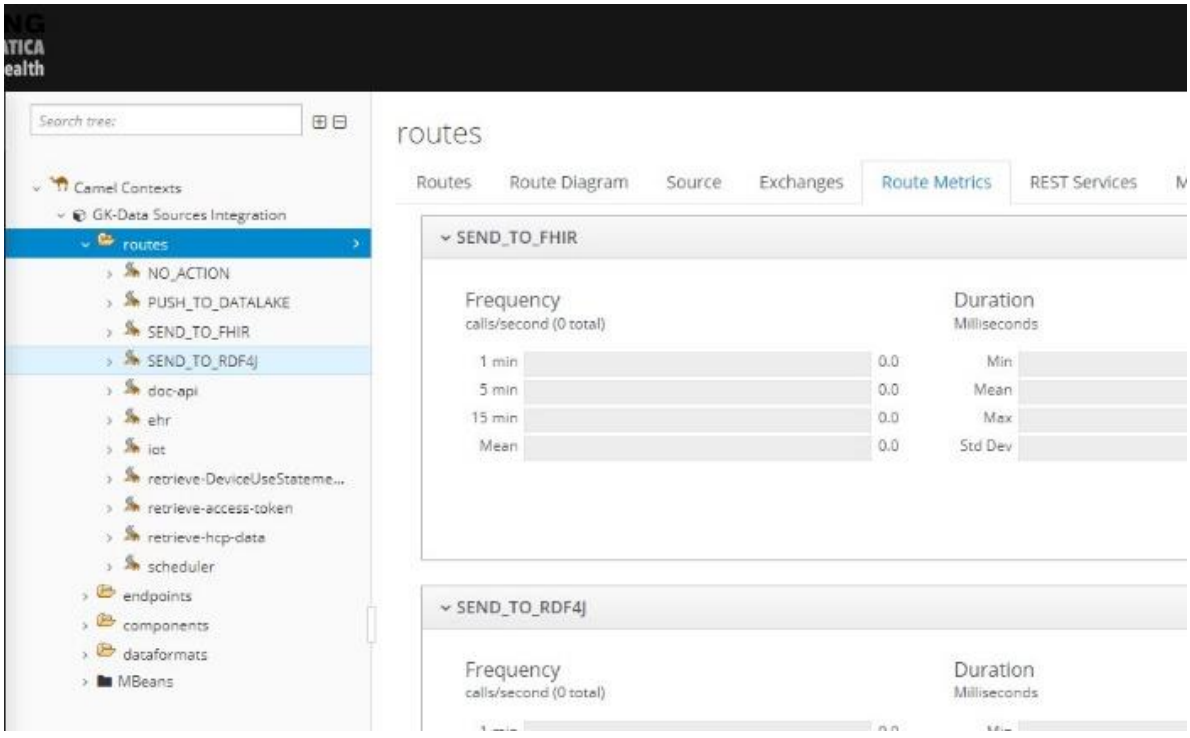


Figure 48 Details about metrics

The different metrics (e.g. completed, mean time, etc.) are made available not only within a table (tab "Routes") but also (tab "Route Metrics") in a graphical modality by means of dedicated widget

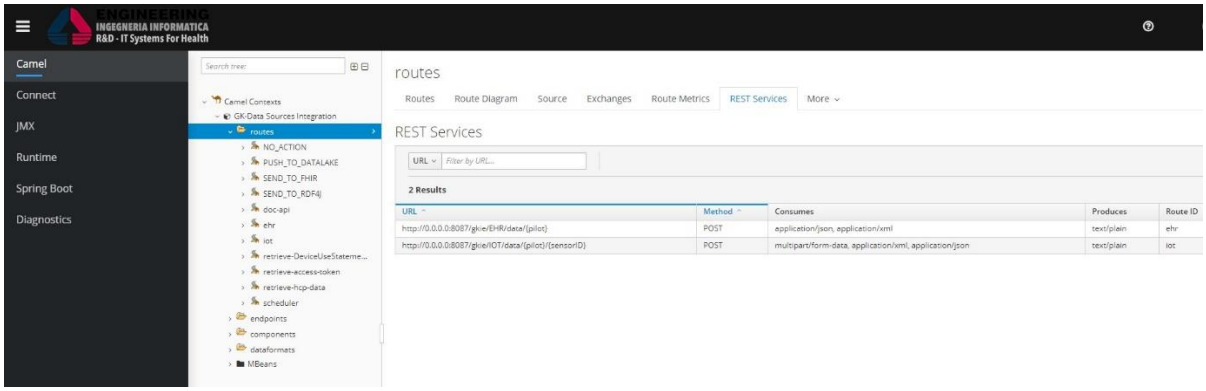


Figure 49 Exposed southbound APIs

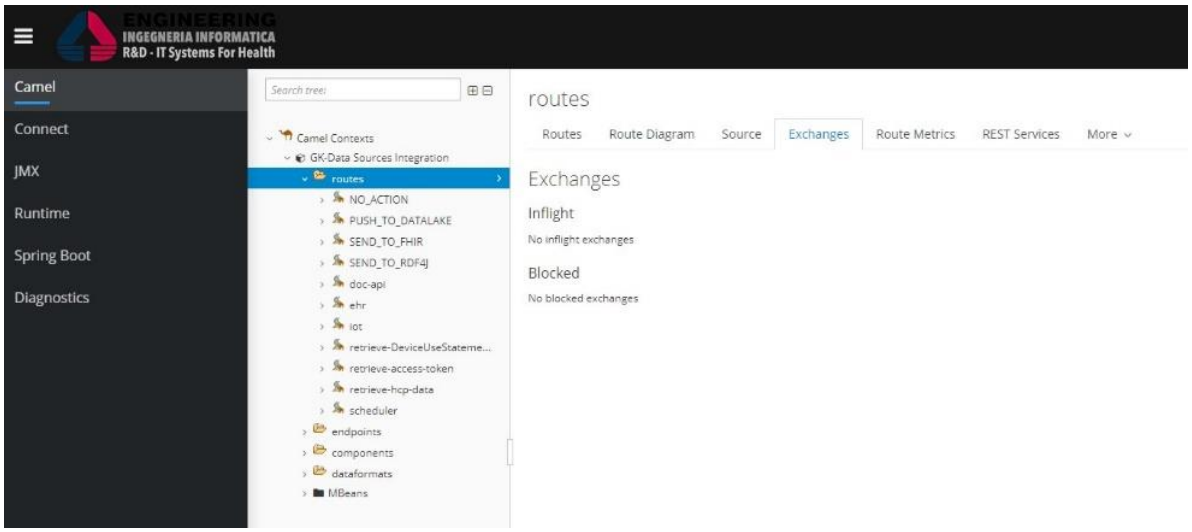


Figure 50 Exchanges

2.1.4 Data Converter [NEW]

2.1.4.1 Implementation [NEW]

Data Federation & Integration includes an engine able to convert from raw data to FHIR format, based on the FHIR Profiles (GK-FHIR Profile from now on) defined in the scope of the task 3.5. To implement this feature, the data models, provided by the server pilots, involved in the project, have been analysed together with the GK-FHIR Profile to define the rules that can be processed to achieve this type of transformation.

When invoked (through the southbound API), the engine takes as input the raw data and exploiting the transformation rules associated to the specific data source, produces as output the converted GK-FHIR data.

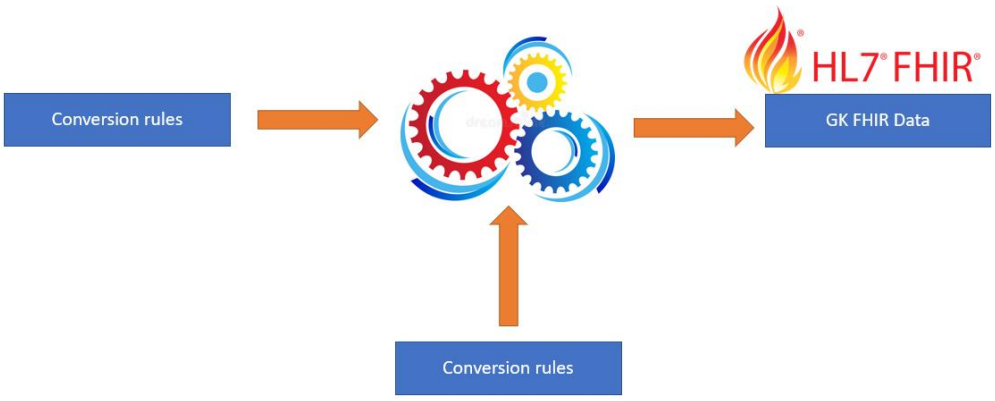


Figure 51 Engine to convert raw data to FHIR

1. The engine can perform the conversion over different type of data: Conversion from raw data to FHIR taking as input a specific third party data model (task 3.4) by applying the defined rules listed in the next section. These rules are processed through complex algorithms written in Java and Python.
2. Conversion from data already compliant to the FHIR standard to GK-FHIR Profile. It takes in input data already in FHIR but not compliant with the GK-FHIR Profiles (as defined in the task 3.5) so it adapts this data to GK-FHIR profiles.

2.1.4.2 Rules [NEW]

As mentioned above, this paragraph presents the rules implemented in order to make the data compliant with the GK-FHIR Profile.

In A.1, A.2 and A.3 are reported in tabular form the details of the rules defined to convert the customs data models collected in task 3.4 in the FHIR format compliant with the profiles defined in the task 3.5. Each table consists of seven columns, the first four are marked in blue (i.e. attribute, type, description and constraint) to represent attributes coming from task 3.4 while the other three columns marked in red (i.e. FHIR Mapping, FHIR Attribute and FHIR note) describe how each field is mapped to FHIR according to the specific profile.

2.2 GK-FHIR Server [UPDATED]

The gk-fhir-server is a Sprint Boot project that implements the HL7 FHIR v4 specification according to the official standard and customized for Gatekeeper project. It uses the HAPI FHIR [9] that is Java software library facilitating a built-in mechanism for adding FHIR's RESTful Server functionalities to a software application. The HAPI FHIR Java library is open source. The HAPI RESTful (Representation State Transfer) Server is based on a Servlet, so it should be deployed with ease to any compliant containers that can be provided. Simple annotations could be used to set up the server on the large part.

Project gk-fhir-server provides all the REST APIs defined by the standard together with the whole data model based on the concept of Resource. Figure 52 shows the GUI of the application exposes on HPE systems.

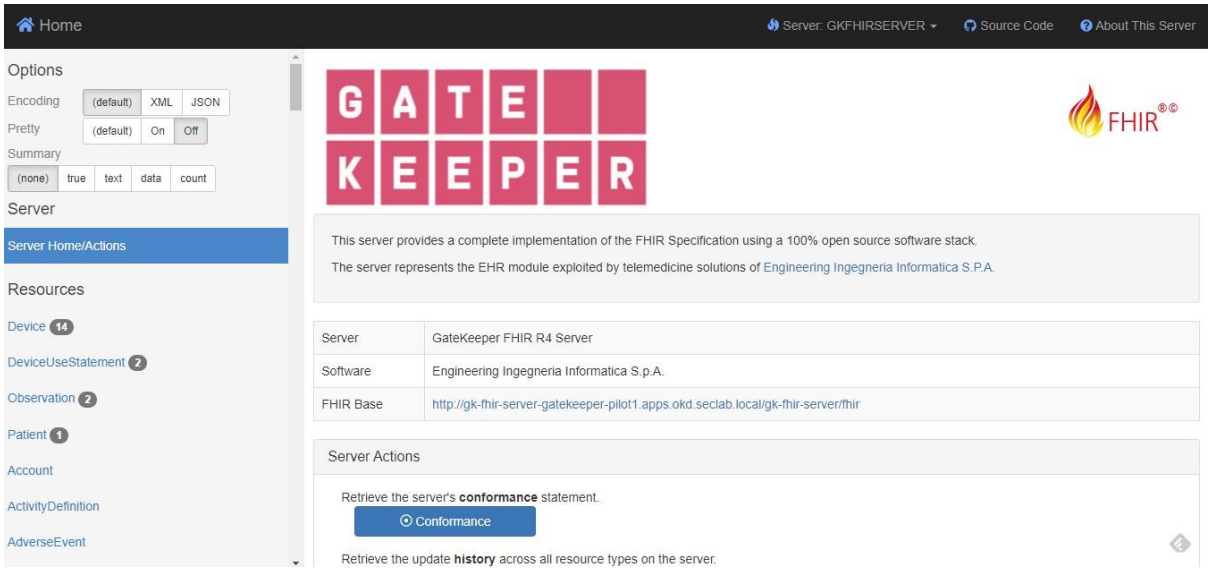


Figure 52 gk-fhir-server GUI

On the left side there is the list of all supported Resources, selecting one of them it is possible to access to the relative Rest APIs.

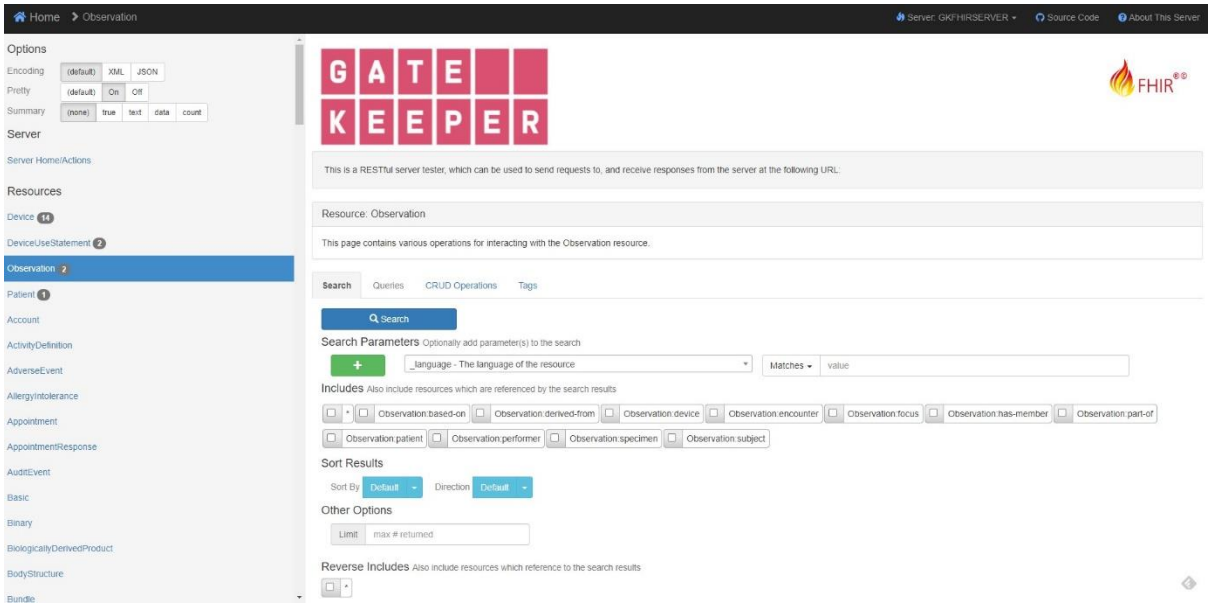


Figure 53 gk-fhir-server GUI Observation page

During the period the deliverable refers to, we continued the customization of the server, according to the definition of the capability statement and the profiles of the resources that are strictly connected with the definition of GK-FHIR-Profile coming from the output of the task 3.5. The details about the last GK-FHIR-Profile the server refers to is documented in [25] and are not reported here for sake of simplicity.

The selected database to persist data in gk-fhir-server is Maria DB [13].

When a REST request is performed on FHIR Server, for example the creation of the Resource passing the JSON in body of the request, it is persisted into database invoking the methods offered by the module named JPA Server Storage. This module returns the

JSON of the persisted Resource that will be added in an OperationOutcome resource and returned in the response of the request.

The workflow has been integrated with an interceptor mechanism that catches each request performed on JPA Server Storage retrieving the resource created, updated, or deleted after the operation completed successfully. The interceptor checks if the operation is a "create" and only in this case saves all the received resources in JSON files into a folder shared with the GK-RDF Watcher. At the same time, if the use of the Kafka channel was enabled, the stored resources will send on the channel and read by every subscriber.

During the second stage of this deliverable, some partners noticed that the FHIR server could not store the binary data linked to some resources.

Related to this problem, we have analysed the new version of the FHIR Server and we have discovered that this problem has been fixed. So we have prepared a new image with the new release of the FHIR server and we have also written the new YAML files; in this way, each pilot will be able to choose the FHIR Server version that is needed for its purposes.

2.3 GK-RDF Watcher [NEW]

The GATEKEEPER RDF-Watcher is the component that executes the RDF conversion.

In the first release of the Data Federation, the RDF conversion and storage processes had been processed within the FHIR Server. During the test stage, we noticed that this evaluation chain caused a heavy slowdown in the FHIR Server lifecycle till to cause, sometimes, a fault within the server processes. For this reason, we have chosen to split the RDF processes from the GK-FHIR Server and put them into a separate docker image.

This component shares a folder with the FHIR Server and listens for all new files that wrote in it; when a new file has been written the watching process launch an RDF conversion procedure. The watching process is made with Inotifyd [5], a Linux integrated mechanism to watch a specific folder or file and be notified if something happens within its context.

After that, the new RDF content is sent to the RDF Server by the same process.

To accomplish the described process, have been realized to Linux scripts and one Python script:

- watcher is responsible to listen for every change in the shared folder and log all changes in this folder.
- fhir-to-rdf-executor-script is responsible to launch the python script for every stored file and log every result. This script is launched by the watcher.
- jsontordf4j is responsible for conversion from FHIR to RDF through fhir_json_to_rdf python library and then sends the result to RDF Server.

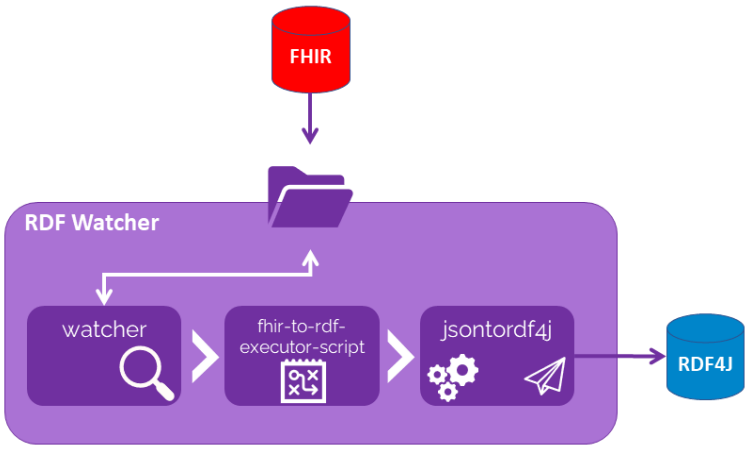


Figure 54 GK-RDF Watcher logic view

Data Federation & Integration provides two kinds of northbound APIs to retrieve data in RDF Format (JSON or XML) and RDF format (JSON or XML). The source code of HAPI FHIR library does not provide any mechanism to retrieve data in RDF format so a dedicated module has been implemented and integration into the tool.

This module is a python script that has two inputs: the JSON of the FHIR resource and the endpoint of the RDF Server where to send transformed RDF data. Internally this script converts the Resource from JSON to RDF format and sends converted data to the specified endpoint. This script is based on the fhirtordf [8] library.

To sum up, the rdf-watcher is the component that launches the python script. In order to better understand how this process works, in the following table is shown an example of a generated RDF representation by the execution of the python script starting from a JSON representation of Bundle resource containing two entries: Condition and Patient where in first column contains the JSON format while the second one contains its equivalent in the FHIR RDF format.

This routine is applied to each resource persisted in gk-fhir-server.

Table 15 Example of conversion from FHIR JSON to RDF format

JSON format	RDF format
{ **"resourceType":"Bundle",** **"entry":[** **{** **"fullUrl":"urn:uuid:80c129ba-dde5-42b8-8cb8-c302f9541e5d",** **"resource":{** **"resourceType":"Patient",** **"identifier":[** **{** **"system":"CAREACROSS",** **"value":"group/1"**	*@prefix fhir: <http://hl7.org/fhir/> .* *@prefix loinc: <http://loinc.org/rdf#> .* *@prefix owl: <http://www.w3.org/2002/07/owl#> .* *@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .* *@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .* *@prefix sct: <http://snomed.info/id/> .* *@prefix v2: <http://hl7.org/fhir/v2/> .* *@prefix v3: <http://hl7.org/fhir/v3/> .* *@prefix w5: <http://hl7.org/fhir/w5#> .* *@prefix xml: <http://www.w3.org/XML/1998/namespace> .* *@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .*

```
        }
      ]
    }
  },
  {
    "fullUrl":"urn:uuid:4dc96d50-454b-4f11-b5e8-70078976e20b",
    "resource":{
      "resourceType":"Condition",
      "extension":[
        {
          "id":"http://hl7.org/fhir/StructureDefinition/is-primary-disease",
          "url":null,
          "valueBoolean":false
        }
      ],
      "category":[
        {
          "coding":[
            {
              "system":"http://www.crowdhealth.eu/hhr-t",
              "code":"diagnosis",
              "display":"Diagnosis"
            }
          ]
        }
      ],
      "code":{
        "coding":[
          {
            "system":"http://snomed.info/sct",
            "code":"441118006",
            "display":"Progesterone receptor negative neoplasm (disorder)"
          }
        ]
      },
      "subject":{
        "reference":"urn:uuid:80c129ba-dde5-42b8-8cb8-c302f9541e5d"
      }
    }
```

```
<http://hl7.org/fhir/Condition/4eaacaec-19e0-4e95-8b8f-be6e3c6e9972> a fhir:Condition ;
  fhir:nodeRole fhir:treeRoot ;
  fhir:Condition.category [
    fhir:index "0"^^xsd:integer ;
    fhir:CodeableConcept.coding [
      fhir:index "0"^^xsd:integer ;
      fhir:Coding.code [
        fhir:value "diagnosis"
      ] ;
      fhir:Coding.display [
        fhir:value "Diagnosis"
      ] ;
      fhir:Coding.system [
        fhir:value "http://www.crowdhealth.eu/hhr-t"
      ]
    ]
  ] ;
  fhir:Condition.code [
    fhir:CodeableConcept.coding [
      a sct:441118006 ;
      fhir:index "0"^^xsd:integer ;
      fhir:Coding.code [
        fhir:value "441118006"
      ] ;
      fhir:Coding.display [
        fhir:value "Progesterone receptor negative neoplasm (disorder)"
      ] ;
      fhir:Coding.system [
        fhir:value "http://snomed.info/sct"
      ]
    ]
  ] ;
  fhir:Condition.subject [
    fhir:link <http://hl7.org/fhir/urn%3Auuid%3A80c129ba-dde5-42b8-8cb8-c302f9541e5d> ;
    fhir:Reference.reference [
      fhir:value "urn:uuid:80c129ba-dde5-42b8-8cb8-c302f9541e5d"
    ]
  ] ;
  fhir:DomainResource.extension [
    fhir:index "0"^^xsd:integer ;
    fhir:Element.id [
      fhir:value "http://hl7.org/fhir/StructureDefinition/is-primary-disease"
    ] ;
    fhir:Extension.url [
      fhir:value "None"
```

}] }	] ; fhir:Extension.valueBoolean [fhir:value "false"^^xsd:boolean]] ; fhir:Resource.id [fhir:value "4eaacaec-19e0-4e95-8b8f-be6e3c6e9972"] . <http://hl7.org/fhir/Condition/4eaacaec-19e0-4e95-8b8f-be6e3c6e9972.ttl> a owl:Ontology ; owl:imports fhir:fhir.ttl . <http://hl7.org/fhir/Patient/5077b199-0160-4358-be29-fc0b7e10cadd> a fhir:Patient ; fhir:nodeRole fhir:treeRoot ; fhir:Patient.identifier [fhir:index "0"^^xsd:integer ; fhir:Identifier.system [fhir:value "CAREACROSS"] ; fhir:Identifier.value [fhir:value "group/1"]] ; fhir:Resource.id [fhir:value "5077b199-0160-4358-be29-fc0b7e10cadd"] . <http://hl7.org/fhir/Patient/5077b199-0160-4358-be29-fc0b7e10cadd.ttl> a owl:Ontology ; owl:imports fhir:fhir.ttl . <http://hl7.org/fhir/urn%3Auuid%3A80c129ba-dde5-42b8-8cb8-c302f9541e5d> a fhir:Resource .

Python fhirtordf script has been integrated in the FHIR Server through an interceptor that works at JPA Server Storage level, as shown in Figure 55.

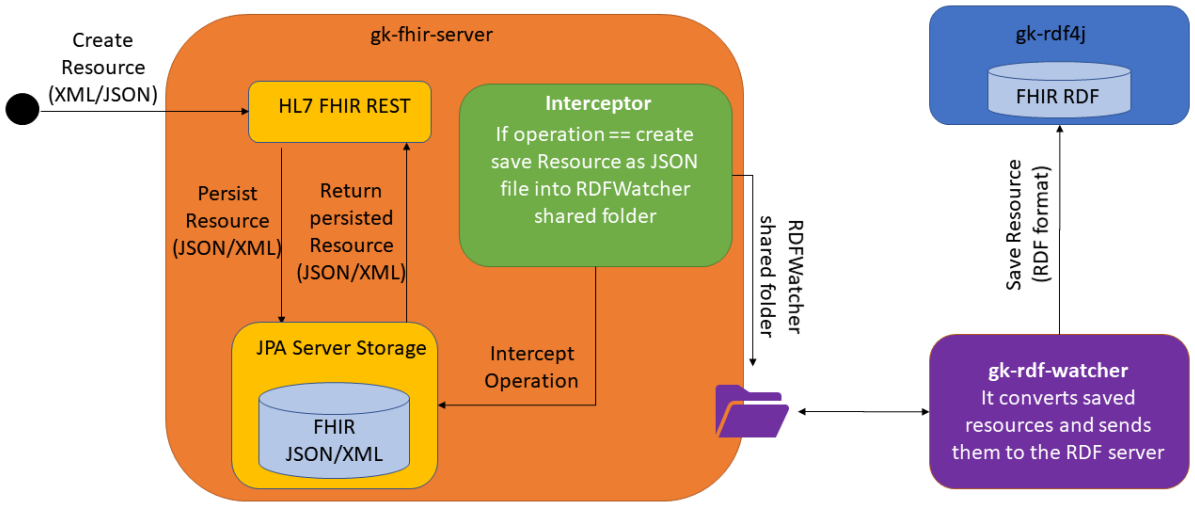


Figure 55 gk-fhir-server interceptor and gk-rdf-watcher

2.4 RDF4J Workbench

Eclipse RDF4J is an open-source framework for storing, querying, and analysing RDF (Resource Description Framework) data distributed under "Eclipse Distribution License 1.0 (BSD)" [31][32]. It contains implementations of an in-memory triplestore and an on-disk triplestore, along with two separate Servlet packages that can be used to manage and provide access to these triplestores, on a permanent server. RDF4J supports two query languages: SPARQL and SeRQL), as well as a set of fully streaming parsers and writers for most common RDF syntax formats, called Rio.

RDF4J's RDF database API differs from comparable solutions in that it offers a stackable interface through which functionality can be added, and the storage engine (SAIL) is abstracted from the query interface. Many other triplestores can be used through the RDF4J API. Through the stackable interface, functionality can be added to all of these stores.

The current core development team consists of individuals and employees of other commercial software vendors that have an interest in continued maintenance and development of the project.

In addition to its primary use as a set of Java libraries, RDF4J also provides a Server web application that can be accessed as a web service for RDF database access, and a Workbench web application which provides a (web-based) client user interface for an RDF4J Server, with a full SPARQL query editor (with completion features and syntax highlighting), and several convenient ways to manipulate or explore the data in any RDF database/SPARQL endpoint.

A general overview of RDF4J framework is shown in Figure 56.

Main features of this RDF solution are synthesized below:

- full support for SPARQL 1.1 query and update;
- fast and efficient parsing of all common RDF formats through the Rio parser toolkit;
- an easy to use, lightweight, modern Java API for handling RDF in code;

- support for RDF Schema reasoning as well as SHACL validation;
- fast in-memory RDF database with optional file-backed persistence;
- fast Native RDF database with full binary persistence to disk;
- convenient access to third-party RDF database implementations and remote SPARQL endpoints

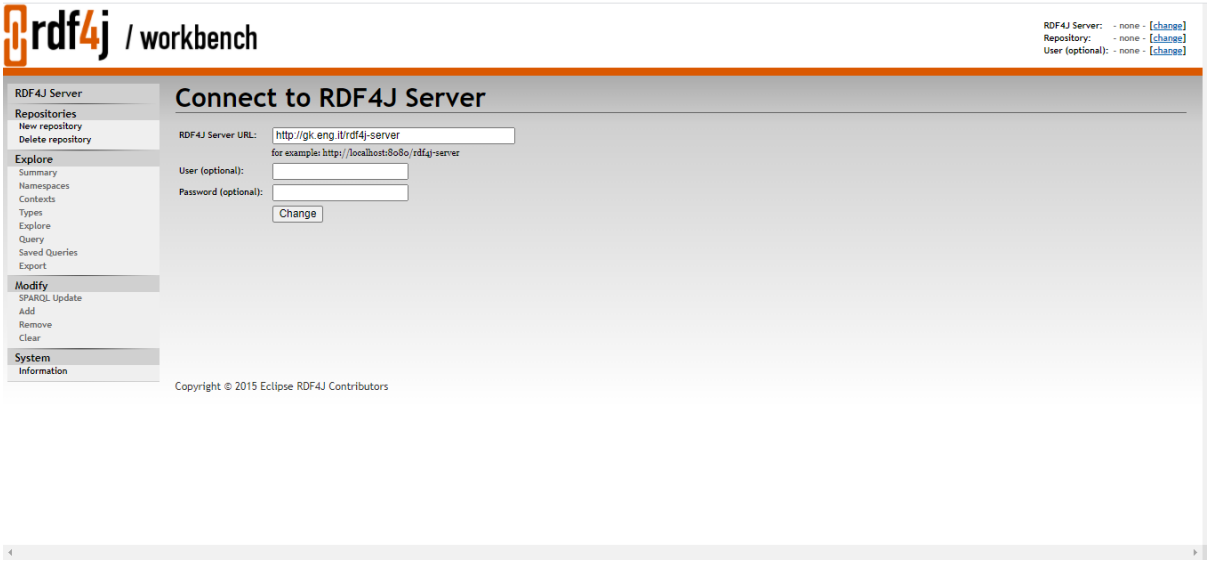


Figure 56 rdf4j workbench: a general overview

2.5 Data Federation & Integration Dockerization [UPDATED]

2.5.1 Docker overview

All the components consisting of Data Federation and Integration has been migrated to Docker [6]. Docker is a set of platforms as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine [7].

A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform. It provides a convenient way to package up applications and preconfigured server environments, which can be used for own private use or share publicly with other Docker users. A Docker image can be created in one of two ways:

- **Interactive Method**: by running a container from an existing Docker image, manually changing that container environment through a series of live steps and saving the resulting state as a new image.

- **Dockerfile Method**: by constructing a plain-text file, known as a Dockerfile, which provides the specifications for creating a Docker image.

The Dockerfile approach is the method of choice for real-world enterprise-grade container deployments. It is a more systematic, flexible and efficient way to build Docker images and the key to compact, reliable and secure container environments. In short, the Dockerfile method is a three-step process whereby you create the Dockerfile and add the commands you need to assemble the image.

The output of build process of Dockerfile is a Docker Image while a Container is a running Image, as shown in

Figure 57 Steps to create a docker container

Compose is a tool for defining and running multi-container Docker applications. With Compose, it is possible to use a YAML file to configure an application's services. Then, with a single command, it is possible to create and start all the services from a custom configuration. Compose works in all environments: production, staging, development, testing, as well as CI workflows. Using Compose is basically a three-step process:

- Define the app's environment with a Dockerfile so it can be reproduced anywhere.
- Define the services that make up the app in docker-compose.yml so they can be run together in an isolated environment.
- Run docker-compose up and Compose starts and runs the entire app.

Container registries are catalogues of storage locations, known as repositories, where it is possible to push and pull container images. The three main types of registries are as follows:

- **Docker Hub**: Docker's own official image resource where it is possible to access more than 100,000 container images, shared by software vendors, open-source projects and Docker's community of users. It is possible also use the service to host and manage your own private images.

- **Third-party registry services**: Fully managed offerings that serve as a central point of access to your own container images, providing a way to store, manage and secure them without the operational headache of running your own on-premises registry.

- **Self-hosted registries**: A registry model favored by organizations that prefer to host container images on their own on-premises infrastructure— typically because of security, compliance or lower latency requirements.

2.5.2 Migration of Data Federation & Integration to Docker [UPDATED]

The prototype of Data Federation & Integration has been released as Docker microservice where some containers are pulled from the Docker Hub registry while other ones are implemented ad hoc starting from the source code of the application. As shown in Figure 58 the DFI framework consists of five containers that can be launched with a single command using the implemented YAML docker-compose file. The image of maria db and gk-rdf4j are pulled the public Docker Hub registry while images for containers gk-integration-engine and gk-fhir-server are written from scratch. Following some details about each container.

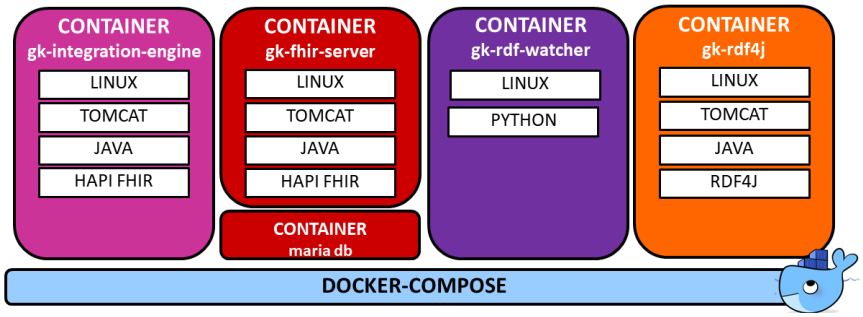


Figure 58 Data Federation & Integration in Docker

The image of gk-rdf4j is pulled from the Docker Hub registry and configurated to be integrated into DFI framework. It is written in Java and deployed on an embedded Tomcat application server.
The image of gk-integration-engine is developed by ENG team, it is based on Tomcat, Java and HAPI-FHIR library. The output is a YAML Dockerfile. This image is pushed on the private ENG Docker registry so that it can be pulled and started when the Docker compose is launched.

```dockerfile
FROM openjdk:15-jdk-alpine

#Install python and pip3
RUN apk upgrade --update && apk add --no-cache python3 python3-dev
RUN apk add --update py-pip

COPY ./target/integration-engine-0.0.1-SNAPSHOT.jar /gkie/

#RUN chmod o+rwx /gkie/integration-engine-0.0.1-SNAPSHOT.jar

#Copy python samsung converter (env1 is the python virtual env for linux)
COPY samsung-converter /gkie/gk_lib/samsung-converter

#COPY Lib files
COPY ./src/main/resources/lib/* /gkie/gk_lib/

#COPY Rml files
COPY ./src/main/resources/rmlmapper-files/* /gkie/gk_rml_data/

#COPY Configuration Files
COPY ./src/main/resources/logback.xml /gkie/conf/

#CREATE LOGS DIRECTORY
RUN mkdir /gkie/logs

#Set environment for python
ENV PATH=/gkie/lib/python3.5:${PATH}

RUN . /gkie/gk_lib/samsung-converter/venv1/bin/activate
RUN pip install -r /gkie/gk_lib/samsung-converter/venv1/requirements.txt

# Make port 8080 available to the world outside this container
EXPOSE 8080

#RUN 'apk update'
#RUN 'apk upgrade'
#RUN 'apk add curl'

# Create a group and user
RUN addgroup -S 1000610000 && adduser -S 1000610000 -G 1000610000 &&\
    chown -R 1000610000:1000610000 /gkie

# Tell docker that all future commands should run as the appuser user
USER 1000610000

WORKDIR /gkie

# This tells Docker to execute a shell command-line within the target system.
# Here we practically just "touch" our file so that
# it has its modification time updated (Docker creates all container files in an "unmodified" state by default).
#RUN sh -c 'touch integration-engine-0.0.1-SNAPSHOT.jar'

#ENTRYPOINT ["/bin/ash", "-c", "sleep infinity"]

ENTRYPOINT ["java","-jar","integration-engine-0.0.1-SNAPSHOT.jar"]
```

Figure 59 Dockerfile for gk-integration-engine

The image of RDFWatcher has been developed by ENG team, it is based on Linux and Python to watch for every new JSON file written into its shared folder and run the routine to convert them from FHIR JSON to RDF (see sec. 2.3 for further details).

```
Dockerfile > ...
1   FROM alpine:3.14
2
3   RUN apk upgrade --update && apk add --no-cache openrc python3 python3-dev cmd:pip3 gcc gfortran freetype-dev musl-dev libpng-dev g++ lapack-dev
4
5   RUN pip3 install --upgrade pip
6
7   # install pip3
8   RUN pip3 install virtualenv
9
10  #install requests
11  RUN pip3 install requests
12
13  # install pip3 rdftofhir
14  RUN pip3 install fhirtordf
15
16  #Add "tests" sources in python installation folder to run fhirtordf library from cli
17  #COPY fhirtordf-depencencies/tests /usr/lib/python3.6/site-packages/tests
18
19  #CREATE WATCHER DIRECTORIES
20  RUN mkdir /gkrw
21  RUN mkdir /gkrw/json
22  RUN mkdir /gkrw/script
23  RUN mkdir /gkrw/log
24  RUN mkdir /.cache
25  #RUN touch /gkrw/log/watcher.log
26
27  #ADD WATCHER SCRIPT AND SERVICES
28  COPY /scripts/watcher /gkrw/script/watcher
29  COPY /scripts/fhir-to-rdf-executor-script.sh /gkrw/script/
30  COPY /scripts/jsontordf4j.py /gkrw/script/jsontordf4j.py
31  #COPY /test /gkrw/test
32
33  #ADD PERIODIC SCRIPTS
34  COPY /scripts/log-splitter.sh /etc/periodic/daily
35  COPY /scripts/log-cleaner.sh /etc/periodic/weekly
36
37  #RUN rc-update add watcher-service
38
39  # Create a group and user
40  RUN addgroup -S 1000610000 && adduser -S 1000610000 -G 1000610000 &&\
41      chown -R 1000610000:1000610000 /gkrw &&\
42      chown -R 1000610000:1000610000 /.cache
43
44  # Tell docker that all future commands should run as the appuser user
45  USER 1000610000
46
47  WORKDIR /gkrw
48
49  #ENTRYPOINT ["/bin/ash", "-c", "sleep infinity"]
50  ENTRYPOINT ["/bin/ash", "-c", "inotifyd /gkrw/script/watcher /gkrw/json:nd >> /gkrw/log/watcher.log 2>&1"]
```

Figure 60 Dockerfile for gk-rdf-watcher

Also, the gk-fhir-server container has been developed by ENG team using the approach of Dockerfile and pushed on the private ENG Docker registry. On the image of this container is installed Tomcat and Java, to run the FHIR Server, gk-fhir-server uses the container maria-db to store data, such container persists data in a docker volume. Figure 61 shows Dockerfile written for gk-fhir-server.

```
1   # Start with a base image containing tomcat
2   FROM tomcat:9.0-alpine
3
4   RUN apk upgrade --update && apk add --no-cache gcc gfortran freetype-dev musl-dev libpng-dev g++ lapack-dev && apk add openjdk8
5
6   # Make port 8080 available to the world outside this container
7   EXPOSE 8080
8
9   COPY  ./target/gk-fhir-server.war /usr/local/tomcat/webapps/
10
11  #Add a user 1000610000 with UID 1000610000
12  # Create a group and user
13  RUN addgroup -S 1000610000 && adduser -S 1000610000 -G 1000610000 &&\
14      chown -R 1000610000:1000610000 /usr/local/tomcat &&\
15      mkdir -p /home/1000610000/lucenefiles &&\
16      chown -R 1000610000:1000610000 /home/1000610000/lucenefiles &&\
17      chown -R 1000610000:1000610000 /usr/local/tomcat/webapps
18
19  #Specify the user
20  USER 1000610000
21
22  # Add Maintainer Info
23  LABEL maintainer="gatekeeper_mantainer"
24
25  CMD ["catalina.sh", "run"]
```

Figure 61 Dockerfile for gk-fhir-server

During the development stage the Docker-compose file is developed in order to start all containers with one command. It pulls images of maria db and rdf4j from Docker Hub

registry and the images of gk-integration-engine and gk-fhir-server from the private Docker registry installed on ENG Server.
After a test stage, we ported the Docker compose file into several YAML files compliant with OKD platform provided by HPE in the last months. We discuss this process, more in detail, in the next chapter.

2.6 Source Code [NEW]

The current prototype of the Data Federation & Integration is shared on the git ENG repository. It consists of three projects: gk-integration-engine, gk-fhir-server and gk-docker.

The source code of gk-integration-engine is available at ENG Gitlab https://production.eng.it/gitlab/GTKEEPER_EU/gk-integration-engine. The used technologies are:

- JAVA 1.8 as programming language.
- Framework Spring [18].
- Apache Camel for the routing.
- Docker to create the image of the software.
- HAPI FHIR library to implement the conversion from raw data to FHIR.
- RML library to convert raw data to RDF.

The source code of gk-fhir-server is available at ENG Gitlab https://production.eng.it/gitlab/GTKEEPER_EU/gk-fhir-server. It is based on the following technologies:

- JAVA 1.8 as programming language.
- HAPI FHIR library to implement the FHIR Rest APIs.
- Tomcat [19] as application server.
- Maria database as storage.
- Docker to create the image of the software.

The source code of gk-rdf-watcher is available at ENG Gitlab https://production.eng.it/gitlab/GTKEEPER_EU/gk-rdf-watcher. It is based on the following technologies:

- Linux Alpine
- Inotifyd to watch a specific folder or file and be notified if something happens within its context
- Python to write the routine to convert FHIR data to RDF.

The source code of gk-docker is available at ENG Gitlab https://production.eng.it/gitlab/GTKEEPER_EU/gk-docker. It contains the docker-compose file that is start the docker images of the components belonging to DFI: gk-integration-engine, gk-fhir-server, maria database, rdf4j workbench.

Moreover it contains the YAML file to deploy the all services on the OKD platform.

3 Data Federation and Integration v2: Deployment Environments [UPDATED]

After the first release of this document, the need to adapt the mentioned solutions in some OpenShift deployable artifacts has emerged. For this reason, after a brief analysis of the OpenShift artifacts, we ported and arrange our components in a series of YAML files releasable over OKD platform.

3.1 Open Shift and HPE Data Centre overview

Now we analyse the migration procedure that was implemented to port all the software over the OKD platform provided by HPE. First of all, we will introduce an overview of the OpenShift artifact that we intend to use, then we examine all the YAML files that are the base of the components; in the end, we present one of the possible deployment scenarios and the expressed pilot's needs.

All the mentioned files are stored to this link:

https://production.eng.it/gitlab/GTKEEPER_EU/gk-docker/-/tree/master/hpe-okd

and each partner will be able to download its configuration looking for its name in the folder name. e.g.: the Puglia pilot will have to download all the files in the "hpe-okd-Puglia" folder and so on for every partner.

Finally, we try to explain a possible way to manage a flow of continues integration and deployment using the tools provided by HPE.

3.1.1 Platform description: OpenShift [NEW]

Red Hat® OpenShift® [3] is a hybrid cloud, enterprise Kubernetes application platform, trusted by thousands of organizations.

OpenShift Artifact used in a generic deployment over an OKD namespace are:

- *Config Map*, an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume. In addition, it allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable
- *Persistent Volume Claim,* a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system. A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see AccessModes

- *Service,* an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector
- *Deployment,* represents a set of multiple, identical Pods with no unique identities. A Deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. In this way, Deployments help ensure that one or more instances of your application are available to serve user requests. Deployments are managed by the Kubernetes Deployment controller
- *Route,* exposes a service at a host name, like www.example.com, so that external clients can reach it by name. DNS resolution for a host name is handled separately from routing; your administrator may have configured a cloud domain that will always correctly resolve to the OKD router, or if using an unrelated host name you may need to modify its DNS records independently to resolve to the router.

3.1.2 GK Integration Engine deployed on HPE Data Centre [NEW]

As we are going to describe the need for every pilot to have its own namespace in the OKD platform has emerged during the design stage. For this reason, we have a separate instance of the Gatekeeper Integration Engine (GKIE) for every of them.

Another thing that emerged during the design stage, is that some pilots already send their data into FHIR format, and these data are already compliant with the GK-FHIR profile. For these pilots, we haven't the necessity to install an instance of the GKIE component.

For any other pilots that use their own data format or their own FHIR profile, it's necessary to use the GKIE beforehand configured with their conversion rules.

For the correct use of the GKIE Southbound APIs, every pilot will have to configure a VPN site-to-site connection between its system and HPE's OKD platform. This connection is a security constraint provided by HPE, if not configured the user will not be able to invoke the Data Federation services.

Every pilot can correctly set up its connection to the HPE platform following the guide released by HPE [22].

Now we introduce and describe all the OpenShift artifacts implemented for the GKIE and their suggested installation order, later in this chapter we provide some steps useful to reproduce their installation into the OKD platform (as described in 3.1.8):

Table 16 GATEKEEPER Integration Engine OpenShift artifacts list

File Name	Kind	#
gk-integration-engine-configmap.yaml	ConfigMap	1
data-federation-pvc-gkie.yaml	PersistentVolumeClaim	2
gk-integration-engine-service.yaml	Service	3
gk-integration-engine-deployment.yaml	Deployment	4
gk-integration-engine-route.yaml	Route	5

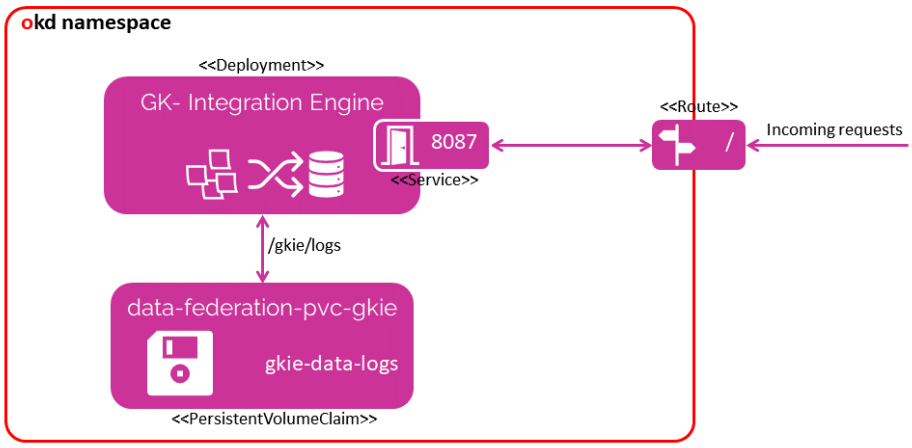


Figure 62 Gatekeeper Integration Engine OKD deployment schema

As shown in the above figure the GK integration engine needs:

- one service for REST communication through different components
- one route for routing incoming connection to its service
- one persistent volume to store its log about its work, mounted on "/gkie/logs" directory

3.1.3 GK-FHIR Server and GK RDF Watcher deployed on HPE Data Centre [NEW]

Like the GKIE, also the FHIR Server will be released one per namespace.

Table 17 Gatekeeper FHIR Server & GK RDF Watcher OpenShift artifacts list

File Name	Kind	#
gk-fhir-server-configmap.yaml	ConfigMap	1
gk-rdf-watcher-log-pvc.yaml	PersistentVolumeClaim	2
data-federation-pvc-server-data.yaml	PersistentVolumeClaim	3
gk-rdf-watcher-json-pvc.yaml	PersistentVolumeClaim	4
gk-fhir-server-service.yaml	Service	5
gk-rdf-watcher-gk-fhir-server.yml	Deployment	6
gk-fhir-server-route.yaml	Route	7

It is possible to configure some options about the FHIR Server from the "env" section of the YAML file, regarding this, here below it shows a screen with the most important option which every pilot must configure.

```
env:
  - name: HAPI_RDF4JWORKBENCH
    value: "http://rdf4j-gatekeeper-dev.apps.okd.seclab.local/rdf4j-server/repositories/gatekeeper/statements"
  - name: HAPI_ENABLE_RDF4J
    value: "true"
  - name: http_proxy
    value: "http://web-proxy.seclab.local:8088/"
  - name: https_proxy
    value: "http://web-proxy.seclab.local:8088/"
  - name: no_proxy
    value: .cluster.local,.seclab.local,.svc
  - name: SPRING_APPLICATION_JSON
    value: >-
      {
      "enable_rdf4j":"true",
      "enable_kafka":"false"
      }
```

Figure 63 env section of the FHIR Server deployment YAML file

If the pilot chose a version of the FHIR standard different from the fourth, it must declare this setting the variable "**fhir_version**" in the env section:

```
env:
  - name: HAPI_RDF4JWORKBENCH
    value: "http://rdf4j-gatekeeper-pilot6b.apps.okd.seclab.local/rdf4j-server/repositories/gatekeeper/statements"
  - name: HAPI_ENABLE_RDF4J
    value: "false"
  - name: http_proxy
    value: "http://web-proxy.seclab.local:8088/"
  - name: https_proxy
    value: "http://web-proxy.seclab.local:8088/"
  - name: no_proxy
    value: .cluster.local,.seclab.local,.svc
  - name: SPRING_APPLICATION_JSON
    value: >-
      {
      "fhir_version":"DSTU3",
      "enable_rdf4j":"false",
      "enable_kafka":"false"
      }
```

Figure 64 env section of the FHIR Server deployment YAML file

As shown in both above figures, in the FHIR Server deployment, we have the possibility to enable two different channels to forward the stored data. In fact, the "enable_kafka" option enables two Apache Kafka channels and forwards every bundle or resource that has been written in the FHIR Server; every subscriber to these channels will receive the sent information.
Instead, the "enable_rdf4j" option enables a procedure that consents to convert the received resources into RDF format and stores them into the provided RDF Server.
For this process, the use of the RDF Watcher is needed. In fact, the use of the FHIR Server only, not guarantee the whole conversion process, because for problems about bad performance, the FHIR Server can only write the received resources into JSON files stored in a folder shared with the RDF Watcher component.
As we mentioned in the paragraph 2.3, when a file is stored in this shared folder, the RDF Watcher takes it in charge and starts a conversion process that ends with sending the converted information to the RDF Server.
Normally these two components can be deployed in separated POD. In that case, must be defined a shared persistent volume with access mode set to "ReadWriteMany"; this mode is the only way to enable access from different POD to the same memory space. This setting is not allowed on the OKD platform provided by HPE; so that, we have chosen

to arrange a unique deployment with two containers. In this case, we can use a shared persistent volume with access mode set to "ReadWriteOnce" because the containers that need to access it are in the same POD.

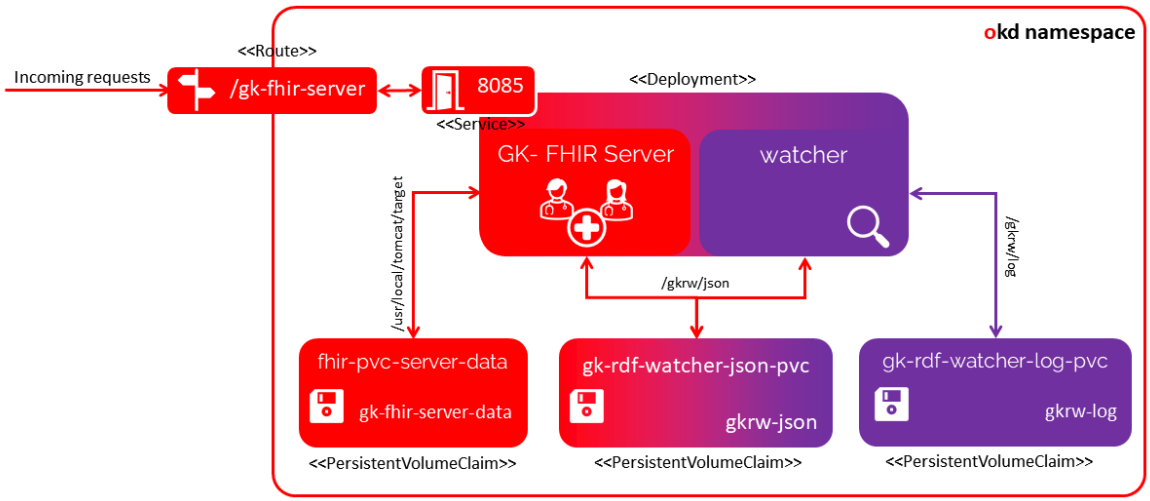


Figure 65 Gatekeeper FHIR Server & Gatekeeper RDF Watcher OKD deployment schema

In the figure above, we can best notice the cohabitation of the FHIR Server and RDF Watcher in the same Deployment. They have a persistent volume for each and one other persistent volume shared between them.

As shown in the figure the components need:

- one service for GK-FHIR Server REST communication with the other components
- one route for routing incoming connection to GK-FHIR Server service
- one persistent volume to store FHIR Server temporary files, mounted on "/usr/local/tomcat/target" directory
- one shared persistent volume to store FHIR resources JSON files, mounted on "/gkrw/json" directory watched by the RDF in search of newly stored files to manage
- one persistent volume to store the RDF Watcher log about its work, mounted on "/gkrw/log" directory

3.1.4 GK-FHIR Database [NEW]

Here below is reported the OpenShift Artifacts list related to the GK-FHIR Database and their suggested installation order.

This component uses a public docker image based on MariaDB.

Table 18 Gatekeeper FHIR Database OpenShift artifacts list

File Name	Kind	#
data-federation-db-cnf-configmap.yaml	ConfigMap	1
data-federation-pvc-db.yaml	PersistentVolumeClaim	2
db-service.yaml	Service	3

db-deployment.yaml	Deployment	4

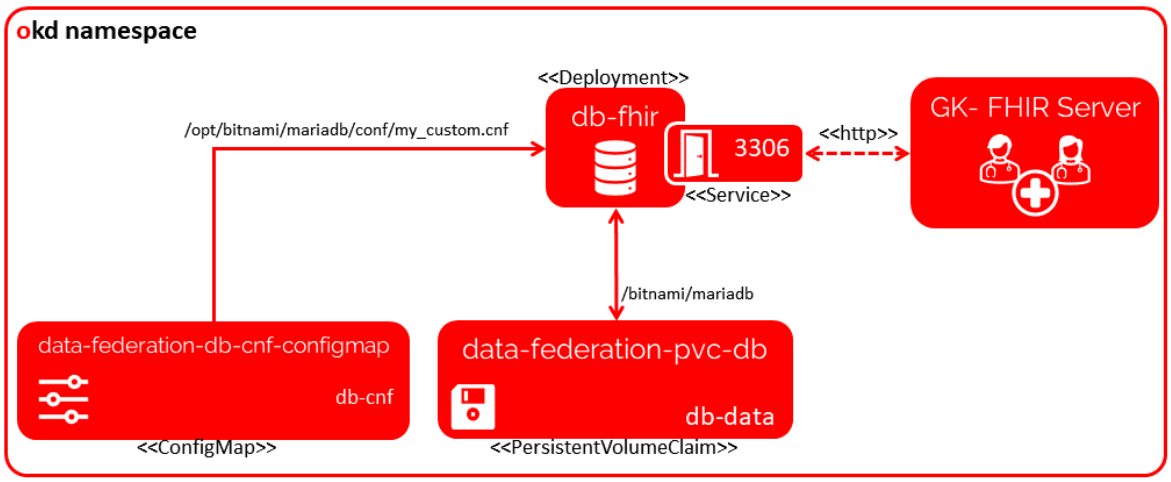


Figure 66 Gatekeeper FHIR Database OKD deployment schema

As shown in Figure 66 the db-fhir needs:

- one service for db-fhir REST communication with the other components, typically the FHIR Server
- one persistent volume to store FHIR Server resources, mounted on "/bitnami/mariadb" directory
- one config map to memorize all the MariaDB options, mounted on "/opt/bitnami/mariadb/conf/custom.cnf" file

3.1.5 RDF4J Workbench deployed on HPE Data Centre [NEW]

This component is an open-source modular Java framework for working with RDF data . This includes parsing, storing, inferencing and querying of/over such data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions. It allows you to connect with SPARQL endpoints and create applications that leverage the power of Linked Data and Semantic Web.

Also, this component uses a public docker image based on RDF4J project.

Table 19 Gatekeeper RDF4J OpenShift artifacts list

File Name	Kind	#
data-federation-pvc-rdf4j-logs.yaml	PersistentVolumeClaim	1
datafederation-pvc-rdf4j.yaml	PersistentVolumeClaim	2
rdf4j-service.yaml	Service	3
rdf4j-deployment.yaml	Deployment	4
rdf4j-route.yaml	Route	5

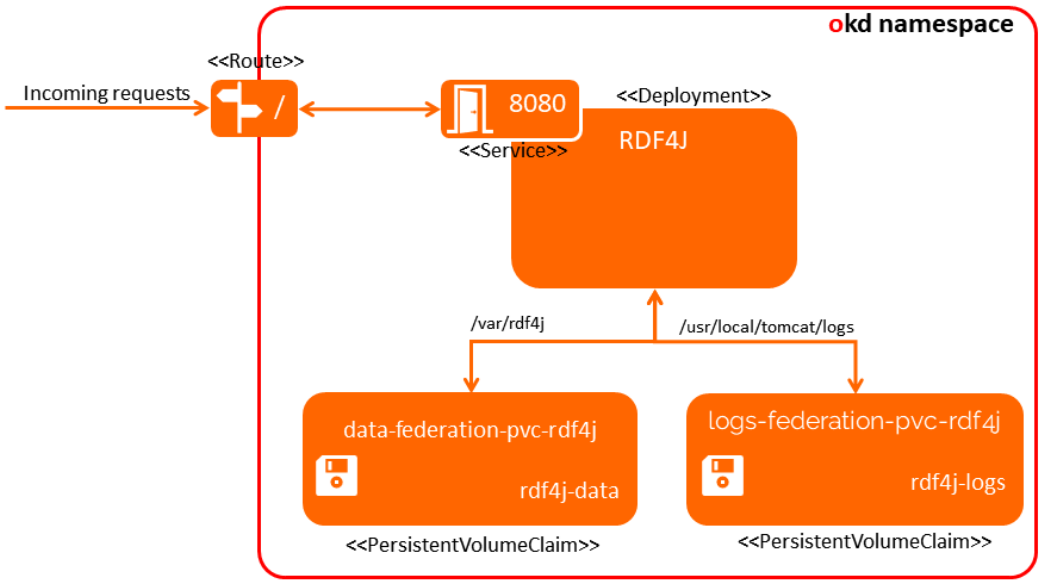


Figure 67 Gatekeeper RDF4J OKD deployment schema

As shown in the above figure the RDF4J Server needs:

- one service for REST communication through different components
- one route for routing incoming connection to its service
- one persistent volume to store its log about its work, mounted on "/usr/local/tomcat/logs" directory
- one persistent volume to store the received data, mounted on "/var/rdf4j" directory

3.1.6 Deployment Scenarios [UPDATED]

This section provides an overview of the possible deployment alternatives of northbound API of the Data Federation & Integration into OKD platform. The OpenShift artifact produced for the DFI will be deployed onto the OKD platform provided by task 4.1. OKD is a distribution of Kubernetes [12] - an open-source system aiming to automatize the deployment, to scale, and to manage of containerized applications - optimized for continuous application development and multi-tenant deployment. OKD embeds Kubernetes and extends it with security and other integrated concepts. OKD adds developer and operations-centric tools on top of Kubernetes to enable rapid application development, easy deployment and scaling, and long-term lifecycle maintenance for small and large teams

OKD is a sibling Kubernetes distribution to Red Hat OpenShift. OpenShift Container Platform (formerly known as OpenShift Enterprise) is Red Hat's on-premises private platform as a service product, built around application containers powered by Docker, with orchestration and management provided by Kubernetes, on Red Hat Enterprise Linux and Container Linux (formerly known as CoreOS or RHCOS).

The figure below about "Data Federation & Integration deployed in OKD cluster" gives a general overview about how the GK Platform could be deployed on (HPE) GK CloudL.

Security, updates, and maintenance can be managed centrally ensuring the highest level of service. GATEKEEPER Platform will be responsible to ensure separation of data and multitenancy. All the accesses from external applications to GK platform are managed by TMS (task 4.2) and GTA (task 4.5). As stated by the deliverable D3.2 GK offers, to the pilots involved in project, the following main alternatives of deployment:

1. **Pilots own a private space (virtual cluster) on GK Cloud and share some data with GK Platform.** In case Pilots require to keep part of their data isolated from the other pilots, GK Cloud can provide private storage spaces in dedicated private "*pilot cloud tenants*", while the GK Platform remain centralized. Pilot systems running in the separate spaces interact with the Platform to exploit its services from within GK Cloud. GK CLOUD PLATFORM - OKD figure shows this alternative where data shared with Data Federation & Integration and persisted in FHIR and RDF format in a private and dedicated space (virtual cluster). In this configuration it is assigned a private virtual cluster to each pilot containing only their data, such data can be accessed only by the pilots owner of the cluster. When a specific pilot invokes southbound APIs of DFI to share its data, this one is able to acquire data, transform them to FHIR and RDF format and, finally, forward them to the specific virtual cluster belonging to the pilot who sent the data. According to this configuration each pilot has only access to its private data.

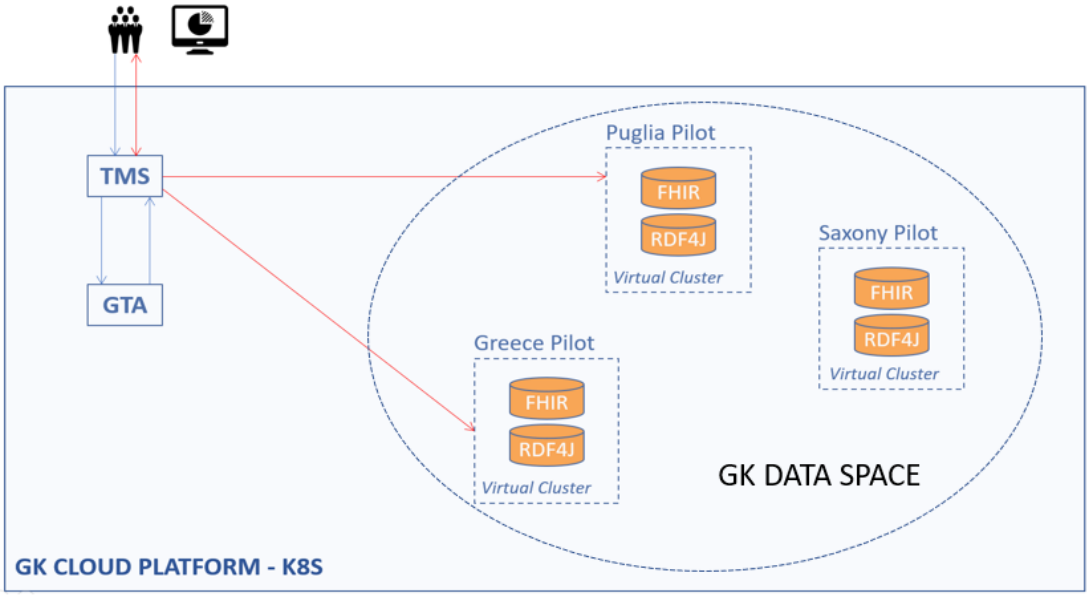


Figure 68 GK CLOUD PLATFORM - OKD

2. **Replicas of the GATEKEEPER platform are deployed separately.**

   To ensure a greater isolation, an alternative deployment is possible which implies the creation of separated "*pilot cloud tenants*" within the (HPE) GK Cloud, where *replicas* of the GATEKEEPER platform are deployed separately (not only storage as in the solution above). (note.: for sake of simplicity only the Data Federation & Integration components (e.g. Integration Engine, RDF-Watcher, FHIR Server, FHIR-DB, RDF4J Server) are reported within each replica but GTA and TMS are deployed in each replica as well).

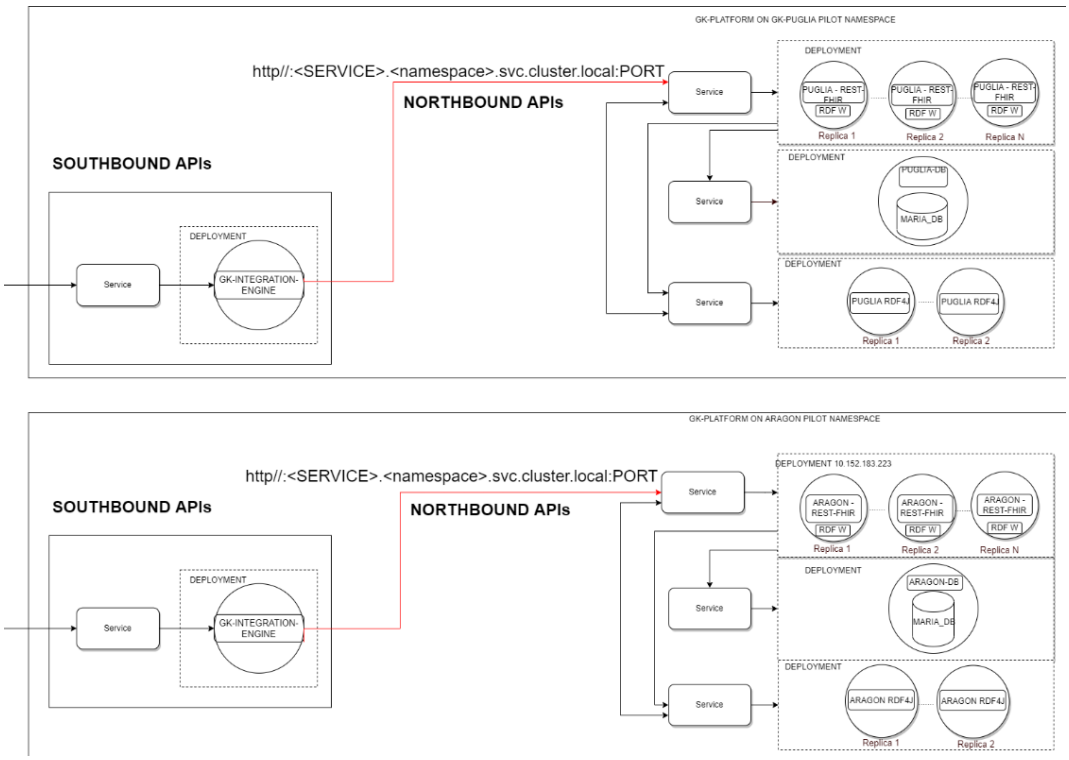


Figure 69 Data Federation & Integration deployed in OKD cluster

The configuration combines all the single components of DFI described in the previous sections, i.e., gk-integration-engine, gk-rdf-watcher, gk-fhir-server and gk-rdf4j. All components are deployed in the same environment but, there is a whole installation per pilot in a dedicated *namespace*.

Each pilot specific installation has a specific k8s namespace: in the table below are listed the pilot names and the related k8s namespace name provided by HPE.

Table 20 Namespace per pilot

Pilot name	Namespace name
Puglia	pilot1
Aragon	pilot2
Saxony	pilot3
Basque Country	pilot4
Greece UC #a	pilot5a
Greece UC #b	pilot5b
Cyprus UC #a	pilot6a
Cyprus UC #b	pilot6b
Milton Keynes (UK)	pilot7

Poland	pilot8
Covid-19	pilot9
BANGOR	pilot10
SINGAPORE	pilot11
TAIWAN	pilot12
HONG KONG	pilot13

Moreover, there are other three *namespaces*, one for **development** purpose, named **'dev'**, and one other for **test** purpose, named **'test'** and one for production purpose, named **'prod'.**

The installations per pilot are replica of the k8s YAML files but with a different namespace.

The interaction among all PODs inside the whole cluster is made with **ClusterIP** node, implemented with k8s Services. A ClusterIP provides a cluster IP address accessible only by other PODs and services within in the cluster. No external IP address is created for the application. To access a POD underlying a cluster service, other applications in the cluster can use the ClusterIP address of the service or send a request using the service name. When reached by requests, the service forwards them to the pods equally, regardless of the clustered IP addresses of the pods and the worker node on which they are deployed. if it is not specified a type in a service YAML configuration file, the ClusterIP type is created by default.

With this configuration all PODs inside the DFI cluster can interact by mean of ClusterIP node using the assigned name and port without setting the IP of the node. The advantage of this configuration is that the configuration YAML file to run the PODs must not be updated each time the IP of the node changes because the interaction among all the PODs is realized using the name and the port associated by the ClusterIP node that are statically defined.

When the POD for gk-integration-engine is started, it reads from the spring_json environment variable the **service names** and **ports** of the gk-fhir-server and gk-rd4j PODs deployed in the pilot specific namespace. Thank to this configuration even if some POD of gk-fhir-server and rdf4j is deleted and recreated, it is not needed to restart the PODs of gk-fhir-server to reload the endpoint of the created POD of fhir-server and rdf4j because they are statically defined into server ClusterIP.

The access point among Pilots applications and DFI Cloud platform are the OKD Routes. Such routes enable the interactions of the PODs running inside the DFI cluster with the external applications using defined host names.

Following are briefly described the steps performed by a pilot X to share its data with the DFI and the steps followed by DFI to store them within GK cloud Service:

1. Pilot X invokes the APIs defined in OKD Route to retrieve the bearer token. (TMS)

a. Route forwarding the request to the service ClusterIP that checks if the credential sent by the PILOT are correct. (TMS)

2. After that the TMS forward the request to the required service.
3. Pilot X invokes one of two defined southbound APIs (through the TMS) defined by the route of specific pilot's gk-integration-engine passing the data in body.
4. The specific pilot's gk-integration-engine has an internal apache camel routine that is able to identify the pilot that is sending data. Based on this information, data are forwarded to the FHIR REST API by the service of the gk-fhir-server deployed in the specific pilot's namespace.
5. fhir-server POD persists data in maria database in FHIR format, store the resource in the folder shared with RDF Watcher and, eventually send the resource through the kafka channel. Then the RDF Watcher convert the stored resources to RDF and invokes the API using the service of the rdf4j POD that persist them.
6. Pilot X can access to persisted FHIR and RDF data by means the northbound APIs hosted on the services associated Routes.

All calls to southbound and northbound APIs are expected to be trusted since the interaction with the Data Federation & Integration is mediated by GTA (D4.14) & TMS (D4.9).

3.1.7 Pilots' needs [NEW]

This information is added in this version of the deliverable in order to summarise within a table the Pilots' needs. We can also find new pilots (i.e. Covid-19, Bangor, Singapore, Taiwan, Hong Kong) who joined in the last period just before the deliverable release. So, they are mentioned here because it was possible to collect their needs, while tasks related to design, data model, and so on (as the ones described in paragraph 1.2) have not yet started. For some Asian pilots also the needs elicitation is in progress and this is the reason why the table is missing of their information.

To continue, the first step was to gather such information through a collaboration tool, Trello [23], that organizes a task/project into boards. In this way, it tells the teamwork what's being worked on, who's working on what, and where something is in a process. In order to give an example of its usage related to pilots' needs collection within the GK project, in Figure 70 a screenshot is reported.

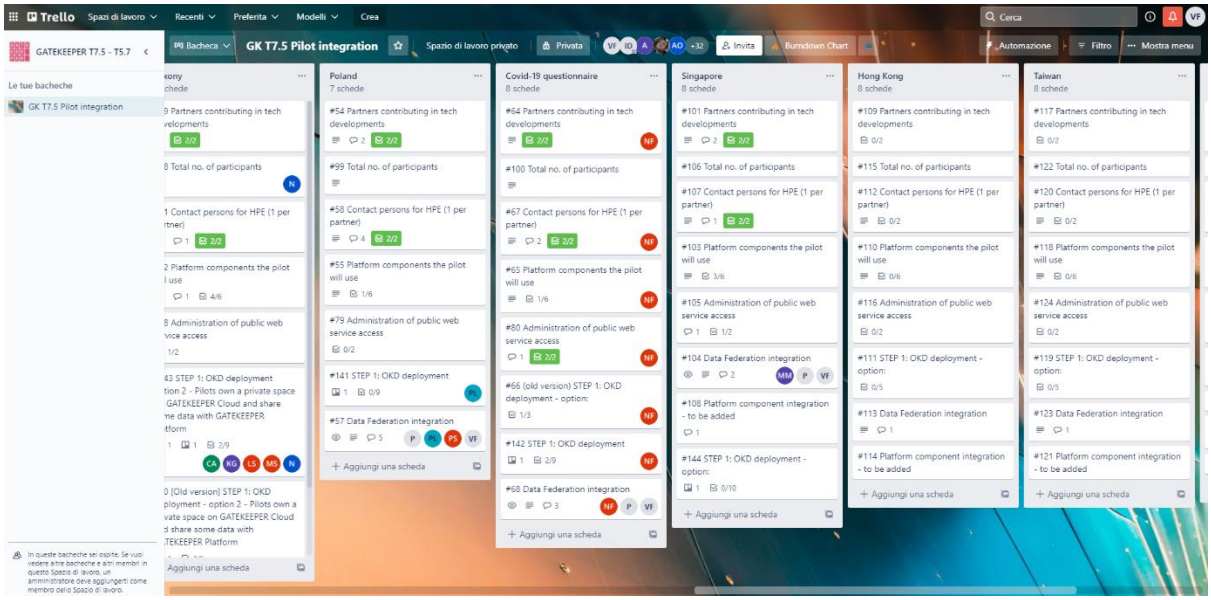


Figure 70 Trello: a general overview of its usage in GK project

Then the needs are summarised in Table 21. For each Pilot an 'X' is reported on the column of the related GK component (FHIR Server, Integration Engine, RDF) for which integration has been requested. If the integration was not needed for a component an '-' is reported.

At the time of the deliverable release, the still unexpressed pilots' needs are represented by empty cells.

Table 21 Pilots' needs

Pilot	GK-FHIR Server	GK-IE	RDF
Puglia	X	X	X
Aragon	X		
Saxony	X	X	X
Basque Country	X	-	-
Greece	X	-	-
Cyprus	X	-	-
Milton Keynes	X	X	X
Poland	X	X	-
Covid-19	X	-	-
Bangor	X	X	X
Singapore	X		
Taiwan	X		

Hong Kong	X		

3.1.8 OKD Installation procedure [NEW]

To explain the OKD installation procedure. here below are shown some screenshots of the overall procedure and each step is described as follows.
First of all (red circle with number 1), the user click the button "+Add" on the OKD left side nav. In the second step (red circle with number 2), the user click the "YAML" tile.

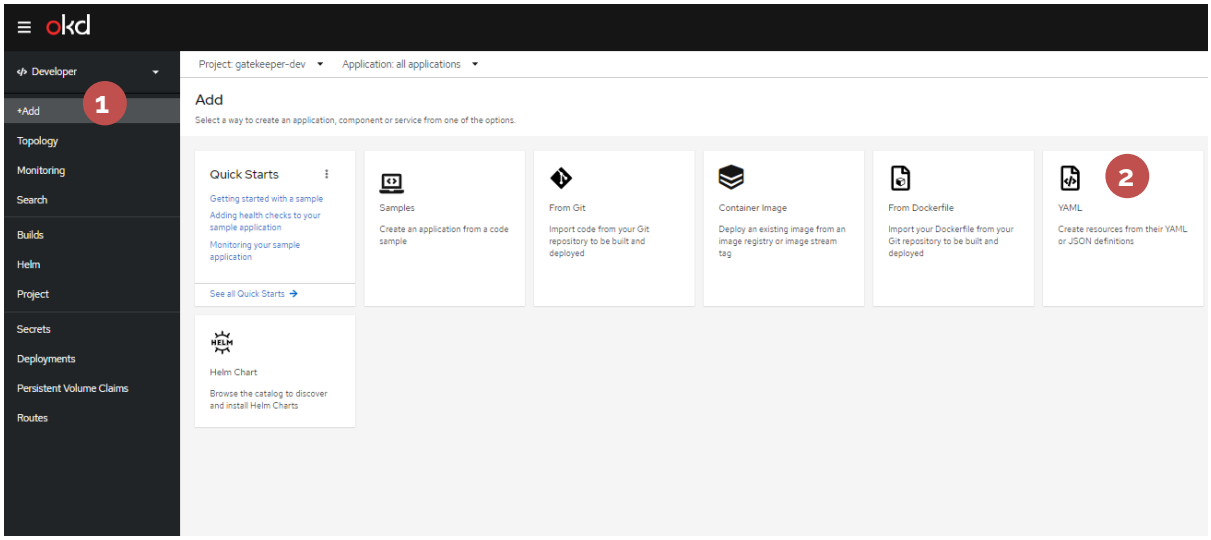


Figure 71 OKD: add resource menu

To continue, in Figure 72 are reported the next two steps of the procedure. In the step number three (red circle with number 3 inside), user paste every YAML file content in the YAML editor following the proposed order (s. 3.1.1). Then, for every described artifact, after pasted it in the YAML editor, user click the "Create" button (red circle with number 4).

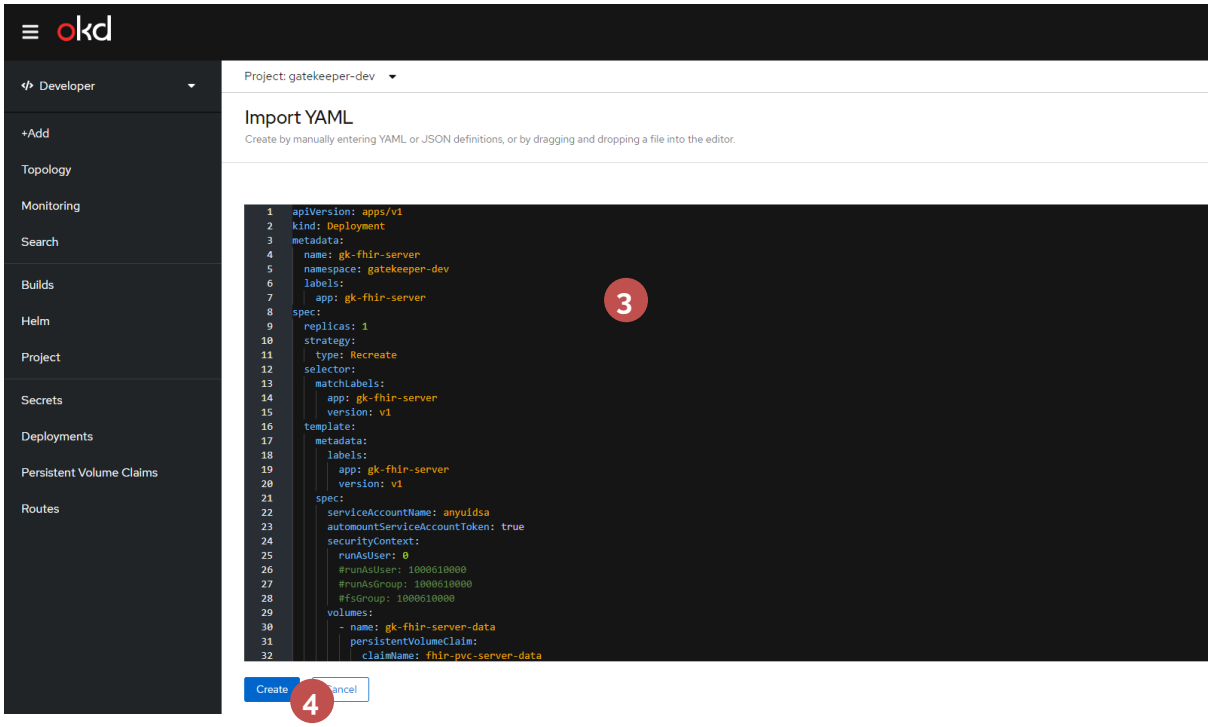


Figure 72 OKD: YAML resource editor

After that all components are deployed and started, you can group them into an application group, as shown in Figure 73.

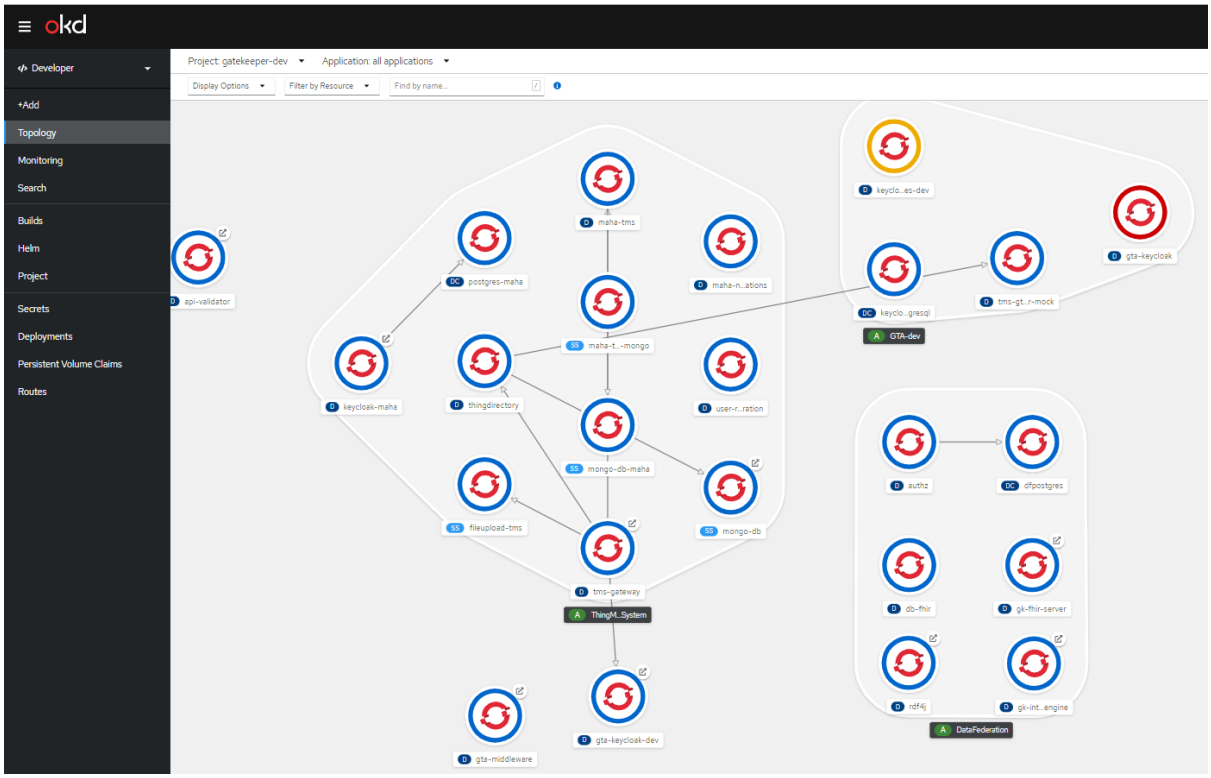


Figure 73 OKD: Platform gatekeeper-dev namespace screenshot

In a nutshell, the steps for the OKD Installation procedure, can be summarised as:

- Click the "+Add" button on OKD left side nav (1)
- Click the "YAML" tile (2)
- Paste every YAML file content in the YAML editor following the proposed order (3)
- For every described artifact, after past it in YAML editor, click the "Create" button (4)

3.1.9 OKD How to test the deployed artifacts [NEW]

Once OKD installation procedure is completed, it is possible to test the deployed artifacts. Here below there are some screenshots in order to lead the user in the testing procedure. At the ending of the paragraph, a bullet point is reported to briefly sum up all the steps.

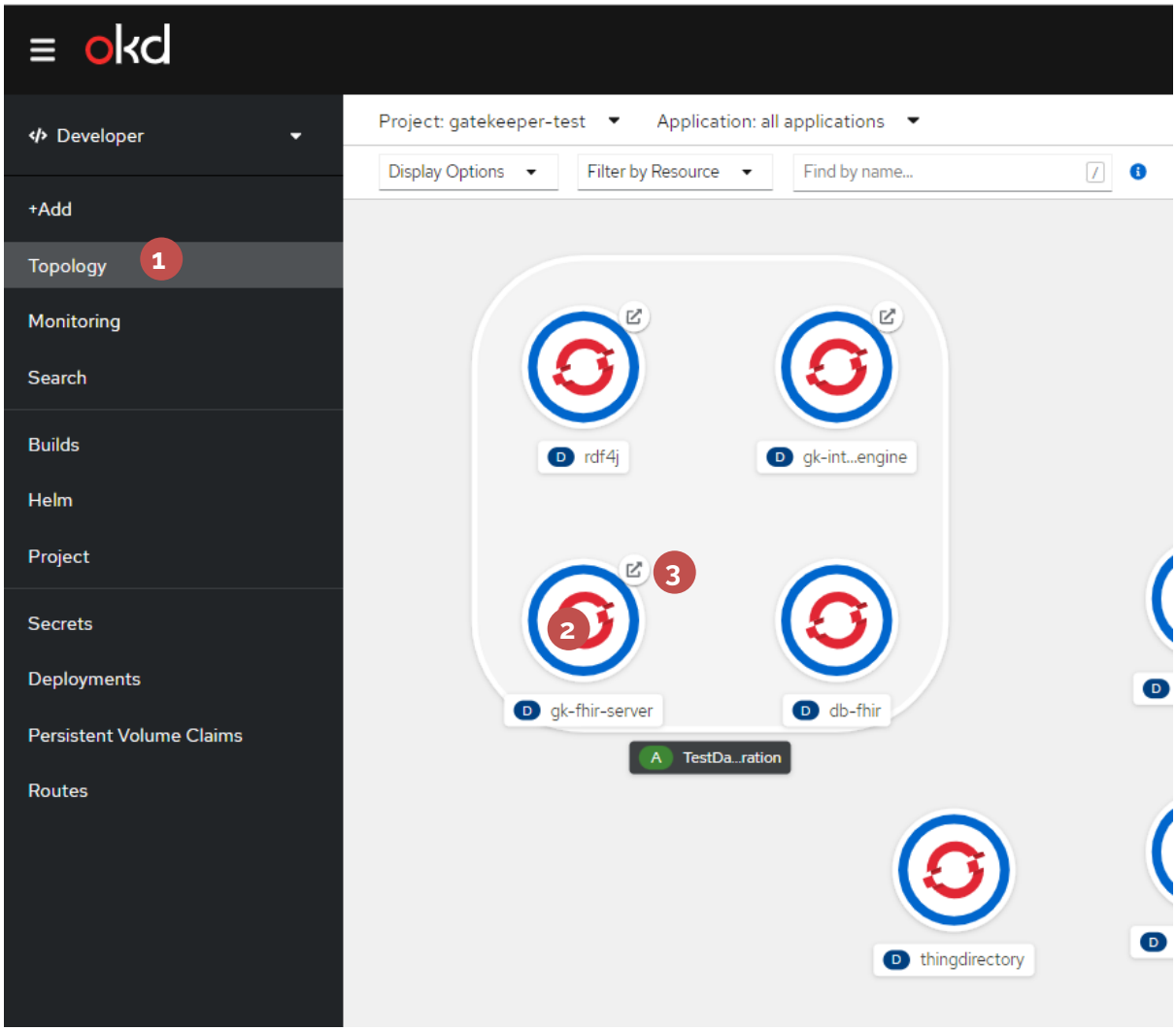


Figure 74 OKD: Platform gatekeeper-test namespace screenshot focused on TestDataFederation pod group

To sum up,

- Click on the "Topology" link from the left side-nav (1)

- Check if all deployed pods are up and running
- If you would like some details about a specific pod, you can click on it (2) and inspect all details in the panel that appear on the right side of the page (more details in the Figure 76)
- If the pod has an associated service, you can see a link on it (3). Then you can click on this link and, if the service has a web page and not only a REST API, you can see the application web page, as shown as example in Figure 75.

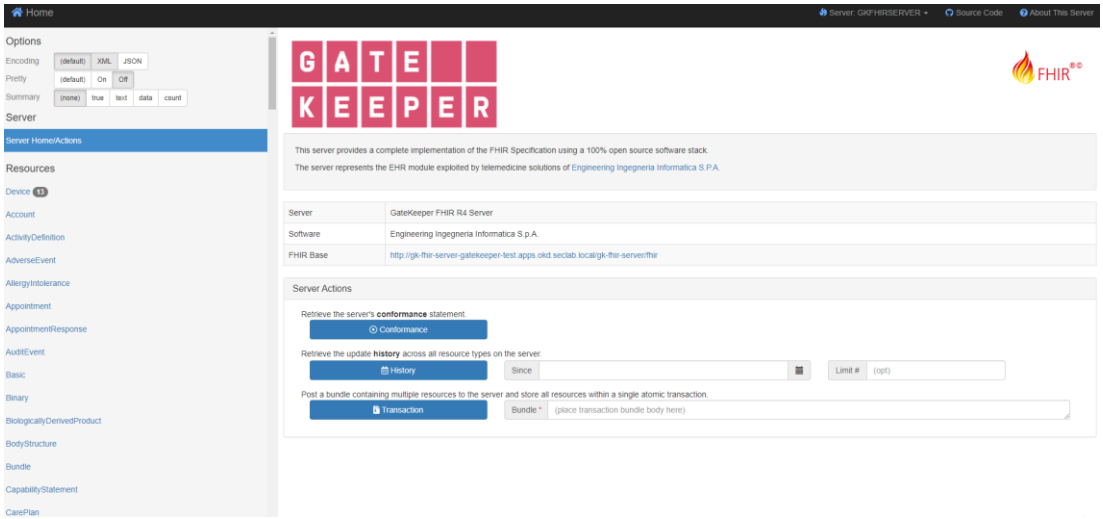


Figure 75 Web application relative to pod linked route

Figure 76 shows some pod details, as described above, relative to gk-fhir-server. In particular, on the top of the pod is reported the name of the "OKD Deploy artifact" (red circle with number 1); then, immediately under the name there is the '*Health Checks*' tile about this deploy (red circle with number 2); below, in '*Pods*' section there are three tabs and in the second one '*Resources*' is represented information about 'Associated POD instance' (red circle with number 3); in '*Services*' section (red circle with number 4) is reported information about the 'Linked service' to this pod; finally, '*Routes*' section (red circle with number 5) gives information on the 'Linked route' associated to this pod.

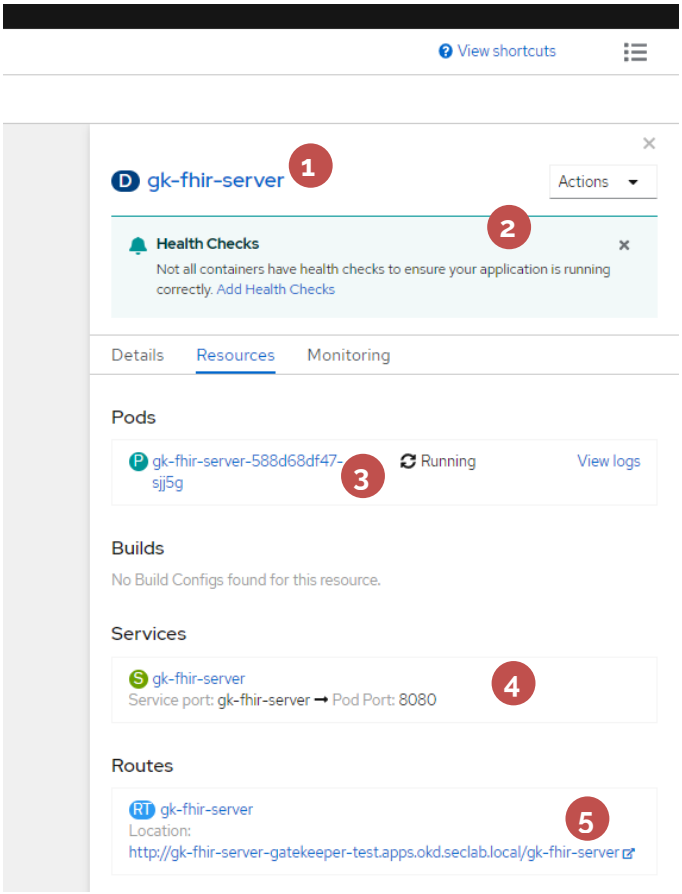


Figure 76 OKD Platform: pod details

To summarise, there are five main details linked to pod details:

- Name of the "OKD Deploy artifact" (1)
- Health check about this deploy (2)
- Associated POD instance (3)
- Linked service (4)
- Linked route (5)

Finally, to collect bugs or issues detected by the pilot through the test, it has been used Slack [24]. In fact, it is a collaboration hub with several features that make easy to contact a teamwork and collaborate with it — whether in the same office or especially around the world (as GK project is)— and also to stay aligned with new development, updates and so on (s. Figure 77).

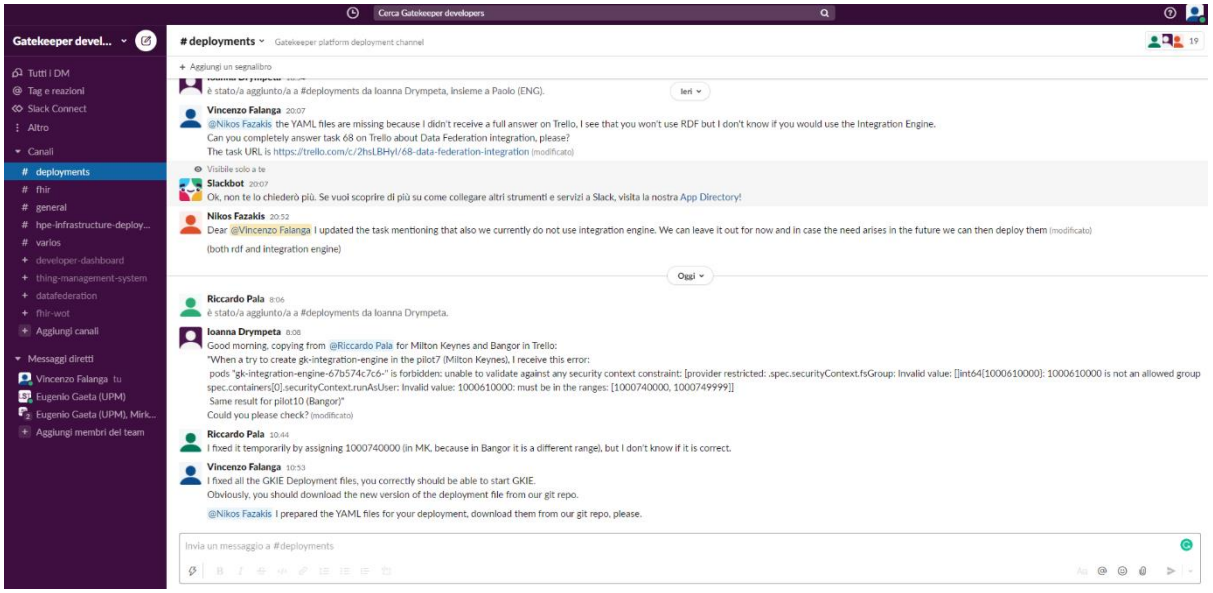


Figure 77 Slack tool: a general overview of its usage in GK project

3.1.10 CI and CD for Data Federation: Jenkins [NEW]

During the last period in which the deliverable was drawn up, HPE made available Jenkins and Git for continuous integration (CI) and continuous development (CD) for Data Federation.

Jenkins [30] is an open source automation server that provides plugins to support building, deploying and automating projects while Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

ENG made up and tested CI/CD chain for the GK-FHIR Server version 4 and GK-IE components. ENG will also support the GK partners regarding automation processes of the other DF components probably after the deliverable release and it will customise YAML files so that these ones will make automatic the build and integration process.

4 Conclusion

This deliverable is providing the final version of Data Federation & Integration as well as the description of how it will interact with the overall Gatekeeper platform. Starting from the new/updated requirements collected during the several remote calls made with each pilot involved in the project, the DFI design has been updated together with the new version of the prototype as a microservice framework, consisting of four main components (gk-integration-engine, gk-fhir-server, gk-rdf-watcher and gk-rdf4j), that expose specific southbound and northbound APIs to collect heterogeneous data coming from electronic health records and devices in order to convert and store such data into FHIR server and RDF repository according to the GK-FHIR-Profile, defined in the task 3.5. In addition, more information about pilots' need have been collected in order to customise and improve the DFI as expected by the projects pilots. Open Callers have been included in this second version of the deliverable and new conversion flow are added in order to support their requests.

Furthermore, such changes have been also useful for improving the performance and integrability with the other platform components and within the HPE deployment environment.

Data Federation & Integration component has been dockerized and deployed on private ENG server to perform some initial integration tests using the Auth 2.0 authentication implemented with keycloak tool. Then, this functionality was in charge of GTA (Gatekeeper Trust Authority) and the TMS (Thing Management System).

Thanks to DFI framework a common semantic model, based on HL7-FHIR, is defined that can be used to retrieve and process persisted data, hiding the problem of having them in heterogeneous formats since they come from different applications where each one uses a different model representation. The main advantage of this approach is that each task can play with Gatekeep data only knowing the defined GK-FHIR-Profile and not the specific models used by the pilot's applications. Moreover, new set of the transformation rules between the data model (task 3.4) to the GK-FHIR-Profile (task 3.5).

5 References

[1] *HAWTIO* https://hawt.io/

[2] *FHIR Implementation Guide* https://build.fhir.org/ig/gatekeeper-project/gk-fhir-ig/index.html

[3] *OpenShift* https://cloud.redhat.com/learn/what-is-openshift

[4] *Apache Camel FHIR Component* . (n.d.). Retrieved from https://camel.apache.org/components/latest/fhir-component.html

[5] *Inotifyd* https://wiki.alpinelinux.org/wiki/Inotifyd

[6] *Docker*. (n.d.). Retrieved from https://www.docker.com/

[7] *Docker Container*. (n.d.). Retrieved from https://www.docker.com/resources/what-container

[8] *fhirtordf*. (n.d.). Retrieved from https://github.com/BD2KOnFHIR/fhirtordf

[9] *HAPI FHIR*. (n.d.). Retrieved from https://hapifhir.io/

[10] *HL7-FHIR*. (n.d.). Retrieved from https://www.hl7.org/fhir/

[11] *keycloak*. (n.d.). Retrieved from https://www.keycloak.org/

[12] *Kubernetes ingress*. (n.d.). Retrieved from https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[13] *Maria DB*. (n.d.). Retrieved from https://mariadb.org/

[14] *Microk8s*. (n.d.). Retrieved from https://microk8s.io/

[15] *Resource Description Framework (RDF) Model and Syntax*. (n.d.). Retrieved from http://www.w3.org/RDF/Group/WD-rdf-syntax/

[16] *RML*. (n.d.). Retrieved from https://rml.io/docs/

[17] *RML*. (n.d.). Retrieved from https://rml.io/specs/rml/

[18] *Spring Java*. (n.d.). Retrieved from https://spring.io/

[19] *Tomcat*. (n.d.). Retrieved from http://tomcat.apache.org/

[20] *URI*. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

[21] *W3C*. (n.d.). Retrieved from https://www.w3.org/

[22] *HPE Guide* "T4.1 How to access HPE GK Data Centre"

[23] *Trello* https://help.trello.com/article/708-what-is-trello

[24] *Slack* https://slack.com/intl/en-au/help/articles/115004071768-What-is-Slack-

[25] *T3.5 D3.9* 'D3.5-GATEKEEPER HL7 FHIR optimization for IoT and Smart and healthy living environments'

[26] *Google Fit* https://www.google.com/fit/

[27] *Fitbit* https://www.fitbit.com/global/eu/home

[28] *iHealth* https://ihealthlabs.eu/it/

[29] *Biobeat* https://www.bio-beat.com/

[30] *Jenkins* https://www.jenkins.io/
[31] *Eclipse RDF4J* https://rdf4j.org/
[32] *RDF4J workbench* https://rdf4j.org/documentation/tools/server-workbench/
[33] *Activage project* http://www.activageproject.eu/
[34] *Open mHealth* https://www.openmhealth.org/
[35] *SenML format* *https://datatracker.ietf.org/doc/html/rfc8428*

Appendix A [NEW]

Here below are reported the mapping rules analysed with the support of the data models related accountable pilot and all the FHIR data types ValueSet used in the mapping process defined by HL7.

A.1 Data models to HL7-FHIR mapping rule, terminologies and FHIR [NEW]

A.1.1 Puglia Data Model to HL7-FHIR

Patient (FHIR::Patient)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
patient_id	String	The identifier of the patient		Patient.identifier[0].value	Patient.identifier[0].system=GK-ID.PATIENT.PUGLIA.SYSTEM	
clinicalExaminetion	ClinicalExaminetion			Observation		

ClinicalExamination (FHIR::Observation | Condition)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					Observation.status="final" Observation.category = GK-VS.OBSERVATION.CATEGORY.VITAL_SIGN Observation.subject=<reference to the patient> (Extension) Observation.GK-EXT..patientAge = <patientAge>	This mapping is applied to all fields of the entity ClinicalExamination that are mapped on Observation resource. An Observation resource is created for each attribute except data and patientAge where these values are added to each Observation or Condition created for this entity.
date	Data	Date of the performed the clinical examination			Observation.effective	This field is set to each observation created in for this entity.

glycosylated Hemoglobin	float	Glycosylated Hemoglobin - outcome variable Unit: mmol/mol	glycosylatedHemoglobin	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIMOLE_PER_MOLE Observation.code = GK-VS.OBSERVATION-CODE.GLYCOSILATED_EMOGLOBIN	
totalCholesterol	float	Total Cholesterol Unit: mg/dL	totalCholesterol	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER Observation.code = GK-VS.OBSERVATION-CODE.TOTAL_CHOLESTEROL	
hdl	float	HDL Unit: mg/dL	hdl	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER Observation.code = GK-VS.OBSERVATION-CODE.HDL	
ldl	float	LDL Unit: mg/dL	ldl	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY	

					Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER Observation.code = GK-VS.OBSERVATION-CODE.LDL	
triglycerides	float	TRIGLYCERIDES Unit: mg/dL	triglycerides	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER Observation.code = GK-VS.OBSERVATION-CODE.TRIGLYCERIDES	
tcHdl	float	TC/HDL Unit: mg/dL	tcHdl	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.code = GK-VS.OBSERVATION-CODE.TC_HDL Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER	
serumCreatinine	float	Serum Creatinine Unit: mg/dL	serumCreatinine	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER Observation.code = GK-VS.OBSERVATION-CODE.SERUM_CREATININE	

albuminuria Creatininuria Ratio	float	Albuminuria/Creatininuria ratio Unit: mg/g	albuminuriaCreatininuriaRatio	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_GRAM Observation.code = GK-VS.OBSERVATION-CODE.ALBUMINURIA_CREATININURIA_RATIO	
gptAlt	float	GPT/ALT Unit: U/l	gptAlt	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.ENZYME_UNIT_PER_LITER Observation.code = GK-VS.OBSERVATION-CODE.GPT_ALT	
gotAst	float	GOT/AST Unit: U/l	gotAst	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.ENZYME_UNIT_PER_LITER Observation.code = GK-VS.OBSERVATION-CODE.GOT_AST	
gammaGt	float	Gamma GT Unit: UI/l	gammaGt	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.INTERNATIONAL_UNIT_PER_LITER	

					Observation.code = GK-VS.OBSERVATION-CODE.GAMMA_GT	
alkalinePhosphatase	float	Alkaline Phosphatase Unit: UI/l	alkalinePhosphatase	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.INTERNATIONAL_UNIT_PER_LITER Observation.code = GK-VS.OBSERVATION-CODE.ALKALINE_PHOSPHATASE	
uricAcid	float	Uric acid Unit: mg/dL	uricAcid	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM. Observation.code = GK-VS.OBSERVATION-CODE.URIC_ACID	
eGFR	float	eGFR Unit: ml/min/1.73m2	eGFR	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER Observation.code = GK-VS.OBSERVATION-CODE.E_GFR	
nitrites	Boolean	Nitrites Present (true)/Absent (false)	nitrites	Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.LABORATORY Observation.value is Boolean	

					Observation.code = GK-VS.OBSERVATION-CODE.NITRITES	
patientAge	Int	Patient age		(Extension) Observation.GK-EXT..patientAge = <patientAge> Condition.onset	Condition.onset is Age	For the Observation is value is mapped as an extension while for Condition this value is mapped on Condition.onset where onset is of the type Age
systolicPressure	Int	Systolic pressure Unit: mmHg		Observation.component[0].value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGN Observation.code = GK-VS.OBSERVATION-CODE.BLOOD_PRESSURE_PANEL_WITH_ALL_CHILDREN_OPTIONAL Observation.component[0].value is Quantity Observation.component[0].value = GK-VS.UCUM.MILLIMETER_OF_MERCURY Observation.component[0].code = GK-VS.OBSERVATION-CODE.SYSTOLIC_BLOOD_PRESSURE	For these two measurements is created a unique Observation resource
diastolicPressure	Int	Diastolic Pressure Unit: mmHg		Observation.component[1].value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGN Observation.component[1].value is Quantity Observation.component[1].value = GK-VS.UCUM.MILLIMETER_OF_MERCURY Observation.component[1].code = GK-VS.OBSERVATION-	

					CODE.DIASTOLIC_BLOOD_PRESSURE	
weight	float	Weight Unit: Kg		Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGN Observation.value is Quantity Observation.value = GK-VS.UCUM.WEIGHT Observation.code = GK-VS.OBSERVATION-CODE.BODY_WEIGHT	
height	float	Heigth Unit: m		Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGN Observation.value is Quantity Observation.value = GK-VS.UCUM.METER Observation.code = GK-VS.OBSERVATION-CODE.BODY_HEIGHT	
yearsOfDiagnosedDiabetes	int	Years of Diagnosed Diabetes Unit: year		Observation.value.value	Observation.category=GK-VS.OBSERVATION.CATEGORY.SURVEY Observation.value is Quantity Observation.value = GK-VS.UCUM.YEARS Observation.code = GK-VS.OBSERVATION-CODE.YEARS_WITH_DIABETES	
hepaticSteatosis	Boolean	Hepatic statosis Present (true)/Absent (false)		Condition.code=GK-VS.CONDITION-CODE.STEATOSIS_LIVER	Condition.suject=<reference to patient> Condition.onset=<patientAge>	

					Condition.recordedData=<date>	
hypertension	Boolean	Hypertension Present (true)/Absent (false)		Condition.code= GK-VS.CONDITION-CODE.HYPERTENSIVE_DISORDER	Condition.suject=<reference to patient> Condition.onset=<patientAge> Condition.recordedData=<date>	
heartFailure	Boolean	Heart failure Present (true)/Absent (false)		Condition.code= GK-VS.CONDITION-CODE.HEART_FAILURE	Condition.suject=<reference to patient> Condition.onset=<patientAge> Condition.recordedData=<date>	
bpco	Boolean	BPCO Present (true)/Absent (false)		Condition.code= GK-VS.CONDITION-CODE.CHRONIC_OBSTRUCTIVE_LUNG	Condition.suject=<reference to patient> Condition.onset=<patientAge> Condition.recordedData=<date>	
chronicKidneyDisease	Boolean	Chronic Kidney disease Present (true)/Absent (false)		Condition.code= GK-VS.CONDITION-CODE.CHRONIC_KIDNEY_DISEASE	Condition.suject=<reference to patient> Condition.onset=<patientAge> Condition.recordedData=<date>	
ischemicHeartDisease	Boolean	Ischemic Heart disease Present (true)/Absent (false)		Condition.code= GK-VS.CONDITION-CODE.ISCHEMIC_HEART_DISEASE	Condition.suject=<reference to patient> Condition.onset=<patientAge> Condition.recordedData=<date>	

A.1.2 Aragon Data Model to HL7-FHIR

Patient/Practitioner (FHIR::Patient | ResearchSubject| ResearchStudy)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					ResearchStudy.status("active") ReserachSubject.status("onstudy") ReserachSubject.individual(Reference(participanti_id)) ReserachSubject.study(Reference(this.ResearchStudy))	
participant_id	String	The unique identifier of the patient.	N.A	Patient.identifier[0].value	Patient.identifier[0].system=GK-VS.GK-ID.PATIENT.ARAGON.SYSTEM	
date	Date	Date of diagnose	dd/mm/yyyy		Condition.recordedDate	
year_birth	Numeric	Year of birth	yyyy		Patient.birthDate	
gender	String	The gender of the participant.	FHIR : http://hl7.org/fhir/gender-identity		Patient.gender	
enter_date	Date	Date of recruitment / entry in the project	dd/mm/yyyy		ResearchSubject.period.start	
exit_date	Date	Date of exit / drop out/ drop off of the project	dd/mm/yyyy		ResearchSubject.period.end	
primary_disease	String	Primary disease at enrolment	ICD-9, ICD-10 and ICPC-2	Condition.code.coding.code	Condition.subject=Reference(participant_id) Condition.category=GK-VS.CONDITION-CATEGORY.PRIMARY_DISEASE Condition.code.cofding.system="http://terminology.hl7.org/CodeSystem/condition-category"	

health_area	String	Health area of the participant (Administrative / Health care center)	ARAGON|01] , [ARAGON|02	ReserachStudy.location.coding.code	ReserachStudy.location.coding.system=""https://www.aragon.es/CodeSystem/health-area""	
death	Date	Date of death or 00/00/0000 if death did not occur	dd/mm/yyyy		Patient.deceased	

Social assessment (FHIR::Observation | Patient)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
participant_id	String	The unique identifier of the patient.	N.A	Observation.subject=Reference(participant_id)		
date	Date	Date of record	dd/mm/yyyy	Observation.effective	Observation.effective is dateTime	
people_living	Numeric	Number of people older than 18 living in the house hold (patient excluded)	N/A	Observation.value.value	Observation.status = GK-VS.OBSERVATION.STATUS.FINAL Observation.subject=Reference(participant_id) Observation.category= GK-VS.OBSERVATION-CATEGORY.SOCIAL_HISTORY Observation.code = GK-VS.OBSERVATION-CODE.PEOPLE_LIVING Observation.value is Quantity	
marital_status	Char	Whether the participant is married or not	http://hl7.org/fhir/ValueSet/marital-status	Patient.maritalStatus		

Habits (FHIR::Observation)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
participant_id	String	The unique identifier of the patient.	N.A	Observation.subject =Reference(participant_id)		
date	Date	Date of record	dd/mm/yyyy	Observation.effective	Observation.effective is dateTime	
tobacco-use	String	Number of cigarettes per day during the last month	https://loinc.org/72166-2/	Observation.value	Observation.status =GK-VS.OBSERVATION.STATUS.FINAL Observation.subject=Reference(participant_id) Observation.category= GK-VS.OBSERVATION-CATEGORY.SOCIAL_HISTORY Observation.code = GK-VS.OBSERVATION-CODE.TOBACCO_SMOKING_STATUS Observation.value is String	
alcohol-consumption	Numeric	Number of days having an alcoholic drink during the last month	1 - Daily 2 - (5-6) days per week 3 - (3-4) days per week 4 - (1-2) days per week 5 - (1-3) days per month 6 - Less than once a month	Observation.value	Observation.status =GK-VS.OBSERVATION.STATUS.FINAL Observation.subject=Reference(participant_id) Observation.category= GK-VS.OBSERVATION-CATEGORY.SOCIAL_HISTORY Observation.code = GK-VS.OBSERVATION-CODE.HISTORY_OF_ALCOHOL_USE Observation.value is String	
physical_activity	Numeric	Level of physical activity during the last month	0 - None (bed) 1 - Very Low (home)	Observation.value	Observation.status =GK-VS.OBSERVATION.STATUS.FINAL	

			2 - Low (basic needs outside home) 3 - Moderate (walk < 10000 steps) 4 - Intense (> 10000 steps) 5 - Vigorous (sport)		Observation.subject=Reference(participant_id) Observation.category= GK-VS.OBSERVATION-CATEGORY.SOCIAL_HISTORY Observation.code = GK-VS.OBSERVATION-CODE.EXERCISE_ACTIVITY Observation.value is String	

Clinical Activity (Admissions – Hospitalization) (FHIR::Encounter)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					Encounter.status=GK-VS.ENCOUNTER-STATUS.FINAL Encounter.class=GK-VS.ENCOUNTER-CLASS.INPATIENT_ENCOUNTER	
participant_id	String	The unique identifier of the patient.	N.A	Encounter.subject=Reference(participant_id)		
date	Date	Date of record	dd/mm/yyyy	(Extension) Encounter.GK-EXT.registeredDate		
admission_date	Date	Date of admission	dd/mm/yyyy	Encounter.period.start		
discharge_date	Date	Date of discharge	dd/mm/yyyy	Encounter.period.end		
indication	String	Main reason for admission	ICD-10	Encounter.reasonCode.[0].coding.code	Encounter.reasonCode.[0].coding.system=<link ICD_10>	

admission_planned	Numeric	Admission planned / unplanned	1 - Planned 2 - Unplanned 999 - Missing answer	Encounter.type		Enconter.type is filled with one of the values listed in the table GK-VS.ENCONTER-TYPE
origin	Numeric	Place of residence before admission	1 - Home 2 - Nursing home 4 - Other 999 - Missing answer	Encounter.hospitalization.admitSource		Encounter.hospitalization.admitSource is filled with of the values listed in the table GK-VS.ENCOUNTER-HOSPITALIZATION
destination	Numeric	Place of residence after admission	1 - Home 2 - Nursing home 3 - Death 4 - Other 999 - Missing answer	Encounter.hospitalization.dischargeDisposition		Encounter.hospitalization.admitSource is filled with of the values listed in the table GK-VS.ENCOUNTER-HOSPITALIZATION

Clinical Activity (Consultation) (FHIR::Encounter)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					Encounter.status=GK-VS.ENCOUNTER-STATUS.FINAL	
participant_id	String	The unique identifier of the patient.	N.A	Encounter.subject=Reference(participant_id)		
date	Date	Date of record	dd/mm/yyyy	(Extension) Encounter.GK-EXT..registeredDate		
contact_date	Date	Date of contact	dd/mm/yyyy	Encounter,period.start		
contact_person	String	Who made the care intervention	1 - GP 2 - Nurse	Encounter.participant[0].type		Encounter.participant.type is one of the value present in the table GK-VS.ENCOUTER-PARTICIPANT-TYPE

			3 - HCP Specialist 4 - Emergency professional 5 - Social worker 6 - Other social profile 7 - Joint intervention 8 - Missing data			
contact_type	String	Type of contact	1 - PC premises 2 - Hospital premises 3 - PC Emergency 4 - Hospital Emergency 5 - Telephone 6 - Home visit 7 - Written (email/sms) 8 - Missing data	Encounter.class		PC premises = Encounter.class=GK-VS.ENCOUNTER-CLASS.PC_PREMISES hospital premises = Encounter.class=GK-VS.ENCOUNTER-CLASS.INPANTIENT_ENCOUNTER PC emergency = Encounter.class=GK-VS.ENCOUNTER-CLASS.PC_EMERGENCY hospital emergency = Encounter.class=GK-VS.ENCOUNTER-CLASS.EMERGENCY telephone = Encounter.class=GK-VS.ENCOUNTER-CLASS.VIRTUAL home visit = Encounter.class=GK-VS.ENCOUNTER-CLASS.HOME_HEALTH written = Encounter.class=GK-VS.ENCOUNTER-CLASS.VIRTUAL
contact_planned	Numeric	Admission planned / unplanned	1 - Planned 2 - Unplanned 999 - Missing answer	Encounter.type		Enconter.type is filled with one of the values listed in the table GK-VS.ENCONTER-TYPE
service	String	Name of the service where the consultation takes place	List codes: 1. PRIMARY CARE CONSULTATIONS 2. PRIMARY CARE EMERGENCIES	Encounter.serviceType.text		

			3. SPECIALIZED CARE CONSULTATIONS 4. SPECIALIZED CARE - EMERGENCY ROOM 5. SOCIAL CARE 6. OTHER Note. Codes for services under the regional EHR are being revised so in case there is a stable version for coding services, it will be used in this field.			

Prescribed Medication (FHIR::MedicationRequest)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					MedicationRequest.intent="order"	
participant_id	String	The unique identifier of the patient.	N.A	Medication.subject=Reference(participant_id)		
date	Date	Date of record	dd/mm/yyyy		MedicationRequest.authoredOn	
status	Binary	Status of the prescription	0 – Inactive 1 - Active	MedicationRequest.status		if status == 0 then MedicationRequest.status="stopped" else if status == 1 then MedicationRequest.status="active"
begin_date	Date	start date of treatment	NA	MedicationRequest.dosageInstruction.timing.bounds.start	MedicationRequest.dosageInstruction.timing.bounds is Period	
end_date	Date	00/00/0000 if end_date is not defined	NA	MedicationRequest.dosageInstruction.timing.bounds.start	MedicationRequest.dosageInstruction.timing.bounds is Period	
active principle	String	Name of the active principle	ATC coding	MedicationRequest.medication.text	MedicationRequest.medication is CodeableConcept	The display of ATC is not provided

sequence	String	Moments of day for medicine intake	String indicating moments of day in which medicine should be taken E.g.: 2 pills at breakfast, or 1 pulse at supper data format: NUMBER | PRESENTATION |FREQUENCY where FREQUENCY can be BREAKFAST|LUNCH|DINNER)	MedicationRequest.dosageInstruction.text		
calendar	String	Weekdays to apply sequence	String with weekdays in English in which medicine should be taken data format, if more than one: mon|tue|sat	MedicationRequest.dosageInstruction.timing.repeat.dayOfWeek		
Dose	Numeric	Value of the dose to be taken		MedicationRequest.dosageInstruction.doseAndRate,dose.value	MedicationRequest.dosageInstruction.doseAndRate,dose is Quantity	
Unit	String	Code of measurement	E.g. mg -> milligram	MedicationRequest.dosageInstruction.doseAndRate,dose.value	MedicationRequest.dosageInstruction.doseAndRate,dose.unit	The code of unit of measure is not provided.

Clinical variables value (FHIR::Observation)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					Observation.status="final"	

					Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGN	
participant_id	String	The unique identifier of the patient.	N.A	Observation.,subject=Reference(participanti_id)		
clinical_variable	String	Name of the clinical variable	LOINC CODES descriptive string	Observatiob.code.text		
Value	String	Value captured		Observation.value.value	Qbservation.value is Quantity	
Unit	String	Unit Code	descriptive string	Observation,value.unit		

Symptom (FHIR:: Observation)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					Observation.category[0]=GK-VS.OBSERVATION-CATEGORY.SURVEY	
participant_id	String	The unique identifier of the patient.	N.A	Observation.,subject=Reference(participanti_id)		
Date	Date of measurement	Date of the collection of symptoms		Observation.effective	Observation.effective is DateTime	
Symptom	String	Name of the symptom	descriptive string	Observation.code.text		
Value / Intensity	String		0 – No 1 – Low 2 - Mild 3– Moderate	Observation.value	Observation,value is String	

			4 - Severe 5 - Extreme			

Form and questionnaire (PROMS) (FHIR::QuestionnaireResponse)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					QuestionnaireResponse.status ="completed"	
participant_id	String	The unique identifier of the patient.	N.A	QuestionnaireResponse.subject=Reference(participant_id)		
completion_date	Date	Date of record	dd/mm/yyyy	QuestionnaireResponse.authored		
record_id	Numeric	Identification of the record that contains all the answers to the same questionnaire		QuestionnaireResponse.identifier.value	QuestionnaireResponse.identifier.system=GK-ID.QUESTIONNAIRE_RESPONSE.QUESTIONNAIRE_RESPONSE_ARAGON	
questionnaire_id	String	Identification of the questionnaire	The list of questionnaires / scales will be published later and may include but not be restricted to Zarit, Barthel, PAM, Barber, E5QD,...	QuestionnaireResponse.questionnaire		
question_id	Numeric	Identification of the number of the question to which the answer corresponds to	N/A	QuestionnaireResponse.item.linkId		
answer	Numeric	Answer to the question in the specific questionnaire	Specific codes apply to each questionnaire	QuestionnaireResponse.item.answer.value	QuestionnaireResponse.item.answer.value is String	

Comorbidity (FHIR::CONDITION)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
					Condition.catergory=GK-VS.CONDITION-CATEGORY.COMORBIDITY	
participant_id	String	The unique identifier of the patient.	N.A	Condition.,subject=Reference(participanti_id)		
date	Date	Date of record	dd/mm/yyyy	Condition.recorded Date		
comorbidity	String	Secondary disease at enrolment	ICD-10 or ICPC-2 Note: Comorbidity does not have an specific code in ICD-10/9, the comorbidity is shown when a patient has multiple codifications	Condition.code.text		
initial_date	Date	Onset / record of the episode		Condtion.onset.start		
end_date	Date	End date of the episode 00/00/0000 if not specified		Condtion.onset.end		
active	Binary		0 – NO / 1 - YES	Condition.clinicalStatus		if active==0 then Condition.clinicalStatus="active" else if active==1 then Condition.clinicalStatus="inactive"

A.1.3 Poland Data Model to HL7-FHIR

Patient (FHIR::Patient)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
ID	String	The unique identifier of the patient.	N.A	Patient.identifier[0].value	Patient.identifier[0].system=GK-ID.PATIENT.POLAND	
email	String	The email of the patient	FHIR: Patient.telecom	Patient.telecom[0].value	Patient.telecom[0].code="email"	
firstname	String	The first name of the patient	N.A.	Patient.name[0].family		
lastname	String	The last name of the patient	N.A.	Patient.name[0].given[0]		
birthdate	Date	The date of birth of the patient	FHIR: Patient.birthDate	Patient.birthDate		
gender	String	The gender of the patient.	FHIR: Patient.gender	Patient.gender		
language	String	The system language that patient wants to use the system	LanguageCode (PL/EN)/EN)	Patient.communication.language	If language="EN" then Patient.communication[0].language.coding[0].code = "en" Patient.communication[0].language.coding[0].display = "English"	

					Patient.communication[0].language.coding[0].system = "http://hl7.org/fhir/ValueSet/languages" If language="PL/EN" then Patient.communication[0].language.coding[0].code = "pl" Patient.communication[0].language.coding[0].display = "Polish" Patient.communication[0].language.coding[0].system = "http://hl7.org/fhir/ValueSet/languages" Patient.communication[1].language.coding[0].code = "en" Patient.communication[1].language.coding[1].display = "English" Patient.communication[1].language.coding[0].system = "http://hl7.org/fhir/ValueSet/languages"	

Observation (FHIR::Observation | Condition)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
				(observation|Condition).code	If Observation.code = "33747003" or "85354-9" or "8480-6" or "8462-4" or "258158006" or, "27113001" or "713137006" or "50373000" or "248365001" or"60621009" or"366319001" or "256668009" or	

					"32457005" then Observation is Mapped to Observation If Observation.code ="I11" ot "I50" or "E78.0" or "E11" or "E03" or "J44" or "N18" the Observation is Mapped to Condition	
ID	Numeric	The unique identifier of the Observation	N.A	N/A		
patientID	Reference to patient unique id	Who and/or what the observation is about	Unique patient ID	(observation|Condition).subject	(observation|Condition).subject is Patient	
datetime	datetime	Clinically relevant time for observation	Timestamp	Observation.effective or Condition.onset	Observation.effective is dateTime Condition.onset is dateTime	
Code	FHIR: CodeableConcept	Type of observation (code / type)	SNOMED Code List, ICD-10 List	Observation.code or Condition.code		See tables Poland SNOMED codes and Poland ICD-10 codes
Category	FHIR: CodeableConcept	Classification of type of observation	http://hl7.org/fhir/observation-category	N/A		

value [value, unity]	Numeric, String	Observation quantity	N.A.	Observation.value Condition.status		

A.1.3.1 Poland SNOMED codes

CODE	FHIR MAPPING
33747003	Observation.code = GK-VS.OBSERVATION-CODE.BLOOD_GLUCOSE Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS Observation.value is Quantity Observation.value = GK-VS.UCUM.MILLIGRAM_PER_DECILITER
85354-9	Observation.code = GK-VS.OBSERVATION-CODE. BLOOD_PRESSURE_PANEL_WITH_ALL_CHILDREN_OPTIONAL
8480-6	Observation.component.code = GK-VS.OBSERVATION-CODE. SYSTOLIC_BLOOD_PRESSURE
8462-4	Observation.component.code = GK-VS.OBSERVATION-CODE. DIASTOLIC_BLOOD_PRESSURE
258158006	Observation.code = GK-VS.OBSERVATION-CODE.SLEEP Observation.category=GK-VS.OBSERVATION.CATEGORY.ACTIVITY Observation.value is Quantity Observation.value = GK-VS.UCUM.MINUTES

27113001	Observation.code = GK-VS.OBSERVATION-CODE.BODY_WEIGHT Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS Observation.value is Quantity Observation.value = GK-VS.UCUM.KG
713137006	Observation.code = GK-VS.OBSERVATION-CODE.STRESS Observation.category=GK-VS.OBSERVATION.CATEGORY.ACTIVITY Observation.value is Quantity Observation.value = GK-VS.UCUM.PERCENT
50373000	Observation.code = GK-VS.OBSERVATION-CODE.BODY_HEIGHT Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS Observation.value is Quantity Observation.value = GK-VS.UCUM.CENTIMETER
248365001	Observation.code = GK-VS.OBSERVATION-CODE.BODY_WAIST Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS Observation.value is Quantity Observation.value = GK-VS.UCUM.BODY_WAIST
60621009	Observation.code = GK-VS.OBSERVATION-CODE.BMI Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS Observation.value is Quantity Observation.value = GK-VS.UCUM.PERCENT
366319001	Observation.code = GK-VS.OBSERVATION-CODE.BODY_FAT Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS

	Observation.value is Quantity Observation.value = GK-VS.UCUM.PERCENT
256668009	Observation.code = GK-VS.OBSERVATION-CODE.BODY_MUSCLE Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS Observation.value is Quantity Observation.value = GK-VS.UCUM.PERCENT
32457005	Observation.code = GK-VS.OBSERVATION-CODE.BODY_WATER Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGNS Observation.value is Quantity Observation.value = GK-VS.UCUM.PERCENT

A.1.3.2 Poland ICD-10 codes

CODE	FHIR MAPPING
I11	Condition.code = GK-VS.CONDITION-CODE.HYPERTENSIVE_DISORDER
I50	Condition.code = GK-VS.CONDITION-CODE.HEART_FAILURE
E78.0	Condition.code = GK-VS.CONDITION-CODE.HYPERCHOLESTEROLEMIA

E11	Condition.code = GK-VS.CONDITION-CODE.TYPE_2_DIABETES_MELLITUS
E03	Condition.code = GK-VS.CONDITION-CODE.HYPOTHYROIDISM
J44	Condition.code = GK-VS.CONDITION-CODE.OTHER_CHROMIC_OBSTRUCTIVE_PULMONARY_DISEASE
N18	Condition.code = GK-VS.CONDITION-CODE.CHRONIC_KIDNEY_DISEASE

A.1.4 ELIOT Hub Collector Data Model to HL7-FHIR

Observation (FHIR::Observation)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
id	String	Logical identity for a resource. This is a temporary identifier according to the standard.		Observation.id		
status	FHIR: code	The FHIR code is equivalent to string and the value is always "final"	status = "final"	Observation.status	Observation.status="final"	

category	FHIR: CodeableConcept	The category of the measure	CodeableConcept.coding[0].system="http://terminology.hl7.org/CodeSystem/observation-category" CodeableConcept.coding[0].value="vital-signs" CodeableConcept.coding[0].value="Vital Signs"	Observation.category	Observation.category=GK-VS.OBSERVATION.CATEGORY.VITAL_SIGN	
code	FHIR: CodeableConcept	This code represents the kind of the measurement	Terms Observation.code[0].text =<free text>	Observation.code		The accepted code are: • GK-VS.OBSERVATION-CODE.BODY_WEIGHT • GK-VS.OBSERVATION-CODE.BODY_HEART_RATE • GK-VS.OBSERVATION-CODE.BODY_BLOOD_PRESSURE_PANEL_WITH_ALL_CHILDREN_OPTIONAL
effectiveDataTime	FHIR: DateTIme	Date consists of Date and time. Example: 2021-01-12T11:19:03.65350112Z		Observation.effective	Observation.effective is DateTime	
contained	FHIR:Reference	The reference to the device that has produced the measurement			Observation.device = <reference to the device>	
value	FHIR: Quantity	For each kind of measure is reported the unit of measure		Observation.value.value	Observation.value is Quantity	if code == GK-VS.OBSERVATION-CODE.BODY_WEIGHT then value = GK-VS:UCUM.KG if code == GK-VS.OBSERVATION-CODE.BODY_HEART_RATE then value = GK-VS.UCUM.HEART_RATE

						if code == GK-VS.OBSERVATION-CODE.BODY_BLOOD_PRESSURE_PANEL_WITH_ALL_CHILDREN_OPTIONAL then value = is not present and are added two components
component	FHIR:Component	FHIR component. for more details. See the FHIR specification.	Component.code = Terms Component.value = Unit of measure	Observation.component[x]	Observation.component[0].code = GK-VS.OBSERVATION-CODE.SYSTOLIC_BLOOD_PRESSURE Observation.component[0].value = GK-VS.UCUM.MILLIMETER_OF_MERCURY Observation.component[1].code = GK-VS.OBSERVATION-CODE.DIASTOLIC_BLOOD_PRESSURE Observation.component[0].value = GK-VS.UCUM.MILLIMETER_OF_MERCURY	These two components are present only for the measure GK-VS.OBSERVATION-CODE.BODY_BLOOD_PRESSURE_PANEL_WITH_ALL_CHILDREN_OPTIONAL

Device (FHIR::Device)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
resourceType	String	The type of the resource	resourceType="Device"	N/A		
id	String	The FHIR code is equivalent to string and the value is always "final"	status = "final"	N/A		

identifier	FHIR:Identifier	The identifier of the device. Only the field value is filled	Identifier[0].value=<identifier>	Device.identifier[0].value	Device.identifier[1].system=GK-ID.DEVICE.MEDISANTE.SYSTEM	

A.1.5 HealthClouldProxy Data Model to HL7-FHIR

HCP does not have its own data model, but it uses the one provided by the reference Health Proxies (i.e. Google Fit, Fitbit, Biobeat, iHealth) and for interoperability reasons it also uses the Open mHealth standard data model. When a conversion is required, HCP normalizes the data retrieved from the Health Proxies in the Open mHealth format.

A.1.6 ENVIRA Data Model to HL7-FHIR

Device (FHIR::Device)

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note
resourceType	String	The type of the resource	resourceType="Device"	N/A		
id	String	The FHIR code is equivalent to string and the value is always "final"	status = "final"	N/A		
identifier	FHIR:Identifier	The identifier of the device. Only the field value is filled	Identifier[0].value=<identifier>	Device.identifier[0].value	Device.identifier[1].system=https://www.gatekeeper-project.eu/sid/envira/device	

Observation

Attribute	Type	Description	Constraint	FHIR Mapping	FHIR Assumption	FHIR Note

status	FHIR: code	The FHIR code is equivalent to string and the value is always "final"	status = "final"	Observation.status	Observation.status="final"	
category	FHIR: CodeableConcept	The category of the measure	CodeableConcept.coding[0].system=" http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper"	Observation.category	Observation.category=GK-VS.OBSERVATION.CATEGORY.LIVING_ENVIRONMENT	
code	FHIR: CodeableConcept	This code represents the kind of the measurement	ENVIRE measures co2 co voc pm10 pm4 pm2.5 pm1 temp hum prb iaqi eiaqi tci covid19RiskIndex	Observation.code		The accepted code are: if (mE.getName().equals("codeName")) • GK-VS.OBSERVATION-CODE.CARBON_DIOXIDE • GK-VS.OBSERVATION-CODE.CARBON_MONOXIDE • GK-VS.OBSERVATION-CODE.VOLATILE_ORGANIC_COMPOUNDS • GK-VS.OBSERVATION-CODE.PM_1 • GK-VS.OBSERVATION-CODE.PM_2_5 • GK-VS.OBSERVATION-CODE.PM_4 • GK-VS.OBSERVATION-CODE.PM_10 • GK-VS.OBSERVATION-CODE.ROOM_TEMPERATURE • GK-VS.OBSERVATION-CODE.HUMIDITIY • GK-VS.OBSERVATION-CODE.PRB • GK-VS.OBSERVATION-CODE.TCI • GK-VS.OBSERVATION-CODE.EIAQI • GK-VS.OBSERVATION-CODE.IAQI • GK-VS.OBSERVATION-CODE.COVID_19_RISK
value	FHIR: Quantity	For each kind of measure is		Observation.value.value	Observation.value is Quantity	

		reported the unit of measure				

A.2 GK-FHIR Data Type [NEW]

A.2.1 FHIR-GK-IDENTIFIERS (GK-ID)

A.2.1.1 PATIENT

PILOT	SYSTEM
PUGLIA	https://www.gatekeeper-project.eu/sid/puglia/patient
ARAGON	https://www.gatekeeper-project.eu/sid/aragon/patient
SAXONY	https://www.gatekeeper-project.eu/sid/saxony/patient
POLAND	https://www.gatekeeper-project.eu/sid/poland/patient

A.2.1.2 DEVICE

PILOT	SYSTEM
MEDISANTE	https://www.gatekeeper-project.eu/sid/medisante/device
DEVICE_HCP	https://www.gatekeeper-project.eu/sid/hcp/device
SAMSUNG	https://www.gatekeeper-project.eu/sid/samsung/device

OPEN CALLER	SYSTEM
ENVIRA	https://www.gatekeeper-project.eu/sid/envira/device

A.2.1.3 QUESTIONNAIRE_RESPONSE

PILOT	SYSTEM
QUESTIONNAIRE_RESPONSE_ARAGON	https://www.gatekeeper-project.eu/sid/aragon/questionnaire-response

A.2.2 FHIR-GK-VALUESETS (GK-VS) [NEW]

A.2.2.1 OBSERVATION-CODE

NAME	System	Code	Display
BODY_WEIGHT	http://loinc.org/	29463-7	Body weight
HEART_RATE	http://loinc.org/	8867-4	Heart rate
BLOOD_PRESSURE_PANEL_WITH_ALL_CHILDREN_OPTIONAL	http://loinc.org/	85354-9	Blood pressure panel with all children optional
SYSTOLIC_BLOOD_PRESSURE	http://loinc.org/	8480-6	Systolic blood pressure
DIASTOLIC_BLOOD_PRESSURE	http://loinc.org/	8462-4	Diastolic blood pressure
BODY_HEIGHT	http://loinc.org/	8302-2	Body height
TOBACCO_SMOKING_STATUS	http://loinc.org/	72166-2	Tobacco smoking status
GLYCOSILATED_EMOGLOBIN	http://loinc.org/	59261-8	Hemoglobin A1c/Hemoglobin.total in Blood by IFCC protocol

TOTAL_CHOLESTEROL	http://loinc.org/	2093-3	Cholesterol [Mass/volume] in Serum or Plasma
HDL	http://loinc.org/	2085-9	Cholesterol in HDL [Mass/volume] in Serum or Plasma
LDL	http://loinc.org/	2089-1	Cholesterol in LDL [Mass/volume] in Serum or Plasma
TRIGLYCERIDES	http://loinc.org/	2571-8	Triglyceride [Mass/volume] in Serum or Plasma
TC_HDL	http://loinc.org/	43396-1	Cholesterol non HDL [Mass/volume] in Serum or Plasma
SERUM_CREATININ	http://loinc.org/	2160-0	Creatinine [Mass/volume] in Serum or Plasma
ALBUMINURIA_CREATININURIA_RATIO	http://loinc.org/	14959-1	Microalbumin/Creatinine [Mass Ratio] in Urine
GPT_ALT	http://loinc.org/	1742-6	Alanine aminotransferase [Enzymatic activity/volume] in Serum or Plasma
GOT_AST	http://loinc.org/	1920-8	Aspartate aminotransferase [Enzymatic activity/volume] in Serum or Plasma
GAMMA_GT	http://loinc.org/	2324-2	Gamma glutamyl transferase [Enzymatic activity/volume] in Serum or Plasma
ALKALINE_PHOSPHATASE	http://loinc.org/	6768-6	Alkaline phosphatase [Enzymatic activity/volume] in Serum or Plasma
URIC_ACID	http://loinc.org/	3084-1	Urate [Mass/volume] in Serum or Plasma
E_GFR	http://loinc.org/	48642-3	Glomerular filtration rate/1.73 sq M.predicted among non-blacks [Volume Rate/Area] in Serum, Plasma or Blood by Creatinine-based formula (MDRD)
NITRITES	http://loinc.org/	5802-4	Nitrite [Presence] in Urine by Test strip
HISTORY_OF_ALCOHOL_USE	http://loinc.org/	63597-9	During the past 30 days, on how many days did you drink one or more drinks of an alcoholic beverage
EXERCISE_ACTIVITY	http://loinc.org/	73985-4	Exercise activity
YEARS_WITH_DIABETES	https://www.phenxtoolkit.org	PX070801190200	PX070801_Diabetes_Mellitus_Year
PEOPLE_LIVING	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	people-living	People living
BLOOD_GLUCOSE	http://snomed.info/sct	33747003	Glucose measurement, blood

	http://loinc.org/	15074-8	Glucose [Moles/volume] in Blood
STEPS	http://loinc.org/	55423-8	Number of steps in unspecified time Pedometer
SLEEP	http://snomed.info/sct http://loinc.org/	258158006 93831-6 93830-8 93829-0 93832-4	Sleep, function Deep sleep duration Light sleep duration REM sleep duration Sleep duration
BODY_WAIST	http://snomed.info/sct http://loinc.org/	248365001 narrower 8280-0	Circumference measure narrower Waist Circumference at umbilicus by Tape measure
BMI	http://snomed.info/sct http://loinc.org/	60621009 is equivalent to 39156-5	Body mass index is equivalent to Body mass index (BMI) [Ratio]
BODY_FAT	http://snomed.info/sct http://loinc.org/	366319001 is related to 73708-0	Body fat observable is related to Body fat [Mass] Calculated
BODY_MUSCLE	http://snomed.info/sct	256668009	Muscle material
BODY_WATER	http://snomed.info/sct	32457005	Body fluid
SMOKING	http://loinc.org/	63629-0	On the number of days you reported you smoked cigarettes during the past 30 days, how many cigarettes did you smoke per day, on average [PhenX]
STRESS	http://snomed.info/sct	713137006	Stress
OXYGEN_SATURATION	http://loinc.org/	59408-5	Oxygen saturation in Arterial blood by Pulse oximetry
RESPIRATORY_RATE	http://loinc.org/	9279-1	Respiratory rate
BODY_TEMPERATURE	http://loinc.org/	8310-5	Body temperature

LIVING_ENVIRONMENT	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	liv-environment	Living Environment Observations
ROOM_TEMPERATURE	http://loinc.org/	60832-3	Room temperature
HUMIDITIY	http://loinc.org/	65643-9	Relative humidity
PRB	http://loinc.org/	76268-2	Pressure.ambient Room
TCI	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	tci	Thermal Comfort Index
EIAQI	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	eiaqi	Environment Indoor Air Quality Index
IAQI	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	iaqi	Indoor Air Quality Index
COVID_19_RISK	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	covid19_risk	Covid 19 Risk Index
CARBON_DIOXIDE	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	co2-conc	Carbon Dioxide concentration (ppm)
CARBON_MONOXIDE	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	co-conc	Carbon Monoxide concentration (ppm)
VOLATILE_ORGANIC_COMPOUNDS	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	voc-conc	Volatile Organic Compounds concentration (ppm)
PM_1	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	pm1-conc	PM 1 particle mass concentration
PM_2_5	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	pm2.5-conc	PM 2.5 particle mass concentration
PM_4	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	pm4-conc	PM 4 particle mass concentration
PM_10	http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper	pm10-conc	PM 10 particle mass concentration

A.2.2.2 OBSERVATION-CATEGORY

NAME	System	Code	Display
VITAL_SIGN	http://terminology.hl7.org/CodeSystem/observation-category	vital-signs	Vital Sign
SOCIAL_HISTORY	http://terminology.hl7.org/CodeSystem/observation-category	social-history	Social History
SURVEY	http://terminology.hl7.org/CodeSystem/observation-category	survey	Survey
ACTIVITY	http://terminology.hl7.org/CodeSystem/observation-category	activity	activity

A.2.2.3 ENCOUNTER-CLASS

NAME	System	Code	Display
INPATIENT_ENCOUNTER	http://terminology.hl7.org/ValueSet/v3-ActEncounterCode	IMP	inpatient encounter
AMBULATORY	http://terminology.hl7.org/ValueSet/v3-ActEncounterCode	AMB	ambulatory
PC_PREMISES	http://terminology.hl7.org/ValueSet/v3-ActEncounterCode	PC_PREMISES	pc premises
PC_EMERGENCY	http://terminology.hl7.org/ValueSet/v3-ActEncounterCode	PC_EMERGENCY	pc emergency
EMERGENCY	http://terminology.hl7.org/ValueSet/v3-ActEncounterCode	EMER	emergency
VIRTUAL	http://terminology.hl7.org/ValueSet/v3-ActEncounterCode	VR	virtual
HOME_HEALTH	http://terminology.hl7.org/ValueSet/v3-ActEncounterCode	HH	home health

A.2.2.4 ENCOUNTER-HOSPITALIZATION

NAME	System	Code	Display
HOME	http://hl7.eu/fhir/ig/gk/CodeSystem/admit-destination	home	Home
NURSING_HOME	http://hl7.eu/fhir/ig/gk/CodeSystem/admit-destination	nursing-home	Nursing home
OTHER	http://hl7.eu/fhir/ig/gk/CodeSystem/admit-destination	other	Other
DEATH	https://www.gatekeeper-project.eu/fhir/CodeSystem/admit-destination	death	Death
MISSING_ANSWER	https://www.gatekeeper-project.eu/fhir/CodeSystem/admit-destination	missing-answer	Missing answer

A.2.2.5 ENCOUNTER-PARTICIPANT-TYPE

NAME	System	Code	Display
GP	http://terminology.hl7.org/CodeSystem/v3-ParticipationType	gp	Gp
NURSE	http://terminology.hl7.org/CodeSystem/v3-ParticipationType	nurse	Nurse
HCP_SPECIALIST	http://terminology.hl7.org/CodeSystem/v3-ParticipationType	hcp-specialist	Hcp specialist
EMERGENCY_PROFESSIONAL	http://terminology.hl7.org/CodeSystem/v3-ParticipationType	emergency-professional	Emergency professional

SOCIAL_WORKER	http://terminology.hl7.org/CodeSystem/v3-ParticipationType	social-worker	Social worker
OTHER_SOCIAL_PROFILE	http://terminology.hl7.org/CodeSystem/v3-ParticipationType	other-social-profile	Other social profile
JOINT_INTERVENTION	http://terminology.hl7.org/CodeSystem/v3-ParticipationType	joint-intervention	Joint intervention

A.2.2.6 ENCOUNTER-TYPE

NAME	System	Code	Display
PLANNED	http://terminology.hl7.org/CodeSystem/encounter-type	planned	Planned
UNPLANNED	http://terminology.hl7.org/CodeSystem/encounter-type	unplanned	Unplanned

A.2.2.7 CONDITION-CODE

NAME	System	Code	Display
STEATOSIS_LIVER	http://snomed.info/sct	197321007	Steatosis of liver
HIGH_BLOOD_PRESSURE	http://snomed.info/sct	38341003	High blood pressure
HEART_FAILURE	http://snomed.info/sct	84114007	Heart failure
CHRONIC_OBSTRUCTIVE_LUNG	http://snomed.info/sct	13645005	Chronic obstructive lung disease
CHRONIC_KIDNEY_DISEASE	http://snomed.info/sct	709044004	Chronic kidney disease
ISCHEMIC_HEART_DISEASE	http://snomed.info/sct	414545008	Ischemic heart disease (disorder)

HYPERCHOLESTEROLEMIA	http://hl7.org/fhir/sid/icd-10	E78.0	Pure hypercholesterolemia
HYPOTHYROIDISM	http://hl7.org/fhir/sid/icd-10	E03	Hypothyroidism
OTHER_CHROMIC_OBSTRUCTIVE_PULMONARY_DISEASE	http://hl7.org/fhir/sid/icd-10	J44	Other chronic obstructive pulmonary disease
TYPE_2_DIABETES_MELLITUS	http://hl7.org/fhir/sid/icd-10	E11	Type 2 diabetes mellitus

A.2.2.8 CONDITION-CATEGORY

NAME	System	Code	Display
SYMPTOM	http://terminology.hl7.org/CodeSystem/condition-category	symptom	Symptom
COMORBIDITY	http://terminology.hl7.org/CodeSystem/condition-category	comorbidity	Comorbidity
PRIMARY_DISEASE	http://terminology.hl7.org/CodeSystem/condition-category	primary-disease	Primary disease

A.2.2.9 CONDITION-STATUS

NAME	System	Code	Display
ACTIVE	http://terminology.hl7.org/CodeSystem/condition-clinical	active	Active
INACTIVE	http://terminology.hl7.org/CodeSystem/condition-clinical	inactive	Inactive

A.2.2.10 UCUM

NAME	System	Unit	Code

KG	http://unitsofmeasure.org	kilogram	kg
MILLIMOLE_PER_MOLE	http://unitsofmeasure.org	millimole per mole	mmol/mol
MILLIGRAM_PER_DECILITER	http://unitsofmeasure.org	milligram per deciliter	mg/dL
ENZYME_UNIT_PER_LITER	http://unitsofmeasure.org	enzyme unit per liter	U/L
INTERNATIONAL_UNIT_PER_LITER	http://unitsofmeasure.org	international unit per liter	[IU]/L
ML_MIN_173_M2	http://unitsofmeasure.org	mL/min/{1.73_m2}	mL/min/{1.73_m2}
MILLIMETER_OF_MERCURY	http://unitsofmeasure.org	millimeter of mercury	mm[Hg]
METER	http://unitsofmeasure.org	meter	m
YEARS	http://unitsofmeasure.org	years	y
HEART_RATE	http://unitsofmeasure.org	bpm	{Beats}/min
MINUTES	http://unitsofmeasure.org	min	minute
CENTIMETER	http://unitsofmeasure.org	cm	centimeter
PERCENT	http://unitsofmeasure.org	%	percent

A.3 FHIR-GK-EXTENSIONS (GK-EXT) [NEW]

Attribute name	FHIR resource	URL	DataType	Description of the extension
patientAge	Observation	http://hl7.org/fhir/StructureDefinition/observation-patientAge	Integer	Age of the patient when the observation is performed
registeredDate	Encounter	http://hl7.org/fhir/StructureDefinition/encounter-registeredDate	DateTime	Date when the encounter is registered (loaded) on the system

Instructions to add a new conversion in the Data Federation

A.4 General description of Data Federation.

Data Federation is a platform consisting of 3 main components:

- **GK-integration engine**: it is able to accepts data coming from heterogeneous data source, convert them into Bundle FHIR and invoke the GK-FHIR Server APIs to store transformed data.
- **GK-FHIR Server:** FHIR Server providing API according the FHIR standard **version R4**
- **GK-RDF Watcher:** the component that executes the RDF conversion and sends the converted data to the RDF server.
- **GK-RDFJ4-WORKBENCH:** Repository containing persisted FHIR data in rdf format. It provides a set of API to retrieve information.

A.5 GK-integration engine

GK-Integration engine is a Maven Java project developed with Spring boot. It provides mainly two kinds of Rest APIs, as shown in the following figure, that accept raw data sent from heterogeneous data sources, convert them into GK-FHIR compliant format and persist such data into the FHIR Server for the storage. In the figure below a screenshot of the API documentation provided via Swagger.

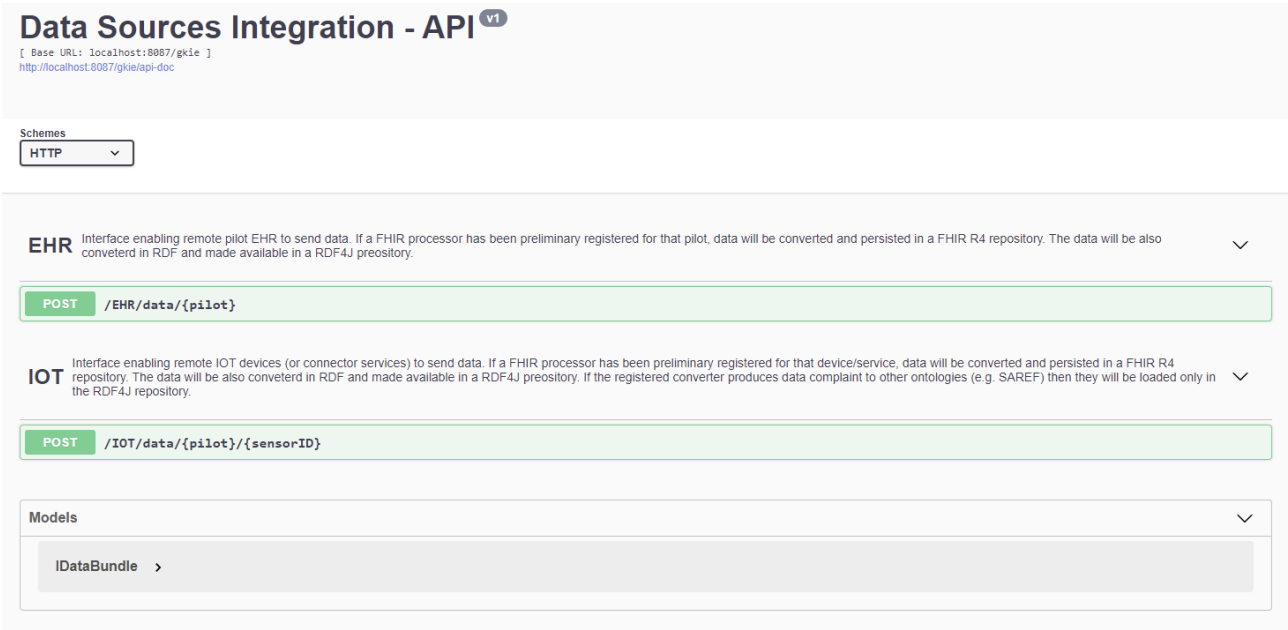


Figure 78 OpenAPI Data Federation Integration Engine

In order to convert the sent data, the integration engine, internally, retrieves the specific **converter** associated to the specific "data source". The "data source"

identifier corresponds to the {pilot} path parameter for the EHR interface while it is a combination of {pilot} and {sensorID} path parameters for the IOT interface.

A.5.1 How to build a new converter

If it is needed to provide a new **converter** for a new data source (it doesn't matter if such data source is going to call EHR or IOT interface), it is needed to implement a new converter following Data Federation framework guidelines. In order to speed up the process a sample eclipse project, with all the needed dependencies already in place, is provided [1]. Once downloaded and imported in eclipse it is simply needed:

Step 1: to provide the java data model used to deserialize data sent by the remote data source.

Step 2: to provide the converter which will include the logic for transforming the deserialized data (see step 1) in GK-FHIR compliant format.

Step 3: preform a test to check the capability of the new converter to properly work

Here below the details of the two steps.

A.5.2 Step 1 details

Browse the sample project and go to:

`it.eng.gk.dataintegration.model.<pilot name>`

or to

`it.eng.gk.dataintegration.model.<sensor ID>`

respectively if the data source we want to federate is an EHR or an IoT sensor (or IoT sensor gateway). The sample project already contains such packages based on our knowledge of the project but they could be easily extended in the case. For instance if the data source to federate is the Samsung IoT gateway android app, then it is needed to access to:

`it.eng.gk.dataintegration.model.samsung`

and modify the class (DataModel.java) by overriding the method getFilledInstance that is appointed to return an instance of the model, valorized with the data received in the request body. If the DataModel.java depends on further classes they can also be added in the same package. The important thing is that they include getter and setter methods as for JAVA BEAN specification . Here below two examples. On the left side an implementation in the case the string sent with the body request is itself a FHIR bundle. On the right instead an example when the body is a generic model in XML.

```java
public class DataModel implements IDataBundle<DataModel>{

    private Bundle inputBundle = null;

    public Bundle getInputBundle() {
        return inputBundle;
    }

    @Override
    public DataModel getFilledInstance(String body) {
        FhirContext ctx = FhirContext.forR4();
        IParser parserJson = ctx.newJsonParser();
        IParser parserXml = ctx.newXmlParser();
        try {
            inputBundle = parserJson.parseResource(Bundle.class, body);
        } catch (DataFormatException e) {
            inputBundle = parserXml.parseResource(Bundle.class, body);
        }
        return this;
    }

}
```

```java
public class DataModel implements IDataBundle<DataModel> {

    public BInformazioniRicovero bInformazioniRicovero;

    public BInformazioniRicovero getbInformazioniRicovero() {
        return bInformazioniRicovero;
    }

    public void setbInformazioniRicovero(BInformazioniRicovero bInformazioniRicovero) {
        this.bInformazioniRicovero = bInformazioniRicovero;
    }


    @Override
    public DataModel getFilledInstance(String body) {
        XmlMapper mapper = new XmlMapper();
        DataModel resource = null;
        try {
            resource = mapper.readValue(body, DataModel.class);
        } catch (JsonProcessingException e) {
            return resource;
        }
        return resource;
    }

}
```

A.5.3 Step 2 details

Once the model has been completed it is possible to develop the converter that is appointed to implement the transformation from the defined JAVA model (see step 1) to BUNDLE FHIR of **type TRANSACTION**. Browse the sample project and go to:

`it.eng.gk.dataintegration.converters.<pilot name>`

or to

`it.eng.gk.dataintegration.converters.<pilot name>_<source ID>`

respectively if the data source we want to federate is an EHR or an IoT sensor (or IoT sensor gateway). The sample project already contains such packages based on our knowledge of the project but they could be easily extended in the case. For instance if the data source to federate is the Samsung IoT gateway android app, then it is needed to move to:

`it.eng.gk.dataintegration.converters.saxony_samsung`

and modify the class (ConverterImpl.java) by:

1. Filling the constructor by initializing the attributes (semanticModel and outputFormat). This information (and related getter(s) methods) are exploited by the engine for routing purposes. Here below an excerpt:

```java
lic class ConverterImpl extends AbstractConverter implements IConverter<DataModel> {

 /*
  * Initilize the converter with the information about the output semantic model
  * (e.g. FHIR) and the syntax (e.g. JSON).
  */
 public ConverterImpl() {
     semanticModel = SemanticModelsEnum.FHIR;
     outputFormat = OutputSyntaxFormatEnum.JSON;
 }
```

2. Overriding the method convertFromHttpBody that is appointed to perform the data format transformation. Here below an excerpt:

```
/*
 * Implement the logic to transofrm the orginal data format to the GK-Profile
 * compliant FHIR profile.
 */
@Override
public Object convertFromHttpBody(String body) {
    DataModel data = new DataModel().getFilledInstance(body);
    Bundle result = null;
    result.setType(BundleType.TRANSACTION);
    /*
     * HERE THE LOGIC TO CONVERT THE ORIGINAL DATA MODEL STRUCTURE TO THE GK PROFILE
     * COMPLIANT BUNDLE.
     */
    return result;
}
```

A.5.4 Step 3 details

Go to the folder: **src/test/java** and in the **package it.eng.gk.dataintegration**. Here you find a test class (IntegrationEngineTests.java) containing three sections:

- The method you can override:

```
@SpringBootTest
class IntegrationEngineTests {

    ////-------------------------------------------------------METHOD TO MODIFY-------------------------------------------------------//
    @Test
    void converter_test_datasource() {
        // STEP 1. Provide the data input as XML\JSON.

        // STEP 2. Invoke the convertFromHttpBody for the specific converter

        // STEP 3. Print the bundle and check if it is compliant with GK-FHIR profile
    }
```

  The compliance must me checked by analyzing the produced FHIR bundle.

- A concrete running example

```
    //-------------------------------------------------------EXAMPLES-------------------------------------------------------//

    // EXAMPLES
    @Test
    void converter_test_puglia_medisante() throws FileNotFoundException {
        // STEP 1
        String body = IntegrationEngineTests.getInputFile("medisante_input.json");
        ConverterImpl converter = new ConverterImpl();
        // STEP 2
        Bundle conversionResult = converter.convertFromHttpBody(body);
        // STEP 3
        FhirContext ctx = FhirContext.forR4();
        IParser parserJson = ctx.newJsonParser();
        IParser parserXml = ctx.newXmlParser();
        System.out.println(parserJson.encodeResourceToString(conversionResult));

    }
```

- an utility method to load the input file you have uploaded in the local folder (src/test/resources/test_files/).

```
//-----------------------------------------------------UTILITY METHODS-----------------------------------------------------//

private static String getInputFile(String fileName) throws FileNotFoundException {
    File text = new File("src/test/resources/test_files/" + fileName);

    // Creating Scanner instnace to read File in Java
    Scanner scnr = new Scanner(text);

    // Reading each line of file using Scanner class
    String outputString = "";
    while (scnr.hasNextLine()) {
        String line = scnr.nextLine();
        outputString = outputString + line;
    }
    return outputString;
}
```

A.6 ENVIRA JSON

Received message


```
{
    "device_info": {
        "uuid": "30afdf79-b363-40d4-ac99-70db778c744b",
        "fw_ver": "V1.4.2"
    },
    "measures": [
        {
            "n": "co2",
            "u": "ppm",
            "v": 673.000
        },
        {
            "n": "voc",
            "u": "ppm",
            "v": 275.000e-3
        },
        {
            "n": "co",
            "u": "ppm",
            "v": 0.206
        },
        {
            "n": "pm10",
            "u": "ug/m3",
            "v": 0.000
        },
        {
            "n": "pm2.5",
            "u": "ug/m3",
            "v": 0.162
        },
        {
            "n": "temp",
            "u": "Cel",
            "v": 32.966
        },
        {
            "n": "hum",
            "u": "%RH",
            "v": 44.355
        },
        {
            "n": "prb",
            "u": "hPa",
            "v": 1002.427
        },
        {
            "n": "pm1",
            "u": "ug/m3",
            "v": 2.813
        },
        {
            "n": "pm4",
            "u": "ug/m3",
            "v": 0.162
        },
        {
            "n": "iaqi",
            "u": "count",
            "v": 65
        },
        {
            "n": "tci",
            "u": "count",
            "v": 37
        },
        {
            "n": "eiaqi",
            "u": "count",
            "v": 3
        }
    ]
}
```


Figure 79 ENVIRA JSON message

Converted information


```
{
  "fullUrl": "http://gk-fhir-server-gatekeeper-dev.apps.okd.seclab.local/gk-fhir-server/fhir/Device/10265",
  "resource": {
    "resourceType": "Device",
    "id": "10265",
    "meta": {
      "versionId": "1",
      "lastUpdated": "2021-11-23T07:40:04.461+00:00",
      "source": "#6oEDJRWAFsw9iHmW"
    },
    "identifier": [
      {
        "system": "https://www.gatekeeper-project.eu/sid/envira/device",
        "value": "00bd30c8-26ba-49ef-9cad-1d6892ab78d7"
      }
    ]
  },
```


Figure 80 ENVIRA Device FHIR conversion


```
{
  "fullUrl": "http://gk-fhir-server-gatekeeper-dev.apps.okd.seclab.local/gk-fhir-server/fhir/Observation/10361",
  "resource": {
    "resourceType": "Observation",
    "id": "10361",
    "meta": {
      "versionId": "1",
      "lastUpdated": "2021-11-24T12:28:34.108+00:00",
      "source": "#5ZCwxQbHyhmvaRxv"
    },
    "status": "final",
    "category": [
      {
        "coding": [
          {
            "system": "http://hl7.eu/fhir/ig/gk/CodeSystem/gatekeeper",
            "code": "liv-environment",
            "display": "Living Environment Observations"
          }
        ]
      }
    ],
    "code": {
      "coding": [
        {
          "system": "http://hl7.eu/fhir/ig/gk//CodeSystem/gatekeeper",
          "code": "co-conc",
          "display": "Carbon Monoxide concentration (ppm)"
        }
      ]
    },
    "valueQuantity": {
      "value": 0.127,
      "unit": "ppm"
    },
    "device": {
      "reference": "Device/10265"
    }
  }
}
```


Figure 81 ENVIRA Observation FHIR conversion