This project has received funding from the European's Union Horizon 2020 research innovation programme under Grant Agreement No. 957258

Architecture for Scalable, Self-human-centric, Intelligent, Secure, and Tactile next generation IoT

D5.3 Transversal Enablers Development Intermediate Version

Deliverable No.	D5.3	**Due Date**	30-APR-2022
Type	Other	**Dissemination Level**	Public
Version	1.0	**WP**	WP5
Description	Core specification and implementation status of vertical enablers developed in ASSIST-IoT.		

Copyright

The ASSIST-IoT consortium consists of the following 15 partners:

UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Spain
PRODEVELOP S.L.	Spain
SYSTEMS RESEARCH INSTITUTE POLISH ACADEMY OF SCIENCES IBS PAN	Poland
ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS	Greece
TERMINAL LINK SAS	France
INFOLYSIS P.C.	Greece
CENTRALNY INSTYUT OCHRONY PRACY	Poland
MOSTOSTAL WARSZAWA S.A.	Poland
NEWAYS TECHNOLOGIES BV	Netherlands
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS	Greece
KONECRANES FINLAND OY	Finland
FORD-WERKE GMBH	Germany
GRUPO S 21SEC GESTION SA	Spain
TWOTRONIC GMBH	Germany
ORANGE POLSKA SPOLKA AKCYJNA	Poland

Disclaimer

Authors

Name	Partner	e-mail
Alejandro Fornés	P01 UPV	alforlea@upv.es
Raúl Reinosa	P01 UPV	rreisim@upv.es
Rafael Vaño	P01 UPV	ravagar2@upv.es
Eduardo Garro	P02 PRO	egarro@prodevelop.es
Miguel Llacer Sanfernando	P02 PRO	mllacer@prodevelop.es
Ernesto Calas Blasco	P02 PRO	ecalas@prodevelop.es
Jose Antonio Clemente Perez	P02 PRO	jclemente@prodevelop.es
Katarzyna Wasielewska-Michniewska	P03 IBSPAN	katarzyna.wasielewska@ibspan.waw.pl
Piotr Lewandowski	P03 IBSPAN	piotr.lewandowski@ibspan.waw.pl
Maria Ganzha	P03 IBSPAN	Maria.ganzha@ibspan.waw.pl
Marcin Paprzycki	P03 IBSPAN	marcin.paprzycki@ibspan.waw.pl
Karolina Bogacka	P03 IBSPAN	k.bogacka@ibspan.waw.pl
Georgios Stavropoulos	P04 CERTH	stavrop@iti.gr
Anastasia Blitsi	P04 CERTH	akblitsi@iti.gr
Evripidis Tzonas	P04 CERTH	tzionasev@iti.gr
Iordanis Papoutsoglou	P04 CERTH	ipapoutsoglou@iti.gr
Anastasia Theodouli	P04 CERTH	anastath@iti.gr
Ron Schram	P09 NEWAYS	Ron.Schram@newayselectronics.com
Alex van den Heuvel	P09 NEWAYS	alex.van.den.heuvel@newayselectronics.com
Oscar López Pérez	P13 S21 SEC	olopez@s21sec.com
Jordi Blasi	P13 S21 SEC	jblasi@s21sec.com
Josue Moret Ruiz	P13 S21 SEC	jmoret@s21sec.com

History

Date	Version	Change
24-Dec-2021	0.1	Table of content
14-Mar-2022	0.2	First round of contributions completed
11-Apr-2022	0.3	Second round of contributions completed. Ready for internal review
28-Apr-2022	0.4	Integration of changes from internal review
30-Apr-2022	1.0	Final version submitted to EC

Key Data

Keywords	Enablers, verticals, self-*, interoperability, manageability, scalability, federated learning, DLT
Lead Editor	Evripidis Tzionas (P04 – CERTH)
Internal Reviewer(s)	Alex van den Heuvel (P09 - NEWAYS), Nick Vrionis (P06 - INFOLYSIS), Alejandro Fornés (P01 – UPV)

Executive Summary

This deliverable is written in the framework of WP5 – Transversal enablers design and development of ASSIST-IoT project under Grant Agreement No. 957258. The document gathers the work and outcomes of the five tasks of the work package, which are devoted to the design and implementation of enablers required to implement the different verticals of the ASSIST-IoT architecture. Tasks 5.1-5.4 started in M4 whereas Task 5.5 started in M9, so respective results presented have different levels of advancement.

The realisation of the ASSIST-IoT architecture requires the design and development of specific elements, both software and hardware, that support the vertical capabilities of ASSIST-IoT: these are Self-*, Scalability, Interoperability, Manageability, and Security, Privacy and Trust. Being the second version of a series of three deliverables, D5.3 updates and extends the specifications presented in the previous deliverable, accompanied by the software artifacts (i.e., enablers) developed so far. In particular, this deliverable provides the necessary updated technical information about the design, implementation (structure and functionalities, technologies, diagrams of use cases, communication endpoints) and ongoing development status of WP5 enablers, jointly with the (software) outcomes developed so far. The concepts presented here are a continuation of work summarised in D5.2 - Transversal Enablers Development Preliminary Version (submitted in M12).

Enablers are the main output of the tasks in WP5. An enabler represents a collection of components, running on hardware nodes, that communicate with each other for delivering a particular functionality to the system. Enablers can only be interacted with via their exposed interfaces. A total of 21 enablers have been identified and formalised, with different degrees of development:

- From Self-*: Self-healing device enabler, Resource provisioning enabler, Monitoring and notifying enabler, Location tracking enabler, Location processing enabler, Automated configuration enabler.
- From Federated Machine Learning: FL Orchestrator, FL Training Collector, FL Repository, FL Local Operations.
- From Cybersecurity: Cybersecurity monitoring enabler, Cybersecurity monitoring agent enabler, Identity manager enabler, Authorization enabler.
- From DLT: Logging and auditing enabler, Data integrity verification enabler, Distributed broker enabler, DLT-based FL enabler.
- From Manageability: Enabler for registration and status of enablers, Enabler for management of services and enablers' workflow, Devices management enablers.

Being the second of a series of three iterations, the software products and the information provided in this deliverable is still susceptible to change, because of the addition of new (or yet not implemented) features.

Table of contents

List of figures

List of tables

List of acronyms

Acronym	Explanation
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
BSON	Binary JSON
CHE	Container Handling Equipment
CPU	Central Processing Unit
CSV	Comma Separated Value
DLT	Distributed Ledger Technology
DoS	Denial of Service
FAIR	Findable, Accessible, Interoperable, Reusable
FML	Federated Machine Learning
FL	Federated Learning
FLS	Federated Learning System
FLTC	Federated Learning Training Collector
GPS	Global Positioning System
HW	Hardware
I/O	Input/Output
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
K8s	Kubernetes
LTS	Long-Term Storage
LTSE	Long-Term Storage Enabler
MANO	Management and Orchestration
NGIoT	Next Generation Internet of Things
NN	Neural Networks
noSQL	Not Only Structured Query Language
MITM	Man-In-The-Middle

ML	Machine Learning
MQTT	MQ Telemetry Transport
OEM	Original Equipment Manufacturer
PAP	Policy Administration Point
PCM	Powertrain Control Module
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
REST	Representational State Transfer
RSSI	Received Signal Strength Indicator
RTG	Rubber-Tyred Gantry (crane)
SDN	Software Defined Network
SoTA	State-of-the-Art
SQL	Structured Query Language
SMC	Secure Multi-Party Computation
SR	Semantic Repository
TBD	To Be Done/Defined
TRL	Technology Readiness Level
TTL/SSL	Time To Live/Secure Sockets Layer
UC	Use Case
WP	Work Package
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

1. About this document

The objective of this deliverable is two-fold: (i) to **update** the **specifications** and **attach additional information** regarding the vertical **enablers** designed, and (ii) to **provide an updated functional version** of the enablers developed so far. These enablers along with horizontal enablers proposed in WP4, are the technological backbone of the project, since they will enable the deployment of an ASSIST-IoT architecture.

It should be highlighted that this deliverable corresponds to the second out of three documents, and therefore its content is a continuation of D5.2 Traversal Enablers Development Preliminary Version. Moreover, it follows deliverable D5.1 Software Structure and Preliminary Design that was first out of two documents. Therefore, the content of D5.3 will be expanded and adapted as the project evolves. This is motivated by different reasons, including the fact that both the requirements and the architecture produced by the work of WP3 are still evolving (and therefore new enablers or modifications in the current ones may be needed), and as a result the interactions between enablers from WP4 and WP5 may require adapting them (in the form of new interfaces, methods, components, etc.).

1.1. Deliverable context

Keywords	**Lead Editor**
Objectives	**O3:** Definition and implementation of decentralised security and privacy exploiting DLT: Specification of DLT-based enablers in Security, Privacy and Trust vertical. **O4:** Definition and implementation of smart distributed AI Enablers: Specification of Federated Machine Learning related enablers. **Other:** Availability of preliminary software artifacts, specifically essential enablers: • DLT logging and auditing enabler • Cybersecurity enablers: Identity manager enabler, Authorization enabler • Manageability enablers
Work plan	D5.3 takes input from: • T3.1 (state-of-the-art): Novel components and technologies research for further design choices • T3.2 & T3.3 (use cases and requirements): To be evaluated and fulfilled with the proposed enablers • T3.5 (architecture): Design principles and high-level functionalities to cover • D5.1 (initial transversal enablers specification): Design of vertical enablers • D5.2 (transversal enablers development preliminary version): Previous version of this deliverable D5.3 influences: • WP7 (pilots and validation): To later on materialize in pilot deployments • WP8 (evaluation and assessment): To evaluate and assess results from testing within pilots D5.3 must be in line with: • WP4 (core enablers): To define functional boundaries and interactions • WP6 (testing, integration and support): To develop, test and deploy according to DevSecOps methodology
Milestones	This deliverable contributes to the realisation of *MS3 – Enablers defined*, that was achieved M12. Although far in time (M24), it is also central part of *MS6 – Software structure finished.*

Deliverables	This deliverable receives inputs from D3.1 (State-of-the-art and Market Analysis Report), D3.2 (Use Cases Manual & Requirements and Business Analysis Initial) and D3.5 (ASSIST-IoT Architecture Definition - Initial), D5.1 (Initial Transversal Enablers Specification), D5.2 Transversal Enablers Development Preliminary Version. Once enablers are being delivered, they will feed the deliverables of WP6 related to testing, integration, distribution, and documentation, they will be the cornerstone of pilots' implementations of WP7, and they will be a key part in the technical evaluation to be performed under the scope of WP8.

1.2. The rationale behind the structure

The document consists of four sections and an appendix. The first section is an introduction that outlines the context of the document. The following section includes specifications of enablers divided into tasks they belong to. Each enabler description includes: Structure and functionalities, communication interfaces, use cases and implementation status at M18 of the project. Note that the above information advance and/or compliment enablers' definitions from the D5.2. Finally the last section is devoted to the future work remaining to be completed until the third and final version of the deliverable.

1.3. Outcomes of the deliverable

The main outcome of this deliverable is the documentation of the intermediate development of WP5 enablers. The work progress of each enabler differs and is summarized in the implementation status subsections. As a continuation of D5.2, this deliverable consolidates an updated and more concrete outline of each enabler.

The most focus is put on essential enablers (identified in D6.5) that include the following enablers from WP5: basic security enablers (Identity manager enabler, Authorization enabler), DLT-based Logging and auditing enabler and manageability enablers. These enablers are not more important or interesting but they are required to be in place in the first order to facilitate use and configuration of other non-essential enablers. Versions of enablers might be ready for Kubernetes or just for Docker (to be deployed via Docker Compose). Although documented more extensively in the Readthedocs documentation related to T6.5, a summary of each enabler implementation status is given here as well.

The deliverable consists not only of the present document, but also of the software artifacts developed and implemented so far. The following enablers have a first functional version: Self-healing device enabler, Resource provisioning enabler,, Monitoring and notifying enabler, Automated configuration enabler, FL Orchestrator, FL Training Collector, FL Local Operations, Authorization enabler, Identity manager enabler, Cybersecurity monitoring enabler, Cybersecurity monitoring agent enabler, Logging and auditing enabler, Management of the enablers existence in a deployment, Management of devices in an ASSIST-IoT deployment.

Included specifications may be modified and/or extended in the last version of this deliverable due to the fact that the work in WP3, WP4 and WP7 is in progress. During the next months, the missing features of the enablers will be implemented, and the needed adaptations to work in a Kubernetes environment and interact between each other will be executed. These outcomes will feed the Work Packages related to integration, deployment and assessment, i.e., WP6, WP7 & WP8.

1.4. Lessons Learnt

During the past months, the partners of the Consortium have focused their effort in developing the design specifications of the enablers that will facilitate the realization of the ASSIST-IoT architecture. From all this work the following insights have been extracted:

- Designing enablers for IoT deployed in K8s clusters is a sensitive task and requires taking into account how K8s works. Integrating such a multitude of interacting enablers can cause multiple issues, extending from connections problems to latency loads.

- Identifying enablers as self-* can be also complex. Autonomous computing, as an uprising technology, has many arbitrary concepts which sometimes overlap each other. Hence, self-* enablers' labelling will be built around their individual functionalities and use cases, differencing from each other.
- Principles of Federated Learning and DLT, as standalone processes, are clear. Integrating them in an IoT environment is risky and requires great effort to ensure the data protection in parallel with constant and effective model updates.
- The early realization of the need for manageability enablers is critical, and it will compensate the effort to develop them in the later stages of the project.

1.5. Deviation and corrective actions

The Consortium formalised in D5.1 and materialised in D5.2 and now D5.3 the envisioned enablers. However, some deviations have slightly altered the initial plan: (geo)Localization enabler from Self-* vertical has been split into two enablers – Location tracking enabler and Location processing enabler. The functionalities assigned to the original enabler have been split so that two new enablers can be used independently. This is justified by the fact that logically gathering localization data is a separate and independent task from their later processing, so the initial scope of the geo(Localization) enabler was too broad. Additionally, there can be different sources of location related data, e.g., UWB-specific, GPS that can be handled by Location processing enabler.

2. Introduction

As it was stated in D3.5, the ASSIST-IoT architecture is structured following a multidimensional approach composed of horizontal Planes and Verticals. The planes represent a classification of logical functions that fall under the scope of a particular domain, whereas verticals target NGIoT properties that exist on different planes, either independently or requiring cooperation of elements from different planes. Verticals in INTER-IoT include: interoperability, self-*, security, privacy and trust, manageability, and scalability.

The main building block in ASSIST-IoT architecture is an enabler - an abstraction term that represents a collection of components, running on nodes, that work together for delivering a particular functionality to the system. D5.1 focused on providing initial specification of transversal (vertical) enablers that belong to specific verticals. D5.2 focused on providing the status of development and advancing technical specifications of the enablers as of M12. D5.3 is a continuation of D5.2, and provides the status of development and advancing/adjusting technical specifications as of M18. What is specific to WP5 is that enablers, besides being distributed between verticals, are also designed and implemented within tasks (indicating problem/application areas) that do not correspond directly to the verticals. The following sections contain descriptions of enablers following a task division.

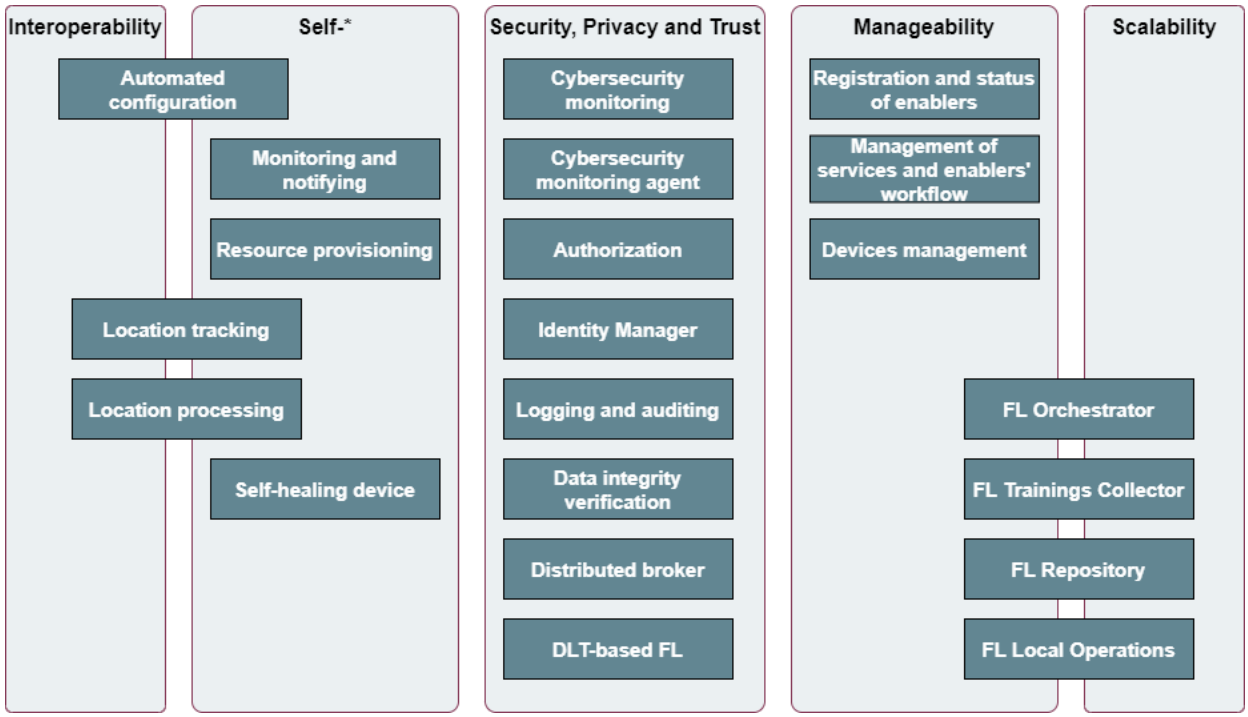

Figure 1. WP5 enablers distribution among verticals

3. Vertical enablers update and implementation status

3.1. Self-* enablers

3.1.1. Self-healing Device enabler

3.1.1.1. Structure and functionalities

This enabler aims at providing the IoT devices with the capabilities of actively attempting to recover themselves from abnormal states, based on a pre-established routines schedule. Hence, it should not require high computation capabilities in order to be deployed on any customizable device.

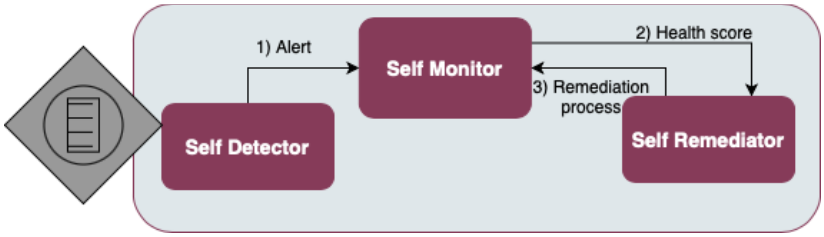

Figure 2. Self-healing Device enabler structure

As described in D5.2, the self-healing device enabler is divided in three components:

- Self-detector: The goal of this component is to collect information from the IoT device.
- Self-monitor: The Self-monitor component is responsible for assessing the device's state of health. It collects and analyses data from multiple sources of information received from the Self-detector, such as memory usage, memory access, network connection metrics (RSSI levels), or CPU usage, providing a health score. The health score metrics are fed to a predefined set of rules (or to an anomaly-detection model) that determines whether the device is in a healthy state or not. The output of this component is used to determine if the remediation has been successful.
- Self-remediator: When the device presents with symptoms of malfunctioning or intrusion, this component's job is to determine from a set of remediation processes, which should be used for a proper treatment. If after the remediation, the device is not back to its normal state, the component is self-triggered to select another remediation process from the list.

Implementation technologies

Table 1. Implementation technologies for the Self-healing Device enabler

Technology	Justification	Component(s)
Node-RED	Is a low-code programming tool for wiring together hardware devices, APIs and online services. Provides all it is needed to implement self-healing devices (hardware and software access)	Self-detector, Self-monitor, Self-remediator
Unix commands	Used to access device hardware & software	Self-detector, Self-monitor, Self-remediator
Javascript	Main language for developing custom functions over all components of the enabler. Selected for its familiarity	Self-detector, Self-monitor, Self-remediator

3.1.1.2. Communication interfaces

Table 2. Communication interfaces (API) of the Self-healing Device enabler

Method	Endpoint	Description
POST	**/cpuusage?threshold=XX**	Change the maximum threshold of CPU usage to XX
POST	**/ramusage?threshold=XX**	Change the maximum threshold of RAM usage to XX
POST	**/network?IP=XX**	Change the IP address over which the service should ping to check network availability

NOTE: Regarding external APIs for the enabler, the above included endpoints have been already developed, although new ones are subject to be added in the following releases of the enabler.

3.1.1.3. Use cases

There are two main use cases related to the self-healing device enabler.

- The first one is related to the monitoring of the CPU and RAM usage (which threshold can be configured by the user via self-healing device enabler API).

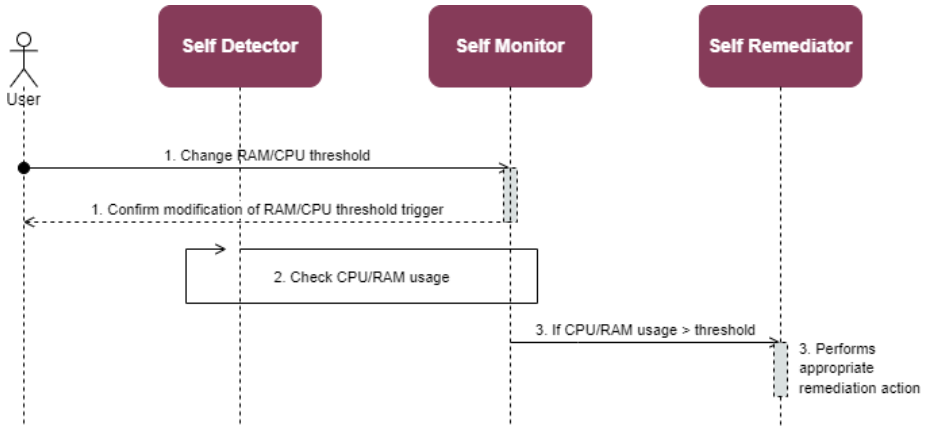

Figure 3. Self-healing Device enabler UC1 (CPU/RAM usage monitoring and threshold update)

STEP 1: The user starts a device, installs the self-device enabler, and configures the CPU/RAM usage thresholds by interacting with the self-monitor via API commands.

STEP 2: Since then, the self-detector and self-monitor have started detecting and monitoring the resources used in a happy path scenario.

STEP 3: If some of the monitored Process of the Operating System of the device surpasses the threshold, the self-monitor informs the self-remediator to carry out the proper remediation action (killing process ID consuming higher CPU/RAM resources).

- The second use case is related to the evaluation of the network interface operation (which accessed IP address can be configured by the user via self-healing device enabler API).

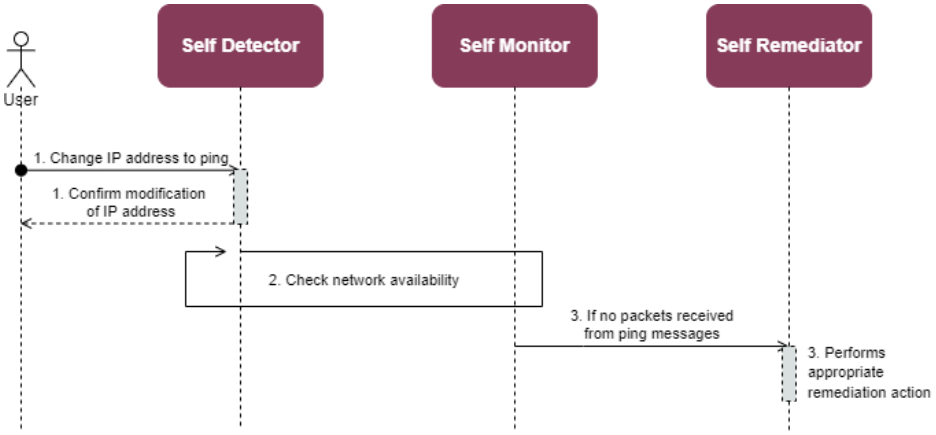

Figure 4. Self-healing Device enabler UC2 (RAM usage monitoring and threshold update)

STEP 1: The user starts a device, installs the self-device enabler, and configures the IP address over which the enabler should ping in order to evaluate the network reachability, by interacting with the self-detector via API commands.

STEP 2: Since then, the self-detector and self-monitor have started detecting and monitoring the network accessibility in a happy path scenario.

STEP 3: If several ping messages do not receive IP packets appropriately, the self-monitor informs the self-remediator to carry out the proper remediation action (restarting network device manager).

3.1.1.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/self_healing_device_enabler.html

Table 3. Implementation status of the Self-healing Device enabler

Category	Status
Components implementation	All the self-healing device enabler components are implemented.
Feature implementation status	For the time being, CPU usage, RAM usage, and network accessibility are the HW resources monitored by the enabler. Regarding remediation actions, only the kill PID (for the CPU, RAM usage monitoring), and restarting of the device's network manager are supported. In next releases, additional metrics (e.g., storage status, battery status), more advanced monitoring options (i.e., smarter ML-based solutions), and other remediation rules (e.g., isolate the device, shut down network ports, or reboot) will be included.
Encapsulation readiness	After several failure tests (the functionalities of the enabler were not reaching the host OS environment if it was encapsulated), the self-healing device enabler has been successfully containerized and tested by adding admin privileges to the Docker version. However, its implementation in a Helm chart, as well as the testing in a K8s cluster is still pending.
Deployed with the Orchestrator in a laboratory environment	Not yet, the self-healing device enabler has been tested only over HW emulators (i.e., not tested over actual HW edge devices), without involving additional enablers (including orchestrator).

3.1.2. Resource Provisioning enabler

3.1.2.1. Structure and functionalities

Working on edge deployments, where resources are not as large as in the cloud, it is unfeasible to set a static resource projection to each node. This is due to the difference in the use of these resources depending on the workload at the time the task is performed, being dependent on several factors. This enabler aims to adapt the auto-scaling of nodes and clusters more dynamically, achieving optimal use in relation to resource utilization and general operation. The updated diagram of the component is given in Figure 5. It is composed by 4 main components and 2 supporting databases:

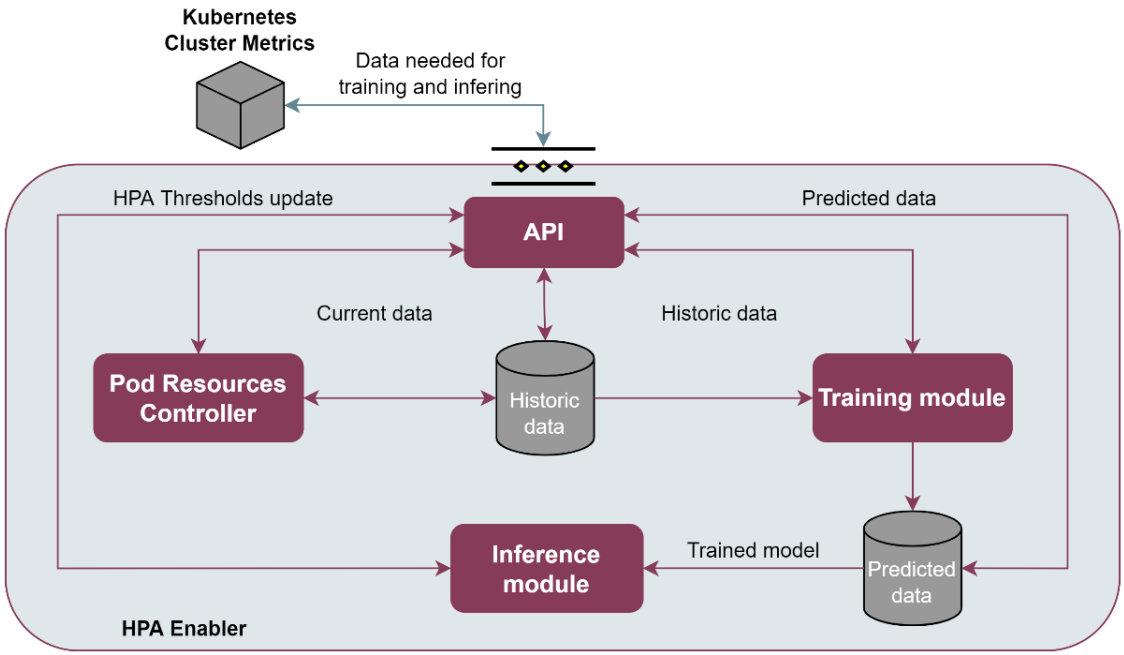

Figure 5. Resource Provisioning enabler structure

Some of its characteristics are:

- Ensuring high QoS and availability of key, selected enablers, considering the current state of the system.
- Monitoring and obtaining historical trends of these enablers, to preventively act upon its scaling requirements (thresholds of resources/usage to instantiate replicas).
- Application of ML techniques and intelligent services to train an optimal data model to predict future behaviour based on historical trends.
- Ability to control several deployments in a cluster independently and selection of minimum, maximum resources and average utilization of each deployment.

Implementation technologies

Table 4. Implementation technologies for the Resource Provisioning enabler

Technology	Justification	Component(s)
Python	Main language for developing custom functions over all components of the enabler. Selected for its familiarity and facility of implementation.	All components except databases
Flask	All the components except the databases will be developed considering Flask as the main technology due to its ease of construction and communication between the other components and the outside in the case of the API.	All components except databases
PyTorch	Main library for building inferences. Optimal for machine learning and deep learning technologies. Compatible with the development of applications in python.	Training module
Neural Prophet	Neural Network based Time-Series model, inspired by Facebook Prophet and AR-Net, built on PyTorch. Used for resource usage inference. We are familiar with management and implementation and it is efficient.	Training module
MySQL	Ease of creating strict relationships between enablers, labels and data. By using large volumes of data, management and functionality is more efficient.	Historic Data Predicted data
Docker images	All components are built as custom Docker images.	All components
Kubernetes resources	The main tool on which the general operation of this enabler is based. It contains services to interconnect pods (containers), deployments to establish the initial configuration of each pod and implements the horizontal pod autoscaler that allows an improved operation based on the need for resources requested at each moment.	All components

3.1.2.2. Communication interfaces

Table 5. Communication interfaces (API) of the Resource Provisioning enabler

Method	Endpoint	Description
GET	**/enablers**	Returns a JSON object showing all enablers with their respective active components in the host cluster.
POST	**/train**	Train the active models of all components stored in the historical database. Adds the predicted data to the future database and acts on the times set by train-values. This call is also automatic.
GET	**/train-values**	Returns a JSON object containing the values that determine the historical days of the data and the prediction days of the data.
POST	**/train-values**	A JSON object is added to modify historical and predictive data values.
GET	**/inference/<enabler>/<component>**	Updates the horizontal pod autoscaler horizontals of the desired components. The data stored in the future database is used. This call is also automatic.
POST	**/inference/<enabler>/<component>**	A JSON object is added to update enablers and component to guarantee high levels of QoS.

3.1.2.3. Use cases

There are six main use cases that apply in this enabler. The **first use case** is related to **get information about the enablers and their components active on the host cluster.** The diagram and involved steps are the following:

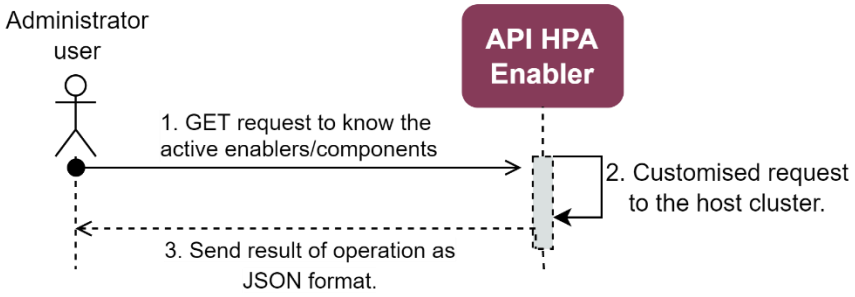

Figure 6. Resource Provisioning enabler UC1 (get active enablers and components)

STEP 1: The user interacts through the enabler API, via GET request, to get the enablers and their components active and correctly configured in the Kubernetes cluster.

STEP 2: The enabler receives the command through its component API and makes a request to the Kubernetes cluster to list the enablers and components.

STEP 3: The corresponding data is returned in JSON format.

The **second use case** is related to an administrator user instructing the enabler to **perform a new data training** (deep learning). This use case is useful if the default values are changed. It should be noticed that the system will automatically start this process on a periodic basis. The diagram and related steps are the following:

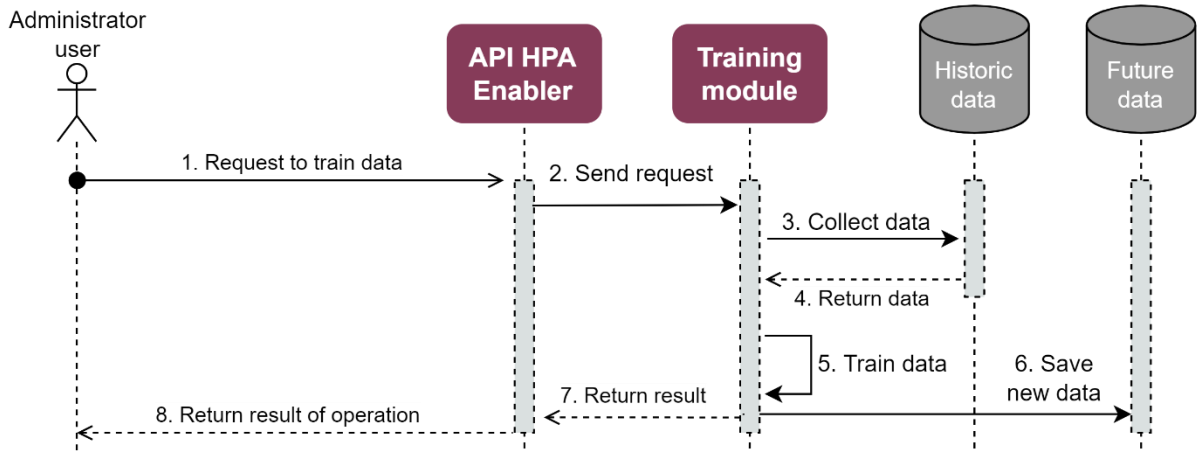

Figure 7. Resource Provisioning enabler UC2 (perform training)

STEP 1: The user interacts through the enabler's API, via a POST request, to instruct the enabler to train the models with the new data.

STEP 2: The enabler receives the command through its component API and redirects the request to the Train Module component.

STEP 3: The train module component collects the necessary data from the history database.

STEP 4: The history database returns the raw data.

STEP 5: The train module component adapts the data and performs the training.

STEP 6: Once the training is done, it adapts the data and saves the results in the future database.

STEP 7: The train module component returns the execution status to the enabler API.

STEP 8: The enabler API component returns the result of the operation.

The **third use case** is related to **an administrator user who gets the range of historical and future data prediction behaviour (in days)**. The response from the enabler is in JSON format. The diagram is following:

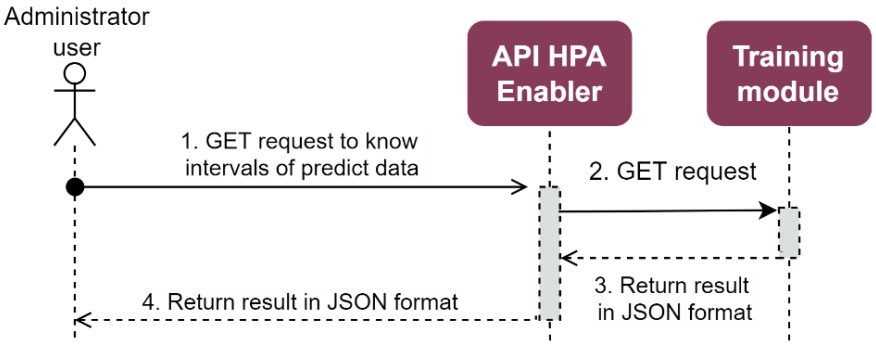

Figure 8. Resource Provisioning enabler UC3 (get interval range for training)

STEP 1: The user interacts through the enabler API, sends a GET request to know the intervals with which the train module component acts.

STEP 2: The enabler receives the request and redirects to the training module.

STEP 3: The training module returns the values in JSON format to the enabler API.

STEP 4: The enabler API returns the output in JSON format.

It is possible to change some default parameters of the behaviour of the enabler. The **fourth use** case is related to **changing the behaviour of the data prediction, indicating the range of historical data for training**. The diagram is following:

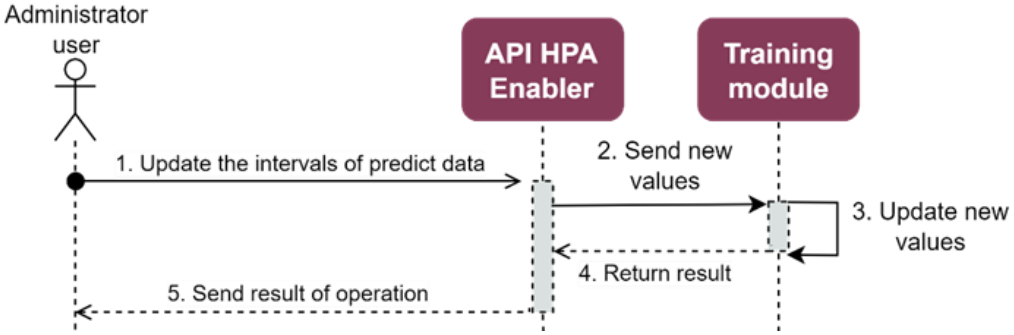

Figure 9. Resource Provisioning enabler UC4 (update data interval for training)

STEP 1: The user interacts through the enabler API, indicating the new intervals of predict data.

STEP 2: The enabler receives the values and sends them to the training module.

STEP 3: The training module updates the values.

STEP 4: The training module returns the result of the update.

STEP 5: Once the process has finished, the enabler API responds to the user with the result of the operation.

The **fifth use case** is related to **an administrator user instructing the enabler to perform inference to the desired enablers/components**. This use case updates the values according to the previous training of each component of the desired enablers. This action is also performed automatically. The diagram and involved steps are the following:

STEP 1: The user interacts through the enabler API, via a GET request, to instruct the enabler to infer the desired enabler components.

STEP 2: The enabler receives the command through its component's API and redirects the request to the inference module component.

STEP 3: The inference module component collects the data needed for training from the future database.

STEP 4: The future database returns the raw data.

STEP 5: The inference module component adapts the data and performs the inference process on all desired components of each enabler.

STEP 6: The inference module component returns the execution state to the enabler API.

STEP 7: The enabler API component returns the result of the operation.

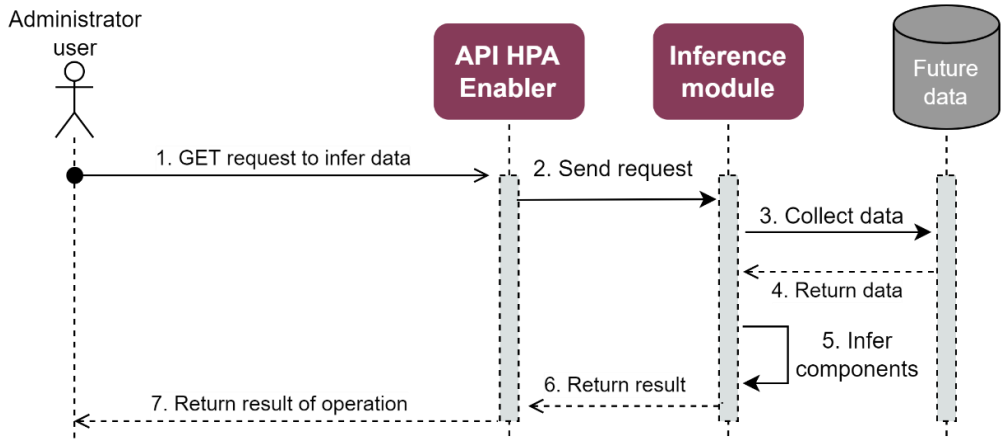

Figure 10. Resource Provisioning enabler UC5 (perform inference)

The **last use case (sixth)** is related to **select the enablers to manage**. This selection can be made for all components of an enabler or for individual components. The diagram is the following:

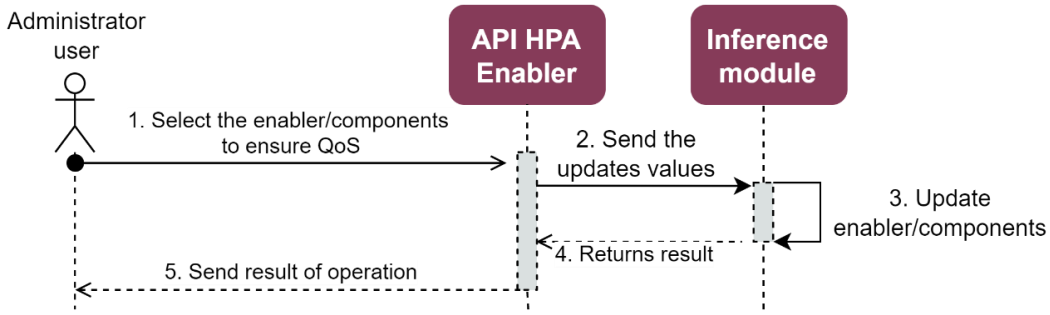

Figure 11. Resource Provisioning enabler UC6 (select enablers to manage)

STEP 1: The user interacts through the enabler API, indicating the prioritised enablers and components.

STEP 2: The enabler receives the assigned cluster selection order and sends a POST request to the inference module along with the data in JSON format for the values to be updated.

STEP 3: The inference module updates the components of these enablers.

STEP 4: The inference module returns the result of the operation.

STEP 5: Once the process has finished, the enabler API responds to the user with the result of the operation.

NOTE: This use case will be available in future releases. Currently, all enablers and active components are target of the enabler, not allowing a more granular selection.

3.1.2.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/resource_provisioning_enabler.html

Table 6. Implementation status of the Resource Provisioning enabler

Category	Status
Components implementation	A first version of all the component is in place
Feature implementation status	Most of the expected features are implemented. Some are still pending: • Finalising the API. • Upgrading the actuation of the Inference Module over the cluster.

Category	Status
Encapsulation readiness	All components are encapsulated in docker images and work under Docker-Compose.
Deployed with the Orchestrator in a laboratory environment	Not yet

3.1.3. Monitoring and Notifying enabler

3.1.3.1. Structure and functionalities

This is an enabler responsible for monitoring the uninterrupted functionality of devices and notifying in case of malfunction incidents. Specifically, it has to ensure the departure of data, the arrival, the validity and its own self-monitoring functionality. **Structure and functionalities have not changed with respect to deliverable D5.2 content.**

Implementation technologies

Table 7. Implementation technologies for the Monitoring and Notifying enabler

Technology	Justification	Component(s)
Kafka	Kafka provides a standardized method to enable a diverse set of technologies to communicate and interact. It is used to build real-time streaming data pipelines and real-time streaming applications which will be very useful in the IoT environment of the project.	Database, Message Queue, Registry
Java	Java is a low complexity programming language and since Kafka is written in Java, it is one of the best choices for the enabler.	Database, Message Queue, Registry, Communication Interface
MongoDB	MongoDB is the choice for the database component. There is no need for relational database, it is fast, scalable and it supports the JSON/BSON data formats.	Database, Registry
MQTT	MQTT is a lightweight publish/subscribe messaging protocol and it is widely used in IoT solutions. Since Edge Data Broker Enabler will use this protocol, it is under consideration to be used for easier integration.	Database, Message Queue, Registry
Python	Python is the best language for developing custom scripts and functions over the components of the enabler.	Database, Message Queue, Registry, Communication Interface

NOTE: It is still under discussion which functionalities can be added to the enabler, in order to be labelled as self-*.

3.1.3.2. Communication interfaces

Table 8. Communication interfaces (API) of the Monitoring and Notifying enabler

Method	Endpoint	Description
POST	**/notifications**	Create Notification
GET	**/notifications/rolling data**	Get input data before the notification occurrence
GET	**/notifications**	Get notification
GET	**/devices**	Get a list of connected devices

3.1.3.3. Use cases

The **first use case** is the same with respect to D5.2 content **(IoT device which stops receiving data from its integrated sensor).** Two more use cases have been added and they are introduced in this version. The **second use case** involves an **IoT wristband device which enters a restricted area** of a specific site.

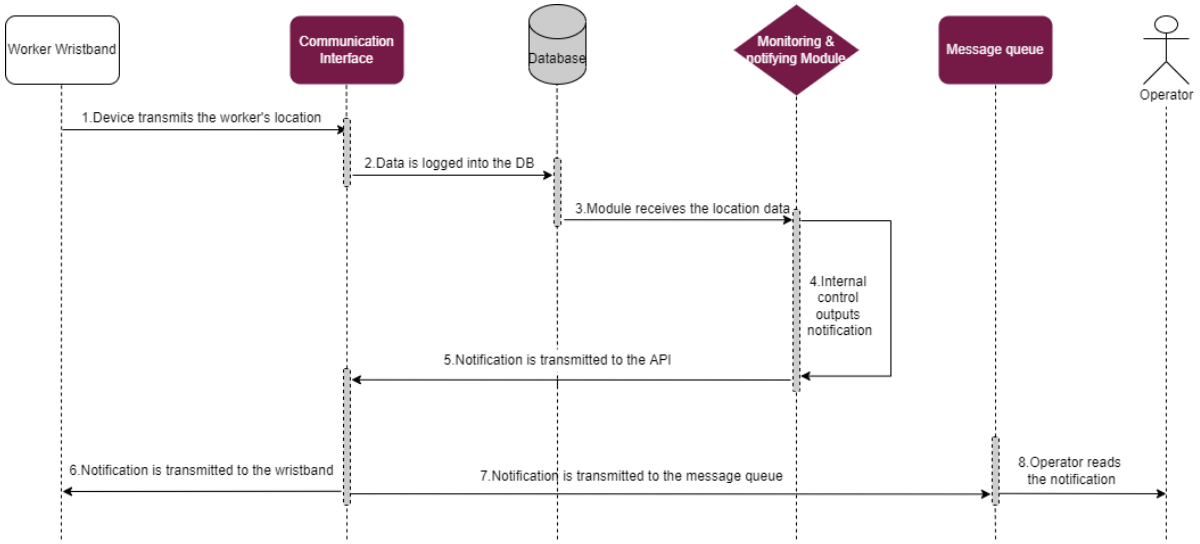

Figure 12. Monitoring and Notifying enabler UC2 (device entering restricted zone)

STEP 1: The wristband monitors and transmits the user's location.

STEP 2: Location data is logged into the database.

STEP 3: The monitoring & notifying module receives the location data.

STEP 4: Since the monitoring module identifies that someone is entering a restricted area, it has to notify both the user itself and a responsible (operator) for the incident.

STEP 5: The notification is transmitted to the API, in order to be sent to the message queue and the user.

STEP 6: The notification is transmitted to the user, by activating a LED on his wristband, indicating he entered a restricted area.

STEP 7: The notification is transmitted to the message queue.

STEP 8: The operator receives the notification and the information (data) before its occurrence and has to act accordingly.

The **third use case** (built around pilot 3A) is related to user **querying a vehicle's conditions**, assuming that there is constant monitoring of the vehicle's metrics.

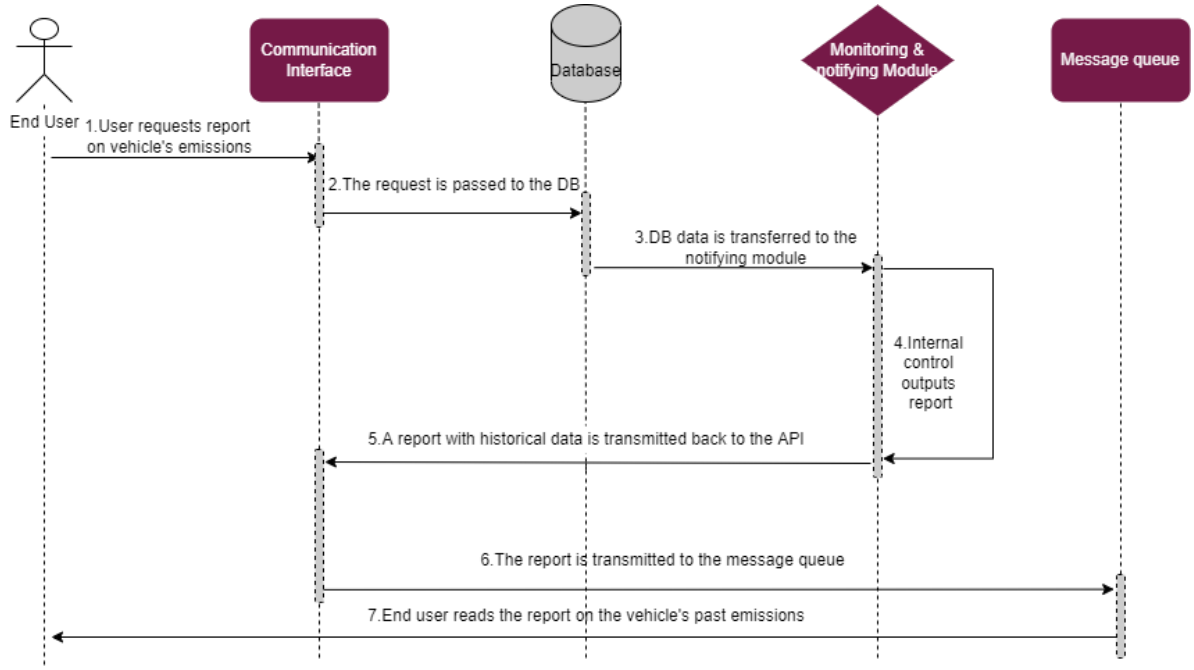

Figure 13. Monitoring and Notifying enabler UC3 (querying vehicle conditions)

STEP 1: An end user requests a report on the vehicle's conditions.

STEP 2: The request is passed to the database.

STEP 3: The database data is transferred to the notifying module in order to create the report with historical data and metrics such as the average emissions.

STEP 4: The report is concluded and it is ready to be sent back to the API.

STEP 5: The report is sent to the API.

STEP 6: The report is transmitted to the message queue.

STEP 7: The end user reads the report he requested.

3.1.3.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/monitoring_and_notifying_enabler.html

Table 9. Implementation status of the Monitoring and Notifying enabler

Category	Status
Components implementation	A first version of all the components are in place
Feature implementation status	The basic features of these components are working in the minimum viable product principle. It remains to: • Connect to the EDBE. • Proceed with encapsulation.
Encapsulation readiness	Components are yet to be encapsulated
Deployed with the Orchestrator in a laboratory environment	Not yet

3.1.4. Location Tracking enabler

The main task of the location tracking enabler is to receive the position of tags. Each tag transmits its position with a fixed repetition rate. This position represents the coordinates of the tag relative to a reference anchor. The localization tracking enabler translates these positions into a format which can be handled by a data broker.

3.1.4.1. Structure and functionalities

The localisation tracking enabler consists of a "tag & anchor configuration" component and a "localisation engine" component, see also Figure 14.

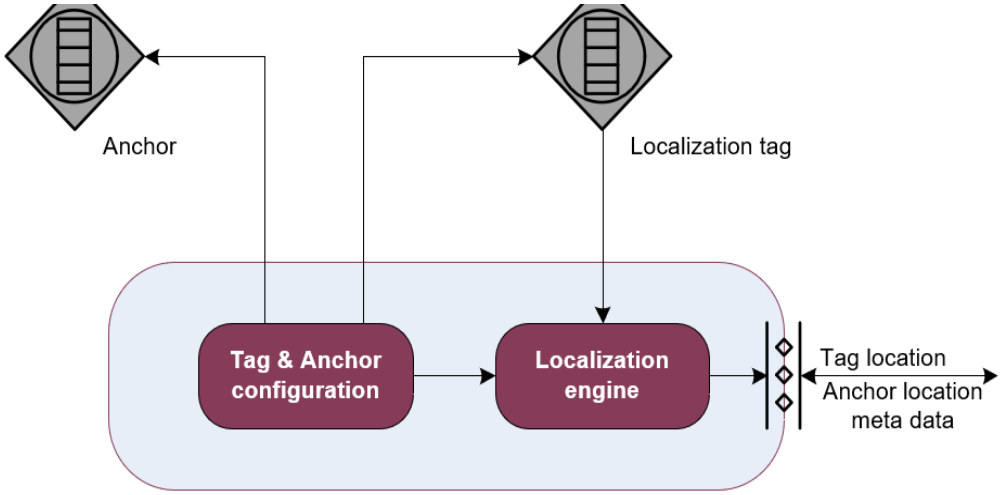

Figure 14. Location Tracking enabler structure

On the one hand, the "tag & anchor configuration" component is used to set parameters like tag ID, position update rate, etc. The exact amount of configuration parameters is under construction at the moment. On the other hand, the "localization engine" receives the location of the tags and translates these locations to a JSON format that can be handled by the data broker. The transmitted location contains, besides the location, a tag ID for which the location is valid and a time stamp. With these parameters the location processing enabler can track when a tag has been at which location.

3.1.4.2. Communication interfaces

The location tracking enabler has the following interfaces. They are under development, and the outcome will be documented in the next version of this deliverable.

- Interface for anchor configuration
- Interface for tag configuration
- Interface for tag location

3.1.4.3. Use cases

The **main use case** is **receiving distances from tags**. A tag delivers distances with a fixed frequency, which is set during configuration. How the use case will be implemented is under construction at the moment and will be documented in the next version of this deliverable.

3.1.4.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/location_tracking_enabler.html

Table 10. Implementation status of the Location Tracking enabler

Category	Status
Components implementation	A first version of all the components are in place
Feature implementation status	The basic features of these components are working in the minimum viable product principle. It remains to: • Implement the localization tag interface and engine. • Determine tag & anchor configuration parameters.
Encapsulation readiness	Components are yet to be encapsulated
Deployed with the Orchestrator in a laboratory environment	Not yet

3.1.5. Location Processing enabler

3.1.5.1. Structure and functionalities

This enabler has been scoped out from the original *(Geo)localization* enabler, whose functionality in M15 was split into two enablers, one concentrating mostly on the hardware part and this one focusing on localisation data processing. Eventually, the Location Processing enabler will provide flexible geofencing capabilities allowing to:

- Define "regions" and "points" of interest, and identify them in a unique way,
- Update the geometry/position data of defined regions and points, and
- Query relationships between a given position and selected region/point(s).

Ultimately, the enabler will handle data updates and queries using both the standard REST-based request-response and a high-speed streaming approach. The "data endpoints" of the enabler will be suitably protected, to prohibit unauthorised data access and manipulation.

Implementation technologies

Table 11. Implementation technologies for the Location Processing enabler

Technology	Justification	Component(s)
Scala	Main language for developing custom functions for all components of the enabler. Selected for its high quality, good support and adequacy for the task.	All components except database
Akka	All the components of the enabler will be developed using Akka (and Akka Streams) libraries, due to the excellent support they provide for features and communication standards/protocols required for the enabler.	All components except database
PostGIS	PostGIS is a mature spatial extension for PostgreSQL. It natively supports all the GIS-oriented functions needed by the Location processing enabler. Additionally, PostGIS is well supported by existing Java-based libraries (hence it is also Scala-friendly).	Location data storage and processing
Apache Kafka	Selected for its state of the art support for asynchronous buffering and stream handling mechanisms.	Streaming updates/queries
Docker images	All components are built as custom docker images.	All components
Docker compose	Docker compose will be used as a tool to define and run applications along with their proper configuration. Future versions of the enabler will utilize **Kubernetes** to improve interoperability with other ASSIST-IoT enablers.	All components

NOTE: Since the decision to create the Location processing enabler was taken in M15, its overall functionality is still under active discussion. As a consequence, it may need to be extended/modified with respect to the description presented above.

3.1.5.2. Communication interfaces

Table 12. Communication interfaces (API) of the Location Processing enabler

Method	Endpoint	Description
GET	**/area/within**	Checks whether the point with specified coordinates (parameters x, y, z) is located within a pre-set area (identified by parameter id), with a specific precision (given by parameter precision).
POST	**/area/ball**	Creates or updates (when passed uuid value) a circular shape area of given dimension and positioning. If the parameter z is not specified a two-dimensional area is created.

NOTE: In the first edition of the enabler, only basic "sphere-oriented" functionality will be available. Features like more complex ways of defining/utilizing areas and stream-oriented processing will be added in future releases. Also implementation of the security mechanisms will be postponed until later release.

3.1.5.3. Use cases

The **first use case** involves an API client **creating an area definition**. The diagram and involved steps are the following ones:

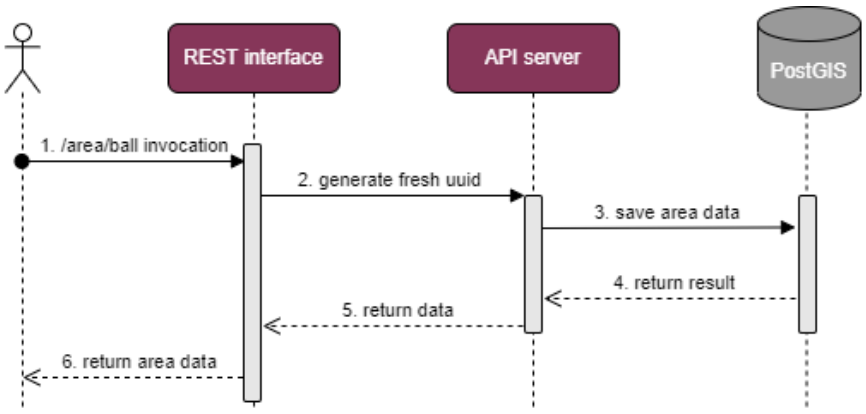

Figure 15. Location Processing enabler UC1 (define an area)

STEP 1: The client invokes the `/area/ball` API endpoint, passing an area specification that does not contain the `id` attribute.

STEP 2: A fresh uuid value is generated and the area data is sent to the PostGIS database.

STEP 3: The area data is saved into the PostGIS database.

STEP 4: The result of the save operation is returned from PostGIS to the API server.

STEP 5: The saved area data is returned to the communication interface.

STEP 6: The client receives notification message containing the complete area data, including the generated uuid value.

The **second use case** involves an API client **updating an area definition**. The diagram flow and related steps are the following:

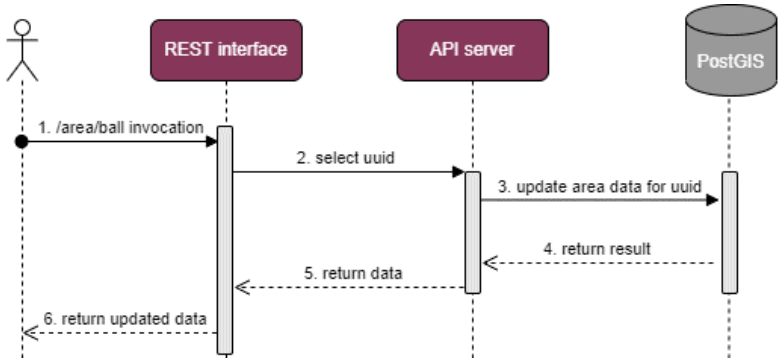

Figure 16. Location Processing enabler UC2 (update an area)

STEP 1: The client invokes the `/area/ball` API endpoint, passing an area specification that contains the `id` attribute.

STEP 2: The API server extracts the uuid.

STEP 3: The area with the given uuid gets updated with new parameter values.

STEP 4: PostGIS returns the result of the update to API server.

STEP 5: The result of the update gets sent to the communication interface.

STEP 6: The client receives notification message containing the complete updated area data.

The **third use case** involves an API client **checking if a given location lies within a specified area**.

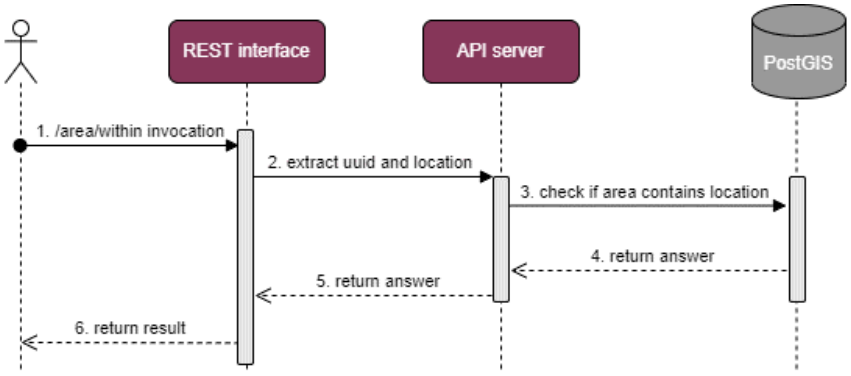

Figure 17. Location Processing enabler UC3 (check location).

STEP 1: The client invokes the `/area/within` API endpoint, passing an area identifier and location data.

STEP 2: The API server receives the uuid and location data from the communication interface.

STEP 3: The API server queries PostGIS server.

STEPS 4-6: The PostGIS returns the answer to the API server, which sends it to the communication interface so it is returned to the client.

3.1.5.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/location_process_enabler.html

Table 13. Implementation status of the Location Processing enabler

Category	Status
Components implementation	The initial design phase is being finalized and first steps of the implementation have been performed.
Feature implementation status	An initial version of the HTTP interface for the enabler has been created.
Encapsulation readiness	A simple containerized configuration for the enabler has been created.
Deployed with the Orchestrator in a laboratory environment	Not yet

NOTE: Since the idea of the *Location processing* enabler emerged in M15, it is currently in the (short and very active) design phase. As a starting point, input from the work on the original *Geolocalization* enabler has been taken. Shortly, the design will be finalised and the implementation will start.

NOTE: To provide the location processing enabler's data persistence and location-processing functionalities, the PostGIS extension for the PostgreSQL database will be used. The stream-oriented (asynchronous) services of the enabler will be based on Kafka streams, utilising the Apache Kafka event streaming platform.

3.1.6. Automated Configuration enabler

The configuration of resources in the Internet of Things environment is a complex task. The main challenges relate to the heterogeneity of resources, the distributed nature of the system, and non-trivial error scenarios. Automated Configuration enabler (AC) responds to these challenges and gives the IoT deployment a degree of independence from the human operator.

3.1.6.1. Structure and functionalities

Structure and functionalities have not changed with respect to deliverable D5.2 content. Internally, the AC represents system configuration as a (possibly disconnected) acyclic directed graph (DAG) with two kinds of vertices: resource and functionality, and edges representing the relation "requires to function". Edges can exist between pairs: (functionality, resource) and (functionality, functionality). Additionally, different "labels" can be associated with each vertex, allowing to categorise and group vertices, without changing the overall structure of the graph. Numerical values, associated with the functionalities, called weights, are used by the ACs to autonomously decide which functionalities should be maintained in the event of an error in any of the system components.

For the AC to function, it must be able to communicate with resources. Communication takes place via connectors. It is the responsibility of the connector to perform direct manipulations on the resource and to inform the AC about the status of the resource. Connectors allow to abstract away the problem of communication between resources and the AC.

The AC reacts to events regarding the resources by adjusting their configurations according to the predefined rules. Supported events include: the resource has been registered, the resource is no longer available, and messages with resource-type specific content and parameters. The available actions are: changing the configuration (all or appropriate nodes in the graph, along with possibly changing the labels), conditional action execution (depending on the received message and the current configuration status), maintaining functionality with the utmost importance, and no action.

3.1.6.2. Communication Interfaces

Communication interfaces have not changed with respect to deliverable D5.2 content.

3.1.6.3. Use cases

The general use case of updating configuration has not change with respect to deliverable D5.2 content. It has been updated, however, to cover handling of non-trivial failover scenarios. For details, see implementation status below.

3.1.6.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/automated_configuration_enabler.html

Table 14. Implementation status of the Automated Configuration enabler

Category	Status
Components implementation	The initial design phase is being finalized and the implementation has started.
Feature implementation status	The essential features of the enabler are under active development.
Encapsulation readiness	Components are yet to be encapsulated.
Deployed with the Orchestrator in a laboratory environment	Not yet

NOTE: Currently, the details of the internal design of the AC is being finalised, considering possible options in which the configuration structure and its application process will be represented and realised as well as how intelligence/automation aspects will be employed/handled. The code currently available from the repository represents some design experiments that have been performed so far.

3.2. Federated machine learning enablers

3.2.1. FL Orchestrator

3.2.1.1. Structure and functionalities

FL Orchestrator is one of the enablers developed in the context of the FL Architecture of the ASSIST-IoT project. It is responsible for specifying FL workflow(s)/pipeline(s) details. Among these details or features are:

- FL job scheduling
- Manage the FL life cycle
- Selecting and delivering initial version(s) of the shared algorithm
- Delivering the version(s) of modules used in various stages of the process, such as training stopping criteria
- Handling the different "error conditions" that may occur during the FL process.

Figure 18 depicts the high-level overview of the FL orchestrator components.

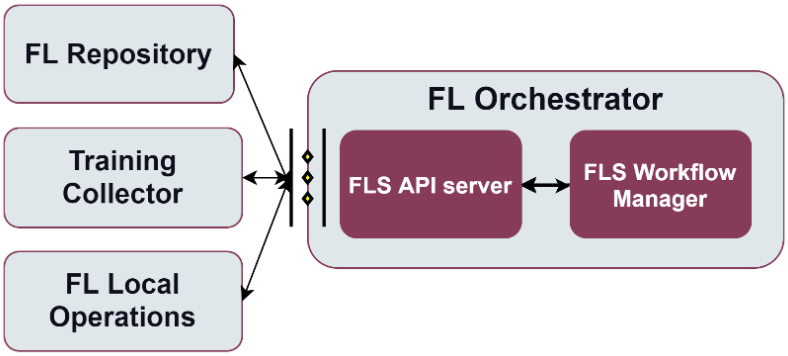

Figure 18. FL Orchestrator enabler structure.

As it can be seen, it is formed by two components:

- **FLS API server:** Offers a REST API to allow the communication and interaction with FL Structure components. Hence, it allows to retrieve information or perform FL management actions, to FL Local Operations, FL Training Collector and FL Repository.
- **FLS Workflow manager:** This component is in charge of defining workflow for a specific incarnation of FL lifecycle. Workflow description specifies, among others, the source of initial configuration (e.g., minimum number of FL Local Operations needed for federated training, number of training rounds for carrying out the federated learning process, the initial shared ML model to be used, evaluation criteria method and required accuracy value, method used for parameter aggregation, required encryption mechanisms), and lifecycle management (e.g., evaluating the number of FL Local Operations connected, or the number of training rounds finished provided by the FL Training Collector).

Implementation Technologies

Table 15. Implementation technologies for the FL Orchestrator enabler

Technology	Justification	Component(s)
Tensorflow/Flower	End-to-end open source platform for federated learning. It is the core of the FL orchestrator. Includes the definition of the API, interaction with other enablers and their main features.	FLS workflow manager, FLS API server
Flask API	Flask API in charge of carrying out all the previously described procedures	FLS API server
MongoDB	Used by the orchestrator to retrieve the list of existing models to the FL repository.	FL Orchestrator DB

3.2.1.2. Communication interfaces

Table 16. Communication interfaces (API) of the FL Orchestrator enabler

Method	Endpoint	Description
GET	**/models**	Receives the models collected in the FL Repository
GET	**/modelID/<id>**	Receives the model <id> collected in the FL Repository
GET	**/configurationsbyModel/<id>**	Recover configurations of model <id> collected in the FL Repository
POST	**/addModel**	Add a new federated learning model to the FL Repository
POST	**/addConfigByModel/<id>**	Add new configuration to a model <id> to be stored in the FL repository
DELETE	**/deleteModel/<id>**	Delete a federated learning model from the FL Repository
GET	**/runModel/<id>**	Request the setup for training of the FL model <id> to the FL Local Operations and FL Training Collector
GET	**/startFLTraining/<id>**	Start training iteration
GET	**/modelsRunning**	Receive the models currently being training from the FL Training Collector
GET	**/status**	Receive the status of all the FL Local Operations involved in the training
GET	**/trainingRound**	Receive the currently finalized training round from the FL Training Collector
GET	**/stopModel/<id>**	Request stopping the FL model <id> to the FL Local Operations and FL Training Collector

NOTE: Several external APIs with other FL enablers of the system are available. The table list above shows the already implemented API methods and endpoints, but new ones will be added in the following releases of the enabler.

3.2.1.3. Use cases

There are 4 use cases that apply to this enabler. The **first use case** is related to the **initial setup/configuration of the FL training process** via the customized web application foreseen for the ASSIST-IoT FL system. Its diagram and involved steps are the following:

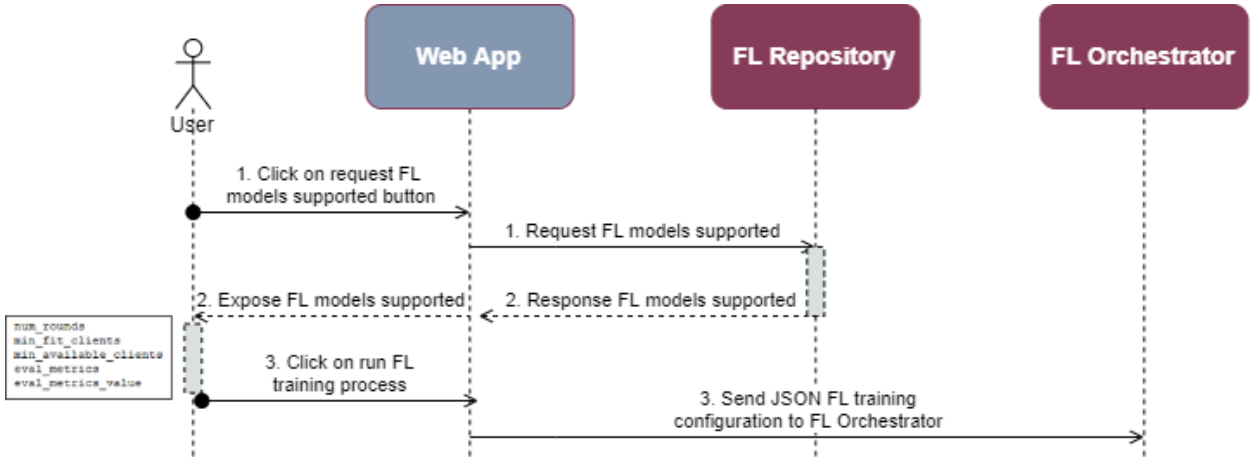

Figure 19. FL Orchestrator UC1 (user configures FL training)

STEP 1: User opens the webpage containing the FL system, and clicks on the request FL models supported button, which communicates directly with the FL repository that exposes the supported ones.

STEP 2: The user modifies according to their preferences/interests the configuration of the provided models supported by the system for starting a new training (e.g., *num_rounds*, *min_fit_clients*, *eval_metrics*, *eval_metrics_value*).

STEP 3: The user finally clicks on the Run button, which sends in JSON format, the new FL training configuration to the FL Orchestrator.

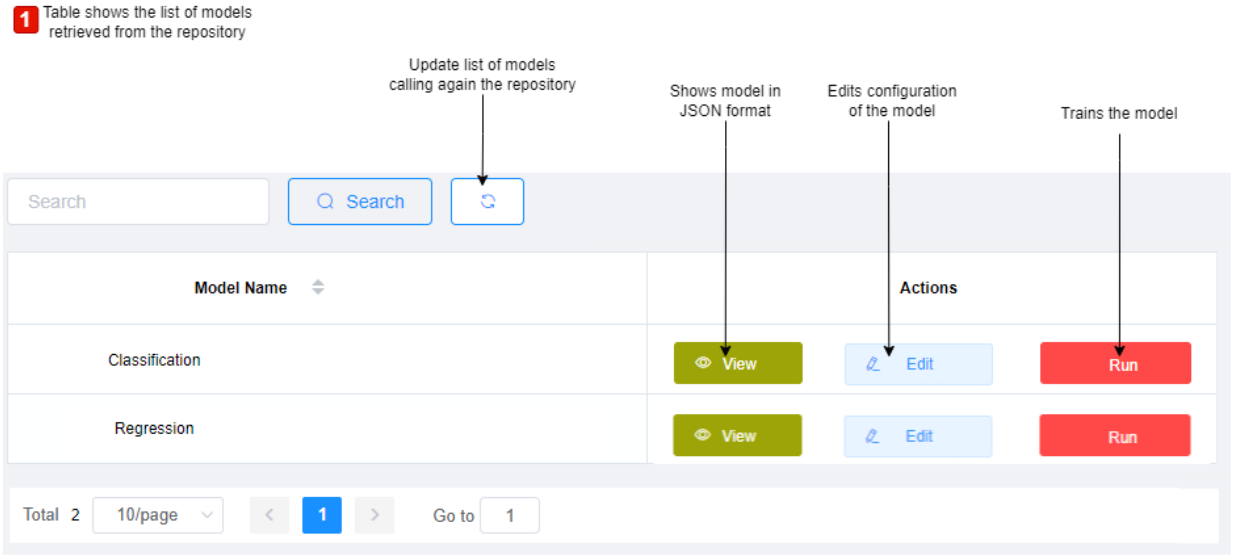

Figure 20. Mock-up FL System Web App (under development)

The **second use case** is related to the **initial connection** of the FL Orchestrator **with the FL Training Collector and FL Local Operations**, which receive the FL training configuration setup defined with the previous use case. The diagram is the following:

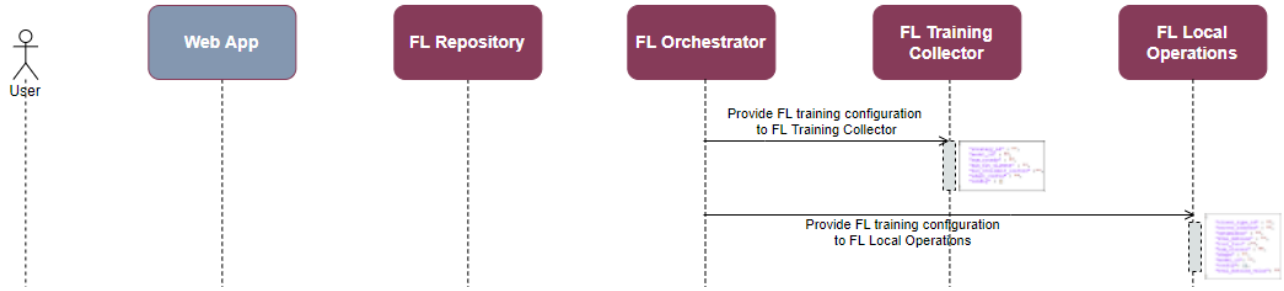

Figure 21. FL Orchestrator UC2 (initial setup of FL architecture)

STEP 1: The FL Orchestrator connects with the FL Training Collector and the different FL Local Operations involved in the FL training (which are defined by the user).

STEP 2: The FL Orchestrator forwards the FL training configuration to the FL Training Collector and the different FL Local Operations connected.

The **third use case** is related to the **lifecycle management of the FL Training Collector**. Two potential scenarios can be devised: a happy path in which all the FL training process is performed successfully, or an error caused due to the decoupling of some FL Local Operations, leading to a handling error situation. Both diagrams are shown in the next figures:

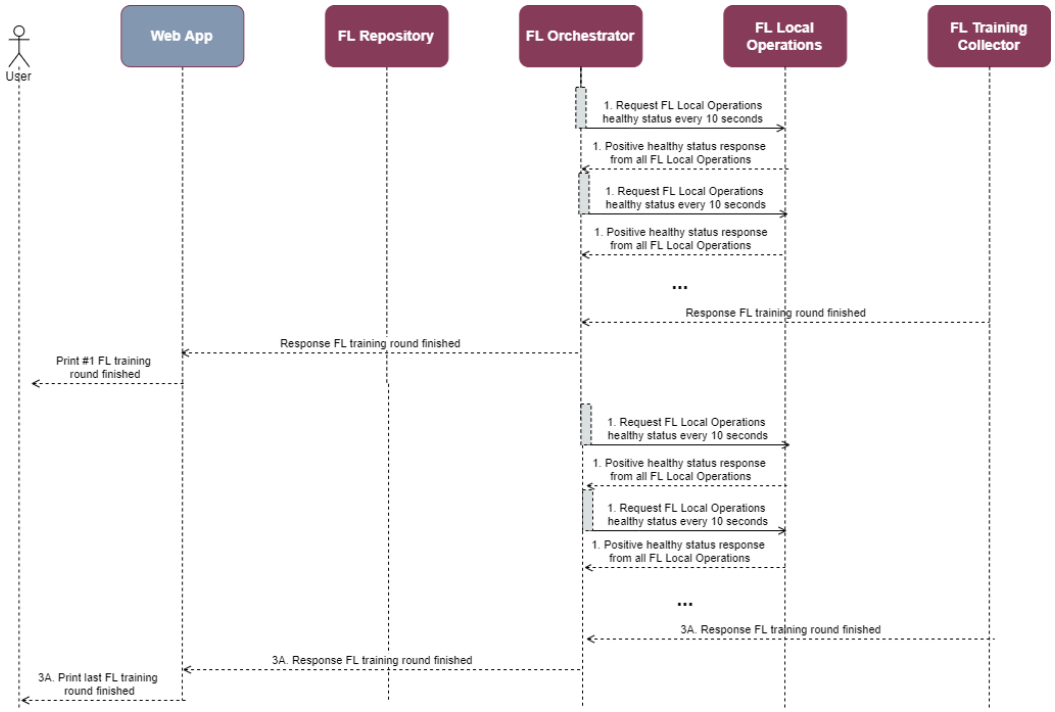

Figure 22. FL Orchestrator UC3 (lifecycle management of FL Training Collector – Option A: happy path)

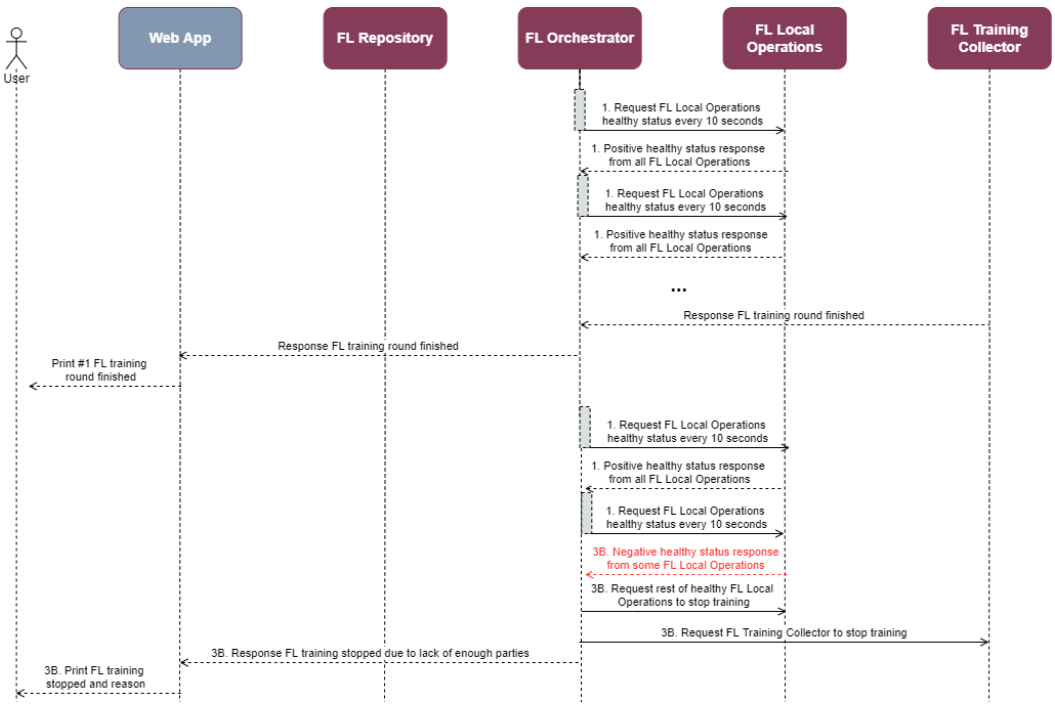

Figure 23. FL Orchestrator UC3 (lifecycle management of FL Training Collector – Option B: FL local operations below minimum)

STEP 0: Once the FL Training Collector and all FL Local Operations have received the FL training configuration, they start the training

STEP 1: The FL Orchestrator requests periodically the status to the FL Local Operations and FL Training Collector informs to the FL Orchestrator when a new FL training round has been ended, which is forwarded to the Web application in order to allow the user be aware of current status of the FL training.

STEP 3A: The FL training process is successfully executed and finished, and a new FL model is available at the FL Repository. The FL Orchestrator informs to the user about the ending of the FL training process as well as about the location of the new trained FL model address in the FL Repository.

STEP 3B: An error caused due to the decoupling of some FL Local Operations occurred, leading to having connected less FL Local Operations than the minimum required by the user. The FL Orchestrator requests to the rest of connected FL Local Operations and FL Training Collector to stop the FL training process, and informs the user about the unexpected ending and about the location of the not finished trained FL model address in the FL Repository.

3.2.1.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_orchestrator.html

Table 17. Implementation status of the FL Orchestrator

Category	Status
Components implementation	All the FL Orchestrator components are implemented in their Minimum Viable Product version.
Feature implementation status	Currently, the FL system is working on a happy paths scenario, i.e., no errors/attacks are contemplated. In next releases, an in-depth error handling criterion, as well as more advanced evaluation metrics will be included.
Encapsulation readiness	The FL Orchestrator is provided as a Docker image, and by means of a Docker-Compose yaml file, it is possible to generate a first complete end-to-end FL system. No k8s environment is supported yet
Deployed with the Orchestrator in a laboratory environment	No. In principle, the FL system enablers will not be used in real environments, only at simulation scenario in laboratories, so that the integration with the orchestrator is not mandated.

3.2.2. FL Training Collector

3.2.2.1. Structure and functionalities

The FL training process involves several independent parties that commonly collaborate in order to provide an enhanced ML model. In this process, the different local updates suggestions shall be aggregated accordingly. This duty within ASSIST-IoT will be tackled by the FL Training Collector (FLTC), which will also be in charge of delivering back the updated model. **Structure and functionalities have not changed with respect to deliverable D5.2 content.**

Implementation technologies

Table 18. Implementation technologies for the FL Training Collector

Technology	Justification	Component(s)
Python	Python is an interpreted high-level general-purpose programming language with a set of libraries. Very popular for data analysis and ML applications.	FLTC I/O, FLTC Combiner
FedML	Research library and benchmark for Federated ML containing federated algorithms and optimizers.	FLTC Combiner
FastAPI	A popular web microframework written in Python, it is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	FLTC I/O
Flower	A FL framework designed to work with a large number of clients. It is both compatible with a variety of ML frameworks and supports a wide range of devices.	FLTC Combiner

3.2.2.2. Communication interfaces

Table 19. Communication interfaces (API) of the FL Training Collector

Method	Endpoint	Description
POST	**/job/config/{id}**	Receive configuration of FL Training Collector components for job with identifier id.
GET	**/job/status/{id}**	Retrieve status of the training process with identifier id.

NOTE: With respect to D5.2 endpoint /model/update/{id}/{version} was removed. The communication between FL Training Collector and FL Local Operations enablers used gRPC protocol and is handled according to Flower framework procedure.

3.2.2.3. Use cases

The **first use case** is about what happens after **instantiation of FL Trainings Collector**, i.e., configuration of appropriate modules (here diagram is not used because components are to be instantiated). The involved steps are:

STEP 1: Receive configuration information from FL Orchestrator.

STEP 2: Establish topology to use, e.g., master-slave, with mediator.

STEP 3: Retrieve from FL Repository appropriate FL Collector (averaging algorithms).

STEP 4: Initialize averaging algorithm e.g. single step, sequential.

The **second use case** is **combining local updates to the model to obtain an updated final model**.

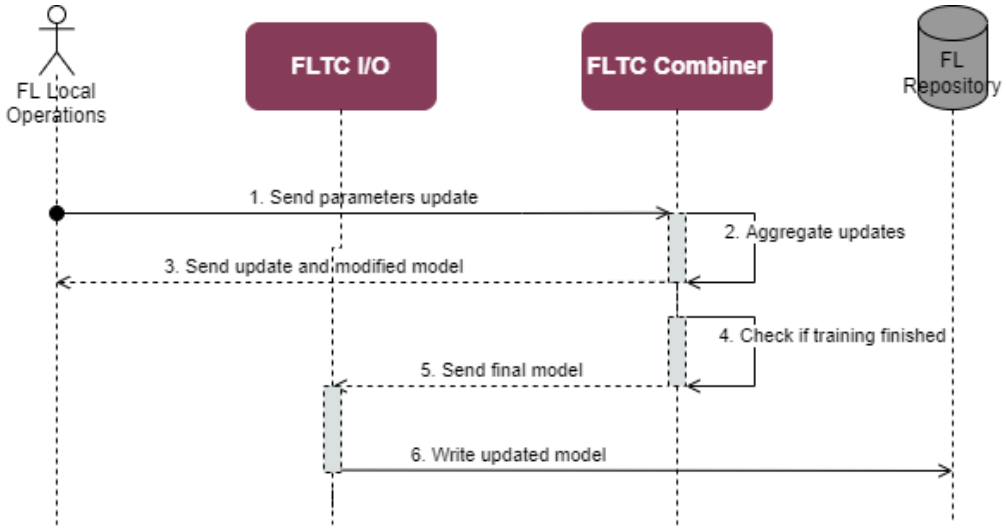

Figure 24. FL Training Collector UC2 (local results aggregation)

STEP 1: FL Local Operations enabler sends local results (parameters updates proposals) of model training to FL Local Training Collector enabler. The proposed update is handled by FLTC Combiner component.

STEP 2: FLTC Combiner combines local results to deliver new a shared model version. Averaging can be completed in one step or can be applied sequentially in a specific order.

STEP 3: FLTC Combiner sends an aggregated weights and model to FL Local Operations (as part of the training process).

STEP 4: FLTC Combiner verifies if model training procedure has been finished or it should still wait for local updates.

STEPS 5-6: If the training process is finished FLTC Combiner sends final model to FLTC I/O which forwards in to FL Repository enabler to be stored and distributes it to FL Local enablers.

3.2.2.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_training_collector.html

Table 20. Implementation status of the FL Training Collector

Category	Status
Components implementation	FLTC I/O - all endpoints API endpoints are ready. FLTC Combiner - in progress (communication with FL Local Operations and FL Repository and running the training process is ready; more strategies and status reporting to FL Orchestrator will be added).
Feature implementation status	• Communication with other FL enablers is ready. • Aggregation of local updates of the ML model prepared by independent parties as part of a model enhancement process is ready, however more strategies will be added. • Saving model to the FL Repository with metadata (final metadata will be specified) is ready.
Encapsulation readiness	Enabler is containerised in Docker.
Deployed with the Orchestrator in a laboratory environment	No (same explanation that FL Orchestrator).

3.2.3. FL Repository

3.2.3.1. Structure and functionalities

The FL Repository will be a set of different databases, including initial ML algorithms, already trained ML models suitable for specific data sets and formats, averaging approaches, and auxiliary repositories for other additional functionalities that may be needed, and are not specifically identified yet. **Structure and functionalities have not changed with respect to deliverable D5.2 content.**

Implementation technologies

Table 21. Implementation technologies for the FL Repository

Technology	Justification	Component(s)
RDF	W3C Resource Description Framework Description (RDF) is a standard for representing information on the Web designed as a data model for metadata. It is one of the foundations for semantic technologies. It will provide flexible and adaptable model for ML algorithms metadata or any auxiliary data.	ML Algorithms library, Auxiliary
FedML	Research library and benchmark for Federated ML containing federated algorithms and optimizers.	FL Collectors, Auxiliary
Python	Python is an interpreted high-level general-purpose programming language with a set of libraries. Very popular for data analysis and ML applications.	Local communication
FastAPI	A popular web microframework written in Python, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	Local communication
MongoDB	MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program.	ML Models Libraries, Auxiliary

3.2.3.2. Communication interfaces

Table 22. Communication interfaces (API) of the FL Repository

Method	Endpoint	Description
POST	**/model**	Adds a new ML model to the library
PUT	**/model/{id}/{version}**	Update model that is already in the repository under identifier id and version

GET	**/model**	Retrieve list of all models stored in the repository
GET	**/model/{id}/{version}**	Retrieve model with a specific identifier and version
DELETE	**/model/{id}/{version}**	Delete a model with a specific identifier and version
POST	**/algorithm**	Add new ML algorithm to the repository
PUT	**/algorithm/{name}/{version}**	Update algorithm that is already in the repository with a given name and version
GET	**/algorithm**	Retrieve list of all ML algorithms stored in the repository
GET	**/algorithm/{name}/{version}**	Retrieve a ML algorithm identified with a given name and version
DELETE	**/algorithm/{name}/{version}**	Delete a ML algorithm with a specific name and version
POST	**/collector**	Add new ML training collector algorithm to the repository
PUT	**/collector/{name}/{version}**	Update ML training collector algorithm that is already in the repository with a given name and version
GET	**/collector**	Retrieve list of all ML training collector algorithms stored in the repository
GET	**/collector/{name}/{version}**	Retrieve a ML training collector algorithm identified with a given name and version
DELETE	**/collector/{name}/{version}**	Delete a ML training collector algorithm with a specific name and version

NOTE1: At this moment, endpoints for accessing to auxiliary data are not defined. They will be added when specific needs are encountered during the project. The table above describes the API methods.

NOTE2: With respect to information in D5.2, PUT endpoints for updating a model/algorithm/collector have been split into two: one for the object and a second one for metadata.

3.2.3.3. Use cases

Use cases have not changed with respect to the deliverable D5.2 content.

3.2.3.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_repository.html

Table 23. Implementation status of the FL Repository

Category	Status
Components implementation	Local communication - all endpoints are ready. ML Algorithms Libraries, ML Model Libraries, FL Collectors - ready as noSQL storage.
Feature implementation status	• External API to insert and retrieve data from the storage is ready. • To be defined is a final set of metadata that should be stored in the repository besides objects of different types.
Encapsulation readiness	Enabler is containerized in Docker.
Deployed with the Orchestrator in a laboratory environment	No (same explanation that FL Orchestrator).

3.2.4. FL Local Operations

3.2.4.1. Structure and functionalities

FL Local Operations enabler is an embedded enabler within each FL involved party/device of the FL systems. **Structure and functionalities have not changed with respect to deliverable D5.2 content.**

Implementation technologies

Table 24. Implementation technologies for the FL Local Operations

Technology	Justification	Component(s)
scikit-learn	A popular machine learning library often used for data pre-processing and transformation, for example encoding labels. It is open source and widely used in the industry.	Data Transformer
pyTorch	An open source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR).	Local Model Trainer
TensorFlow	A free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.	Local Model Trainer, Local Model Inferencer
Flower	A federated learning framework designed to work with a large number of clients. It is both compatible with a variety of ML frameworks and supports a wide range of devices.	Local Model Trainer
OpenVINO	A free toolkit facilitating the optimization of a deep learning model. It is cross-platform and free to use.	Local Model Inferencer
OpenCV	A real-time computer vision library providing already optimized models. It is cross-platform and open-source.	Local Model Inferencer
Python	Python is an interpreted high-level general-purpose programming language with a set of libraries. Very popular for data analysis and ML applications.	Data Transformer, Local Communication
Pailier Encryption, Affine Homomorphic Encryption	Two homomorphic encryption algorithms that will be used to preserve the privacy of the data without affecting the performance of the model.	Privacy
FastAPI	A popular web microframework written in Python, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	Local Communication

3.2.4.2. Communication interfaces

Table 25. Communication interfaces (API) of the FL Local Operations

Method	Endpoint	Description
POST	**/job/config/{id}**	Receive configuration for training job
PUT	**/model/{id}/{version}**	Receive new shared model
GET	**/status**	Get current status of the enabler

NOTE: For communication between FL Local Operations and FL Training Collector during the training process gRPC protocol is used (as in Flower framework).

3.2.4.3. Use cases

Use cases have not changed with respect to the deliverable D5.2 content.

3.2.4.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_local_operations.html

Table 26. Implementation status of the FL Local Operations

Category	Status
Components implementation	Local communication - API endpoints for configuration, model and status are ready. Endpoints for data transformation and prediction need to be implemented. Local model trainer - is in progress. The functionality of conducting training and communicating with FL Training Collector is ready. Integration with FL Repository needs to be added. Local model inferencer - to be done. Privacy - to be done. Data transformation - to be done.
Feature implementation status	• Communication with FL Training Collector and FL Orchestrator is ready. • Local model training is ready. • Retrieving model from FL Repository needs to be added. • Verification of local data formats compatibility with data formats required by FL and handling data transformations needs to be added. • Communication of model updates via encryption mechanisms needs to be added.
Encapsulation readiness	Enabler is containerised in Docker.
Deployed with the Orchestrator in a laboratory environment	No (same explanation that FL Orchestrator).

3.3. Cybersecurity enablers

3.3.1. Cybersecurity Monitoring enabler

3.3.1.1. Structure and functionalities

Cybersecurity monitoring enabler will consolidate the necessary information for cyber threat detection over the deployed architecture and pilots. Cybersecurity monitoring enabler provides cyber security awareness and visibility on cybersecurity objectives and will provide infrastructure cybersecurity monitoring.

The cybersecurity monitoring server will be responsible of collecting, processing, and analysing the incoming information from the infrastructure under study. It will consolidate an output that will provide cybersecurity monitoring information related to different events. Figure 25 describes cybersecurity monitoring components and describes how cybersecurity monitoring output will be alerts resulted from the processing of security events using rules.

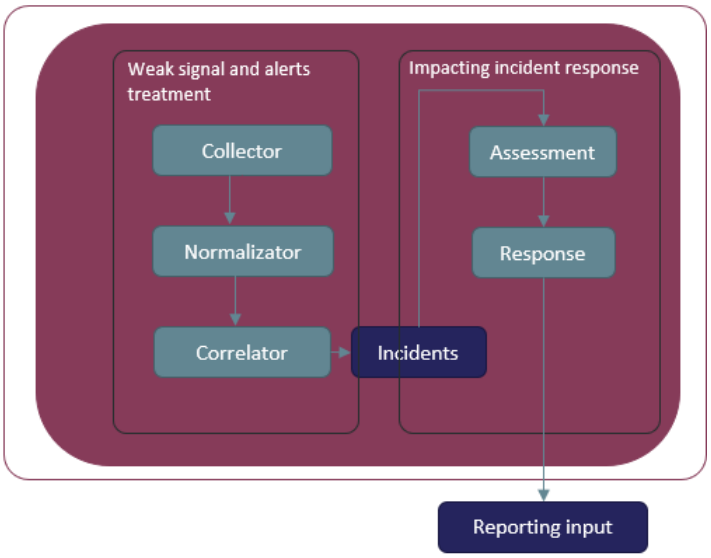

Figure 25.Cybersecurity Monitoring enabler high level structure

The functionalities of the Cybersecurity Monitoring enabler are presented below:

- It will receive logs and information from the agents deployed.
- It will decode the log, identify the log type and extract some useful fields.
- It will have a ruleset to be applied to the received logs.
- It will apply the active rules to the received log, and if there is a match, it will generate an alert.
- It will normalize the alert event and correlate until it determines if it is only a simple alert or a real incident.
- It will enrich the incident with useful information, to facilitate the assignment of the risk level of the incident and the response actions to be done.
- It can do predefined actions for incident mitigation depending on the incident, such as communicate with the agent so that it performs an action, send an email or send the incident to a ticketing system.

Cybersecurity enabler will update information on a GUI so that the admin user can see the status of the agents and the alert/incident information.

Implementation technologies

Table 27. Implementation technologies for the Cybersecurity Monitoring enabler

Technology	Justification	Component(s)
Wazuh server	Analysis	Decoder, rule engine, correlator
Elasticsearch, Filebeat, Logstash and Kibana	Data gathering, storage and visualization	Associated to visualization, and data storage
The Hive	Security orchestration and response	Incident Response
Cortex and MISP	Threat intelligence and threat sharing platforms for digital forensics and incident response	External enrichment

3.3.1.2. Communication interfaces

Table 28. Communication interfaces (API) of the Cybersecurity Monitoring enabler

Method	Endpoint	Description
GET	**/manager/status**	Return the status of the monitoring server
GET	**/manager/info**	Return basic information such as version, compilation date, installation path
GET	**/manager/configuration**	Return enabler configuration used.
PUT	**/manager/configuration**	Replace configuration with the data contained in the API request
GET	**/manager/stats**	Return statistical information for the current or specified date
PUT	**/manager/restart**	Restart the manager
GET	**/agents**	Obtain a list with information of the available agents
DELETE	**/agents**	Delete all agents or a list of them based on optional criteria
POST	**/agents**	Add a new agent with basic info
POST	**/agents/insert**	Add an agent specifying its name, ID and IP. If an agent with the same ID already exists, replace it using 'force' parameter
PUT	**/agents/{agent_id}/restart**	Restart the specified agent
PUT	**/agents/restart**	Restart all agents or a list of them
PUT	**/active-response**	Run an Active Response command on all agents or a list of them

NOTE: Cybersecurity monitoring server will implement a restful API to manage monitoring server basic configuration and cybersecurity agents connected.

3.3.1.3. Use cases

The **main use case** behind the cybersecurity monitoring server is described in the following flow, showing the steps that will happen to be **protected against cyberthreats**, from the processing of a log to its evaluation and, if needed, a generated response:

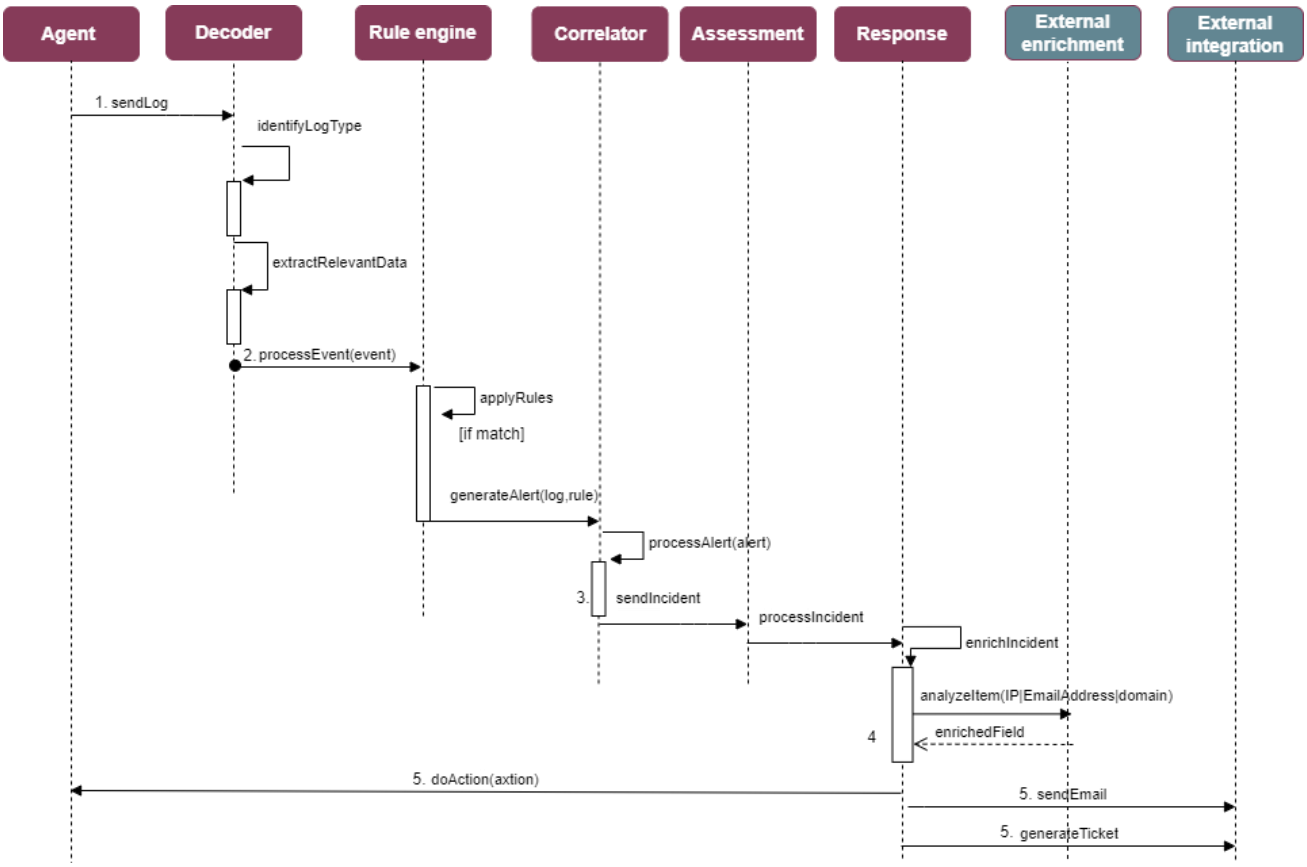

Figure 26. Cybersecurity Monitoring enabler UC (cyberthreats protection)

STEP 1: Agent detected event associated to system log monitoring running in the agent side.

STEP 2: Decoder at server component side extract the relevant data and forward to the rule engine component.

STEP 3: Rule engine process and apply rules accordingly and forward to the Assessment.

STEP 4: Response components will automate and orchestrate cybersecurity response, gathering and enriching the information on the cybersecurity incident using external enrichment services if needed.

STEP 5: External interaction component will be triggered from the Response component to arise any action using the agent or any other external interaction.

Use cases and **additional user stories** associated to cybersecurity monitoring server **are**:

- Agent detect events associated to identification, authentication, and authorization.
- Agent detects installation of new and non-permitted software, on the system under monitoring and report to the server.
- Agent detects abuse of authorization on the system under monitoring and report to the server.
- Agent detects unauthorised changed of configuration files and report to the server.

3.3.1.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/cybersecurity_monitoring_enabler.html

Table 29. Implementation status of the Cybersecurity Monitoring enabler

Category	Status
Components implementation	All the components of the enabler have a first functional version
Feature implementation status	The basic features are in place. However, some developments are pending: • Orchestrate the enabler and integrate with other enablers.
Encapsulation readiness	Components have been containerised as Docker images and deployed using Docker Compose. The Helm chart is under development. Manifests in Kubernetes are done, but they are in testing-phase for the integration of all related technologies.
Deployed with the Orchestrator in a laboratory environment	Not yet. The enabler is in Docker-Compose orchestration tested on a virtual environment and with two common use cases with Windows and Linux operating systems, monitoring all the agents, the status and security of the hosts.

3.3.2. Cybersecurity Monitoring Agent enabler

3.3.2.1. Structure and functionalities

The Cybersecurity Monitoring Agent enabler will report to the security monitoring server. It enables the execution of processes on the system target under study to provide relevant information if a cybersecurity breach is produced. This enabler will perform functions of an endpoint detection and response system, monitoring and collecting activity from end points that could indicate a cybersecurity threat.

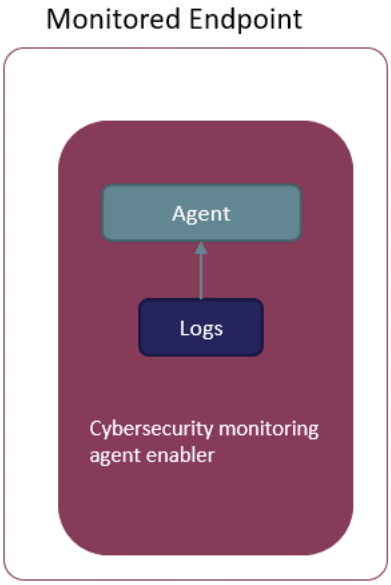

Figure 27. Cybersecurity Monitoring Agent enabler structure

The functionalities of the Cybersecurity Monitoring Agent enabler are presented below:

- It will collect and process the system events and system log messages.
- It will monitor file integrity of critical files and audit data of the system.
- It will monitor the security of the docker engine API and the container at runtime.
- It will be able to perform some actions such as blocking network connection or stopping running processes if the Cybersecurity monitoring enabler requests it.

Table 30. Implementation technologies for the Cybersecurity Monitoring Agent enabler

Technology	Justification	Component(s)
Wazuh agent	Collection of logs	Agent
Rsyslog	Collection of logs	Agent

3.3.2.2. Communication interfaces

Table 31. Communication interface (TCP/UDP) of the Cybersecurity Monitoring Agent enabler

Without TLS (to Wazuh-server)	**Dedicated port (1514 by default)**	To communicate with the Cybersecurity Monitoring enabler (to be register on it and to send the collected data)
With TLS (to Wazuh-server)	**Dedicated port (1515 by default)**	
rsyslog-based implementations	**Dedicated port (standard 514)**	

3.3.2.3. Use cases

The **main use case** behind the Cybersecurity Monitoring Agent enabler is described in the following flow, from the **collection of data** to their send to the Cybersecurity Monitoring enabler and, if needed, the **execution of an action**:

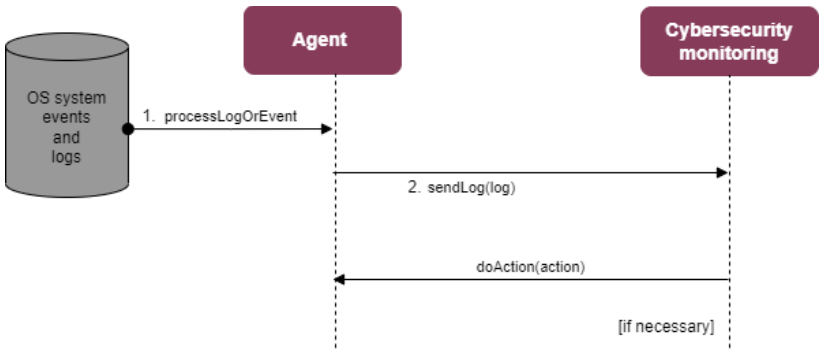

Figure 28. Cybersecurity Monitoring Agent enabler UC (send collected data and actuate)

STEP 1: Agent detected event associated to system log monitoring running in the agent side and collected by the agent daemon.

STEP 2: Cybersecurity monitoring server receives agent information and process the relevant data using the components described in the and forward to components described in the cybersecurity monitoring enabler.

Use cases and additional user stories associated to Cybersecurity Monitoring agent are

- Detection of events associated to identification, authentication, and authorization.
- Agent detects installation of new and non-permitted software, on the system under monitoring and report to the server.
- Agent detects abuse of authorization on the system under monitoring and report to the server.
- Agent detects unauthorised changed of configuration files and report to the server.

3.3.2.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/cybersecurity_monitoring_agent_enabler.html

Table 32. Implementation status of the Cybersecurity Monitoring Agent enabler

Category	Status
Components implementation	All the components of the enabler have a first functional version
Feature implementation status	The basic features are in place. However, some developments are pending: • Orchestrate the enabler and integrate with other enablers.
Encapsulation readiness	Components have been containerised as Docker images and deployed using Docker Compose. The Helm chart is under development. Manifests in Kubernetes are done, but they are in testing-phase for the integration of all related technologies.
Deployed with the Orchestrator in a laboratory environment	Not yet

3.3.3. Identity manager enabler

3.3.3.1. Structure and functionalities

Identity Manager (IdM) enabler will be responsible for managing identities on the access control process. Authentication is a process by which the credentials provided by an entity (computer, application, or person) are compared with those stored in the system to ensure that said entity is effectively who or what it claims to be. Its main goal is to ensure that only authenticated entities are granted access to the specific resource (systems, applications, or IT environments) for which they are authorized. This includes control over entities (i.e., user provisioning, or entities provisioning) and the process of onboarding new entities (i.e. users, systems, etc.).

IdM enabler will perform the authentication phase of access control process, processing and validating the identity for later control of the access to the resources by the authorization enabler. It will rely on OAuth2 protocol, which allows the delegation of the authentication process to a remote server, granting a communication that keeps entity (user or system) authentication data secure. Afterwards, the IdM will communicate with the Authorization enabler also by means of OAuth2, implementing XACML policies. Figure 29 depicts the general structure of the enabler:

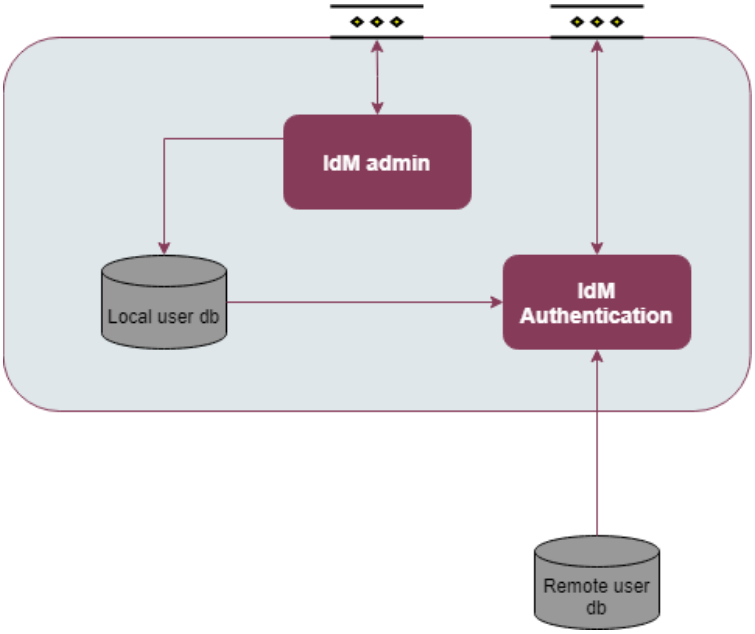

Figure 29. Identity Manager enabler structure

The functionalities of the IdM are presented below:

- IdM will provide a central user database and management console.
- IdM will be able to work federated with remote user databases, unifying remote user stores.
- IdM will provide Single-Sign-On capabilities through OAuth2 protocol.
- IdM will integrate with the Authorization enabler in order to offer a common authorization and authentication process.

Table 33. Implementation technologies for the Identity Manager enabler

Technology	Justification	Component(s)
Keycloak	Identiy and Access Management core component	IdM authentication
Oauth2	Standard web federated identity	IdM authentication
LDAP connector	External user store	IdM authentication
Web interface	Manage user database	IdM Admin

3.3.3.2. Communication interfaces

Table 34. Communication interfaces (API) of the Identity Manager enabler

Method	Endpoint	Description
GET	**/v1/users**	Create a user
POST	**/v1/users**	List users
GET	**/v1/users/user_id**	Read info about a user
PATCH	**/v1/users/user_id**	Update a user
DELETE	**/v1/users/user_id**	Delete a user

NOTE1: Enabler endpoints (REST API assumed, but may be others). The enabler should have a primary interface for communicating with other enablers or applications (its components communicate through internal communication mechanisms).

NOTE2: The following table describe initial descriptions for basic operations on Identity Manager enabler, just as a reference. Complete API specification is still to be extended.

3.3.3.3. Use cases

The following flow describes the two main use cases; the **first use case** is related to an **administrator user which registers a new user** to the system (step #1). The **second use case** involves a **user authenticating in the system**, and the actions performed by the IdM to accept it or not.

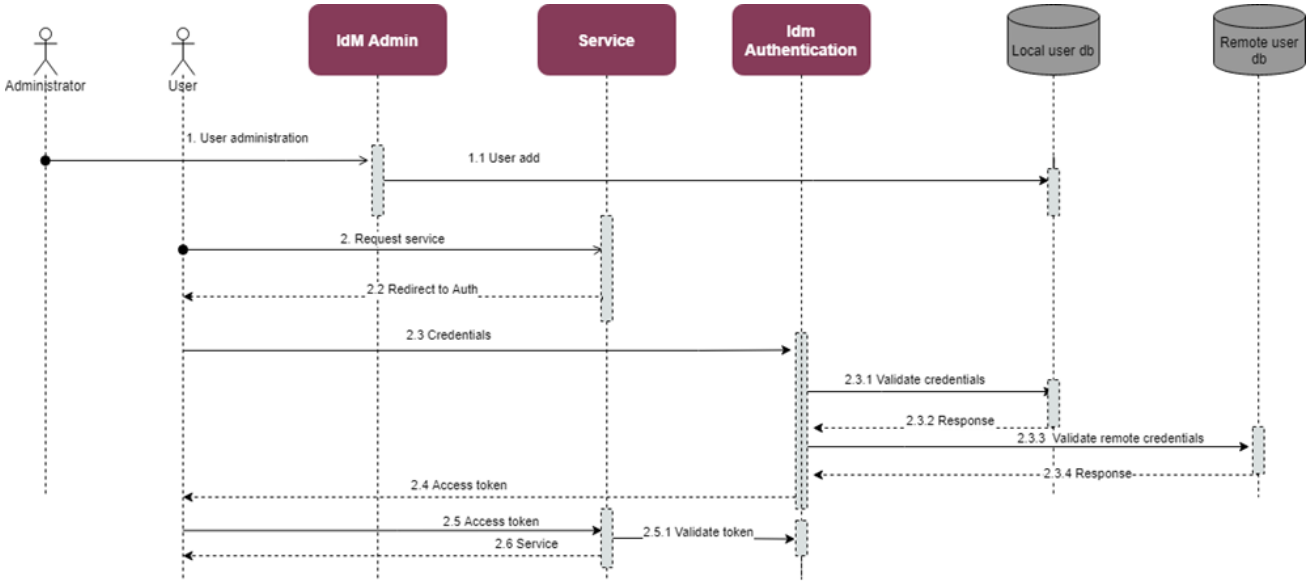

Figure 30. Identity manager enabler UCs (add user, authenticate user)

STEP 1: An administrator populates user database.

STEP 2: A user requests a service from an APP.

STEP 2.2: If the user has no previous identification active, it is redirected to the Authentication server.

STEP 2.3: User identifies himself in the IdM and obtains a session token

STEP 2.4. If local user store has no identity for credentials, request may be federated to a remote user DB.

STEP 2.5: User presents the token to the application server.

STEP 2.5.1: Token is validated against the IdM.

STEP 2.6: If the token is valid, the client can access the server.

3.3.3.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/identity_manager_enabler.html

Table 35. Implementation status of the Identity Manager enabler

Category	Status
Components implementation	All the components of the enabler have a first functional version.
Feature implementation status	The basic features are in place, including basic API. However, some developments are pending: • Definition of final interface requirements. • Definition of use case needs and architecture to complete integration
Encapsulation readiness	Components have been containerised as Docker images and deployed using Docker Compose. The Helm chart is under development.
Deployed with the Orchestrator in a laboratory environment	Not yet

3.3.4. Authorization enabler

3.3.4.1. Structure and functionalities

The Authorization enabler will be responsible for authorization phase in the access control process. Authorization is a process of granting, or automatically verifying, permission to an entity (computer, application, or person) to access requested information after the entity has been authenticated. The enabler is based on XACML standard security policies, which results on obligations actions to be deployed after the evaluation process. Authorization enabler is composed with different components as described below:

- Policy Administration Point (PAP). Point which manages access authorization policies.
- Policy Decision Point (PDP). Point which evaluates access requests against authorization policies before issuing access decisions.
- Policy Enforcement Point (PEP). It responds to where enforcement is going to take place.
- Policy Information Point (PIP). It provides attribute values upon request from the PDP context.

Its general structure is presented in the figure below. It describes two different modes of deploying the same enabler. It can function as federated server, an autonomous edge service, or interact between both.

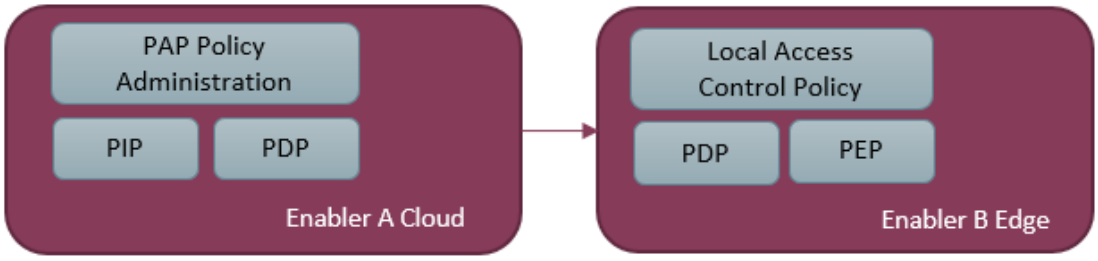

Figure 31. Authorization enabler structure

In ASSIST-IoT, a federated Authorization enabler will distribute a security policy from cloud to the edge to be locally evaluated by the PDP and enforced locally by the PEP. Federated PAP policy will be controlled by an administrator team and replicated locally in a local Access Control Policy. The functionalities of the enabler are the following:

- PAP will provide a Web administrator to create and deploy the security policy to the different devices.
- A service that wants to use the authorization service will have a PEP, enforcement point to make request to the authorization server, this is asking whether the access should be granted or not.
- PDP provides a REST interface available to the PEP to receive the request and orchestrate the process.

- PIP will be responsible of generating the context for the request and obtaining any data that external provider can offer to be incorporated to the request.
- The Policy repository will store locally to the PDP the policy to be applied.
- PDP will evaluate the request against the policy and will respond with the response.
- Obligation server will launch external requests to perform the derived actions (obligations) obtained as a result of the policy decision. This will have the form of REST requests.

Table 36. Implementation technologies for the Authorization enabler

Technology	Justification	Component(s)
XACML	Policy definition and evaluation	PAP, PDP, PEP, PIP
REST interfaces	Inter module comunications	PAP, PDP
MQTT	Trace publication	PDP

3.3.4.2. Communication interfaces

Table 37. Communication interfaces (API) of the Authorization enabler

Method	Endpoint	Description
POST	**/evaluate?resource=<domain>@<resource>&action=<action>&code=<id>**	Evaluates a request performed and authorises it or not depending on the stored policies.

3.3.4.3. Use cases

The **main use case** behind the Authorization server is described in the following diagram, which describes the **entire authorization flow**:

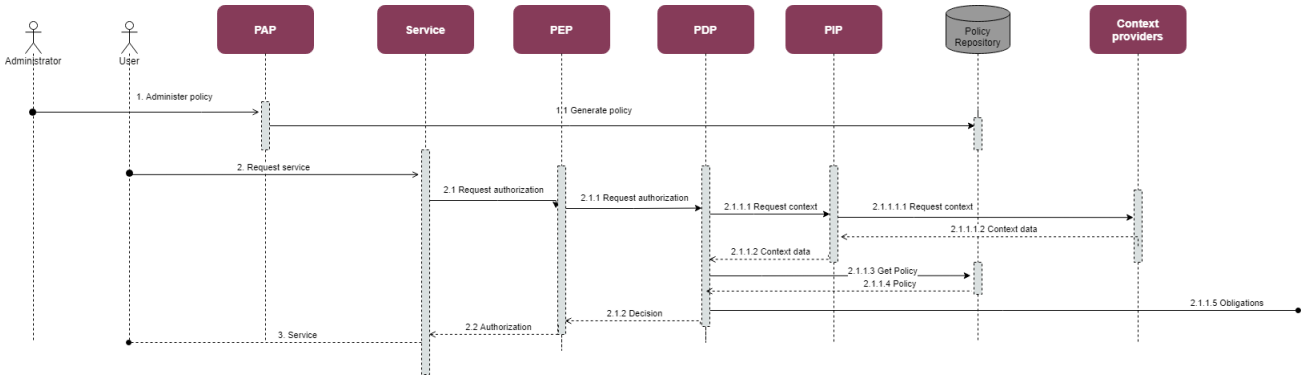

Figure 32. Authorization enabler UC (authorization flow)

STEP 1-1.1: An Administrator defines the data elements to be used in the validation process (conditions, pre-shared keys, context data…) and exports it to the policy storage.

STEP 2: A user requests the access to the service provided in the device. After identification, the PEP will generate an access request **STEP 2.1** and send it to the PDP **STEP 2.1.1.**

STEP 2.1.1.1: PDP will request the PIP to gather the context required for the decision **STEP 2.1.1.1.1.**

STEP 2.1.1.2: PDP will complete the request, get the policy from the storage **STEP 2.1.1.3** and obtain a decision **STEP 2.1.2.**

STEP 2.1.2.1: PDP will launch the external obligations **STEP 2.1.2.1.1.**

STEP 2.2: PEP will redirect the decision to the App.

3.3.4.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/authorization_enabler.html

Table 38. Implementation status of the Authorization enabler

Category	Status
Components implementation	All the components of the enabler have a first functional version
Feature implementation status	Authorization Server first version development completed. Pending developments: • Definition of related enabler integration details, to complete next version.
Encapsulation readiness	Components have been containerised as Docker images and deployed using Docker Compose. Kubernetes deployment is ongoing.
Deployed with the Orchestrator in a laboratory environment	Not yet

3.4. DLT-based enablers

3.4.1. Logging and Auditing enabler

3.4.1.1. Structure and functionalities

Structure and functionalities have not changed with respect to deliverable D5.2 content.

Implementation technologies

Table 39. Implementation technologies for the Logging and Auditing enabler

Technology	Justification	Component(s)
Hyperledger Fabric Chaincode (Smart Contracts)	The Hyperledger is a fitting choice for building a private network to support the creation of a consortium blockchain. The technology provides permissions to handle the network along with a good scalability. Hyperledger Fabric can have its value augmented by deploying smart contracts to automate functions.	Logging and Auditing business logic
Hyperledger Fabric peers, orderers		Hyperledger Fabric peers and orderers
Hyperledger Fabric Certificate Authority (CA)		Certification Authorities (CAs)
REST (Enabler's API)	A popular web microframework written in Python, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	DLT API

3.4.1.2. Communication interfaces

Table 40. Communication interfaces (API) of the Logging and Auditing enabler

Method	Endpoint	Description
POST	**/log**	Create logs
GET	**/gets**	Get list of logs
GET	**/getbyid/{id}**	Get log by specific id

3.4.1.3. Use cases

Use cases have not changed with respect to deliverable D5.2 content.

3.4.1.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/logging_and_auditing_enabler.html

Table 41. Implementation status of the Logging and Auditing enabler

Category	Status
Components implementation	A first version of all the components are in place
Feature implementation status	The basic features of these components are working in the minimum viable product principle. It remains to: • Integrate with other enablers. • Proceed with encapsulation.
Encapsulation readiness	Components are yet to be encapsulated
Deployed with the Orchestrator in a laboratory environment	Not yet

3.4.2. Data Integrity Verification enabler

3.4.2.1. Structure and functionalities

Structure and functionalities have not changed with respect to deliverable D5.2 content.

Implementation technologies

Table 42. Implementation technologies for the Data Integrity Verification enabler

Technology	Justification	Component(s)
Hyperledger Fabric Chaincode (Smart Contracts)	The Hyperledger is a fitting choice for building a private network to support the creation of a consortium blockchain. The technology provides permissions to handle the network along with a good scalability. Hyperledger Fabric can have its value augmented by deploying smart contracts to automate functions.	Logging and Auditing business logic
Hyperledger Fabric peers, orderers		Hyperledger Fabric peers and orderers
Hypeledger Fabric Certificate Authority (CA)		Certification Authorities (CAs)
REST (Enabler's API)	A popular web microframework written in Python, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	DLT API

3.4.2.2. Communication interfaces

Table 43. Communication interfaces (API) of the Data Integrity Verification enabler

Method	Endpoint	Description
POST	**/updatedata**	Update data
GET	**/getdata**	Download data

3.4.2.3. Use cases

Use cases have not changed have not changed with respect to deliverable D5.2 content.

3.4.2.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/data_integrity_verification_enabler.html

Table 44. Implementation status of the Data Integrity Verification enabler

Category	Status
Components implementation	A first version of all the components are in place
Feature implementation status	The basic features of these components are working in the minimum viable product principle. It remains to: • Integrate with other enablers. • Proceed with encapsulation.
Encapsulation readiness	Components are yet to be encapsulated
Deployed with the Orchestrator in a laboratory environment	Not yet

3.4.3. Distributed Broker enabler

3.4.3.1. Structure and functionalities

Structure and functionalities have not changed with respect to deliverable D5.2 content.

Implementation technologies

Table 45. Implementation technologies for the Distributed Broker enabler

Technology	Justification	Component(s)
Hyperledger Fabric Chaincode (Smart Contracts)	The Hyperledger is a fitting choice for building a private network to support the creation of a consortium blockchain. The technology provides permissions to handle the network along with a good scalability. Hyperledger Fabric can have its value augmented by deploying smart contracts to automate functions.	Logging and Auditing business logic
Hyperledger Fabric peers, orderers		Hyperledger Fabric peers and orderers
Hypeledger Fabric Certificate Authority (CA)		Certification Authorities (CAs)
REST (Enabler's API)	A popular web microframework written in Python, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	DLT API

3.4.3.2. Communication interfaces

Table 46. Communication interfaces (API) of the Distributed Broker enabler

Method	Endpoint	Description
POST	**/insert**	Post data source metadata to the Data Consumer, given the source id
GET	**/get**	Get data source metadata from Data Provider (source)

3.4.3.3. Use cases

Use cases have not changed with respect to deliverable D5.2 content.

3.4.3.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/distributed_broker_enabler.html

Table 47. Implementation status of the Distributed Broker enabler

Category	Status
Components implementation	A first version of all the components are in place
Feature implementation status	The basic features of these components are working in the minimum viable product principle. It remains to: • Integrate with other enablers.

Category	Status
	• Proceed with encapsulation.
Encapsulation readiness	Components are yet to be encapsulated
Deployed with the Orchestrator in a laboratory environment	Not yet

3.4.4. DLT-based FL enabler

3.4.4.1. Structure and functionalities

Structure and functionalities have not changed with respect to deliverable D5.2 content.

Implementation technologies

Table 48. Implementation technologies for the DLT-based FL enabler

Technology	Justification	Component(s)
Hyperledger Fabric Chaincode (Smart Contracts)	The Hyperledger is a fitting choice for building a private network to support the creation of a consortium blockchain. The technology provides permissions to handle the network along with a good scalability. Hyperledger Fabric can have its value augmented by deploying smart contracts to automate functions.	Logging and Auditing business logic
Hyperledger Fabric peers, orderers		Hyperledger Fabric peers and orderers
Hypeledger Fabric Certificate Authority (CA)		Certification Authorities (CAs)
REST (Enabler's API)	A popular web microframework written in Python, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	DLT API

3.4.4.2. Communication interfaces

Table 49. Communication interfaces (API) of the DLT-based FL enabler

Method	Endpoint	Description
POST	**/post**	Post the aggregated model (or the global model)
GET	**/get**	Get the updated models from FL Local operations (FL Privacy should take place prior to the data transmission)

3.4.4.3. Use cases

Use cases have not changed with respect to deliverable D5.2 content.

3.4.4.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/dlt_based_fl_enabler.html

Table 50. Implementation status of the DLT-based FL enabler

Category	Status
Components implementation	A first version of all the components are in place
Feature implementation status	The basic features of these components are working in the minimum viable product principle. It remains to: • Integrate with other enablers. • Proceed with encapsulation.
Encapsulation readiness	Components are yet to be encapsulated
Deployed with the Orchestrator in a laboratory environment	Not yet

3.5. Manageability

3.5.1. Enabler for Registration and Status of enablers

3.5.1.1. Structure and functionalities

Integrated in the tactile dashboard, this enabler will serve as a registry of enablers and, in case they are deployed, a means of retrieving their status. In particular, it will: (i) allow the registration of an enabler (this is, from an ASSIST-IoT repository). Essential enablers will be pre-registered; (ii) retrieve a list of currently running enablers; (iii) depict the status and the specific logs of an enabler (the latter only if the enabler with log collection capabilities is in place); and (iv) facilitate the deployment of standalone enablers (mostly for those that have to be present at any deployment).

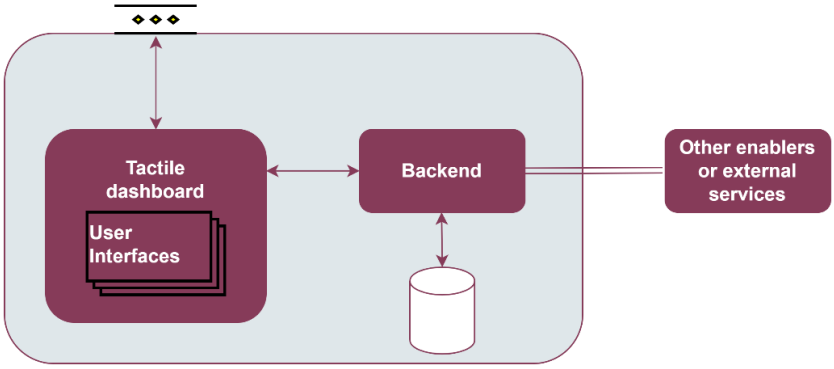

Figure 33. Enabler for registration and status of enablers structure

Implementation technologies

Table 51. Implementation technologies for the Registration and Status of enablers

Technology	Justification	Component(s)
PUI9	Framework developed and provided by Prodevelop to build dashboards and user interfaces. The dashboard and the dashboard backend are developed using this framework.	Dashboard and dashboard backend
Vue.js	A JavaScript framework used by the PUI9 client to create user interfaces.	Dashboard
Java	The object-oriented programming language used to create the PUI9 backend.	Dashboard backend
Spring framework	One of the most popular Java frameworks to develop microservices, completely oriented to build microservices ready to be deployed at the cloud. The PUI9 backend is built using the Spring framework.	Dashboard backend
PostgreSQL	Database needed to interact with the PUI9 backend and to persist the dashboard features.	Database
Docker images	All components are built as custom Docker images.	All components

3.5.1.2. Communication interfaces

Table 52. Communication interfaces (API) of the Registration and Status of enablers

Method	Endpoint	Description
GET	**/dashboard/enablers**	Enablers view of the dashboard

3.5.1.3. Use cases

There are five use cases that apply this enabler. The **first use case** is to **show the list of the deployed enablers in a table.** The diagram and related steps are the following:

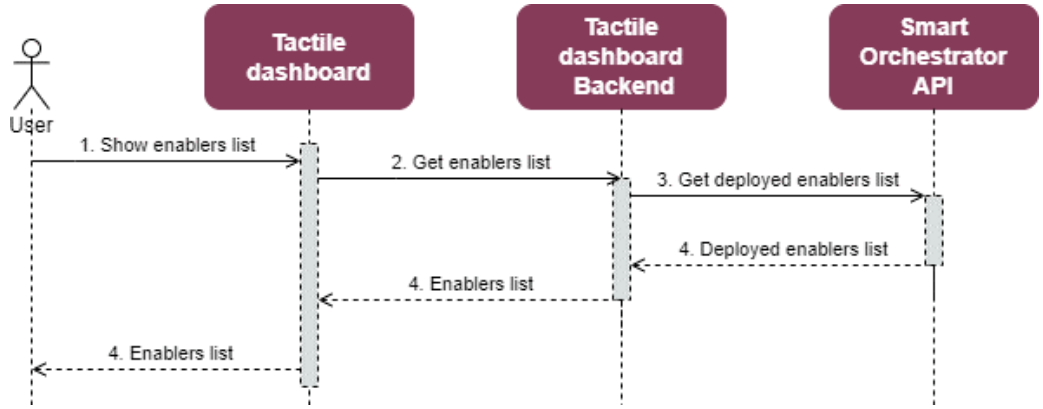

Figure 34. Enabler for Registration and Status of enablers UC1 (show deployed enablers)

STEP 1: The user interacts with the tactile dashboard and selects the enablers view on the menu.

STEP 2: The dashboard sends an HTTP GET request to its backend to obtain the list of deployed enablers.

STEP 3: The backend sends an HTTP GET request to the Smart Orchestrator API to obtain a list with the deployed enablers, which returns the demanded list.

STEP 4: The backend provides to the dashboard the list with the deployed enablers, then shown on it.

The **second use case** is to **deploy a new enabler**. The diagram and involved steps are the following:

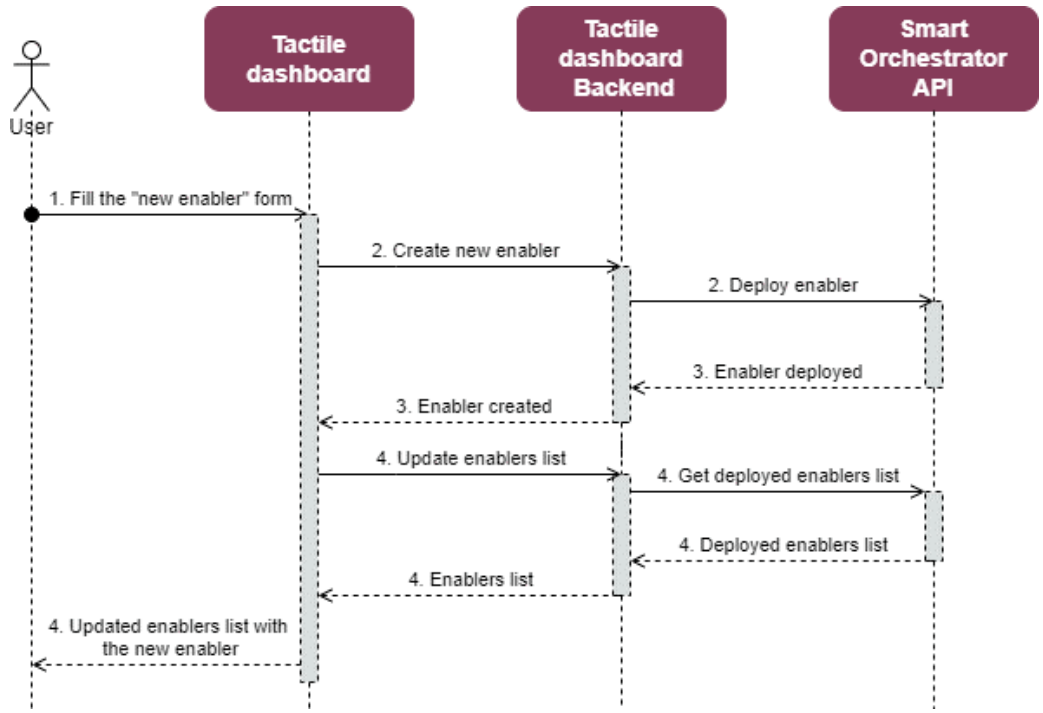

Figure 35. Enabler for Registration and Status of enablers UC2 (deploy an enabler)

STEP 1: The user interacts with the tactile dashboard, clicks on the "Add new enabler" button and fills in the "new enabler" form.

STEP 2: The dashboard sends an HTTP POST request to its backend to deploy the new enabler, which forwards the request to the Smart Orchestrator API.

STEP 3: The Smart Orchestrator API returns the result of the operation.

STEP 4: If the enabler has been deployed successfully, the dashboard shows to the user the updated list of enablers. To that end, the first use case is instantiated. It should include the recently-added enabler.

The **third use case** is to **terminate a deployed enabler**, the step before deleting an enabler. The diagram and related steps are the following:

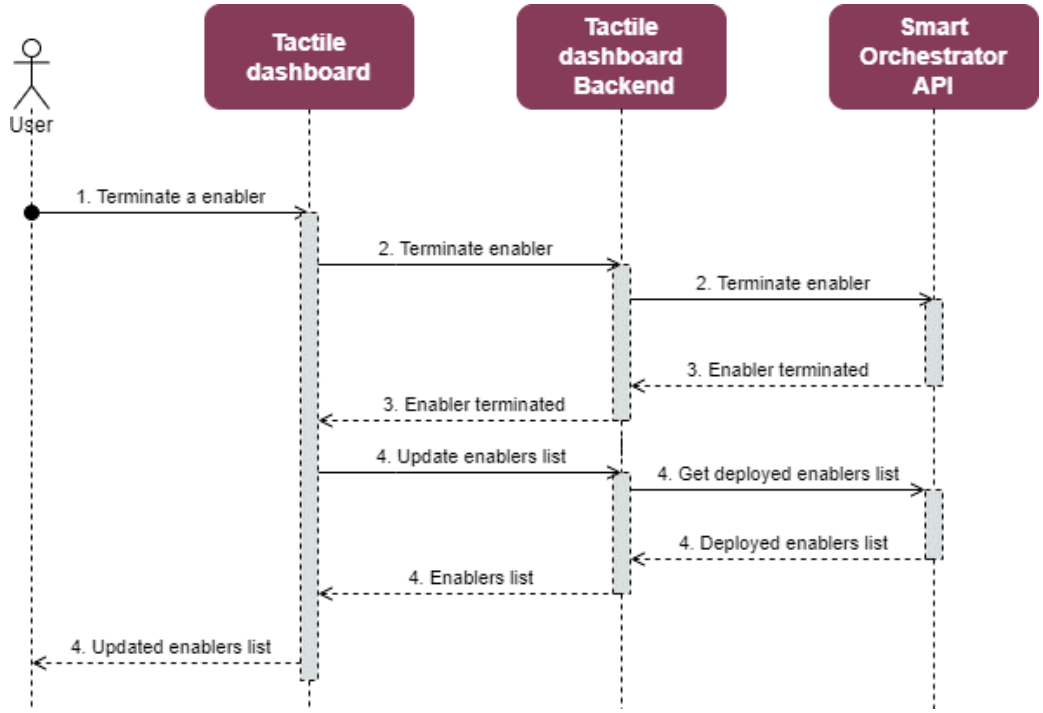

Figure 36. Enabler for Registration and Status of enablers UC3 (terminate an enabler)

STEP 1: The user interacts with the tactile dashboard and clicks on the "Terminate enabler" button.

STEP 2: The dashboard sends an HTTP PUT request to its backend to terminate the selected enabler, which forwards the request to the Smart Orchestrator API (in this case, via POST).

STEP 3: The Smart Orchestrator API returns the result of the operation.

STEP 4: If the enabler has been terminated successfully, the dashboard shows to the user the updated list of enablers. To that end, the first use case is instantiated. It should mark the recently-terminated enabler as inactive.

The **fourth use case** is to **delete a terminated enabler**. The diagram and involved steps are the following:

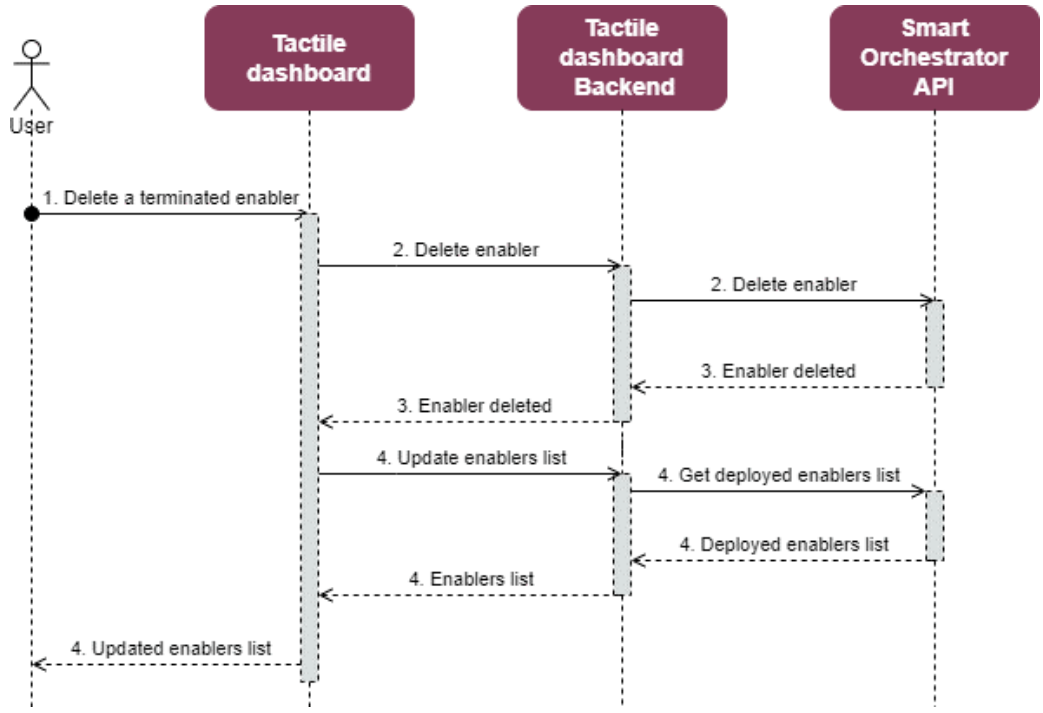

Figure 37. Enabler for Registration and Status of enablers UC4 (delete an enabler)

STEP 1: The user interacts with the tactile dashboard and clicks on the "Delete enabler" button of a terminated enabler.

STEP 2: The dashboard sends an HTTP DELETE request to its backend to delete the selected enabler, which forwards the request to the Smart Orchestrator API.

STEP 3: The backend returns the result of the operation.

STEP 4: If the enabler has been deleted successfully, the dashboard shows to the user the updated list of enablers. To that end, the first use case is instantiated. It should not plot the recently-terminated enabler.

The **last (fifth) use case** is to **show the logs of a deployed enabler**. Its diagram and related steps are the following:

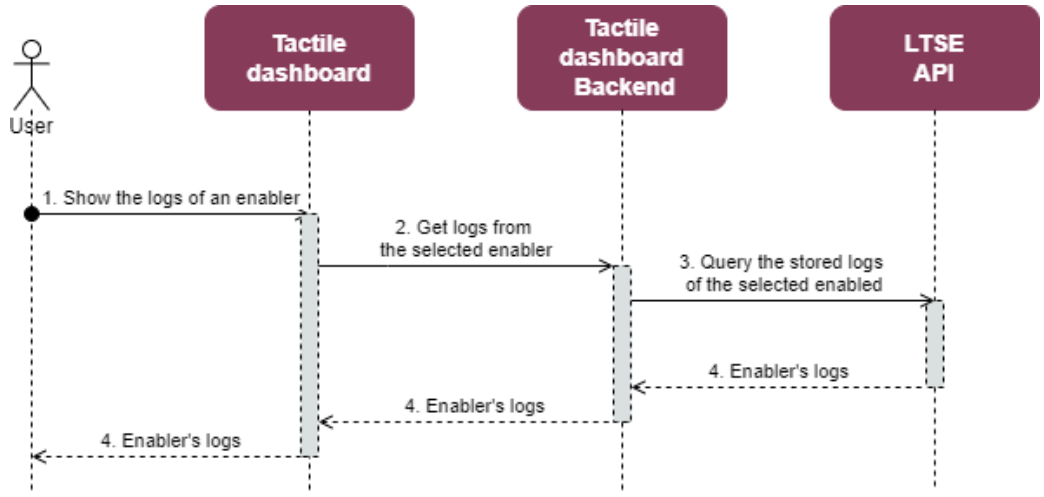

Figure 38. Enabler for Registration and Status of enablers UC5 (show enabler logs)

STEP 1: The user interacts with the tactile dashboard and clicks on the "Show enabler logs" button.

STEP 2: The dashboard sends an HTTP GET request to its backend to delete the selected enabler.

STEP 3: The backend performs a search query making an HTTP GET request to the LTSE API.

STEP 4: The LTSE API returns the result of the query to the backed, which passes it to the dashboard so the user can visualise the list of logs of the selected enabler.

3.5.1.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/registration_and_status_enabler.html

Table 53. Implementation status of the Registration and Status of enablers

Category	Status
Components implementation	All the components of the enabler have a first functional version
Feature implementation status	The basic features are in place, however, some developments are pending: • To configure the default parameters of an enabler, a form must be shown based on a dedicated template per enabler, instead of introducing a raw JSON object. The challenge is to customize this form for each enabler, including all the customizable parameters in a user-friendly way. • The use case related to logs has not been implemented yet.
Encapsulation readiness	Components have been containerised as Docker images and deployed using Docker Compose. The Helm chart is under development.
Deployed with the Orchestrator in a laboratory environment	Not yet

3.5.2. Enabler for Management of Services and Enablers' Workflow

3.5.2.1. Structure and functionalities

Integrated in the tactile dashboard, this enabler will present a graphical environment where ASSIST-IoT administrators can instantiate the enablers required to work, and also to connect them to compose a chain, or service workflow, making use of a Directed Acyclic Graph (DAG) interface. Having information about the physical topology and available k8s nodes/clusters, it will allow the user to decide whether to select the proper node or cluster for deploying an enabler, or let the system decide based on pre-defined architectural rules.

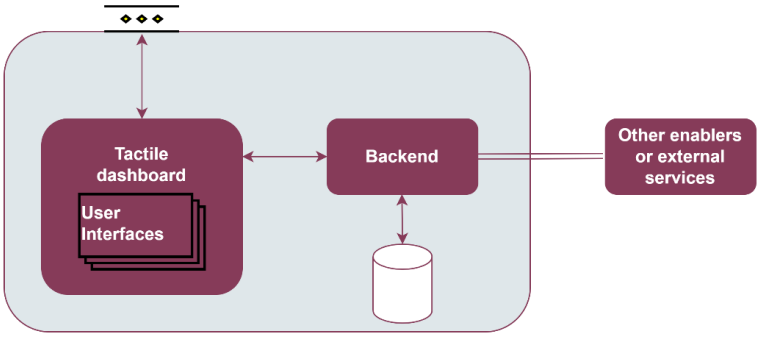

Figure 39. Enabler for the Management of Services and Enablers' Workflow structure

NOTE: The development is at very preliminary stage. For the sake of avoiding technological choices that might be modified, no DAG-related solution is indicated at this moment.

Implementation technologies

Table 54. Implementation technologies for the Management of the Services and Enablers' Workflow structure

Technology	Justification	Component(s)
PUI9	Framework developed and provided by Prodevelop to build dashboards and user interfaces. The dashboard and the dashboard backend are developed using this framework.	Dashboard and dashboard backend
Vue.js	A JavaScript framework used by the PUI9 client to create user interfaces.	Dashboard
Java	The object-oriented programming language used to create the PUI9 backend.	Dashboard backend
Spring framework	One of the most popular Java frameworks to develop microservices, completely oriented to build microservices ready to be deployed at the cloud. The PUI9 backend is built using the Spring framework.	Dashboard backend
PostgreSQL	Database needed to interact with the PUI9 backend and to persist the dashboard features.	Database
Docker images	All components are built as custom Docker images.	All components

3.5.2.2. Communication interfaces

Table 55. Communication interfaces (API) of the Services and Enablers' Workflow structure

Method	Endpoint	Description
GET	**/dashboard/workflow**	Services and enablers' workflow view of the dashboard

3.5.2.3. Use cases

The component is in an early development stage, as it greatly depends on its interaction with other enablers (and hence, need to have their APIs and environment variables in place). At the moment, it is not possible to describe concise use cases, therefore for the sake of avoiding adding content that might be likely modified, use cases are not indicated yet.

3.5.2.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/management_of_services_and_enablers.html

Table 56. Implementation status of the Services and Enablers' Workflow structure

Category	Status
Components implementation	The design is in place, as is identical to the first manageability enabler. However, the dedicated interface and the backend functions related to it are yet to be implemented.
Feature implementation status	Not implemented yet.
Encapsulation readiness	As components have not been yet particularised to perform the operations related to this enabler, Docker images have not been produced.
Deployed with the Orchestrator in a laboratory environment	Not yet

3.5.3. Devices Management enabler

3.5.3.1. Structure and functionalities

Integrated in the tactile dashboard, the main functionalities of this enabler will be to register: (i) a smart IoT device in a deployment, and (ii) a cluster in an ASSIST-IoT deployment, including in the latter case all the necessary messages to notify it to the smart orchestrator. It will also execute all the required actions related to networking for enabling connectivity among isolated/independent clusters, including those that have been added via VPN/SD-WAN technology. Besides, it will allow monitoring any registered node and device in the deployment, including its status (i.e., availability and used resources) and current instantiated enablers' components.

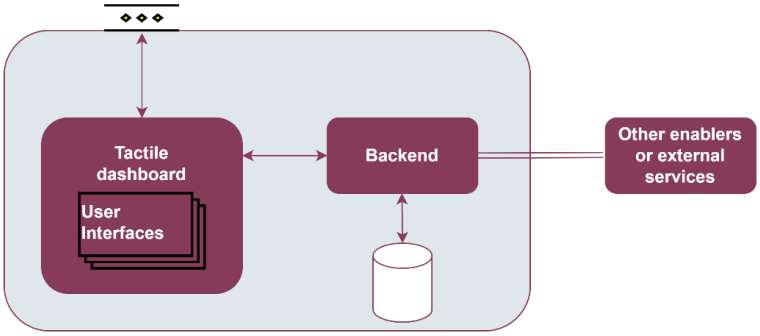

Figure 40. Devices Management enabler structure

Implementation technologies

Table 57. Implementation technologies for the Devices Management enabler

Technology	Justification	Component(s)
PUI9	Framework developed and provided by Prodevelop to build dashboards and user interfaces. The dashboard and the dashboard backend are developed using this framework.	Dashboard and dashboard backend
Vue.js	A JavaScript framework used by the PUI9 client to create user interfaces.	Dashboard
Java	The object-oriented programming language used to create the PUI9 backend.	Dashboard backend
Spring framework	One of the most popular Java frameworks to develop microservices, completely oriented to build microservices ready to be deployed at the cloud. The PUI9 backend is built using the Spring framework.	Dashboard backend
PostgreSQL	Database needed to interact with the PUI9 backend and to persist the dashboard features.	Database
Docker images	All components are built as custom docker images.	All components

3.5.3.2. Communication interfaces

Table 58. Communication interfaces (API) of the Devices Management enabler

Method	Endpoint	Description
GET	**/dashboard/devices**	Services view of the dashboard

3.5.3.3. Use cases

There are three use cases that apply this enabler. The **first use case** is to **show a list of the registered K8s clusters in a table**. The diagram and related steps are the following:

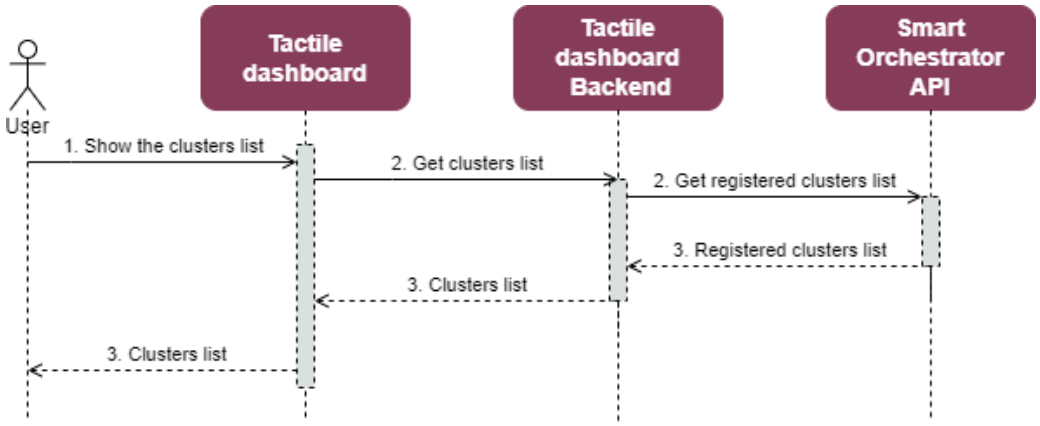

Figure 41. Devices Management enabler UC1 (show registered clusters)

STEP 1: The user interacts with the tactile dashboard and selects the K8s clusters view on the menu.

STEP 2: The dashboard sends an HTTP GET request to its backend to obtain the list of registered clusters, which forwards it to the Smart Orchestrator API to obtain a list with the registered clusters.

STEP 3: The Smart Orchestrator API returns the list with the registered clusters, and the backend sends this information to the dashboard to print a list with the registered clusters.

The **second use case** is to **register a new K8s cluster**. The diagram and the involved steps are the following:

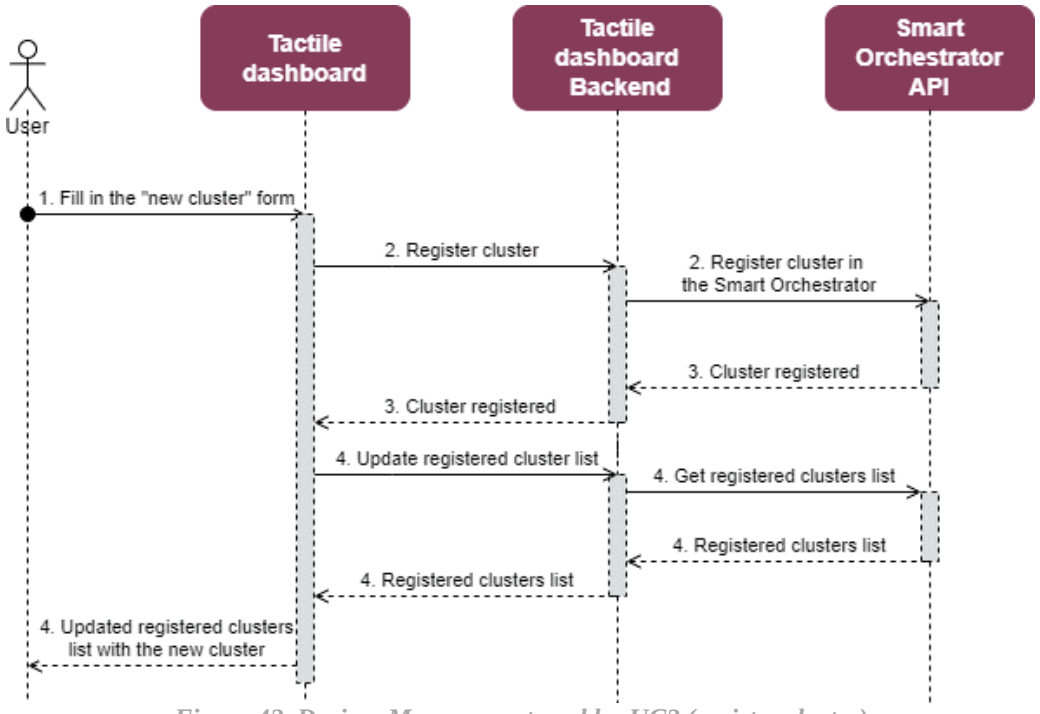

Figure 42. Devices Management enabler UC2 (register cluster)

STEP 1: The user interacts with the tactile dashboard, clicks on the "Add new cluster" button and fills in the "new cluster" form.

STEP 2: The dashboard sends an HTTP POST request to its backend to register the new cluster, which forwards it to the Smart Orchestrator.

STEP 3: The Smart Orchestrator API returns the result of the operation.

STEP 4: If the cluster has been registered successfully, the dashboard shows to the user the updated list of registered clusters. To that end, the first use case is instantiated.

The **last (third) use case** is to **delete a registered K8s cluster**. The diagram and related steps are the following:

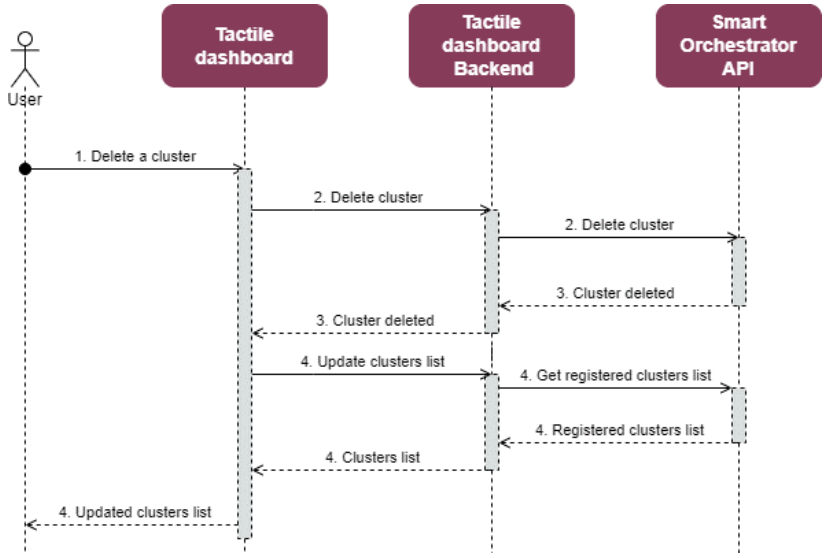

Figure 43. Devices Management enabler UC3 (delete cluster)

STEP 1: The user interacts with the tactile dashboard and clicks on the "Delete cluster" button.

STEP 2: The dashboard sends an HTTP DELETE request to its backend to delete the selected cluster, which forwards it to the Smart Orchestrator.

STEP 3: The Smart Orchestrator API returns the result of the operation.

STEP 4: If the cluster has been deleted successfully, the dashboard shows to the user the updated list of registered clusters. To that end, the first use case is instantiated.

NOTE: The use cases related to Smart IoT devices management are yet to be designed as they have not been implemented yet.

3.5.3.4. Implementation status

Link to Readthedocs (structure defined with WP6 documentation task): https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/devices_management_enabler.html

Table 59. Implementation status of the Devices Management enabler

Category	Status
Components implementation	All the components of the enabler have a first functional version
Feature implementation status	The basic features to manage k8s clusters are in place. However, some developments are pending: • Manage clusters that make use of SD-WAN or VPN connections. • Manage Smart IoT devices.
Encapsulation readiness	Components have been containerised as Docker images and deployed using Docker Compose. The Helm chart is under development.
Deployed with the Orchestrator in a laboratory environment	Not yet

4. Future Work

This document provides an update and extension of the specifications provided in the first iteration of this deliverable series. Apart from providing insight about technical information for the transversal enablers, it also encompasses software outcomes as of M18. These software outcomes are at different levels of development: some of them are containerised, others integrated with K8s (manifests ready) or prepared for packaging (in Helm charts), whereas the implementation of a few of them are still in an immature stage. It should be highlighted that the development of the enablers identified as essential has been prioritised for this release, and hence a functional version is already available.

The enablers developed so far allows for continuing efforts related to this and other work packages:

- To finish the development of the components of the enablers (WP5).
- To containerise, and/or generate the K8s manifests required to deploy them in those cases that have not virtualized the overall solution (WP5).
- To perform the testing and integration methodologies for each enabler (WP6).
- To perform the necessary modifications, in order to ensure the proper interactions between enablers from WP4 & WP5.
- To package, publish and release the enablers as Helm charts (WP6).
- To start implementing them in pilots for further validation and assessment (WP7), either fully or partially packaged.

In the next (and last) iteration of the deliverable, all the enablers will have a functional packaged version available. They will be accompanied by another document, in which all the modifications and deviations will be reported, as well as an update with the final enabler templates.