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

D5.2 Transversal Enablers Development – Preliminary Version

Deliverable No.	D5.2	**Due Date**	31-OCT-2021
Type	Other	**Dissemination Level**	Public
Version	1.0	**WP**	WP5
Description	Technical specification of vertical enablers identified and 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
Ignacio Lacalle	P01 UPV	iglaub@upv.es
Alejandro Fornés	P01 UPV	alforlea@upv.es
Paco Mahedero	P01 UPV	framabio@upv.es
Katarzyna Wasielewska-Michniewska	P03 IBSPAN	katarzyna.wasielewska@ibspan.waw.pl
Piotr Lewandowski	P03 IBSPAN	piotr.lewandowski@ibspan.waw.pl
Wiesław Pawłowski	P03 IBSPAN	wieslaw.pawlowski@ibspan.waw.pl
Maria Ganzha	P03 IBSPAN	maria.ganzha@ibspan.waw.pl
Marcin Paprzycki	P03 IBSPAN	marcin.paprzycki@ibspan.waw.pl
Eduardo Garro	P03 PRO	egarro@prodevelop.es
Miguel Llacer	P03 PRO	mllacer@prodevelop.es
Sergio Vivó	P03 PRO	svivo@prodevelop.es
Georgios Stavropoulos	P04 CERTH	stavrop@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 S21SEC	olopez@s21sec.com
Jordi Blasi	P13 S21 SEC	jblasi@s21sec.com

History

Date	Version	Change
1-Oct-2021	0.1	Table of content
22-Oct-2021	0.8	Version ready for internal review
29-10-2021	1.0	Final version

Key Data

Keywords	Enablers, verticals, self-*, interoperability, manageability, scalability, federated learning, DLT
Lead Editor	Katarzyna Wasielewska-Michniewska (P03 IBSPAN)
Internal Reviewer(s)	P09 NEWAYS, P06 INFOLYSIS

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 identified for 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 advancement.
This document is included in deliverable D5.2 that is of type Other-software (besides document deliverable includes the content of code repositories). It provides technical information necessary for the implementation of WP5 enablers and serving as a guiding information to understand ongoing development. The concepts presented here advance work summarized in D5.1 Initial Transversal Enablers Specification (submitted in M9). By M12 of the project (October 2021), a total of 17 enablers have been identified and formalised:

- From Self-*: Self-healing device enabler, Resource provisioning enabler, Monitoring and notifying enabler, geo (Localization) 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.

Technical information included in this document: updated structure diagrams, outline of enabler functionalities, communication interfaces specification, use cases for inter-component communication. For each enabler progress information is included to indicate advancement of work since D5.2 is a preliminary version of the development.

Table of contents

List of tables

List of figures

List of acronyms

Acronym	Explanation
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
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 main goal of this deliverable is **to provide the status of development works** that are going under the scope of WP5.

It should be highlighted that this deliverable corresponds to the first out of three, and therefore its content 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.
Work plan	D5.2 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 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 D5.2 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 DevSecOp methodology
Milestones	This deliverable contributes to the realisation of *MS3 – Enablers defined*, that will be achieved in 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). 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 one appendix. It starts with an introduction that outlines the context of the document. Next section includes specifications of enablers divided into tasks that they belong to. For each

enabler the description includes: structure and functionalities, communication interfaces, selected technologies, use cases realized by a given enabler and work progress at M12 of the project (note that WP5 started in M4). These information advance and/or compliment enablers definitions from the deliverable D5.1. Finally, the last section of this document concludes with a summary of the future work carried out in the work package that will be included in the second version of the deliverable. The appendix contains template tables and diagrams for identified enablers

1.3. Outcomes of the deliverable

The main outcome of this deliverable is a preliminary development of WP5 enablers. Depending on the enabler, the advancement of work differs and is summarized in the work progress subsections. Additionally, in the deliverable additional technical information was given that compliments specifications in D5.1 and provides context to better understand the ongoing development. Included specifications may be modified and/or extended in the following versions of this deliverable due to the fact that the work in WP3, WP4 and WP7 is in progress.

1.4. Deviation and corrective actions

In Federated Learning, FL Privacy enabler's functionality has been moved to Privacy component in FL Local Operations enabler. The two initial enablers would have had a strong dependency on each other and FL Privacy provided functionality that did not require a separate encapsulated enabler. As a result, it was decided that a better architectural decision would be to replace this enabler with a component of the latter.

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 focuses on providing the status of development and advancing technical specifications of the enablers. 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.

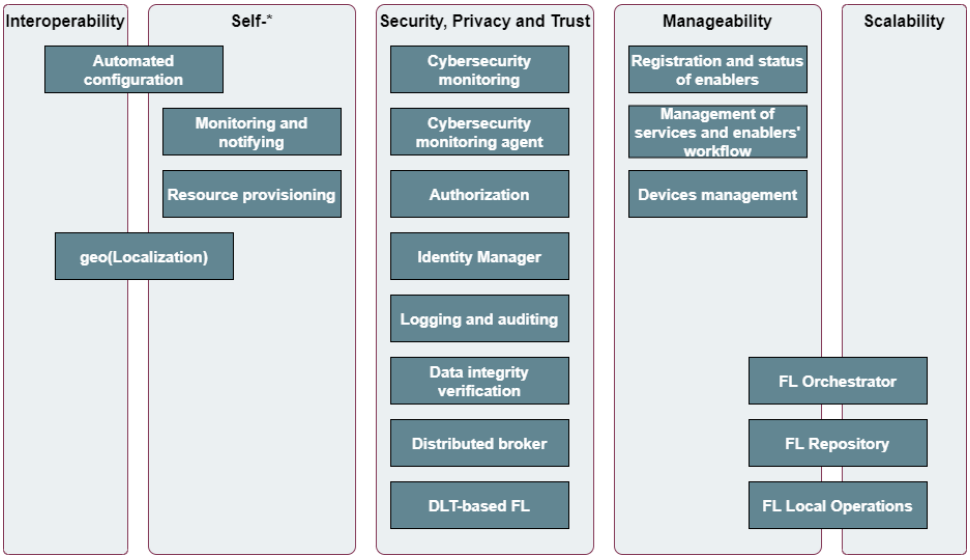

Figure 1. WP5 enablers distribution among verticals.

3. Vertical enablers definitions

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.

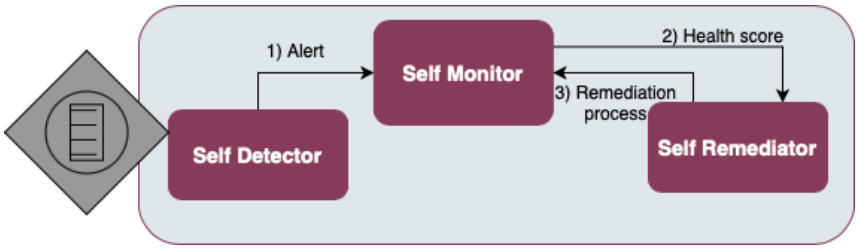

Figure 2. Self-healing device enabler structure.

As described in D5.1, 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 self-remediator is triggered to select another remediation process from the list.

3.1.1.2. Communication interfaces

The enabler is being developed by means of NODE-Red libraries. Therefore, there are no writable communications within their internal components. Regarding external APIs for the enabler, the following two has been already developed, although new ones are subject to be added in the following releases of the enabler.

External APIs for the enabler

Method	Endpoint	Description
POST	/cpuusage?threshold=XX	Let's user change the maximum threshold of CPU usage
POST	/ramusage?threshold=XX	Let's user change the maximum threshold of RAM usage

3.1.1.3. Technologies

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)	All of them
Unix commands	Used to access device hardware & software	All of them

Javascript	Main language for developing custom functions over all components of the enabler. Selected for its familiarity	All of them

Gaps

Since the self-healing device needs to have access to the host OS resources (CPU, RAM, network, etc), there is a limitation about its encapsulation. Due to its characteristics, if the enabler that is based on Node-Red directives would be running within a Docker container, it would not be possible to access to carry out manage UNIX sentences properly (stop, restart, kill…).

3.1.1.4. Use cases

There are two use cases detected to take into account. The first one is related to **change the CPU usage threshold** via API.

STEP 1: The user interacts via API with the enabler to change the CPU usage threshold.

STEP 2: When finished, the API must response with some result.

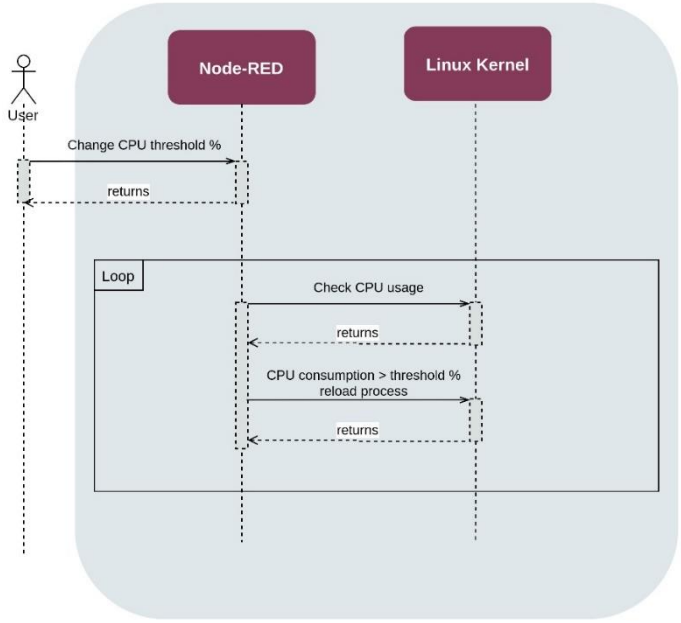

Figure 3. Self-healing CPU usage monitoring and threshold update UC.

The second use case is related to **change the RAM usage threshold via API**.

STEP 1: The user interacts via API with the enabler to change the RAM usage threshold.

STEP 2: When finished, the API must response with some result.

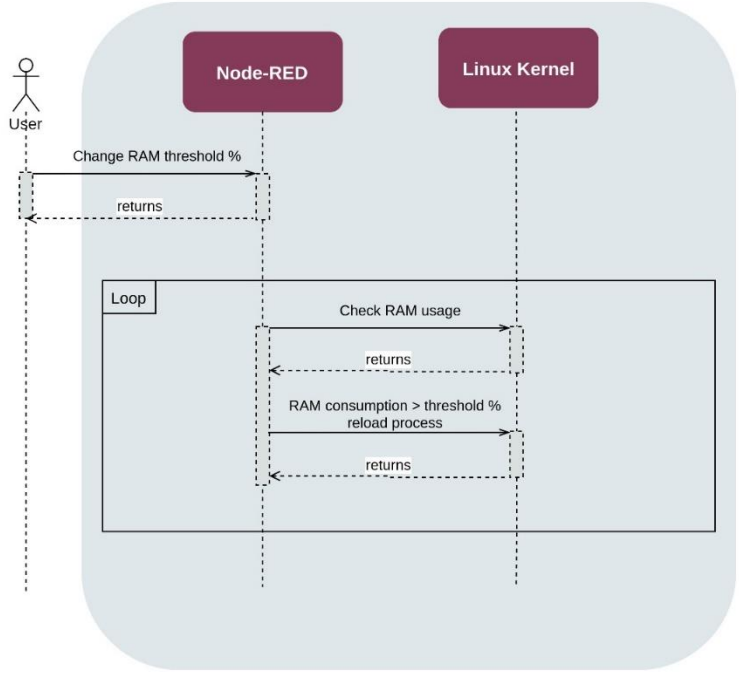

Figure 4. Self-healing RAM usage monitoring and threshold update UC.

3.1.1.5. Work progress

For the time being, CPU and RAM usage are the only two metrics that are monitored with the enabler. In addition, only the Kill PID remediation action is supported. In next releases, additional metrics (e.g., network status) will be included, and other remediation rules such isolate the device, shut down network ports, or reboot will be added. Furthermore, it is expected that by encapsulating the self-healing device enabler within a K8s cluster, self-healing functionalities will be possible in a containerized environment.

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, the auto-scaling thresholds cannot be set as trivially. This enabler aims at modifying the scaling response of nodes and clusters into a more dynamic fashion, by:

- Ensuring high QoS and availability of key, selected enablers, considering current state of the system.
- Monitoring historic trends of these enablers, to preventively act upon its scaling requirements (thresholds of resources/usage to instantiate replicas).
- Applying ML techniques and intelligent services to modify the predetermined scaling-related values.

The updated diagram of the component can be seen in Figure 5. It is composed by 5 main components and 3 supporting databases.

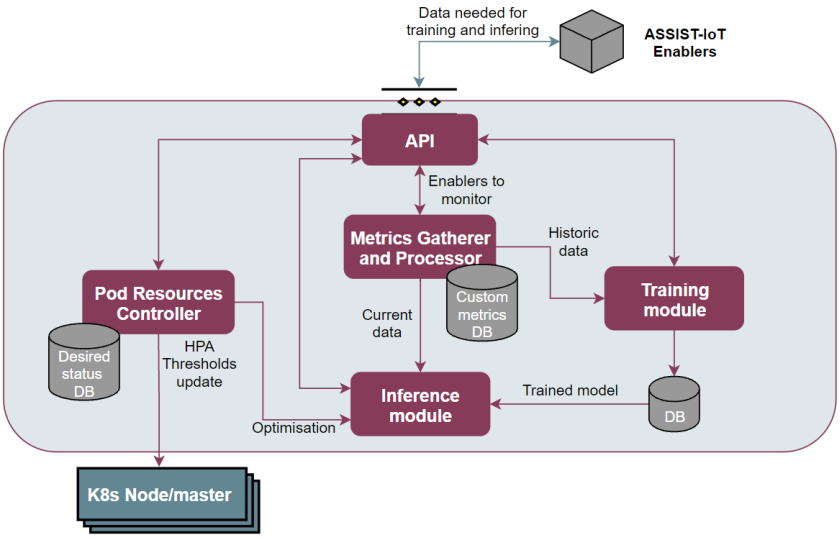

Figure 5. Resource provisioning enabler structure.

3.1.2.2. Communication interfaces

Internal APIs for communication between enablers

Communication between API component and the rest is not included (essentially, the API forwards the calls specified in the second table).

Component	Method	Endpoint	Description
Training module	POST	/train	Starts a process to train/update the model, aiming at scheduling potential up-/down-scaling of its managed components
Inferring module	POST	/inference	Starts an inference process to evaluate the necessity of preventively modify the values of horizontal autoscaling
Pod resources controller	POST	/initiate	Starts a process of inference and optimization. This should only occur when new enablers are added to be monitored
Pod resources controller	POST	/optimize	Receives the results of an inference process, communicates to the k8s nodes/masters and modifies the desired state database
Metrics gatherer and processor	GET	/metrics/historic	Returns historic, aggregated/processed metrics of the system and selected enablers
Metrics gatherer and processor	GET	/metrics/current	Returns current, aggregated/processed metrics of the system and selected enablers

External APIs for the enabler

Method	Endpoint	Description
POST	/enablers	Updates the list of enablers to monitor (i.e., receives a JSON object, with data related to the addition of a new enabler to the list or modifying the whole current list of monitored enablers). Returns the list of currently
GET	/enablers	Returns a JSON object with the list of enablers that must ensure high QoS
POST	/enablers/interval	To specify interval between scaling evaluation processes (i.e., inference and appliance of results)
POST	/metrics	Obtains metrics from the enablers devoted to metrics gathering for the specified enablers, and send them to the metrics gatherer and processor
POST	/metrics/interval	To specify interval between metrics gathering processes
POST	/metrics/span	To specify the span of days of information to store in the custom metric database. Older information is deleted
POST	/training/interval	To specify interval between metrics gathering processes

GET	/desired-state	Returns information related to a cluster, including available and used resources of its managed nodes, and deployed components of ASSIST-IoT enablers

3.1.2.3. Technologies

Technology	Justification	Component(s)
kubectl	Alongside with ServiceAccount, RoleBinding/ClusterRoleBinding objects, allows pods to make calls to the k8s API. Alternative: Daemon object	Pod Resources Controller
MongoDB	Since there is no need of strict relational data, implementation of both custom metrics database and desired database will be performed and MongoDB collections	Custom metrics database & desired state database
Python	Main language for developing custom functions over all components of the enabler. Selected for its familiarity.	Metrics gatherer and processor, custom functions
scikit-learn/ Tensorflow	Main libraries for inference construction. One will be selected once the model is chosen (e.g., Tensorflow is better for Deep Learning models)	Inference & training module
Flask	The APIs will be developed considering Flask as the main technology for its ease of construction. For the main API, Node.js will be explored	All APIs

Gaps

The limits of the horizontal autoscaller (HPA) of k8s are static, per object (i.e., deployments, StatefulSets, etc.), and defined in a specific manifest file. The aim of the project is to make it more dynamic, managed by an external enabler that communicates with the corresponding nodes (via kubectl).

Risks

The technologies selected are mature and dedicated to fulfilling the components' functionalities, so they should not pose any risk by themselves. Still, testing its functionality will rely on the existence of other enablers to get realistic data, and the presence of a monitoring stack to retrieve data from them and from the system, as without them the enabler cannot be fully developed.

3.1.2.4. Use cases

There are 4 main use cases that apply in this enabler. The first one is related to **an administrator user that specifies those enablers, or specific components of them, that must guarantee high levels of QoS or high availability**. The diagram is the following:

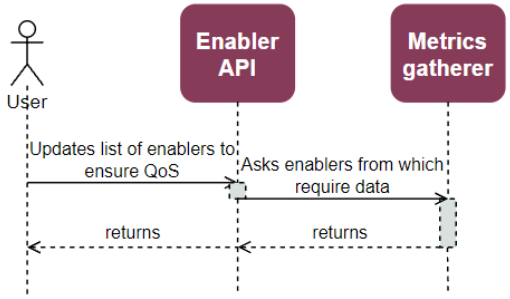

Figure 6. Resource provisioning enabler UC 1.

STEP 1: The user interacts via the enablers' API, which in turn communicates to the Metrics Gatherer from which enablers it has to obtain historic data.

STEP 2: When finished, the API must response with some response code and the list of currently-managed enablers.

Other use cases, in which the previous use case is repeated in a similar fashion, are the following ones: (i) when a user specifies the frequency for gathering current usage/resources data from the monitored enablers; (ii) when

a user indicates the time between training processes (e.g., once per day, once per week, etc.). In this case, the API interacts internally with the training module; and (iii) when the user indicates the granularity in which changes over the scaling should be applied (e.g., each 10 minutes, each hour, etc.), in this case, internally communicating with the Pod resources controller.

The second use case is related to the **gathering of metrics**. Since a component must access to data from other enablers, it requires that the API components perform this action on his behalf, as it is shown in the diagram:

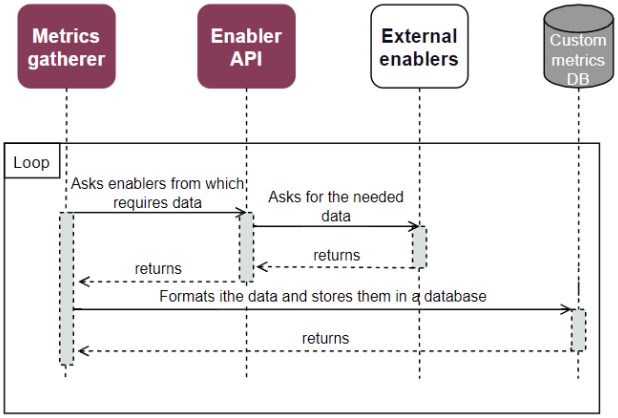

Figure 7. Resource provisioning enabler UC 2.

STEP 1: The Metrics Gatherer starts the process according to the frequency specified by an administrator user **or** the first time that a new enabler has been added to be monitored. It communicates with the API to ask data from other enablers.

STEP 2: The API obtains the data and passes them back to the Metrics Gatherer.

STEP 3: The Metrics Gatherer processes the data from the different sources and stores them in a valid format for the latter training phase.

The third use case responds to **the necessity of training**. Its diagram is the following:

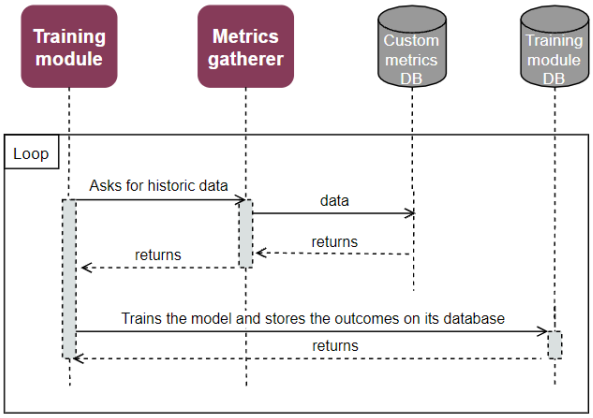

Figure 8. Resource provisioning enabler UC 3.

STEP 1: The Training Module starts a process according to the frequency specified by an administrator user **or** the first time that a new enabler has been added to be monitored. It communicates with the Metrics Gatherer to retrieve the historic data of the monitored enablers.

STEP 2: The Metrics Gatherer sends the data back to the training module.

STEP 3: Then, the training module starts the training with the data received from the Metrics gatherer, and once it is finished, stores the outcomes in its own internal database.

The last use case is related to **the actual up-/down-scaling of the components of the enablers**. This is controlled by the pod resources controller, which, in the end, will apply the actions over the system. The diagram and involved steps are depicted below:

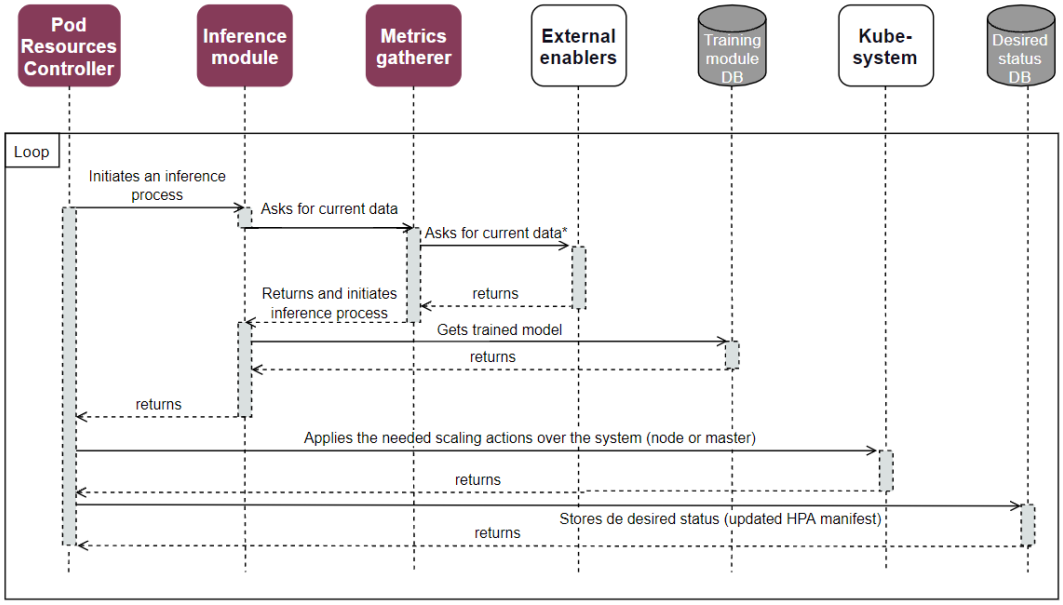

Figure 9. Resource provisioning enabler UC 4.

STEP 1: The Pod Resources Controller starts a process according to the frequency specified by an administrator user **or** the first time that a new enabler has been added to be monitored. It communicates with the Inference Module, which will obtain the optimal values for the modifying the scaling of the components of the enablers or not.

STEP 2: The Inference Module retrieves the data info from the Metrics Gatherer, which obtain current data regarding system status and resources from other enablers* (the Metrics Gatherer retrieves these data from them similarly to the second use case, via the API, although not indicated in the figure).

STEP 3: The Inference Module retrieves the trained model and performs the inference with current data.

STEP 4: The Pod Resources Controller obtains the results and applies the necessary actions over the involved k8s horizontal autoscalerers, if needed. Current status is stored in the desired status database.

3.1.2.5. Work progress

Work performed so far includes a Proof of Concept (PoC) of the enabler, and an initial version of some of the needed components. The resource provisioning PoC is composed of three components:

- **Python server**: Developed with flask, it exposes the required custom.metrics.k8s.io API paths. Here is where the metric calculations are made to up-scale and down-scale a set of k8s pods, and returned in the desired response format. Essentially, it serves as (i) a very initial Pod resources controller, and (ii) a server for providing a set of (dummy) metrics to the HPA (k8s' Horizontal Pod Autoscaler).
- **APIService**: This service registers the custom.metric.k8s.io API, so k8s HPA can access it, which allows it to extend his functionality. The metrics registered are dummy, and provided by the Python server.

The HPA will now use the API exposed from the server and check if the custom metric reaches the target value to scale the pods of the deployment chosen for the testing purposes (the deployment chosen in the POC is a php-apache server). Currently, the enabler has a hardcoded range time where the target metrics reach the target value, that is how the pods autoscale. Passed this time the pods return to their initial state. Current status is depicted in the following figure:

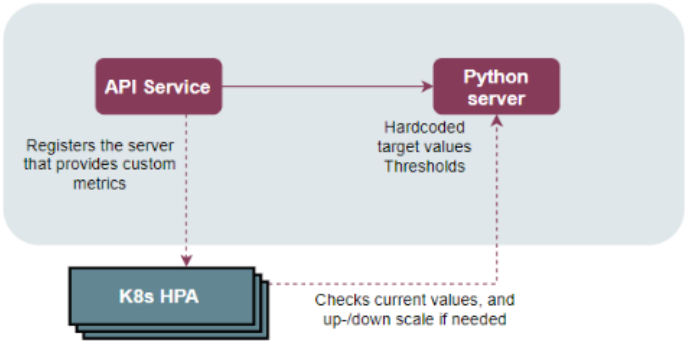

Figure 10. Diagram of the PoC of the Resource provisioning enabler.

Before deploying the PoC, once the git repository has been cloned in a computer with a k8s distribution (e.g., k3s, microk8s, k8s, etc.), the docker image of the server has to be built. To that end, the following command has to be run inside the folder /Server:

```
$ docker image build -t server .
```

To deploy the k8s objects related to the enabler and the testing service (i.e., the php-apache server), the following command has to be executed from the root folder, where the manifests files (i.e., .yaml) files are located:

```
$ kubectl apply -f .
```

Then, the user can observe that replicas are increased in the specific time range and downscaled outside of that range. This PoC shows how to set and apply custom metrics to control the behaviour of the replicas of a deployment. The general idea of this enabler will be to also modify the directives of the HPA for specific enablers. This is done by modifying thresholds of the metrics monitored to control when to increase or decrease current number of replicas, based on ML techniques.

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.

- **Device Monitoring**: Another functionality of the enabler is the device monitoring. The enabler ensures that the IoT device reads the required data in fixed time intervals, in order to control data flooding or data interruption. If not, a notification will be created.
- **Edge Monitoring**: Furthermore, the enabler is to guarantee the edge monitoring. In more details, the enabler ensure communication with connected IoT devices. If communication between the linked components is lost, a notification will be created. Additionally, it will check for attacks (i.e. sybil attack).

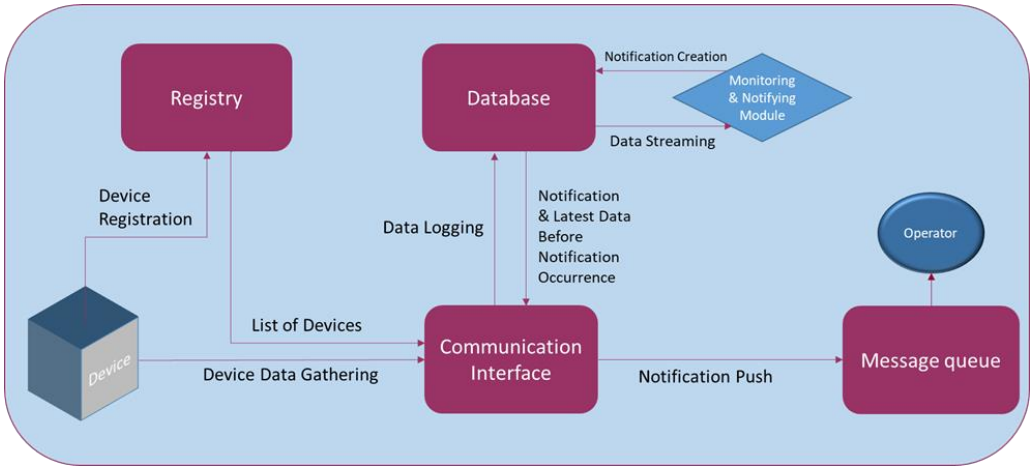

Figure 11. Monitoring & Notifying enabler structure.

3.1.3.2. Communication interfaces

The standard API between the enabler's components is already being developed. Any additional API with external components will be submitted in the upcoming releases of the 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. Technologies

The candidate technologies for the enabler's completion are the following:

Technology	Justification	Component(s)
MySQL	It is the standard and most popular relational database management system based on SQL.	Database
Kafka	Kafka provides a unified, high-throughput, low-latency platform for handling real-time data pipelines.	Database, Message queue, Registry
Scala	Scala is one of the best high-level programming languages which provides static types to avoid bugs in complex applications.	Registry
Java	Low complexity programming language serving the same purpose as Scala, to use as an alternative.	Registry
Python	Main language for developing custom functions over all components of the enabler. Selected for its familiarity.	Database, Message queue, Registry

Gaps

It is still unclear whether a data gathering and processing unit is necessary for the imported data.

3.1.3.4. Use cases

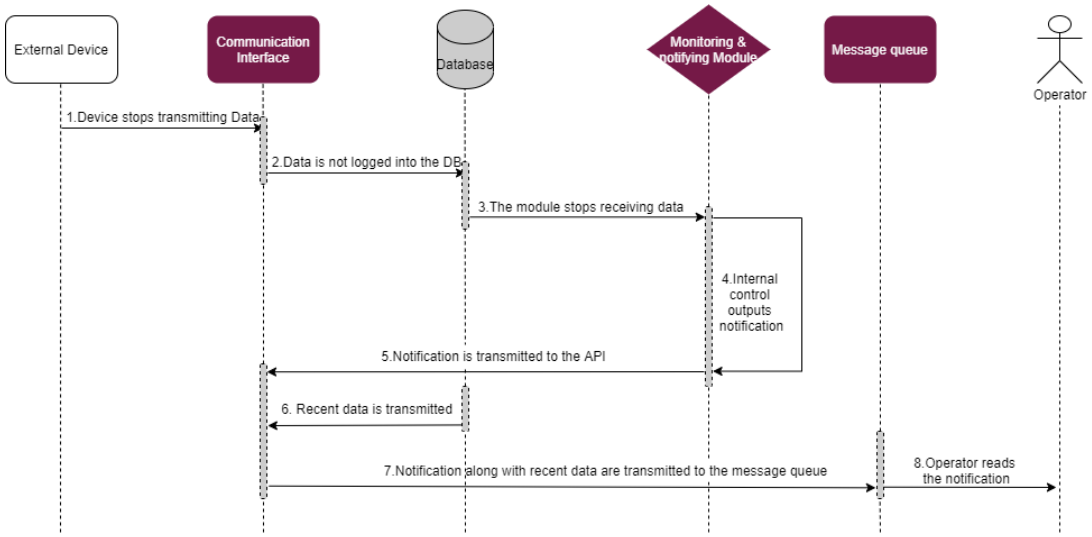

Figure 12. Monitoring & Notifying enabler UC.

The first use case involves an IoT device which stops receiving data from its integrated sensor.

STEP 1: The communcation interface stops receiving data from the IoT device.

STEP 2: Data stops being logged into the database.

STEP 3: The monitoring module stops receiving data.

STEP 4: Since the monitoring module stops receiving data, it is clrear that the sensor is malfunctioning and creates a notification.

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

STEP 6: The latest data before the notification occurence is also transmitted to the API to help the operator diagnose the problem.

STEP 7: The notification along with the recent data are transmitted to the message queue.

STEP 8: Operator receives the notification and the information (data) before its occurence and has to act accordingly.

3.1.3.5. Work progress

The software development of the enabler is currently under way and an initial version of the components has already been set, including the database and the API. A preliminary version will be presented in less than a month.

3.1.4. Geo (Localization) enabler

3.1.4.1. Structure and functionalities

This enabler is responsible for determining worker tag positions, mapping the tag positions, and issuing warnings to the tags.

- **Positioning & Alert:** The purpose of the localisation positioning and alert is to; 1) collect information about (geo)location coordinates of the IoT device; 2) to give an alert to both the smart IoT device and the OSH (Occupational Safety & Health) manager in case the smart IoT device is located in an unauthorized zone or danger zone, or if an unsafe situation is created due to an incident.
- **Localisation Monitor:** The purpose of the localisation monitor is to collect the location coordinates and alarms of all connected smart IoT devices, to determine if the IoT device is within an authorized area and if there is a possible unsafe situation. The MAP component serves as 3) input to gain insight

into the latest map details and location status. If an IoT device is in an unauthorized zone or danger zone, or if an unsafe situation is created due to an incident, an alert is issued to both the smart IoT device and the OSH manager, and an incident log is generated saved, and the map updated.

- **Map:** The purpose of MAP is to provide the latest building, floors and environment information, including geofencing, border enforcement, danger zones and escape and security routes. The incident log on the map is updated for the OSH management system.

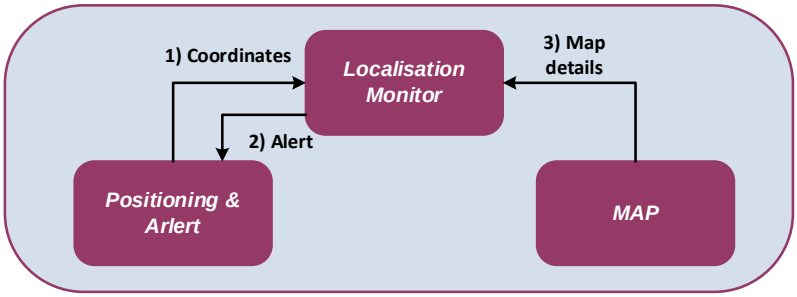

Figure 13. Positioning and alert.

3.1.4.2. Technologies

There is a lot to consider when designing (indoor) location systems, such as choosing the location technology that is best and meets the needs of each application. The available technologies differ in range, accuracy, reliability, secure ranging, localization service latency, scalability, infrastructure costs and other aspects. Based on the requirements of the pilot use cases for all these criteria, Ultra Wideband (UWB) localization was chosen. The main rationale: accuracy <10cm, immunity to multi-path and interference, Range/ coverage typically 50 to 70 meters, very secure, location service latency less than 1 millisecond, scalability more than thousands of tags and a relative low infrastructure cost.

Key components of this Real Time Localisation System (RTLS) are Tags, Anchors, a Gateway / Server, RTLS software and BIM maps. The system consists of:

- Powerful multiple UWB anchors with both a wired Power over Ethernet (PoE) and wireless Wi-Fi connectivity.
- Tags, rechargeable battery powered.
- Gateway/Edge node (central processing engine) calculated based on self-learning algorithms the real-time location of the tags.
- RTLS Manager for easy set-up and visualisation.
- OSH management system with BIM maps.

Technology	Justification	Component(s)
UWB	Ultra Wideband (UWB) localization was chosen. The main rationale: accuracy <10cm, immunity to multi-path and interference, Range/ coverage typically 50 to 70 meters, very secure, location service latency less than 1 millisecond, scalability more than thousands of tags and a relative low infrastructure cost.	UWB Chipsets and FW
Ethernet	If the infrastructure allows the communication between the UWB Anchors and the Edge node / Gateway shall make use of a wired Ethernet or Power over Ethernet (PoE), and between the Edge node/gateway and OSH management system shall be Ethernet.	Hardware and FW
WiFi	The communication between the UWB Anchors and the Edge node/Gateway shall make use of a wireless Wi-Fi connectivity is tis is required	WiFi Chipset and FW
4/5G	The communication between the Edge node/Gateway and shall make use of a wireless 4/5G connectivity is this is required to communicate with the OSYH Management system	4/5G chipset and FW

Yocto	Yocto[1] operating system (OS) an open-source collaboration project based on Linux that helps developers creating custom Linux-based systems regardless of the hardware architecture.	Operating System
HAL	Hardware Abstraction Layer (HAL), consists of device driver needed as interface between the electronics and the OS	FW
Embedded	Configuration and initialisation of the standard interfaces (Ethernet, Serial, etc.), SSH and a default user will be preconfigured on the Edge node, making the node fully functional and ready to run enablers on.	FW

Risks

- It is currently unclear who will take responsibility for the data processor and necessary software to map the position of each mobile tag device on a BIM map layer(s) as part of the OSH management system.
- We miss the availability of application software from an RTLS Manager for easy installation and visualization of the localisation infrastructure.
- The use of Wi-Fi connected anchors will reduce the infrastructure cost against a wired infrastructure. The drawback of a wireless clock synchronisation is that it impacts the tag device position accuracy. A literature survey showed that the clock synchronisation for UWB most suitable localisation positioning requires that all anchors should be synchronised at a nano second level accuracy. In case of a Wireless Clock synchronisation for UWB positioning this could be hard to reach. Based on Clock skew and drift compensation and by averaging over multiple anchor measurements, the position error good be reduced to 51 cm. This could mean that a mixture of wired and wireless anchors will be used.

3.1.4.3. Use cases

The figure below shows a communication flow diagram of **the mobile tag localisation data flow and messages** in the smart safety of workers use case.

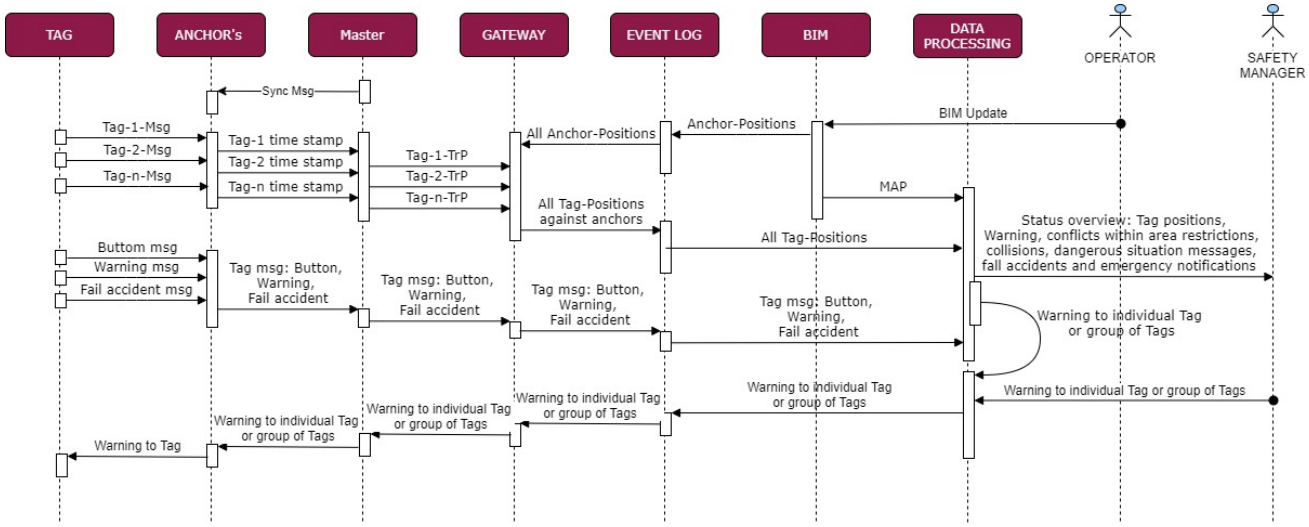

Figure 14. Location and event structure UC.

STEP 1: Tag devices send (broadcast) periodically messages toward the anchors.

STEP 2: Multiple anchors (reference points) are deployed at fixed and known locations and are accurately time synchronized. When an anchor receives the periodically send messages of a tag device, the anchor will time-stamped this tag beacon message signal related to the common synchronized time base, record the Tag receive time and send it to the master anchor. The timestamps from multiple anchors are then forwarded to a central master anchor engine. The gateway fulfill the master functionality in this case.

[1] https://www.yoctoproject.org/

STEP 3: The purpose of the master is to synchronize all anchors with accurate time, this can be done over a wired or wireless link and send the record mobile Tag device arrival time to the Gateway localisation engine.

STEP 4: The gateway, a location engine, performs multi-lateration algorithms based on the time difference of the tag devices' beacon signal coming from each anchor. The result will be a 2D or 3D location of mobile tag device. The gateway and master are in many cases combined into one device.

STEP 5: The location of each of the mobile tag devices is real-time logged by the event logger inside the OSH manager.

STEP 6: The BIM represents the construction map, for example of a construction site or container area and shows the position of the anchors and gateway(s) on it.

STEP 7: The Operator updates the BIM database, including an accurate (geo)localisation position of each of the anchors and gateway(s).

STEP 8: The data processor of the OSH Manager shall calculate position and map each of the mobile tag devices on a BIM map layer(s). It shall also monitor and warning in case of conflicts within area restrictions, collisions, dangerous situation messages, fall accidents and emergency notifications. In case OSH data processor detects a conflict within area restrictions, collision or a hazardous situation a messages or emergency notifications will be send to individual mobile tag device or to a group of mobile tag devices.

STEP 9: The Safety Manager received status overview about mobile tag device positions, Warning, conflicts within an area restriction, collisions, dangerous situation messages, emergency notifications and accidents.

STEP 10: The Safety Manager is capable to send Warning to an individual mobile tag device or group of mobile tag device.

3.1.4.4. Work progress

The work includes a Proof of Concept (PoC) of the enabler and the development of a HW/FW version of some of the required components. The resource provisioning PoC consists of the following components:

- Edge Node/Gateway (electronics and enclosure)
- Anchors (electronics and enclosure)
- Worker location tracking tag (electronics and enclosure)
- Worker Smart fall arrest tag (electronics and enclosure)
- Embedded FW for Edger node/gateway, Anchors, Worker tag, and Fall Arrest Tag
- Hardware Abstraction Layer for Edger node/gateway, Anchors, Worker tag, and Fall Arrest Tag
- Pre-Install software on gateway/ edge node: *Python, Software update support, Docker*

3.1.5. Automated configuration enabler

3.1.5.1. Structure and functionalities

This enabler aims at keeping heterogeneous devices and services synchronised with their configurations. User can update configuration and define its alternative versions in case of errors. Self-* component will detect if a fallback configuration should be used and will apply it in reaction to changes in the environment as necessary/required.
For each of the devices and services under its control, the enabler requires a flexible representation of the available configurations. Additionally, to adequately react to errors and other events/conditions, it needs an intelligent mechanism for changing/applying the “fallback” configurations. To achieve its goals the enabler will utilise the following components:

- **Database**: A place to store details about configuration, configuration rules and information about whether all connected devices received configuration (updates).
- **Configuration applier**: Component responsible for checking and applying configuration updates via Device/Service Connector as well as reacting to failure conditions

- **Message Queue**: Infrastructure for transporting messages internally between components and connectors
- **Registry**: Component responsible for registering devices/connector types and providing list of devices for defined connector types.
- **Intelligence**: Module responsible for deciding which configuration and configuration parameters should be applied. This could be realised in one of two ways:
  - User chooses/defines the desired state of the system and intelligence component performs the required changes
  - User defines rules and in case of specific events configuration gets updated.

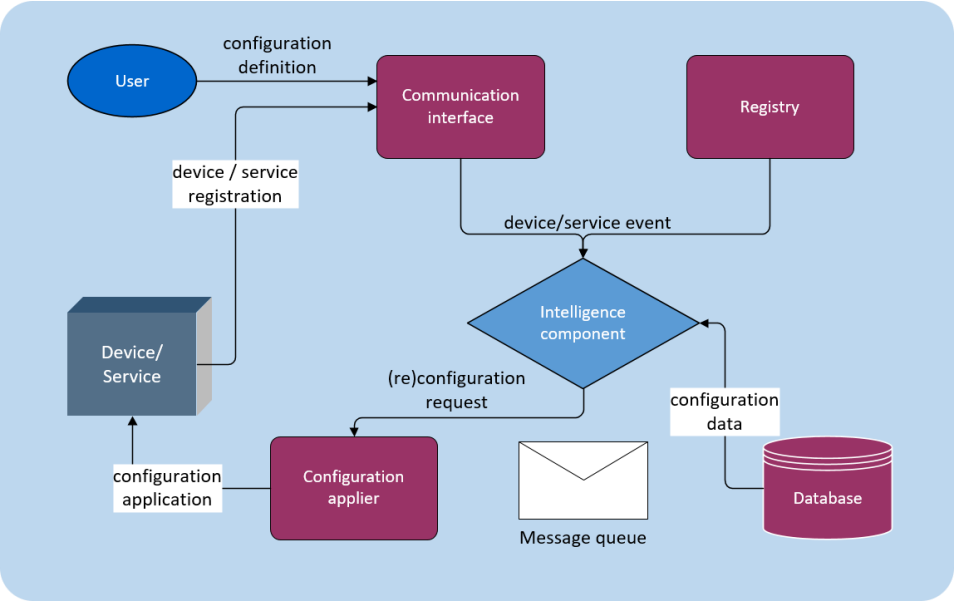

Figure 15. Automated configuration enabler structure.

3.1.5.2. Communication interfaces

Method	Endpoint	Description
POST	/config/{config_id}/schema	Define configuration schema
PUT	/config/{config_id}	Update configuration
DELETE	/config/{config_id}	Remove configuration
POST	/thing/{device_type}/{id}	Register device (or service)
POST	/thing/{device_type}/{device_id}	Define connector for device of a given type
POST	/thing/provider/{provider_id}	Register service for retrieving list of devices
GET	/thing/{device_type}/{device_id}	Get device details
GET	/config/{config_id}	Get config details
GET	/thing/{device_type}/	Get config details

3.1.5.3. Technologies

Technology	Justification	Component(s)
Scala[2]	Scala is a modern, mature, statically-typed programming language, providing support for both functional and imperative, object-oriented style. Those features, together with the library-level compatibility with Java, and familiarity with the language within	Configuration applier, Registry, Intelligence

[2] https://www.scala-lang.org/

	the SRIPAS group are arguments for choosing Scala as the programming environment for implementing the enabler.	
Akka[3]	Akka is a highly-regarded Scala framework supporting the Actor concurrency model. This library is a de facto standard for creating concurrent and/or distributed systems in Scala. Additionally, Akka provides connectors for REST[4], MQTT[5], Kafka[6], gRPC[7], and other contemporary technologies, which makes it a natural fit for heterogenous and distributed environment of IoT.	Configuration applier, Registry, Intelligence
Kafka[8]	Kafka is an open-source, distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. Kafka's high reliability seems like a good fit for internal component communication. Its large number of available connectors will also help with various analytical needs we might have.	All of them
MQTT[9]	MQTT is an OASIS[10] standard messaging protocol for the IoT. It is designed to provide an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. Today, MQTT is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc. MQTT provides IoT specific features like Last Will and Testament[11]. PAHO[12] provides a broad range of MQTT clients.	Configuration applier

Gaps

- Configuration: Overall trend is currently to use declarative configuration (sometimes exposing an API to execute imperative configuration). How this mechanism should be executed in heterogeneous, smart solution is not clear yet.
- Self-*: The concept of the enabler and its high-level design seem clear. The details of the abstract model underpinning the design are still under consideration. Additionally, what remains to be investigated and decided are the lower-level implementation details and methods.
- Scala 3: Not all of the required libraries/tools directly support Scala 3 yet. This problem can be temporarily mitigated but eventually it is expected that Scala 3 will be the language of choice for the implementation of the enabler and its components.

3.1.5.4. Use cases

Although there are multiple use cases, right now we are focusing our work on the most basic one – updating configuration. In following months additional use cases will be described, but first the configuration representation problem has to be solved – at least partially (see Work Progress below).

[3] https://akka.io/
[4] https://doc.akka.io/docs/akka-http/current/introduction.html
[5] https://doc.akka.io/docs/alpakka/current/mqtt.html
[6] https://doc.akka.io/docs/alpakka-kafka/current/home.html
[7] https://doc.akka.io/docs/akka-grpc/current/index.html
[8] https://kafka.apache.org/
[9] https://mqtt.org/
[10] https://en.wikipedia.org/wiki/OASIS_(organization)
[11] https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/
[12] https://www.eclipse.org/paho/

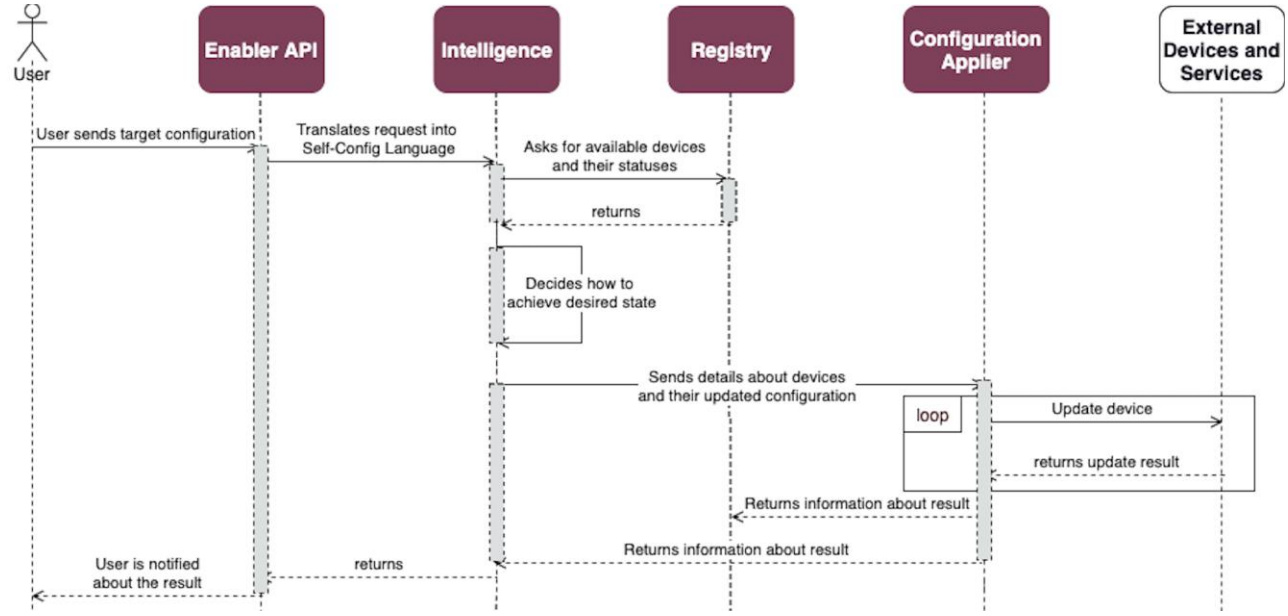

Figure 16. Automated configuration enabler UC.

STEP 1: User sends request containing *target configuration* that they would like the system to achieve. This might be done, for example, by defining *post condition* that system needs to adhere to.

STEP 2: Intelligence component communicates with Registry component to check the current status of the devices.

STEP 3: Intelligence component checks how to achieve *target configuration* using available devices. This requires understanding *actions* a particular device can perform and what are the results of those actions. Based on that, Intelligence component will be able to create series of device updates. The hard part is to understand how to model that capability in a flexible and user-friendly way.

STEP 4: Intelligence component sends request to Configuration Applier.

STEP 5: Configuration Applier updates all devices.

STEP 6: Configuration Applier sends results to both Registry and Intelligence component.

STEP 7: Intelligence component returns result status to User via Enabler API.

3.1.5.5. Work progress

In the project code repository, there is a scaffolding of the multi-module sbt project. The project uses Scala 3, but it cross compiles to Scala 2.

Currently, most work is done in configuration representation. This is a non-trivial and highly impactful problem, but after it will be solved, the tempo of actual software development will increase substantially. Non-triviality of the representation problem comes from the fact that the configuration has to reflect the dynamic nature of IoT system and includes “levers” that will allow to control or endow Self-* aspects. The high impact follows from the fact that the *automated configuration* enabler will be a part of every non-trivial ASSIST-IoT deployment.

3.2. Federated machine learning enablers

3.2.1. FL Orchestrator

3.2.1.1. Structure and functionalities

The FL orchestrator is responsible of specifying details of FL workflow(s)/pipeline(s). This includes FL job scheduling, managing the FL life cycle, selecting and delivering initial version(s) of the shared algorithm, as

well as modules used in various stages of the process, such as training stopping criteria-. Finally, it can specify ways of handling different "error conditions" that may occur during the FL process.

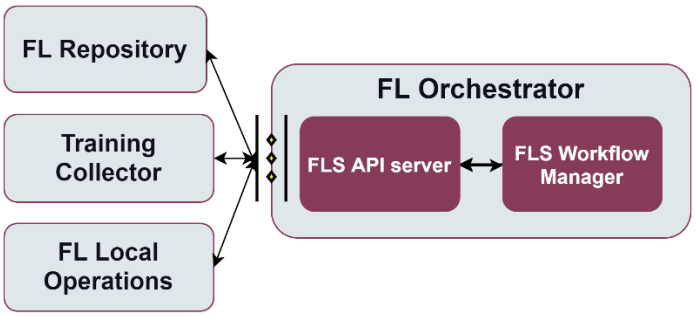

Figure 17. FL Orchestrator enabler structure.

There is a client and a server master files, which include the defined configurable parameters. When a request is made, with the value of the configurable parameters, the FL orchestrator returns the files already prepared for deployment and use. One the one hand, master files will be stored in master folder and modified files in mod folder. Subsequently, a compressed zip file is generated, which returns the compressed files to the receiver. Currently, the sample from the Flower documentation is included (https://flower.dev/docs/quickstart_tensorflow.html).

3.2.1.2. Communication interfaces

External APIs for the enabler

Method	Endpoint	Description
GET	/params	JSON received is retrived from request that will allow to modify the master configuration files
POST	/params	JSON configuration file is sent to the FL Local Operations

3.2.1.3. Technologies

The FL orchestrator has been conceived as an API in Flask, which from an input request with a predefined set of requirements, it is capable of generating the necessary files to set up the federated training process. The framework used is Flower, which allows the use of TensorFlow, PyTorch and MXNet models.

The use cases with the client / server part must be defined before introducing them into the procedure, as well as all their possible configurations.

The FL orchestrator is formed by:

- A *master* subdirectory, that includes templates with client / server training files
- A *mod* subdirectory, with Client / server training files already modified
- fl_main.py: Flask API in charge of carrying out all the described procedure
- request.py: Request example
- params.json: Json sample with the request parameters

3.2.1.4. Use cases

First use case show flow of action when **a user configures the training process and requests the configuration files to the FL Orchestrator**.

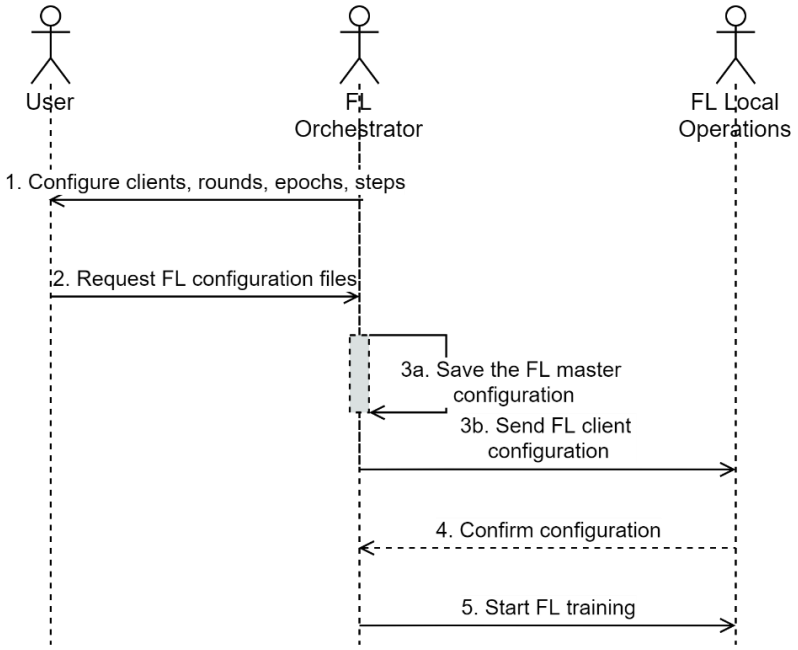

Figure 18. FL Orchestrator configures and requests parties to start FL training.

STEP 1: User configures the number of clients needed to start the FL training, as well as the number of rounds, the epochs and the steps by epoch FL.

STEP 2: User requests the FL Orchestrator the generation of the FL master and FL client configuration files.

STEP 3: FL Orchestrator generates both files

STEP 3a: FL Orchestrator saves within its environment the FL master configuration file

STEP 3b: FL Orchestrator sends to the involved parties the FL client configuration files.

STEP 4: FL Local Operations acknowledges its reception.

STEP 5: FL Orchestrator mandates connected parties to start the FL training.

3.2.1.5. Work progress

For the time being, these are the configurable parameters foreseen in the FL orchestrator: Number of clients needed to start the training, Number of training rounds, Epochs, and Steps by epoch. Additional parameters such as Stopping FL Training criteria, Privacy mechanism will be analysed for the next release of the enabler, as well as additional API calls with the FL Training Collector and the FL repository. In addition, the configuration within the different aggregation strategies available is also considered. While Flower already allows use its implemented strategies like Federated Average, new implementations are also considered through the use of callback functions.

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, which will also be in charge of delivering back the updated model.

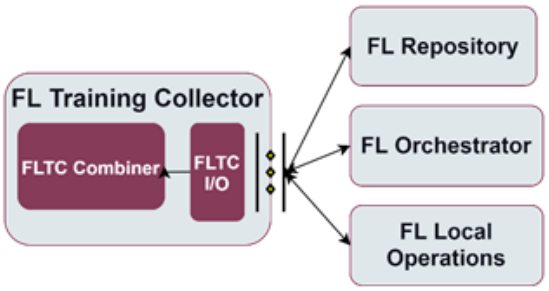

Figure 19. Training Collector enabler structure.

Functionalities:

- Aggregate local updates of the ML model prepared by independent parties as part of a model enhancement process. Responsible components: FLTC Combiner, FLTC I/O.
- Delivering back to the parties the updated model. Responsible component: FLTC I/O.

3.2.2.2. Communication interfaces

External APIs for the enabler

Method	Endpoint	Description
PUT	/model/update/{id}/{version}	Receive request with updated parameters for model with identifier id and a given version (as part of federated training).
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.

3.2.2.3. Technologies

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, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	FLTC I/O
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.	FLTC Combiner

3.2.2.4. Use cases

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

STEP 1: Receive configuration information from FL orchestator.

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

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

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

The second use case is **combaning local updates to the model to obtained new final model to be shared with involved parties**.

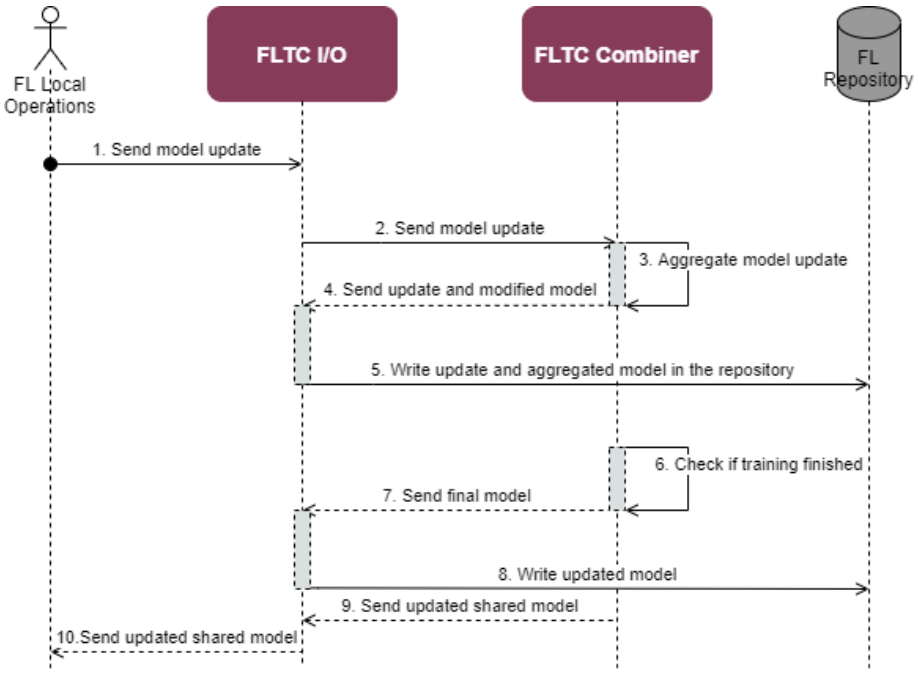

Figure 20. Local results aggregation UC.

STEP 1: FL Local Operations enabler sends local results (parameter updates proposals) of model training to FL Local Operations enabler.

STEP 2: FLTC I/O handles the request. If it is correct the proposed update is forwarded to FLTC Combiner component.

STEP 3: FLTC Combiner combines local results to deliver new shared model version. Averaging can be completed in one step or can be applied sequentially in a specific order.

STEP 4, 5: FLTC Combiner sends the received local update and aggregated model (after application of the update; intermediate results) to FLTC I/O which sends in to the FL Repository enabler to be stored.

STEP 6: FLTC Combiner verifies if model training procedure has been finished or it should still wait for local updates.

STEP 7, 8: 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.

STEP 9, 10: Send updated shared model to involved local parties.

3.2.2.5. Work progress

An outline of the project has been created, with a basic working implementation of a FastAPI server with endpoints as defined in the documentation. An appropriately triggered endpoint also starts a Flower server, which works to train the model according to a provided configuration. The endpoint can be tested using a built-in FastAPI tool in localhost /docs.

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.

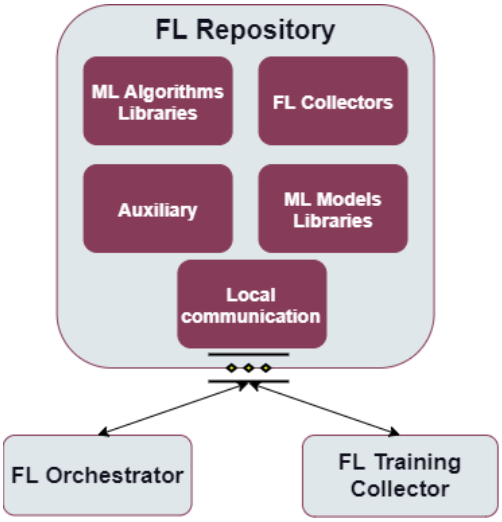

Figure 21. FL Repository enabler structure.

Functionalities:

- Provide storage for FL related data like: 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.
- Provide interfaces to put and retrieve data from different components of the enabler.
- Communication with other FL enablers. Responsible component: Local communication.

3.2.3.2. Communication interfaces

External APIs for the enabler

At this moment endpoints for access to auxiliary data are not defined. They will be added when specific needs are encountered during the project.

Method	Endpoint	Description
POST	/model	Adds new ML model to the library
PUT	/model/update/{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 lis 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 lis 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

3.2.3.3. Technologies

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

3.2.3.4. Use cases

The first use case show flow of action when **FL Orchestrator retrieves ML algorithm** that is available in the library. Assumption is that the requester knows the name and version of the algorithm to retrieve.

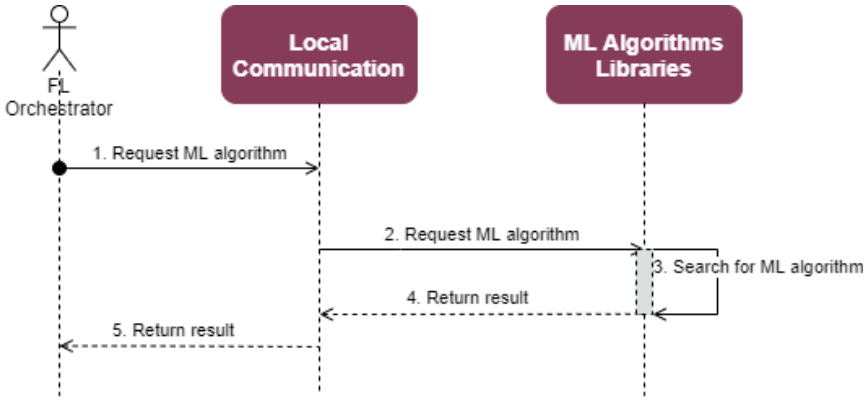

Figure 22. FL Repository UC 1.

STEP 1: FL Orchestrator sends request to Local Communication component that is responsible for enabler's communication with external entites.

STEP 2: If request is correct it is forwarded to ML Algorithms Libraries component.

STEP 3: ML Algorithms Libraries searches for an algorithm with a given name and version.

STEP 4: If algorithm was found it is returned (possibly with any required metadata) to the Local Communication. If algorithm was not found then respective information is returned to Local Communication component.

STEP 5: Local Communication forwards response to the requester.

The second use case shows flow of action when **FL Training Collector retrieves ML collector algorithm** that is available in the library. Assumption is that the requester knows the name and version of the algorithm to retrieve.

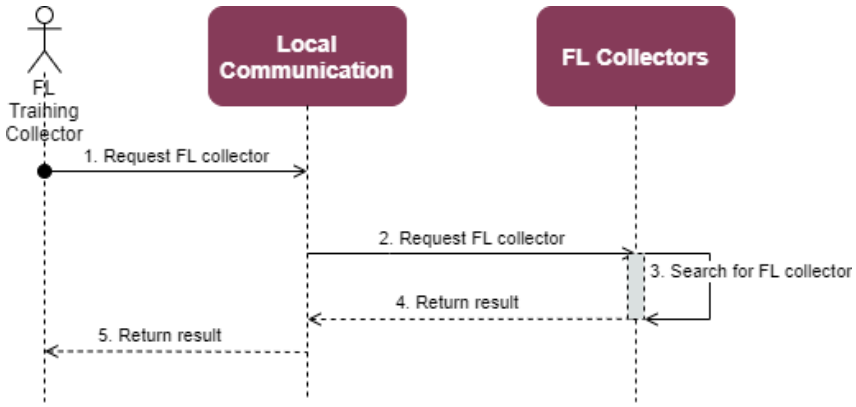

Figure 23. FL Repository UC 2.

STEP 1: FL Training Collector sends request to Local Communication component that is responsible for enabler's communication with external entities.

STEP 2: If request is correct it is forwarded to FL Collectors component.

STEP 3: FL Collectors searches for an algorithm with a given name and version.

STEP 4: If algorithm was found it is returned (possibly with any required metadata) to the Local Communication. If algorithm was not found, then respective information is returned to Local Communication component.

STEP 5: Local Communication forwards response to the requester.

The third use case shows flow of action when **FL Training Collector sends updated model (final or intermediate)** to be stored in a repository.

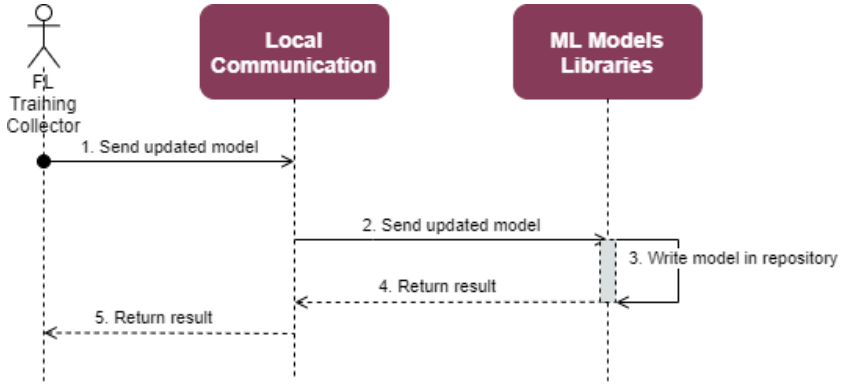

Figure 24. FL Repository UC 3.

STEP 1: FL Training Collector sends request to Local Communication component that is responsible for enabler's communication with external entities.

STEP 2: If request is correct it is forwarded to ML Models Libraries component.

STEP 3: ML Models Libraries saves a model with a given name and a new version.

STEP 4: If operation is successful confirmation is returned (possibly with any required metadata) to the Local Communication. If any problem occurs, then respective information is returned to Local Communication component.

STEP 5: Local Communication forwards response to the requester.

The list of use cases is not exhaustive however others operate in the same manner when it comes to putting and retrieving data from the repository.

3.2.3.5. Work progress

An outline of the project has been created, with a basic working implementation of a FastAPI server with endpoints as defined in the documentation. The endpoints will allow to upload and retrieve information about and data stored in the repository (right now no storage is deployed). Information exchanged can include binary data and metadata (key-value pairs) that described them. The endpoints can be tested using a built-in FastAPI tool in localhost /docs. In progress is work on specification of formats of data and metadata information that will be stored in the repository.

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.

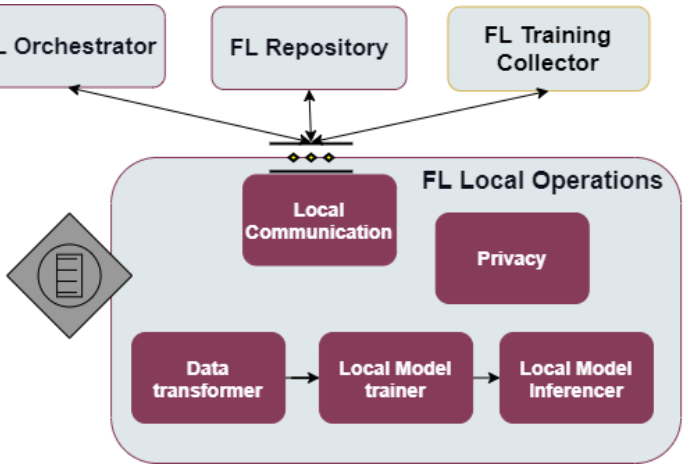

Figure 25. FL Local Operations enabler structure.

Functionalities:

- Enabler embedded in each FL involved party performing local training.
- Verification of local data formats compatibility with data formats required by FL. Responsible component: Data transformer.
- Transformation of local data formats to format required by the ML system (possibly using predefined transformers). Responsible component: Data transformer.
- Local model training. The local results will be sent to the FL training collector in order to carry out the appropriate aggregation methodology over the common shared model.
- Inference with the final shared ML model. Responsible component: Local Model Inferencer.
- Communication of model updates via encryption mechanisms. A homomorphic encryptor will not permit outsiders to see the output model of each device/party (MITM attacks), whereas methods for creating differentially private noise will guarantee that Malicious Aggregator cannot be allowed to infer which records are actual models and which not. Responsible components: Privacy, Local communication.

3.2.4.2. Communication interfaces

External APIs for the enabler

Method	Endpoint	Description
POST	/job/config/{id}	Receive configuration for training job
POST	/model	Receive new shared model
POST	/job/transformer/{id}	Receive any required data transformer for job with identifier id

Internal APIs for the enabler

Component	Method	Endpoint	Description
Data transformer	POST	/data/status	Check if local data is correctly formated for FL.
Data transformer	POST	/data/transform	Transform data to required format if appropriate transformer was provided.
Data transformer	POST	/data/transform/config	Receive transformation to be applied to transform data if they are not in a required format.
Local model training	POST	/training/model	Locally train model
Local model training	POST	/training/model	Receive model
Local training model	POST	/training/config	Receive configuration for the training process
Local model inferencer	POST	/predict/model	Inference with model
Local model training	POST	/predict/model	Receive model
Privacy	POST	/encrypt	Encrypt data
Privacy	POST	/config	Receive configuration for Privacy component

Local communication component's API is equivalent to external enabler's API.

3.2.4.3. Technologies

Technology	Justification	Component(s)
scikit-learn	A popular machine learning library often used for data preprocessing and transformation, for example encoding labels. It is open source and widely used in the industry.	Data Transformer
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

3.2.4.4. Use cases

The basic use case shows a flow of actions when FL Local Operations enabler **performs a new training job**.

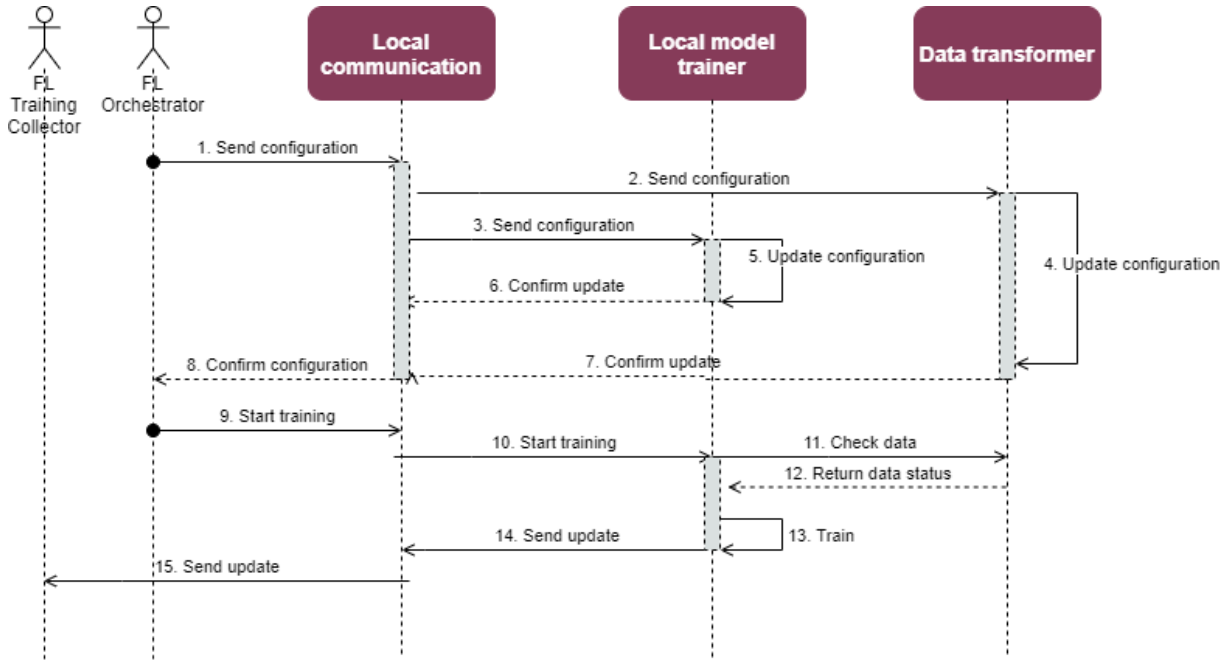

Figure 26. FL Local Operations UC.

STEP 1: FL Orchestrator send a new job configuration to FL Local Operations.

STEP 2, 3: Configuration is propagated to Local model trainer and Data transformer components.

STEP 4, 5: Components update their setup to correspond to a received configuration/

STEP 6, 7: Component confirm configuration updates to Local communications.

STEP 8: Local communication confirms to FL Orchestrator that FL Local Operations has been configured.

STEP 9: FL Orchestrator requests to start the training process.

STEP 10: Start training command is propagated to Local model trainer component.

STEP 11: Local model trainer checks with Data transformation component if local data are in correct format for the algorithm.

STEP 12: Data transformation component responds.

STEP 13: Local model trainer trains using an algorithm specified in configuration and local data.

STEP 14: Updated parameters are sent to Local communication.

STEP 15: Local communication sends updated parameters to FL Training Collector. Here, Privacy component will be utilized to protect the message send. For brevity, it was omitted on the current diagram.

3.2.4.5. Work progress

A working FastAPI server with external endpoints as defined in documentation has been added. An appropriate message sent to a specific endpoint can now trigger a Flower client with a specific configuration. The Flower client begins sample model training.

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 and incident response over the deployed architecture and pilots.

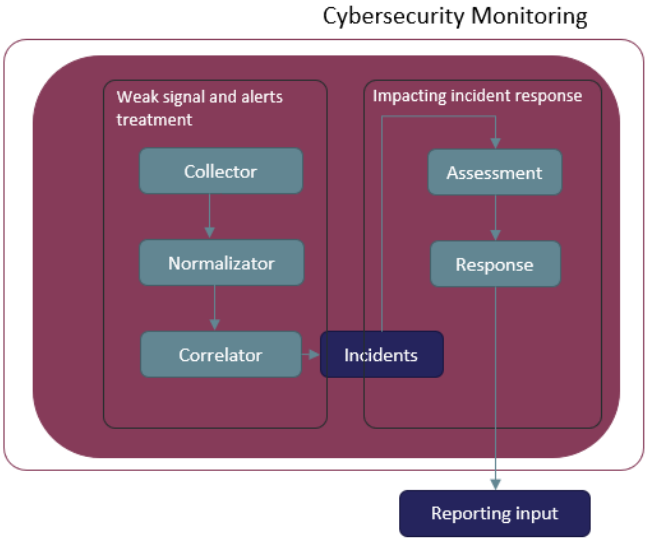

Figure 27. High-level structure of Cybersecurity monitoring enabler.

Functionalities:

- Cybersecurity enabler will receive logs and information from the agents deployed.
- Cybersecurity enabler will decode the log, identify the type of log and extract some useful fields.
- Cybersecurity enabler will have a ruleset to be applied to the received logs.
- Cybersecurity enabler will apply the active rules to the received log, and if there is a match, it will generate an alert.
- Cybersecurity enabler will normalize the alert event and correlate until determine if it is only a simple alert or a real incident.
- Cybersecurity enabler 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.
- Cybersecurity enabler 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.

3.3.1.2. Communication interfaces

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

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

3.3.1.3. Technologies

Although the definitive system to be deployed is still pending decision, the design requires the following:

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

Cybersecurity monitoring enabler can also adapt input from different interfaces such as syslog, rsyslog, or inputs from message brokers like MQTT.

3.3.1.4. Use cases

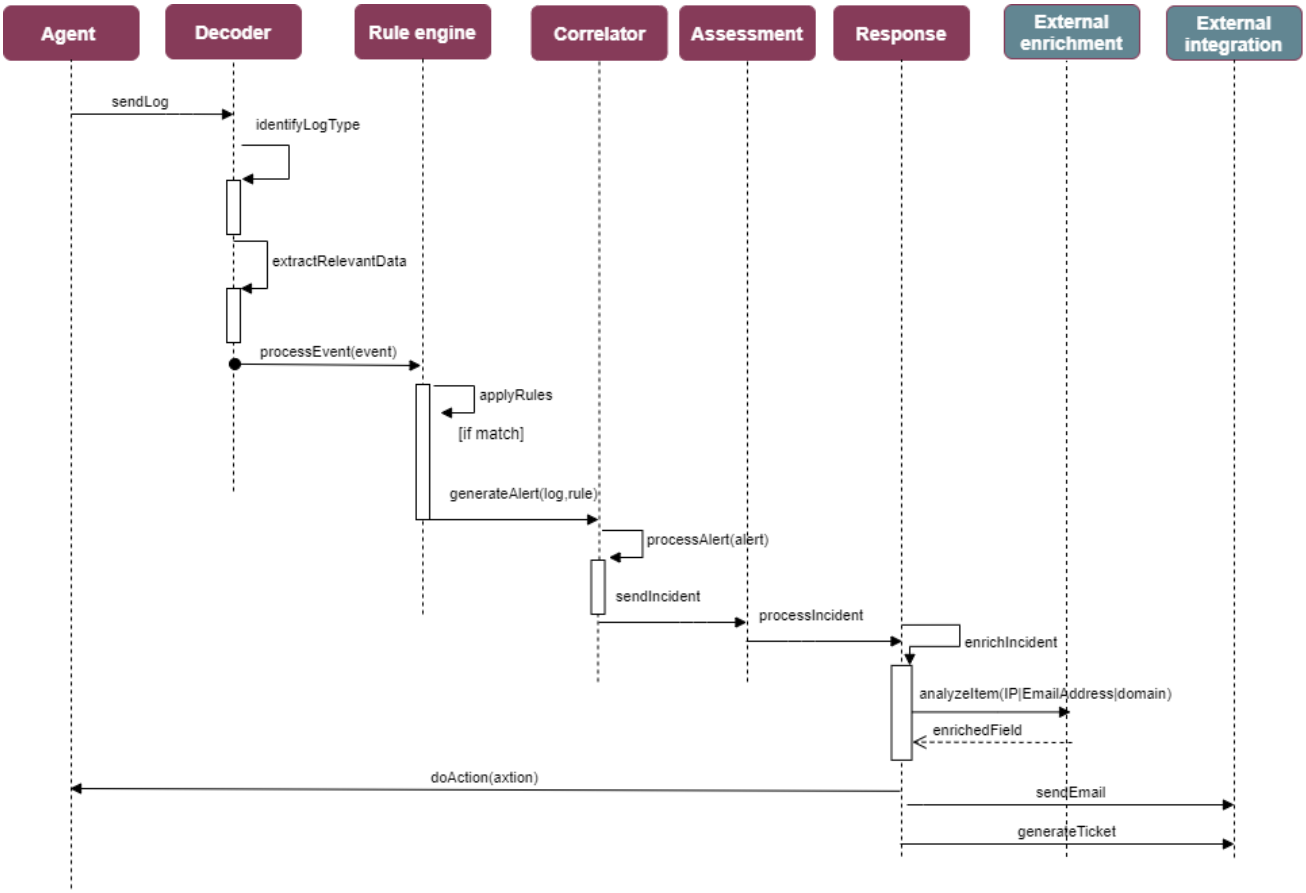

Figure 28. Cybersecurity monitoring flow process UC.

The general use case behind the cybersecurity monitoring server is described in the following flow:

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.

3.3.1.5. Work progress

Docker file specification to build a docker image with a preconfigured installation of a cybersecurity monitoring server.

The progress has been done focusing to provide to package in a docker image for cybersecurity monitoring server as long as a default security policy.

3.3.2. Cybersecurity monitoring agent enabler

3.3.2.1. Structure and functionalities

Cybersecurity monitoring agent will report to cybersecurity monitoring server. Cybersecurity monitoring agent will collect information from target system to provide relevant information if a cybersecurity breach is produced.

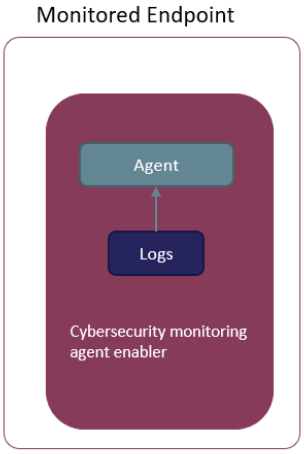

Figure 29. Cybersecurity monitoring server agent structure.

Functionalities:

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

3.3.2.2. Communication interfaces

Agent communication with server communication using wazuh implementation will use 1514 TCP/UDP port. Other implementations using rsyslog will use standard rsyslog 514 port.

3.3.2.3. Technologies

Although the definitive system to be deployed is still pending decision, the design requires the following:

Technology	Justification	Component(s)
Wazuh agent	Collection	Agent
rsyslog	Collection	Agent

Other agent-based technologies that also can be applied are based on unified logging layer like fluentd, that are also researched.

3.3.2.4. Use cases

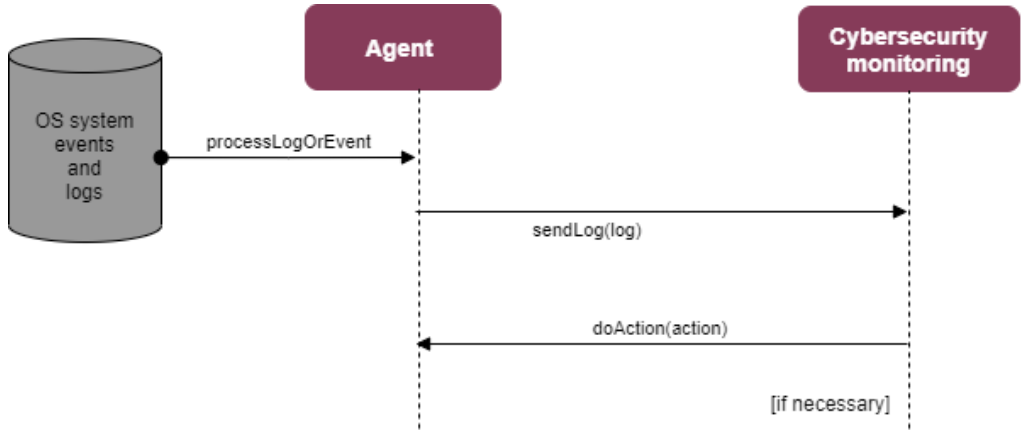

Figure 30. Cybersecurity monitoring agent flow process UC.

The general use case behind the cybersecurity monitoring agent is described in the following flow:

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 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

3.3.2.5. Work progress

Docker file specification to build a docker image with a preconfigured installation of a cybersecurity monitoring agent based on wazuh and also on rsyslog.

The progress has been done focusing to provide to package in a docker image for cybersecurity monitoring agent installed with a linux version of cybersecurity monitoring agent.

3.3.3. Identity manager enabler

Identity manager enabler will be responsible for managing identities on the access control process.

3.3.3.1. Structure and functionalities

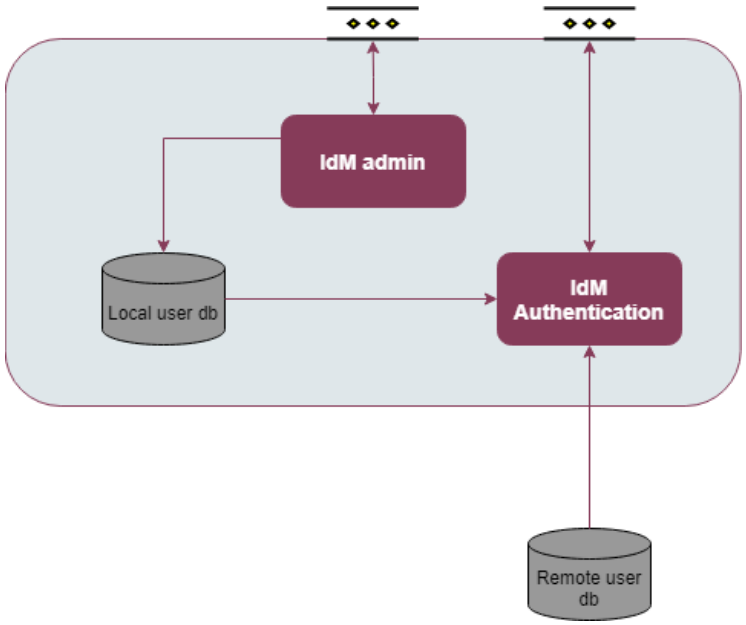

Figure 31. Identity manager enabler structure.

Functionalities:

- 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.

3.3.3.2. Technologies

Although the definitive system to be deployed is still pending decision, the design requires the following:

Technology	Justification	Component(s)
Oauth2	Standard web federated identity	IdM Authentication
LDAP connector	External user store	IdM Authentication
Web interface	Manage user database	IdM Admin

3.3.3.3. Use case

The main use case behind the identity server is described in the following flow:

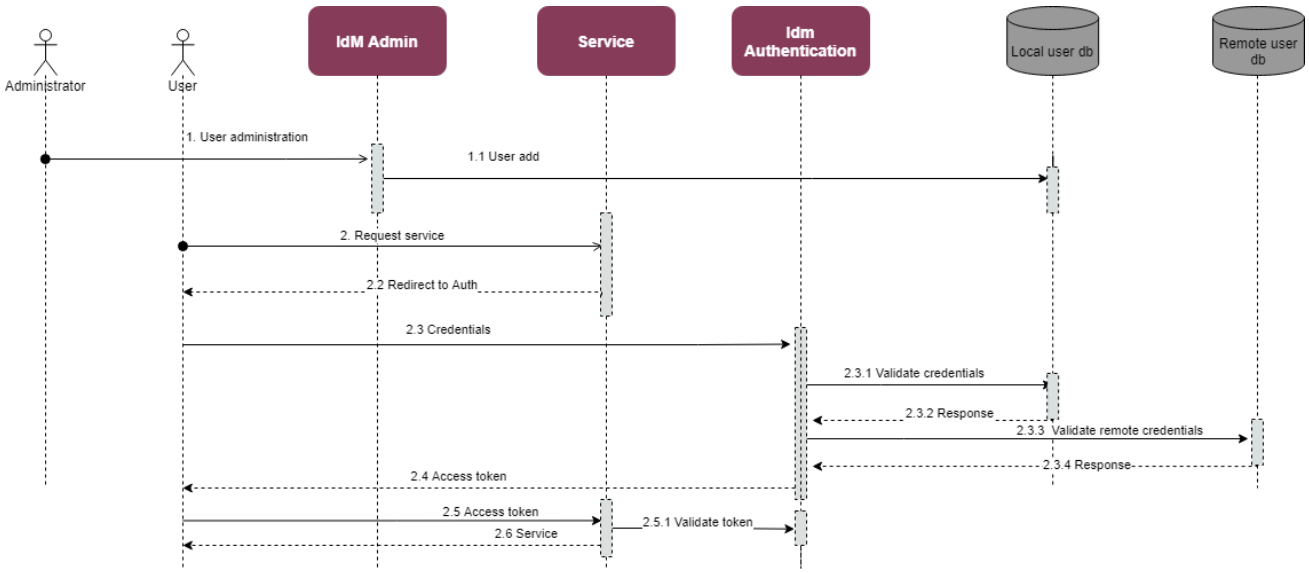

Figure 32. Identity server flow UC.

STEP 1: An administrator populates user database

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

STEP 2.1: If the user has no previous identification active, it is redirected to the Authentication server **STEP 1.2**.

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. Work progress

Multiple alternatives are being evaluated to identify the most adequate identity server that fits the requirements of the use cases. No definitive decision has been achieved yet.

3.3.4. Authorization enabler

3.3.4.1. Structure and functionalities

Authorization enabler will be responsible for the authorization phase in the access control process. Authorization enabler will be based on XACML standard security policies, results on obligations actions to be deployed after the evaluation process.

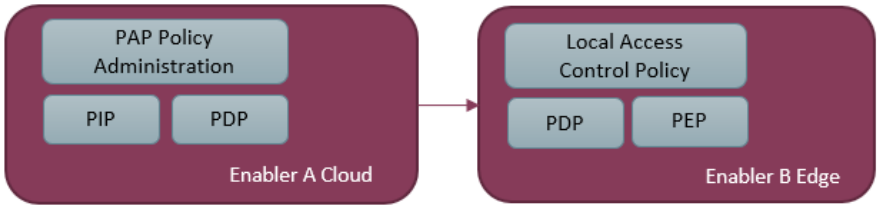

Figure 33. Authorization enabler structure.

Figure 33 above describes two different modes of deploying the same enabler. They can function as federated server, autonomous edge service or interact between both.

Functionalities:

- 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 ask 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 request to perform the derived actions (obligations) obtained as a result of the policy decision. This will have the form of REST requests.

3.3.4.2. Technologies

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

3.3.4.3. Use case

The main use case behind the Authorization server is described in the following flow:

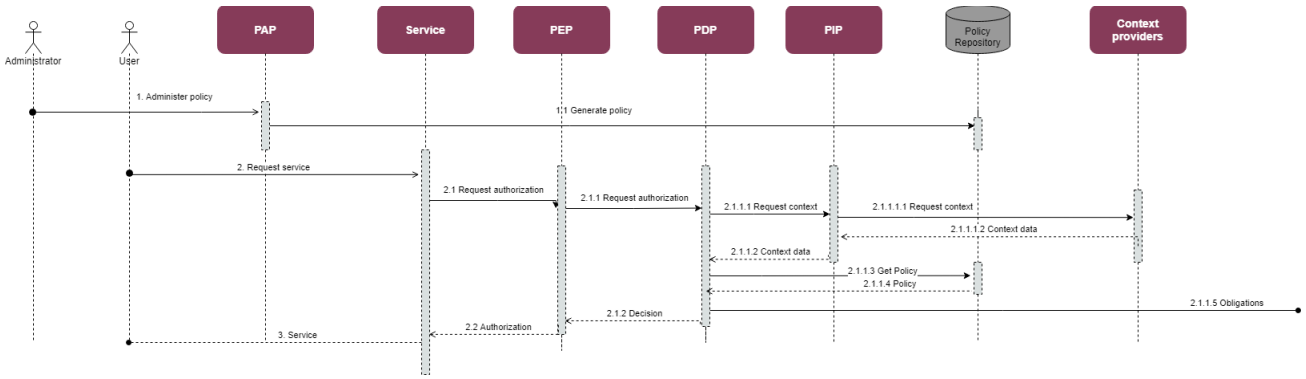

Figure 34. Authorization server UC.

STEP 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 1.1**.

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.

STEP 3: Service will be provided.

3.3.4.4. Work progress

Server core has been developed and dockerized. Current work is focused on separation of functionalities in order to federate policy generation and use.

3.4. DLT-based enablers

3.4.1. Logging and auditing enabler

3.4.1.1. Structure and functionalities

This enabler will log critical actions that happen during the data exchange between ASSIST-IoT stakeholders to allow for transparency, auditing, non-repudiation and accountability of actions during the data exchange. It will also log resource requests and identified security events to help to provide digital evidence and resolve conflicts between stakeholders, when applicable. If any requirement of filtering prior to logging, a filtering module will be considered to be deployed. The DLT API is the candidate component for performing any filtering.

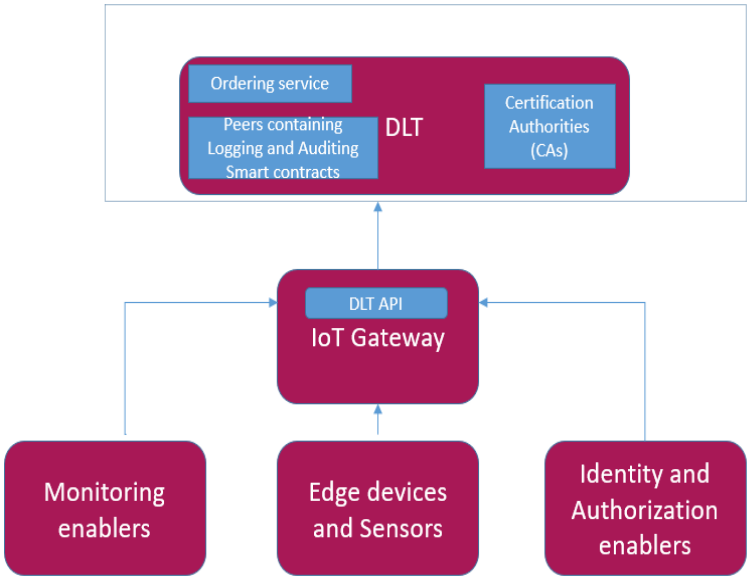

Figure 35. Logging and auditing enabler structure. . The upper (DLT) block contains the internal components of the enabler

3.4.1.2. Communication interfaces

Method	Endpoint	Description
POST	/logs	Create a new log
GET	/logs/{id}	Get log with specific ID
GET	/logs	Get all logs

3.4.1.3. Technologies

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)	Currently it is decided as project-wide standard. REST is overall a web standard.	**DLT API**

3.4.1.4. Use cases

The main use cases where this enabler is planned to be used are the immutable logging of the configurations, the logging of security incidents and the logging of the resources access. A sequence diagram depicting the main flow of the UC for logging security incidents is depicted in Figure 36 below.

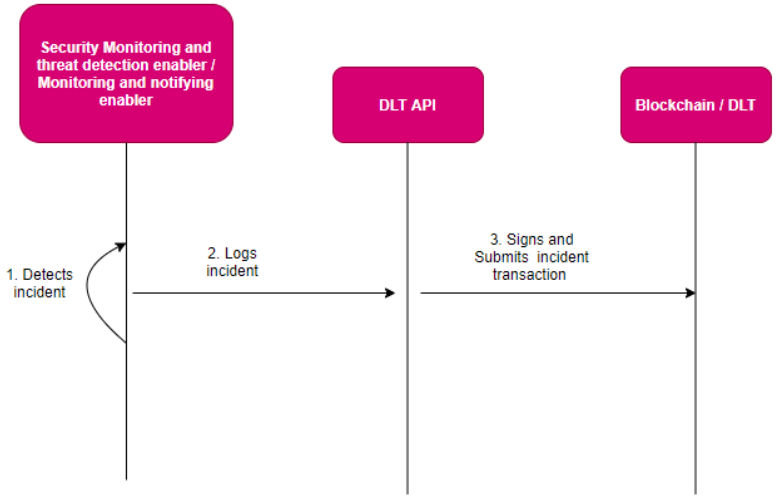

Figure 36. Incident logging flow UC.

The steps of the flow are described below:

STEP 1: The interacting enabler (either security Monitoring and Threat detection enabler or Monitoring and Notifying enabler) detects internally an incident.

STEP 2: The interacting enabler (either security Monitoring and Threat detection enabler or Monitoring and Notifying enabler) posts the log containing the incident information to the DLT API.

STEP 3: The DLT API signs and submits to the Blockchain/DLT a transaction containing the log information.

3.4.1.5. Work progress

The software development of the enabler is currently under way and an initial version of the componets has already been set, including the Hyperledger Fabric network, Smart contracts, and the DLT API. A preliminary version will be presented in less than a month.

3.4.2. Data integrity verification enabler

This is an enabler responsible for providing DLT-based data integrity verification mechanisms that allow data consumers to verify the integrity of any data at question. Network peers host smart contracts (chaincode) which includes the data integrity business logic. It stores hashed data in a data structure and it compares it with the hashed data of the queries made by clients in order to verify their integrity.

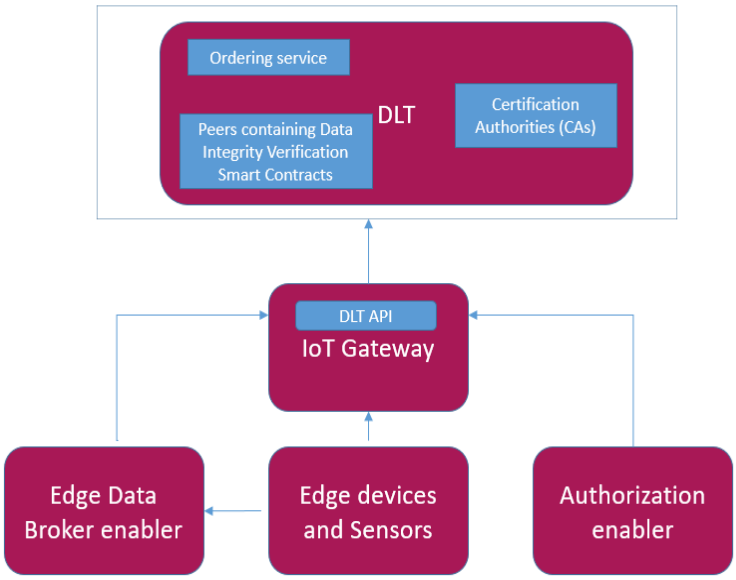

Figure 37. Data Integrity Verification enabler. The upper (DLT) block contains the internal components of the enabler.

3.4.2.1. Communication interfaces

Method	Endpoint	Description
POST	/data/{Hash}	Stores hashed data
GET	/data/{Hash}	Returns if the hashes data exist in the ledger or not

3.4.2.2. Technologies

The technologies that are to be implemented for the execution of the enabler's components are the following:

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.	**Data Integrity Verification business logic**
Hyperledger Fabric peers, orderers		**Hyperledger Fabric peers and orderers**
Hypeledger Fabric Certificate Authority (CA)		**Certification Authorities (CAs)**
REST (Enabler's API)	Currently it is decided as project-wide standard. REST is overall a web standard.	**DLT API**

3.4.2.3. Use cases

The potential use cases where this enabler is planned to be used are the integrity verification of the XACML policies and the defined thresholds of the fleet emissions to avoid their alteration. A sequence diagram depicting the main flow of the UC for the integrity verification of the XACML policies is depicted in Figure 38 below.

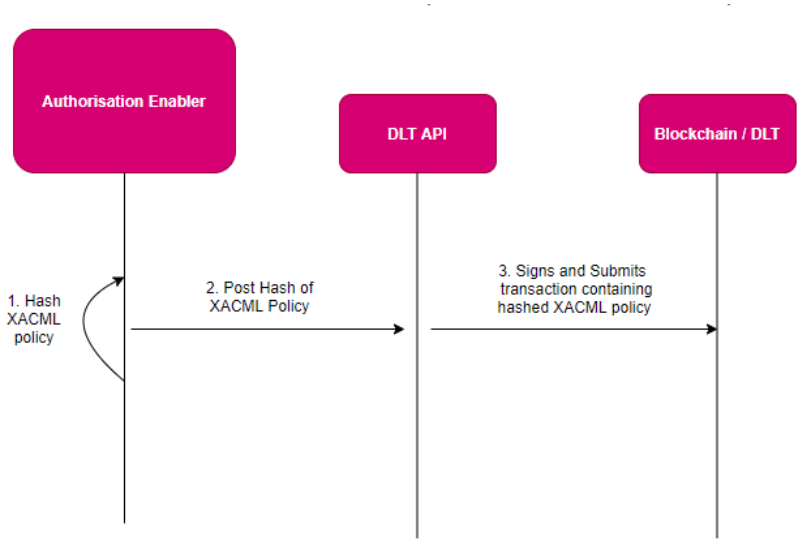

Figure 38. XACML Integrity Verification Flow UC.

The steps of the flow are described below:

STEP 1: The interacting enabler (e.g. the Authorisation enabler) prepares internally the hash of the XACML policy.

STEP 2: The interacting enabler (e.g. the Authorisation enabler) posts the hash of the XACML policy to the DLT API.

STEP 3: The DLT API signs and submits to the Blockchain/DLT a transaction containing the the hash of the XACML policy.

3.4.2.4. Work progress

The software development of the enabler is currently under way and an initial version of the components has already been set, including the Hyperledger Fabric network, Smart contracts, and the DLT API. A preliminary version will be presented in less than a month.

3.4.3. Distributed broker enabler

3.4.3.1. Structure and functionalities

This enabler will provide a mechanism that will facilitate data sharing between different heterogeneous IoT devices belonging to various edge domains and/or between different enablers of the architecture. In coordination with other enablers that will ensure trust between data sources (i.e. Identity and Authorisation providers), it will deal with data source metadata management and provide trustable, findable, and retrievable metadata for the data sources.

Another functionality for the distributed broker enabler is the data source registration. The enabler will serve as a trusted registry of all the IoT domains/devices and/or ASSIST-IoT enablers that act as data producers. Indexing and querying services will facilitate the efficient retrievability of the stored metadata of the registered producers by consumers in compliance with the FAIR principles.

The last enabler's functionality is about the semantic interoperability facilitation. This enabler can act as a facilitator to the enablers that will provide semantic interoperability (e.g. the semantic repository) by providing searchable metadata of the interoperable IoT domains (e.g. references to the semantic repository locations of the data, data model references, data models/schemata, data characteristics and data descriptions, data usage constraints, etc).

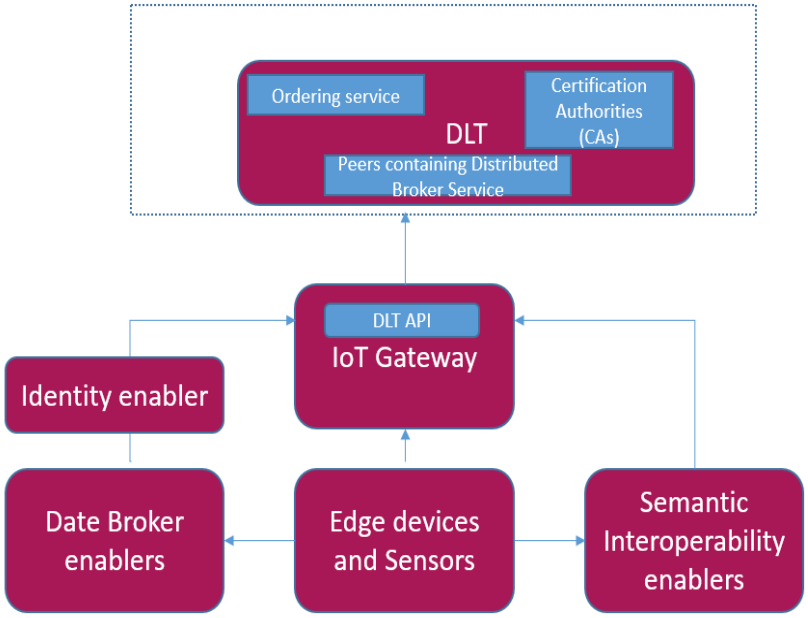

Figure 39. Distributed data enabler overview. The upper (DLT) block contains the internal components of the enabler.

3.4.3.2. Communication interfaces

Method	Endpoint	Description
POST	/metadata/{sourceID}	Post data source metadata to the Data Consumer given the data source id
GET	/metadata/{sourceID}	Get data source metadata from Data Provider (source)

3.4.3.3. Technologies

The technologies that are to be implemented for the execution of the enabler's components are the following:

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.	**Distributed Broker service (data sources registry) business logic**
Hyperledger Fabric peers, orderers		**Hyperledger Fabric peers and orderers**
Hypeledger Fabric Certificate Authority (CA)		**Certification Authorities (CAs)**
REST (Enabler's API)	Currently it is decided as project-wide standard. REST is overall a web standard.	**DLT API**

3.4.3.4. Use cases

The main use cases where this enabler is planned to be used is the data sources registration that will facilitate the semantic interoperability among different IoT domain pertaining heterogeneous data sources. A sequence diagram depicting the main flow of the UC for data source registration is depicted in Figure 40 below.

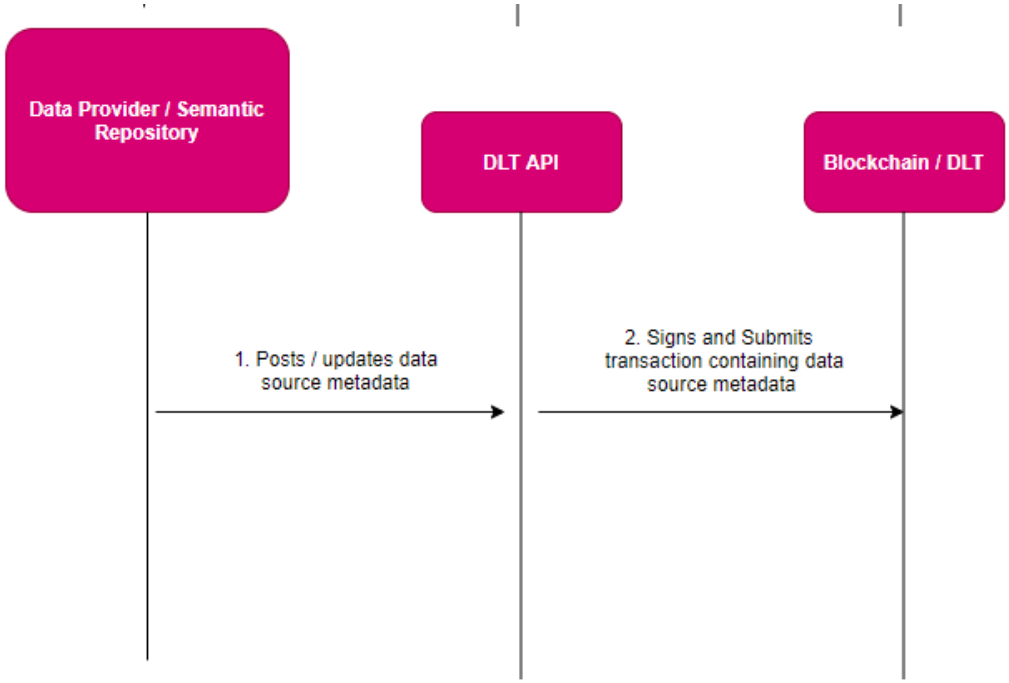

Figure 40. Data source metadata posting/updating Flow UC.

The steps of the flow are described below:

STEP 1: The interacting enabler (the Data Provider / the Semantic Repository enabler) posts the data source metadata (e.g. data source URLs and access points) to the DLT API.

STEP 2: The DLT API signs and submits to the Blockchain/DLT a transaction containing the data source metadata.

3.4.3.5. Work progress

The software development of the enabler is currently under way and an initial version of the components has already been set, including the Hyperledger Fabric network, Smart contracts, and the DLT API. A preliminary version will be presented in less than a month.

3.4.4. DLT-based FL enabler

3.4.4.1. Structure and functionalities

This enabler will foster the use of DLT-related components to exchange the local, on-device models (or model gradients) in a decentralised way. The DLT can act as a component to manage AI contextual information and prevent any alteration to the data. The alteration of data is a threat to the Federated Learning approach and the DLT can help in mitigating the threat. Moreover, the enabler will allow mitigating single-point of failures. Finally, the enabler can be charged with validating the individually trained models to rule out malicious updates that can harm the global model.

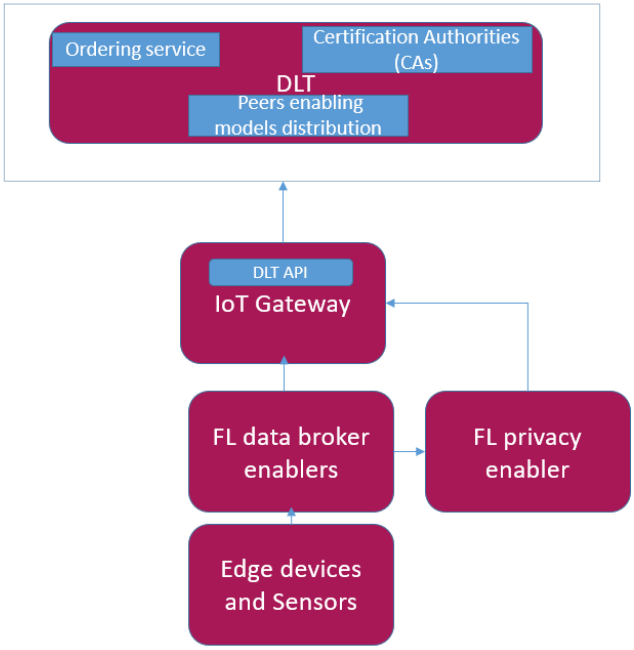

Figure 41. DLT-based FL enabler overview. The upper (DLT) block contains the internal components of the enabler.

3.4.4.2. Communication interfaces

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

3.4.4.3. Technologies

The technologies that are to be implemented for the execution of the enabler's components are the following:

Technology	Justification	Component(s)
Hyperledger Fabric Chaincode (Smart Contracts) / Swarm Learning	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.	**Models distribution business logic**
Hyperledger Fabric peers, orderers		**Hyperledger Fabric peers and orderers**
Hypeledger Fabric Certificate Authority (CA)		**Certification Authorities (CAs)**
REST (Enabler's API)	Currently it is decided as project-wide standard. REST is overall a web standard.	**DLT API**

3.4.4.4. Use cases

The main use case for which this enabler is planned to be used is the model verification and exchange/distribution. The idea is that after securing the models using Privacy Enhancing Techniques (PETs), the models will be forwarded to the nodes of the DLT for model verification and exchange/distribution.

3.4.4.5. Work progress

The software development of the enabler is currently under way and an initial version of the components has already been set, including the Hyperledger Fabric network, Smart contracts, and the DLT API. A preliminary version will be presented in less than a month.

3.5. Manageability

This last section differs from the ones provided for the rest of the verticals. The reason lies in the fact that the task started later in time, and thus its maturity is in a lower level. Hence, this subsection will follow the structure of D5.1, in which the identified enablers are presented and later on expanded in a dedicated Appendix A - , where their respective templates are presented. The following enablers aim at allowing the system owner to handle an ASSIST-IoT deployment, from registering and managing devices and enablers, to compositing flows of services.

3.5.1. Enabler for registration and status of enablers

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).

3.5.2. Enabler for management of services and enablers' workflow

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.

3.5.3. Devices management enabler

The main functionalities of this enabler will be to register: (i) a k8s node in an ASSIST-IoT k8s cluster, (ii) a smart IoT device in a deployment, and (iii) 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., available and used resources) and current instantiated enablers' components.

4. Future Work

This document is part of the second deliverable for WP5 along with preliminary software version (status as of M12). Since this deliverable will have a second and third iterations, the specifications included here may be extended or updated as the project evolves.

The finalization of this deliverable will speed up the development of related software, therefore next tasks will be to:

- Continue the development along with technical description provided here,
- Extend or fill in missing information such as endpoint/API specification for each enabler and component,
- Verify issues listed in gaps sections and propose how to mitigate identified risks,
- Any adjustments necessary (e.g., slight modification in provided functionalities, change in enabler structure, change in selected technologies),
- Management of the backlog of tasks, distribution of work and progress in implementation activities.

Appendix A - Manageability Enablers templates

A.1 - Enabler for registration and status of enablers

Table 1. General information of the Enabler for registration and status of enablers

Enabler	***Enabler for registration and status of enablers***
id	*T55E1*
Owner and support	*UPV, SRIPAS, CERTH*
Description and main functionalities	*This enabler will serve as a registry of enablers and, in case they are deployed, the retrieval of their status. In particular, it will:* • *Allow the registration of an enabler (this is, from an ASSIST-IoT repository). Essential enablers will be pre-registered* • *Retrieve a list of currently-running enablers* • *Depict the status and the specific logs of an enabler (the latter only if the enabler with log collection capabilities is in place).* • *Facilitate the deployment of standalone enablers (mostly for those that have to be present at any deployment)*
Plane/s involved	*Smart network and control, Data management, and Application and Services planes*
Vertical, related capabilities and features	*Manageability*
Relation with other enablers	*This enabler will guide the deployment of the essential enablers of an ASSIST-IoT deployment. The smart orchestrator will be deployed with the aid of this one, and the rest will be managed by it.* • *T44E1: Tactile dashboard enabler* • *T42E1: Smart Orchestrator* • *T42E3: Performance and usage diagnosis enabler* • *T43E8: LTSE* • *T55E4: Monitoring and notifying enabler*
Requirements mapping	• *RC7: Edge-oriented deployment*
Use case mapping	*This enabler will be present at all use cases, for administration purposes.*
Required components	*Enablers management server, Enablers management UI, Enablers registry*
TRL information	*Current: TBD* *Target: TRL6*
Identified by	*UPV*
Modification date	*11-10-21*

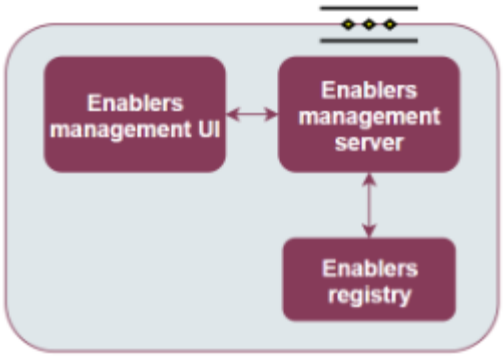

Figure 42. High-level diagram of the Enabler for registration and status of enablers.

Endpoints

Method	URL	Description
POST	**/enablers**	Registers an enabler from an ASSIST-IoT repository, and adds it to the orchestrator
GET	**/enablers**	Returns a list of all the enablers registered in a deployment
POST	**/deploy**	Deploy a standalone enabler (i.e., not part of a flow, but that could be later on used by one or many)
GET	**/deploy**	Returns the status of all the deployed enablers
GET	**/deploy/${id}**	Returns the status of a deployed enabler and its components
GET	**/deploy/logs/${id}**	Return the logs of a particular enabler
PUT	**/deploy/${id}**	Modifies the configuration parameters of an enabler, and updates it.
DELETE	**/deploy/${id}**	Deletes an enabler. It can affect any flow that consumes it.

Components

Enabler component	***Enablers management server***
id	*T53E1_enablers*
Description and main functionality	*This server will allow to: (i) register/remove enablers from an ASSIST-IoT enablers repository, or any repository with enablers compatible with an ASSIST-IoT ecosystem; (ii) instantiate/modify/remove a standalone enabler, this is, not part of a flow (but that can be later on connected to one, or many); and (iii) access the logs of any deployed enabler and its components.*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*Python custom functions, Node.js*

Enabler component	***Enablers management UI***
id	*T53E1_enablersUI*
Description and main functionality	*Set of interfaces to allow end-users to execute the functions provided by the server.*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*PUI9, Vue*

Enabler component	***Enablers registry***
id	*T53E1_registry*

Description and main functionality	*This registry will keep track of all the enablers that can be part of a particular deployment, and well as a registry of currently-deployed enablers and components.*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*Docker, Helm*

A.2 - Enabler for management of services and enablers' workflow

Table 2. General information of the Enabler for management of services and enablers' workflow

Enabler	***Enabler for management of services and enablers' workflow***
id	*T55E3*
Owner and support	*UPV, PRO, SRIPAS*
Description and main functionalities	*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* ***composite service*** *(i.e., a workflow). 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.*
Plane/s involved	*Smart network and control, Data management, and Application and Services planes*
Vertical, related capabilities and features	*Manageability*
Relation with other enablers	*All the enablers should be deployable with this one. This list only refers to those enablers that will take part on the orchestration process.* • *T42E1: Smart Orchestrator* • *T55E1: Management of enablers existence* • *T55E4: Management of devices in an ASSIST-IoT deployment*
Requirements mapping	• *RC7: Edge-oriented deployment*
Use case mapping	*All use cases will require of this enabler to be fulfilled*
Required components	*Workflow management UI, Workflow management API, Workflow registry*
TRL information	*Current: TBD* *Target: TRL6*
Identified by	*UPV*
Modification date	*11-10-21*

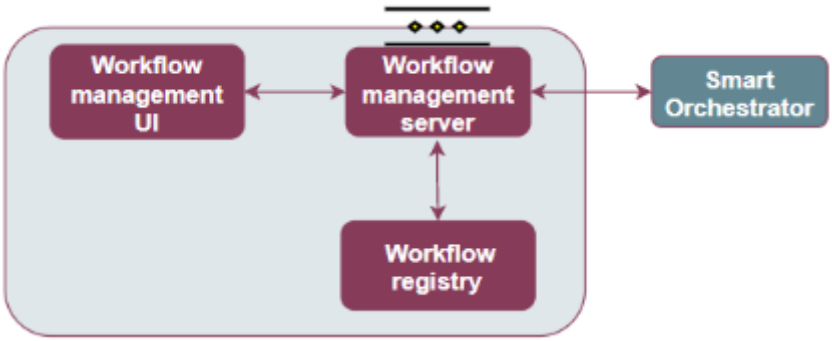

Figure 43. High-level diagram of the enabler for management of services and enablers' workflow.

Endpoints

Method	**URL**	**Description**
POST	**/workflow/deploy**	Starts the deployment of a service workflow

GET	**/workflow/deploy**	Returns a list of currently deployed service workflows
GET	**/workflow/deploy/${id}**	Get current status of a particular service workflow and its managed enablers
DELETE	**/workflow/deploy/${id}**	Deletes a deployed workflow and its related enablers (it does not remove any enabler which has been deployed in other workflows or in a standalone way)

Components

Enabler component	***Workflow management UI***
id	*T53E3_UI*
Description and main functionality	*It will provide a high-level dashboard to declare, in a flow-based/DAG fashion, an end-to-end service workflow consisted of a set of enablers. To that end, it will retrieve info related to available (implementable) enablers, deployed enablers, cluster/s topology, etc. In addition, pop-ups to declare pre-deploying configuration for the enablers will be present in this interface.*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*Custom/Open source DAG generator, Vue*

Enabler component	***Workflow registry***
id	*T53E3_registry*
Description and main functionality	*It will register all the workflows (instantiated, running, terminated) and their current status. It will also contain policies for completing information of manifests before being sent to the orchestrator.*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*SQLite, MongoDB*

Enabler component	***Workflow management server***
id	*T53E3_API*
Description and main functionality	*This API will be primarily in charge of receiving a workflow command (with workflow manifest file and configuration parameters) and distribute the necessary actions to instantiate (or reuse existing) and connect enablers to compose an end-to-end service.*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*Express, Flask, Python*

A.3 - Devices management enabler

Table 3. General information of the Devices management enabler

Enabler	***Devices management enabler***
id	*T55E4*
Owner and support	*UPV, NEWAYS*
Description and main functionalities	*The main functionality of this enabler will be to register: (i) a k8s node in an ASSIST-IoT k8s cluster, (ii) a smart IoT device in a deployment, and (iii) 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., available and used resources) and current instantiated enablers' components.*

Plane/s involved	*Device and Edge plane, Smart network and control*
Vertical, related capabilities and features	*Manageability*
Relation with other enablers	*All the enablers should be deployable with this one. This list only refers to those enablers that will take part on the orchestration process.* • *T42E1: Smart Orchestrator* • *T44E3: Performance and Usage diagnosis enabler* • *T51E3: Geolocation* • *T51E4: Monitoring and Notifying*
Requirements mapping	• *RC7: Edge-oriented deployment*
Use case mapping	*This enabler will be present at all use cases, for administration purposes.*
Required components	*Devices management UI, Devices management server, Devices registry*
TRL information	*Current: TBD* *Target: TRL6*
Identified by	*UPV*
Modification date	*11-10-21*

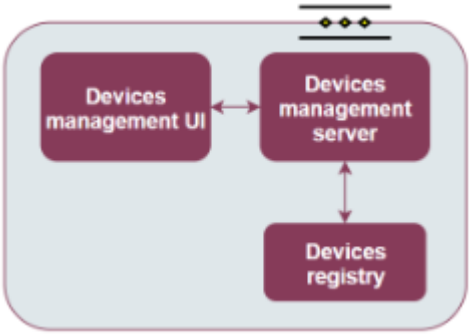

Figure 44. High-level diagram of the Devices management enabler.

Endpoints

Method	URL	Description
POST	**/clusters**	Registers a k8s cluster to an ASSIST-IoT deployment, and notifies it to the orchestrator
GET	**/clusters**	Returns the list of clusters registered in an ASSIST-IoT deployment
GET	**/clusters/${id}**	Return information related to a cluster, including available and used resources of its managed nodes, and deployed components of ASSIST-IoT enablers
DELETE	**/clusters/${id}**	Deletes a cluster from an ASSIST-IoT deployment and unsubscribes it from the orchestrator. It can be done only if any ASSIST-IoT enabler is running
POST	**/devices/**	Registers a device to an ASSIST-IoT deployment
GET	**/devices/**	Returns the list of devices registered in an ASSIST-IoT deployment
DELETE	**/devices/${id}**	Unregisters a smart IoT device from the deployment
POST	**/nodes**	Adds a k8s node to a k8s cluster
GET	**/nodes**	Retrieves all the information related to a cluster, including available and used resources of its managed nodes, and deployed components of ASSIST-IoT enablers

Components

Enabler component	***Devices management server***
id	*T53E4_devices*
Description and main functionality	*This server will allow to: (i) register/remove existing k8s nodes and clusters to ASSIST-IoT ecosystem (and to subscribe clusters to the orchestrator); (ii) register/removing smart*

	IoT devices to the ecosystem; and (iii) retrieve a list of components and metrics related hardware usage and availability in the different nodes.
Target node/s	*High-level node, or Cloud*
Candidate technologies	*Python custom functions, Node.js*

Enabler component	***Devices management UI***
id	*T53E4_devicesUI*
Description and main functionality	*Set of interfaces to allow end-users to execute the functions provided by the server.*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*PUI9, Vue*

Enabler component	***Devices registry***
id	*T53E4_API*
Description and main functionality	*This registry will keep track of all the clusters, nodes and devices that are part of a particular deployment*
Target node/s	*High-level node, or Cloud*
Candidate technologies	*SQLite, MongoDB*