

*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



# D5.5 – Transversal Enabler Development Final Version

<b>Deliverable No.</b>	D5.5	<b>Due Date</b>	31-OCT-2023
<b>Type</b>	Other	<b>Dissemination Level</b>	Public
<b>Version</b>	1.0	<b>WP</b>	WP5
<b>Description</b>	Final specification of the vertical enablers identified and developed in ASSIST-IoT.		



# Copyright

Copyright © 2020 the ASSIST-IoT Consortium. All rights reserved.

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 INSTYTUT 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

This document contains material, which is the copyright of certain ASSIST-IoT consortium parties, and may not be reproduced or copied without permission. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

The information contained in this document is the proprietary confidential information of the ASSIST-IoT Consortium (including the Commission Services) and may not be disclosed except in accordance with the Consortium Agreement. The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the Project Consortium as a whole nor a certain party of the Consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and accepts no liability for loss or damage suffered by any person using this information.

The information in this document is subject to change without notice.

The content of this report reflects only the authors' view. The Directorate-General for Communications Networks, Content and Technology, Resources and Support, Administration and Finance (DG-CONNECT) is not responsible for any use that may be made of the information it contains.

## Authors

Name	Partner	e-mail
Alejandro Fornés	P01 UPV	alforlea@upv.es
Raúl Reinoso	P01 UPV	rreisim@upv.es
Rafael Vaño	P01 UPV	ravagar2@upv.es
Eduardo Garro	P02 PRO	egarro@prodevelop.es
Katarzyna Wasielewska-Michniewska	P03 IBSPAN	katarzyna.wasielewska@ibspan.waw.pl
Przemysław Hołda	P03 IBSPAN	przemyslaw.holda@ibspan.waw.pl
Karolina Bogacka	P03 IBSPAN	k.bogacka@ibspan.waw.pl
Wiesław Pawłowski	P03 IBSPAN	wieslaw.pawlowski@ibspan.waw.pl
Anastasia Blitsi	P04 CERTH	akblitsi@iti.gr
Evripidis Tzonas	P04 CERTH	tzionasev@iti.gr
Johan Schabbink	P09 NEWAYS	johan.schabbink@newayselectronics.com
Oscar López Pérez	P13 S21 SEC	olopez@s21sec.com
Rafael Borné	P13 S21 SEC	rborne@s21sec.com

## History

Date	Version	Change
18-May-2023	0.1	Table of content
02-Oct-2023	0.2	First round of contributions completed
13-Oct-2023	0.3	Second round of contributions integrated. Ready for internal review
27-Oct-2023	0.4	Integration of changes from internal review
31-Oct-2023	1.0	Final version submitted to EC

## Key Data

<b>Keywords</b>	Enablers, verticals, self-*, security, manageability, scalability, federated learning, DLT
<b>Lead Editor</b>	Katarzyna Wasielewska-Michniewska (P03 - IBSPAN)
<b>Internal Reviewer(s)</b>	Alejandro Fornés (P01 - UPV), Konstantinos Fragkos (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 collects the work and outcomes of the five tasks of the work package, which focus on the design and implementation of the enablers required to implement the different verticals of the ASSIST-IoT architecture. These enablers (which may entail not just software but also hardware-related design and development) belong to at least one of the following verticals: (i) Self-\*, (ii) Scalability, (iii) Interoperability, (iv) Manageability, and (v) Security, Privacy and Trust.

This deliverable is the third in a series of three iterations devoted to the formalisation and design of the vertical enablers identified within the scope of the project, hence presenting their final software structure. It updates and extends the specifications presented in the previous deliverable, accompanied by the developed software artifacts (i.e., enablers). In particular, this deliverable provides the necessary updated technical information about the design and implementation (structure and functionalities, technologies, diagrams of enabler stories, communication interfaces), jointly with the (software) outcomes developed so far. The concepts presented here are a continuation of work summarised in D5.3 - Transversal Enablers Development Intermediate Version.

It should be mentioned that the other series of deliverables belonging to this Work Package, devoted mainly to the design of the actual artifacts, also included an update of specifications. Thus, the present document is built upon both series (particularly, after D5.3 & D5.4). The final design of the following enablers is included:

- From Self-\*: Self-healing device enabler, Resource provisioning enabler, Monitoring and notifying enabler, Location tracking enabler, Location processing enabler, and Automated configuration enabler.
- From Federated Machine Learning: FL Orchestrator, FL Training Collector, FL Repository, and FL Local Operations.
- From Cybersecurity: Cybersecurity monitoring enabler, Cybersecurity monitoring agent enabler, Identity manager enabler, and Authorisation enabler.
- From DLT: Logging and auditing enabler, Data integrity verification enabler, Distributed broker enabler, and DLT-based FL enabler.
- From Manageability: Clusters and topology manager, Enablers manager, and Composite services manager (the names of these enablers were modified in the last architecture document, D3.7).

For each of them, their respective (i) tables of general information, (ii) high-level structure and components description with implementation technologies, (iii) communication, and (v) enabler stories are presented in their final versions.

# Table of contents

Table of contents .....	5
List of figures .....	6
List of tables .....	7
List of acronyms .....	10
1. About this document.....	12
1.1. Deliverable context .....	12
1.2. The rationale behind the structure.....	13
1.3. Outcomes of the deliverable.....	13
1.4. Lessons learnt.....	13
1.5. Deviation and corrective actions .....	13
1.6. Version-specific notes .....	13
2. Introduction .....	14
3. Vertical enablers.....	14
3.1. Self-* enablers.....	16
3.1.1. Self-healing device enabler.....	16
3.1.2. Resource provisioning enabler.....	19
3.1.3. Monitoring and notifying enabler .....	24
3.1.4. Location tracking enabler .....	29
3.1.5. Location processing enabler .....	29
3.1.6. Automated configuration enabler .....	32
3.2. Federated learning enablers.....	36
3.2.1. FL Orchestrator.....	36
3.2.2. FL Training collector.....	42
3.2.3. FL Repository .....	45
3.2.4. FL Local operations .....	51
3.3. Cybersecurity enablers.....	56
3.3.1. Identity manager enabler .....	56
3.3.2. Authorisation enabler.....	58
3.3.3. Cybersecurity monitoring enabler.....	62
3.3.4. Cybersecurity monitoring agent enabler.....	65
3.4. DLT-based enablers .....	68
3.4.1. Logging and auditing enabler .....	68
3.4.2. Data integrity verification enabler .....	71
3.4.3. Distributed broker enabler .....	73
3.4.4. DLT-based FL enabler.....	75
3.5. Manageability enablers .....	78
3.5.1. Enablers manager.....	78

3.5.2.	Composite services manager .....	85
3.5.3.	Clusters and topology manager.....	89
4.	Enabler's Technical Documentation and Demo Videos.....	95
5.	Conclusions .....	96

## List of figures

Figure 1.	WP5 enablers distribution among verticals.....	14
Figure 2.	Example of high-level diagram.....	15
Figure 3.	High-level diagram of the Self-healing device enabler.....	17
Figure 4.	Self-healing device enabler ES1 (CPU/RAM usage monitoring and threshold update).....	18
Figure 5.	Self-healing device enabler ES2 (RAM usage monitoring and threshold update).....	18
Figure 6.	High-level diagram of the Resource provisioning enabler.....	20
Figure 7.	Resource provisioning enabler ES1 (get enablers and components inferred).....	21
Figure 8.	Resource provisioning enabler ES2 (update enablers and components to get inferred) .....	21
Figure 9.	Resource provisioning enabler ES3 (perform training) .....	22
Figure 10.	Resource provisioning enabler ES4 (get interval range for training).....	22
Figure 11.	Resource provisioning enabler ES5 (update data interval for training) .....	23
Figure 12.	Resource provisioning enabler ES6 (perform inference).....	23
Figure 13.	High-level diagram of the Monitoring and notifying enabler .....	25
Figure 14.	Monitoring and notifying enabler ES1 (IoT device not receiving data) .....	26
Figure 15.	Monitoring and notifying enabler ES2.....	27
Figure 16.	Monitoring and notifying enabler ES3.....	28
Figure 17.	Monitoring and notifying enabler ES4 (checking health of devices or gateways) .....	28
Figure 18.	High-level diagram of the Location processing enabler .....	30
Figure 19.	Location processing enabler ES1 (query configuration).....	31
Figure 20.	Location processing enabler ES2 (running query input).....	31
Figure 21.	High-level diagram of the Automated configuration enabler .....	33
Figure 22.	Automated configuration enabler ES1 (model creation).....	34
Figure 23.	Automated configuration enabler ES2 (model deletion).....	35
Figure 24.	Automated configuration enabler ES3 (updating configuration).....	35
Figure 25.	High-level diagram of FL Orchestrator enabler .....	37
Figure 26.	FL Orchestrator ES1 (user configures FL training).....	38
Figure 27.	FL WebUI homepage.....	39
Figure 28.	Screenshot of FL WebApp initial FL algorithm parameters.....	39
Figure 29.	Screenshot of FL WebApp final configuration setup.....	39
Figure 30.	FL Orchestrator ES2 (initial setup of FL architecture) .....	40
Figure 31.	FL Orchestrator ES3 (lifecycle management of FL Training collector – Option A: happy path) ...	40
Figure 32.	Screenshot of FL WebApp lifecycle management for the Option A: Happy Path .....	41
Figure 33.	FL Orchestrator ES3 (lifecycle management of FL Training collector – Option B: FL local operations below minimum).....	41
Figure 34.	High-level diagram of the FL Training collector .....	43
Figure 35.	FL Training collector ES1 (training mechanism instantiation).....	44
Figure 36.	FL Training collector ES2 (local results aggregation) .....	44
Figure 37.	High-level diagram of the FL Repository .....	46
Figure 38.	FL Repository ES1 (retrieve algorithm) .....	48
Figure 39.	FL Repository ES2 (retrieve ML collector).....	49
Figure 40.	FL Repository ES3 (send updated model results).....	49
Figure 41.	FL Repository ES4 (download FL serialised Pickle file) .....	50
Figure 42.	Screenshot of the FL WebApp download page.....	50
Figure 43.	FL Repository ES5 (data transformation).....	51
Figure 44.	High-level diagram of the FL Local operations .....	53

Figure 45. FL Local operations ES1 (FL training).....	54
Figure 46. FL Local operations ES2 (inference).....	55
Figure 47. High-level diagram of the Identity Manager enabler.....	57
Figure 48. Identity manager enabler ES1 and ES2 (add user, authenticate user).....	58
Figure 49. High-level diagram of the Authorization enabler .....	60
Figure 50. Authorization enabler ES1 (authorization flow) .....	61
Figure 51. High-level diagram of the Cybersecurity monitoring enabler.....	63
Figure 52. Cybersecurity Monitoring enabler ES1 (cyberthreats protection) .....	65
Figure 53. High-level diagram of Cybersecurity monitoring agent enabler.....	66
Figure 54. Cybersecurity monitoring agent enabler ES1 (send collected data and actuate).....	67
Figure 55. High-level diagram of the Logging and auditing enabler .....	69
Figure 56. Logging and auditing enabler ES1 (logging) .....	70
Figure 57. High-level diagram of the Data integrity verification enabler .....	71
Figure 58. Data Integrity Verification enabler ES1 (data integrity check).....	72
Figure 59. High-level diagram of the Distributed broker enabler .....	74
Figure 60. Distributed broker enabler ES1 (immutable metadata logging).....	75
Figure 61. High-level diagram of DLT-based FL enabler.....	76
Figure 62. DLT-based FL enabler ES1 (secure reputation) .....	77
Figure 63. High-level diagram of the Enablers manager.....	79
Figure 64. Enablers manager ES1 (list repositories) .....	80
Figure 65. Repositories view of the enablers manager.....	80
Figure 66. Form to add new repositories .....	80
Figure 67. Enablers manager ES2 (register repository).....	81
Figure 68. Update repository button.....	81
Figure 69. Enablers manager ES3 (update repositories) .....	81
Figure 70. Remove repository button.....	82
Figure 71. Enablers manager ES4 (delete repository) .....	82
Figure 72. Enablers manager ES5 (list enablers) .....	82
Figure 73. Enablers view of the enablers manager.....	83
Figure 74. Enablers manager ES6 (deploy enabler) .....	83
Figure 75. Form to add new enablers .....	83
Figure 76. Enablers manager ES7 (upgrade enabler) .....	84
Figure 77. Upgrade enabler button.....	84
Figure 78. Enablers manager ES8 (delete enabler) .....	84
Figure 79. Delete enabler button .....	85
Figure 80. High-level diagram of the Composite services manager .....	86
Figure 81. Composite services manager ES1 (list pipelines) .....	87
Figure 82. Composite services manager ES2 (deploy pipeline).....	87
Figure 83. Composite services manager ES3 (delete pipeline).....	88
Figure 84. High-level diagram of the Cluster and topology manager .....	90
Figure 85. Clusters and topology manager ES1 (list clusters).....	90
Figure 86. K8s clusters view of the clusters and topology manager .....	91
Figure 87. Clusters and topology manager ES2 (register cluster).....	91
Figure 88. Form to register new clusters .....	91
Figure 89. Clusters and topology manager ES3 (delete cluster) .....	92
Figure 90. Delete cluster button .....	92
Figure 91. Clusters and topology manager ES4 (depict topology).....	92
Figure 92. Topology view of the clusters and topology manager .....	93
Figure 93. Form to add new enabler in specific node.....	93

## List of tables

Table 1. Template table to report the general information of the enablers.....	15
Table 2. Template to report the components and implementation technologies of the enablers.....	15
Table 3. Template table to report the API of the enablers.....	16



Table 4. Template table to report the implementation status of the enablers .....	16
Table 5. General information of the Self-healing device enabler.....	16
Table 6. Components and implementation of the Self-healing device enabler .....	17
Table 7. Communication interfaces (API) of the Self-healing device enabler.....	17
Table 8. Implementation status of the Self-healing device enabler.....	19
Table 9. General information of the Resource provisioning enabler.....	19
Table 10. Components and implementation of the Resource provisioning enabler .....	20
Table 11. API of the Resource provisioning enabler.....	20
Table 12. Implementation status of Resource provisioning enabler.....	24
Table 13. General information of the Monitoring and notifying enabler .....	24
Table 14. Components and implementation of the Monitoring and notifying enabler.....	26
Table 15. API of the Monitoring and notifying enabler .....	26
Table 16. Implementation status of the Monitoring and notifying enabler .....	28
Table 17. General information of the Location processing enabler .....	29
Table 18. Components and implementation of the Location processing enabler .....	30
Table 19. API of the Location processing enabler .....	31
Table 20. Implementation status of the Location processing enabler.....	32
Table 21. General information of the Automated configuration enabler .....	32
Table 22. Components and implementation of the Automated configuration enabler .....	34
Table 23. Communication interfaces of the Automated configuration enabler.....	34
Table 24. Implementation status of the Automated configuration enabler.....	36
Table 25. General information of the FL Orchestrator.....	36
Table 26. Components and implementation of the FL Orchestrator .....	37
Table 27. API of the FL Orchestrator.....	37
Table 28. Implementation status of the FL Orchestrator .....	42
Table 29. General information of the FL Training collector enabler .....	42
Table 30. Components and implementation of the FL Training collector.....	43
Table 31. API of the FL Training collector .....	44
Table 32. Implementation status of the FL Training collector .....	45
Table 33. General information of the FL Repository .....	45
Table 34. Components and implementation of the FL Repository.....	46
Table 35. API of the FL Repository .....	47
Table 36. Implementation status of the FL Repository .....	51
Table 37. General information of the FL Local operations .....	51
Table 38. Components and implementation of the FL Local operations.....	53
Table 39. API of the FL Local operations .....	53
Table 40. Implementation status of the FL Local operations .....	55
Table 41. General information of the Identity manager enabler .....	56
Table 42. Components and implementation of the Identity manager enabler .....	57
Table 43. API of the Identity manager enabler .....	57
Table 44. Implementation status of the Identity manager enabler.....	58
Table 45. General information of the Authorisation enabler.....	58
Table 46. Components and implementation of the Authorisation enabler .....	60
Table 47. API of the Authorisation enabler.....	60
Table 48. Implementation status of the Authorization enabler.....	61
Table 49. General information of the Cybersecurity monitoring enabler.....	62
Table 50. Components and implementation of the Cybersecurity monitoring enabler .....	63
Table 51. API of the Cybersecurity monitoring enabler.....	64
Table 52. Implementation status of the Cybersecurity monitoring enabler.....	65
Table 53. General information of the Cybersecurity monitoring agent enabler .....	65
Table 54. Components and implementation of the Cybersecurity monitoring agent enabler .....	67
Table 55. Communication interfaces (TCP/UDP) of the Cybersecurity monitoring agent enabler .....	67
Table 56. Implementation status of the Cybersecurity monitoring agent enabler.....	68
Table 57. General information of the Logging and auditing enabler .....	68
Table 58. Components and implementation of the Logging and auditing enabler.....	69



Table 59. API of the Logging and auditing enabler .....	70
Table 60. Implementation status of the Logging and auditing enabler .....	70
Table 61. General information of the Logging and auditing enabler .....	71
Table 62. API of the Data integrity verification enabler .....	72
Table 63. API of the Data integrity verification enabler .....	72
Table 64. Implementation status of the Data integrity verification enabler .....	73
Table 65. General information of the Distributed broker enabler .....	73
Table 66. Components and implementation of the Distributed broker enabler .....	74
Table 67. API of the Distributed broker enabler .....	74
Table 68. Implementation status of the Distributed broker enabler .....	75
Table 69. General information of the DLT-based FL enabler .....	75
Table 70. Components and implementation of the DLT-based FL enabler .....	76
Table 71. API of the DLT-based FL enabler .....	77
Table 72. Implementation status of the DLT-based FL enabler .....	78
Table 73. General information of the Enablers manager .....	78
Table 74. Components and implementation of the Enablers manager .....	79
Table 75. API of the Enablers manager .....	79
Table 76. Implementation status of the Enablers manager .....	85
Table 77. General information of the Composite services manager .....	85
Table 78. Components and implementation of the Composite services manager .....	86
Table 79. API of the Composite services manager .....	87
Table 80. Implementation status of the Composite services manager .....	88
Table 81. General information of the Cluster and topology manager .....	89
Table 82. Components and implementation of the Cluster and topology manager .....	90
Table 83. API of the Cluster and topology manager .....	90
Table 84. Implementation status of the Cluster and topology manager .....	94

## List of acronyms

Acronym	Explanation
<b>AI</b>	Artificial Intelligence
<b>API</b>	Application Programming Interface
<b>AR</b>	Augmented Reality
<b>CHE</b>	Container Handling Equipment
<b>CPU</b>	Central Processing Unit
<b>CSV</b>	Comma Separated Value
<b>DLT</b>	Distributed Ledger Technology
<b>DoS</b>	Denial of Service
<b>FAIR</b>	Findable, Accessible, Interoperable, Reusable
<b>FML</b>	Federated Machine Learning
<b>FL</b>	Federated Learning
<b>FLS</b>	Federated Learning System
<b>FLTC</b>	Federated Learning Training Collector
<b>GPS</b>	Global Positioning System
<b>HW</b>	Hardware
<b>I/O</b>	Input/Output
<b>JSON</b>	JavaScript Object Notation
<b>JVM</b>	Java Virtual Machine
<b>K8s</b>	Kubernetes
<b>LTS</b>	Long-Term Storage
<b>LTSE</b>	Long-Term Storage Enabler
<b>MANO</b>	Management and Orchestration
<b>NGIoT</b>	Next Generation Internet of Things
<b>NN</b>	Neural Networks
<b>noSQL</b>	Not Only Structured Query Language
<b>MITM</b>	Man-In-The-Middle
<b>ML</b>	Machine Learning
<b>MQTT</b>	MQ Telemetry Transport
<b>OEM</b>	Original Equipment Manufacturer
<b>PAP</b>	Policy Administration Point
<b>PCM</b>	Powertrain Control Module
<b>PDP</b>	Policy Decision Point
<b>PEP</b>	Policy Enforcement Point

<b>PIP</b>	Policy Information Point
<b>REST</b>	Representational State Transfer
<b>RSSI</b>	Received Signal Strength Indicator
<b>RTG</b>	Rubber-Tyred Gantry (crane)
<b>SDN</b>	Software Defined Network
<b>SoTA</b>	State-of-the-Art
<b>SQL</b>	Structured Query Language
<b>SMC</b>	Secure Multi-Party Computation
<b>SR</b>	Semantic Repository
<b>TBD</b>	To Be Done/Defined
<b>TRL</b>	Technology Readiness Level
<b>TTL/SSL</b>	Time To Live/Secure Sockets Layer
<b>UC</b>	Use Case
<b>WP</b>	Work Package
<b>XACML</b>	eXtensible Access Control Markup Language
<b>XML</b>	Extensible Markup Language

# 1. About this document

The main objective of this document is to present the final design and specifications of the vertical enablers developed within the scope of ASSIST-IoT project. These enablers, along with horizontal enablers proposed in WP4, are the technological backbone of the project since they will enable the deployment of an ASSIST-IoT architecture.

Although this is the final iteration of a series of three deliverables devoted to the development and outcomes of enablers, the complementing WP5 deliverable series (which focuses on design and specification) has also been providing specification updates, and hence this report has been arranged as a final self-containing document built upon the content of its previous version (D5.3 Traversal Enablers Development Intermediate Version) and the complementary series (D5.4 Software Structure and Final Design). This final deliverable of WP5 includes all the information produced in an all-encompassing document, along with the developed artifacts.

## 1.1. Deliverable context

Keywords	Lead Editor
<b>Objectives</b>	<p><b><u>O3:</u></b> Definition and implementation of decentralised security and privacy exploiting DLT: Specification of DLT-based enablers in Security, Privacy and Trust vertical.</p> <p><b><u>O4:</u></b> Definition and implementation of smart distributed AI Enablers: Specification of Federated Machine Learning related enablers.</p> <p><b><u>Other:</u></b> Availability of preliminary software artifacts, specifically essential enablers:</p> <ul style="list-style-type: none"> <li>• Cybersecurity enablers: Identity manager enabler, Authorization enabler</li> <li>• Manageability enablers</li> </ul>
<b>Work plan</b>	<p>D5.5 takes input from:</p> <ul style="list-style-type: none"> <li>• T3.2 &amp; T3.3 (use cases and requirements): To be revisited and linked to the proposed enablers.</li> <li>• T3.5 (architecture): Depicts the design principles and high-level functionalities to consider and follow.</li> <li>• D5.1 (initial transversal enablers specification): Initial design of vertical enablers, and previous iteration of this deliverable.</li> <li>• D5.2, D5.3, D5.4 (transversal enablers development preliminary, intermediate and final versions): Included all-encompassing specifications from D5.1 and D5.4 and potential updated design-related content.</li> </ul> <p>D5.5 influences:</p> <ul style="list-style-type: none"> <li>• WP7 (pilots and validation): To later on materialise in pilot deployments.</li> <li>• WP8 (evaluation and assessment): To evaluate and assess results from testing within pilots.</li> </ul> <p>D5.4 must be in line with:</p> <ul style="list-style-type: none"> <li>• WP4 (core enablers): To define functional boundaries and interactions.</li> <li>• WP6 (testing, integration and support): To develop, test and deploy according to DevSecOps methodology.</li> </ul>
<b>Milestones</b>	This deliverable contributes to and is a key part of <i>MS6 – Software structure finished</i> , along with the previous content presented in D5.3. It also contributes to <i>MS7 – Integrated solution</i> .
<b>Deliverables</b>	Being a final all encompassing document built upon previous deliverables of this WP, it receives inputs from D5.4 – Final Transversal Enablers Specification and the deliverables related to the outcomes and artifacts, which included an update of the initial

	specifications (D5.2 & D5.3 – Transversal Enablers Development Preliminary/ Intermediate Version). It is also influenced by the main architecture document (D3.7) and linked to the project requirements and use cases (D3.3).
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 1.2. The rationale behind the structure

The document consists of five sections. The first section is an introduction that outlines the context of the document. The following section includes specifications of enablers divided into tasks they belong to. Each enabler description includes: general specification and features, structure, components and implementation technologies, enabler stories and implementation status at M30 of the project. Note that the above information finalizes enablers' definitions from the D5.4 and implementation and outcomes from D5.3. Section 4 points to the resources related to technical documentation, produced in the scope of WP6. Finally, the last section is devoted to conclusions.

## 1.3. Outcomes of the deliverable

The main outcome of the present deliverable is the final implementation and specification of the enablers envisioned within the scope of WP5. As updates (and enlargement) over the initial design (i.e., D5.1, D5.4) were also documented in the complementary series of deliverables of the WP (the one related to actual developments and outcomes, i.e., D5.2 & D5.3), most of the content remains valid and designs have not suffered many major changes. All vertical enablers have updated their respective tables with general information, pointing to the updated project's requirement and use cases. Enablers stories have been extended and updated.

## 1.4. Lessons learnt

There are no lessons learnt identified after the submission of D5.4. Lessons learnt included in previous WP5 deliverables are still valid.

## 1.5. Deviation and corrective actions

The Consortium formalised in D5.1 and D5.4, and materialised in D5.2, D5.3 and D5.5 the envisioned enablers. Since preparation of D5.4 no corrective actions were needed.

## 1.6. Version-specific notes

The Consortium formalised the initial design of the vertical enablers in M9. Being so early in the project, many relevant information required for their development was missing, and therefore these needs were covered in the subsequent deliverables of WP5, namely D5.2, D5.3 & D5.4, despite D5.2 and D5.4 should have focused just on releasing artifacts. Deliverables D5.3 and D5.4 were constructed as incremental documents containing only changes with respect to previous deliverables in respective series. Therefore:

- This version encompasses the final content of the enablers' template table from D5.1 and D5.4, and presents the final high-level schema, components and implementation technologies, communication endpoints and enabler stories from D5.2 and D5.3.
- This deliverable is the main document for enablers design and specifications, presenting the updated and current data from D5.3 and D5.4.

## 2. Introduction

Considering the ASSIST-IoT reference architecture, verticals represent properties or features that are present on different horizontal planes (i.e., Device and edge, Smart network and control, Data management and Application and services), either independently or requiring cooperation with them. In this architecture, vertical capabilities can be grouped in five types, namely: (i) Self-\*, (ii) Interoperability, (iii) Security, Privacy and Trust, (iv) Scalability, and (v) Manageability. Some of these capabilities become features of the system by design choices (e.g., using an underlying container orchestration framework such as Kubernetes, provides features that can be allocated under scalability and self-\*), while others require of dedicated artifacts (i.e., enablers) to be incorporated.

A total of 21 enablers have been formalised in the project as deemed necessities (or at least, encouraged) to be part of Next Generation IoT system realisations. Before implementing them, a set of initial specifications with expected features and candidate technologies were outlined in D5.1 and later on extended in D5.2, D5.3, D5.4.

It should be mentioned that enablers, despite been originally envisioned belonging to a specific Vertical, might provide features that can be allocated under the umbrella of other ones, as it can be seen in Figure 1. One clear example are the enablers related to Federated Learning (FL). These enablers work together to provide a framework that can be deployed to train models in different computing nodes locally and combine their results in a central node, hence contributing to scalability as processing effort is shared. Also, data do not travel through the network to the central node, just the trained models and the inferred weights, preserving data privacy (hence enablers can be assigned to Security, Privacy and Trust vertical as well).

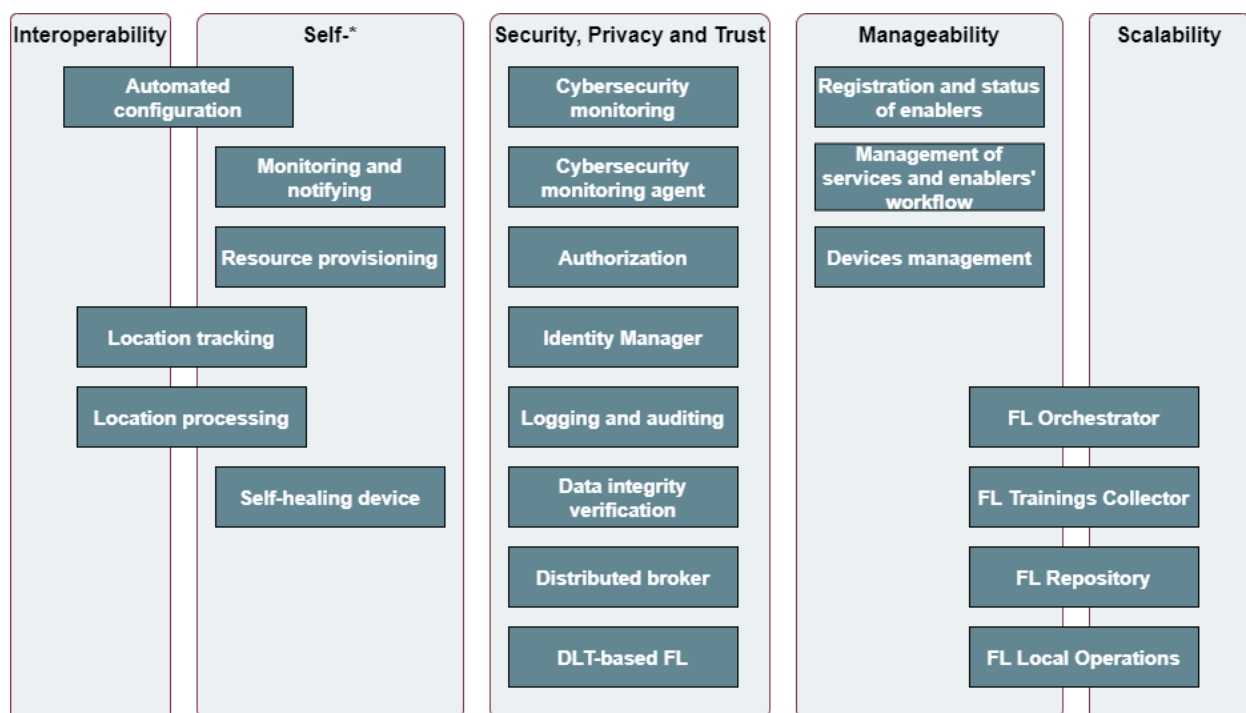


Figure 1. WP5 enablers distribution among verticals

## 3. Vertical enablers

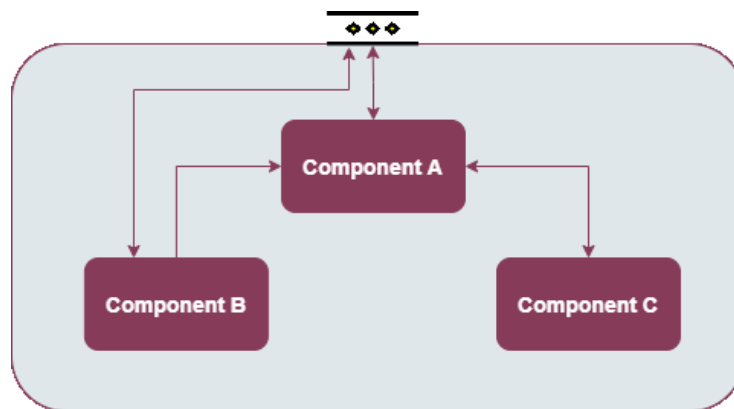
The specification of the enablers of the project are presented following a common structure. First, a table with **general information** about a given enabler, including its description, is attached (see Table 1). Then, a basic diagram presenting the **high-level structure** of its internal components is depicted, followed by a table listing the **components and their implementation technologies**, frameworks and/or programming languages leveraged to realise each of its components. Afterwards, the **endpoints** (generally APIs) that will be exposed to consume the enabler (either by end users, administrators or other enablers) are documented. Finally, a collection

of **Enabler Stories** (ES), being the cases in which an enabler will perform an action as a response to a given event or condition (initiated by a user, or not), will be presented.

*Table 1. Template table to report the general information of the enablers*

Enabler	Name of the enabler
<b>Id</b>	Short unique identifier/acronym
<b>Owner and support</b>	Lead and supporting beneficiaries
<b>Description and main functionalities</b>	Functional description of the enabler (description paragraph and bullet points for describing its features)
<b>Key features</b>	Bullet points for describing its features, focusing on advancements over SotA
<b>Vertical, related capabilities and features</b>	<p>Vertical to which this enabler belongs. Vertical groups together logically connected features and functionalities of a system, regardless of the plane on which they may be implemented. ASSIST-IoT defines 5 verticals:</p> <ul style="list-style-type: none"> <li>• Manageability</li> <li>• Scalability</li> <li>• Security, privacy and trust</li> <li>• Interoperability</li> <li>• Self-* (autonomy)</li> </ul> <p>Every vertical involves capabilities, a concretisation of a capability is called a feature.</p>
<b>Plane/s involved</b>	Horizontal plane or planes on which the enabler's features are delivered
<b>Requirements mapping</b>	List of the IDs of the requirements addressed or considered. Update 5.1 using D3.3 data
<b>Use case mapping</b>	List of the IDs of the use cases related to this enabler. Update 5.1 using D3.3 data
<b>Internal components</b>	List of the internal components of this enabler

## Components and technologies



*Figure 2. Example of high-level diagram*

The description and implementation technologies of each one of the components will be reported in a table as the following one:

*Table 2. Template to report the components and implementation technologies of the enablers*

Component	Description	Technology/s
	It is in charge of / Id deals with / It provides ...	

## Communication interface/s

The third section reports the communication interfaces. Generally, this refers to API endpoints, following the table below. In case that other interfaces are present (e.g., MQTT connections, VPN enabler via dedicated TCP/UDP connection, etc.) they will be reported accordingly.



Table 3. Template table to report the API of the enablers

Method	Endpoint	Description
GET/POST/PUT/DELETE		

### Enabler stories

#### STEPS 1-2:

#### STEP 3:

#### STEP 4:

...

### Additional information

The fifth and last section reports additional information related to documentation, encapsulation readiness, integration with other enablers of the project, and features that could be extended in future releases.

Table 4. Template table to report the implementation status of the enablers

Category	Status
Link to ReadtheDocs	Link to documentation
Potential features	Additional features that could be added/extended in the future, now that more knowledge about the enabler is available
Encapsulation readiness	Row to explain if an enabler is an encapsulation exception, and why, or if it has a full functional Helm package ready
Integration with other enablers	Expresses if the enabler require others to offer all its functions, or if it works in a complete standalone fashion

## 3.1. Self-\* enablers

### 3.1.1. Self-healing device enabler

#### 3.1.1.1. General specifications and features

Table 5. General information of the Self-healing device enabler

Enabler	Self-healing device enabler
Id	T51E1
Owner and support	PRODEVELOP
Description and main functionalities	This enabler aims at providing to IoT devices with the capabilities of actively attempting to recover themselves from abnormal states, mainly divided in two categories: network security (lack of connection, jamming), and long-term resources (HW's end-of-life, HW unsupported capabilities), based on a pre-established routines schedule.
Key features	Self-* (Autonomy)
Vertical, related capabilities and features	Self-*
Plane/s involved	Device and edge plane; Smart network and control plane
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-5: Local processing capabilities</li> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-21: Reduction of computing demands for AI training</li> <li>• R-P2-16: Device reliability and durability</li> <li>• R-P3A-9: Edge intelligence</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P1-2: CHE location tracking</li> <li>• UC-P2-1: Worker's health and safety assurance</li> </ul>

Enabler	Self-healing device enabler
	<ul style="list-style-type: none"> <li>UC-P3A-2: Vehicle non-conformance causes identification</li> </ul>
Internal components	Self-detector, Self-Monitor, Self-remediator node-red nodes FlaskAPI for configuring monitoring thresholds

### 3.1.1.2. Structure, components, and implementation technologies

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.

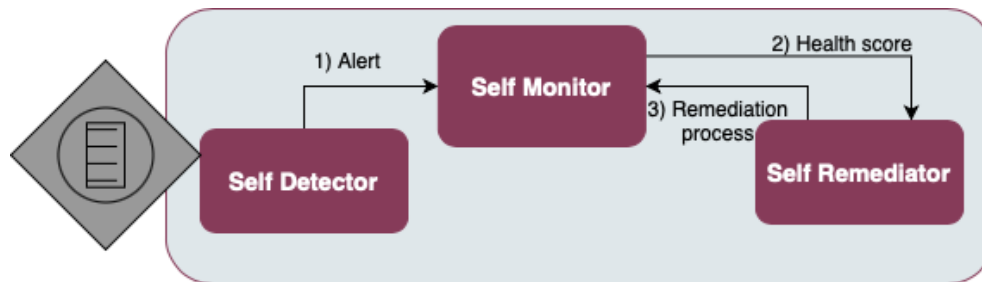


Figure 3. High-level diagram of the Self-healing device enabler

Specifically, a description of each one of the components depicted is provided in the table below, along with the technologies used for implementing them:

Table 6. Components and implementation of the Self-healing device enabler

Component	Description	Technology/s
Self-detector	The goal of this component is to collect information from the IoT device.	Node-RED, JavaScript, Unix commands
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 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 a proper treatment. If after the remediation, the device is not back to its normal state, the component is self-triggered to select another remediation process from the list.	

### 3.1.1.3. Communication interfaces

Table 7. Communication interfaces (API) of the Self-healing device enabler

Method	Endpoint	Description
POST	/cpuusage?threshold=XX	This endpoint changes the maximum threshold of CPU usage to the XX value defined by the user
POST	/ramusage?threshold=XX	This endpoint changes the maximum threshold of RAM usage to the XX value defined by the user
POST	/network?IP=XX	This endpoint changes the IP address over which the service should ping to check network availability

### 3.1.1.4. Enabler stories

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

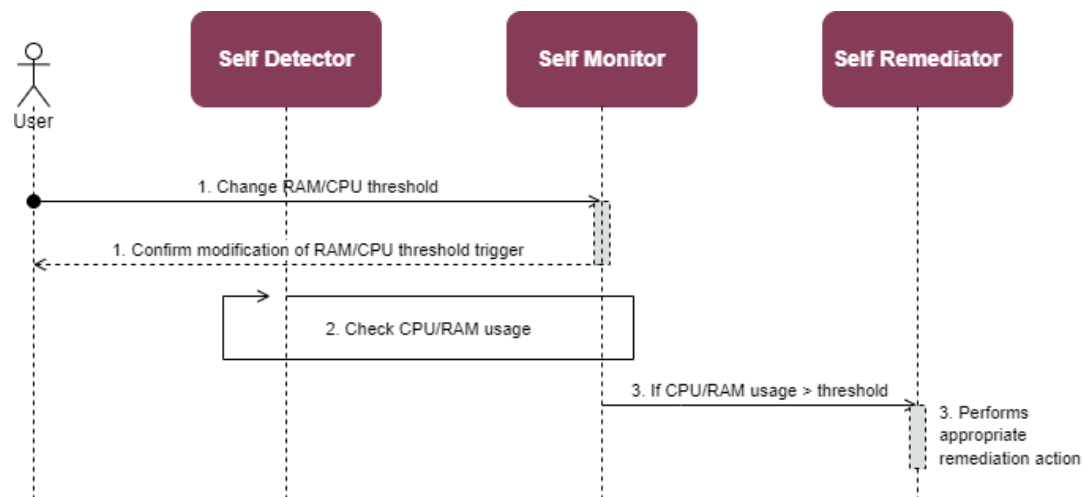


Figure 4. Self-healing device enabler ES1 (CPU/RAM usage monitoring and threshold update)

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

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

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

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

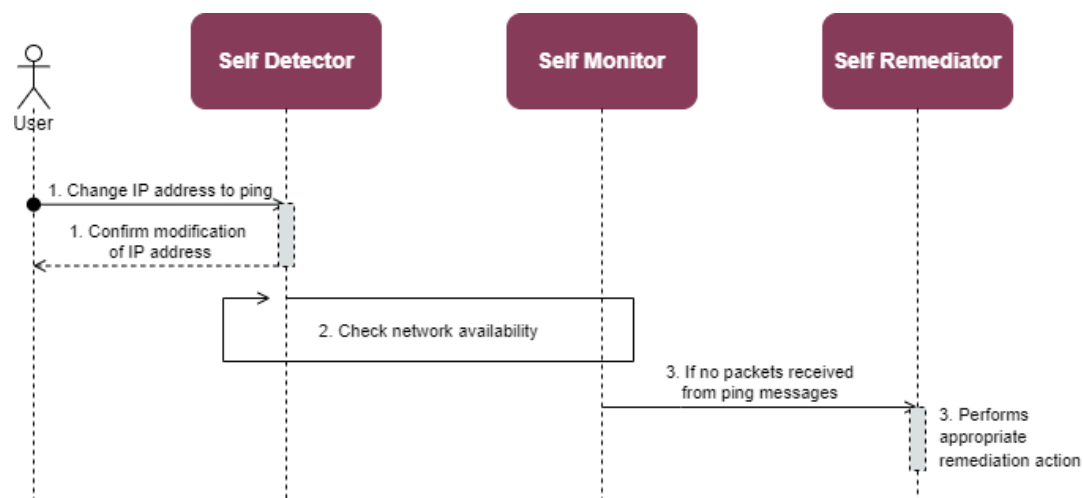


Figure 5. Self-healing device enabler ES2 (RAM usage monitoring and threshold update)

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

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

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

### 3.1.1.5. Implementation information

Table 8. Implementation status of the Self-healing device enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/self_healing_device_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/self_healing_device_enabler.html</a>
Potential features	Ensure ASSIST-IoT devices and platform reliability.
Encapsulation readiness	After several failure tests (the functionalities of the enabler were not reaching the host OS environment if it was encapsulated), the self-healing device enabler became a non-encapsulated exception of the ASSIST-IoT enablers list. It can be downloaded as a source file from the public GitLab's repository.
Integration with other enablers	This enabler does not require any integration with other enablers of the project.

## 3.1.2. Resource provisioning enabler

### 3.1.2.1. General specifications and features

Table 9. General information of the Resource provisioning enabler

Enabler	Resource provisioning enabler
Id	T51E2
Owner and support	UPV
Description and main functionalities	Working on edge deployments, where resources are not as large as in the cloud, it is unfeasible to set a static resource projection to each node. This is due to the difference in the use of these resources depending on the workload at the time the task is performed, being dependent on several factors. This enabler aims to adapt the auto-scaling of nodes and clusters more dynamically, achieving optimal use in relation to resource utilisation and general operation.
Key features	Self-* (Autonomy)
Vertical, related capabilities and features	Self-*
Plane/s involved	All Planes – it monitors and adapts the resources of enablers from all planes.
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-9: Workload placement</li> <li>• R-C-16: Resource monitoring</li> <li>• R-C-18: Support for autonomous processing</li> <li>• R-C-19: Support for self-aware systems</li> </ul>
Use case mapping	This enabler is agnostic to any use case, and can be applied to all use cases of the project.
Internal components	API, Pod Resources Controller (PRC), Training module (TM), Inference module (IM), History data – MySQL Database, Predicted data – MySQL Database

### 3.1.2.2. Structure, components and implementation technologies

When the administrator user enables the resources provisioning controller enabler it automatically starts working. It accesses the metrics and stores them in its internal database, performs the deep learning process and infers to create the horizontal objects pod autoscalers dynamically. All this with pre-set values in the initial configuration. Its structure is presented in Figure 6, consisting of the following elements:

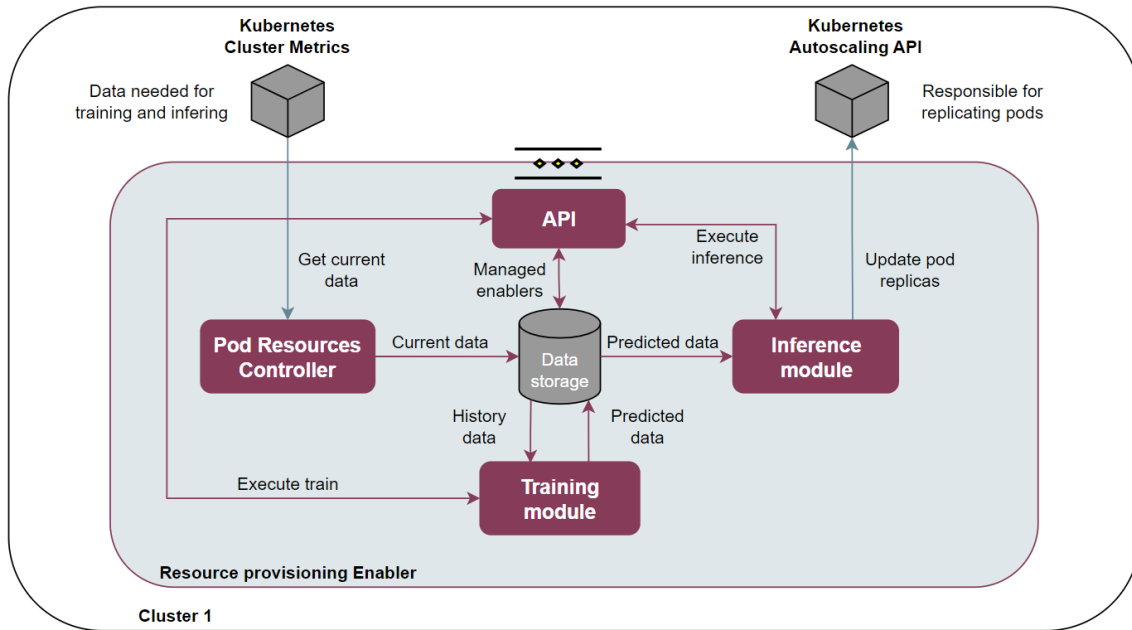


Figure 6. High-level diagram of the Resource provisioning enabler

Particularly, the Resource provisioning enabler is composed of five main components:

Table 10. Components and implementation of the Resource provisioning enabler

Component	Description	Technology/s
<b>API</b>	Contains the logic necessary to make GET and POST calls to intervene with the system behaviour, change default values or collect information.	Python, Flask
<b>Pod Resources Controller (PRC)</b>	Performs the collection of metrics and is responsible for storing the values in 15-minute intervals.	Python, Flask
<b>Inference Module (IM)</b>	Adds logic to the data in the future database and generates the inference process. Creates or replaces the horizontal pod autoscaler objects. Changes the previous values to the new ones based on the results obtained.	Python, Flask
<b>Training Module (TM)</b>	Collects the raw data from the history databases and converts it to the format needed for the deep learning process. Executes the data predictions and stores them in a new database.	Python, Flask, Neural Prophet, PyTorch
<b>Data storage</b>	Contains the history and predicted data of all components of each active enabler. Also, contains the relative information about if each enabler are activated to infer or not.	MySQL

### 3.1.2.3. Communication interfaces

Table 11. API of the Resource provisioning enabler

Method	Endpoint	Description
GET/POST	/v1/enablers	Create, update or get the enablers and components active and managed by the resource provisioning enabler.
GET/POST	/v1/train-values	Endpoint in charge of defining IPSec proposals that can be used for tunnels in an overlay.
GET	/v1/train	Defines a traffic hub in an overlay. Requires certificate and kubeconfig file to be able to manage it.
GET	/v1/inference	Defines the overlay IP range used for the edge clusters.
GET	/v1/version	Defines an edge cluster location (with WAN Acceleration enabler). Among other input, it required kubeconfig file and certificate information.
GET	/v1/health	Defines a connection between a hub and an edge cluster.
GET	/v1/api-export	Export API Swagger.

### 3.1.2.4. Enabler stories

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

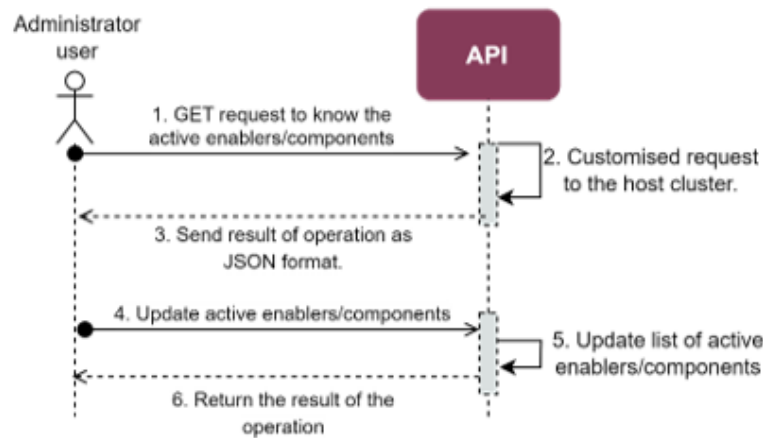


Figure 7. Resource provisioning enabler ES1 (get enablers and components inferred)

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

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

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

**STEP 4:** The user can resend a POST request with updating the active enablers and components of each enabler to be inferred.

**STEP 5:** The enabler updates the list of enablers/components active to be inferred in order to the POST request received.

**STEP 6:** The enabler returns the result of the operation.

The **second enabler story** is related to updating enablers and components to perform inference. The diagram and related steps are the following:

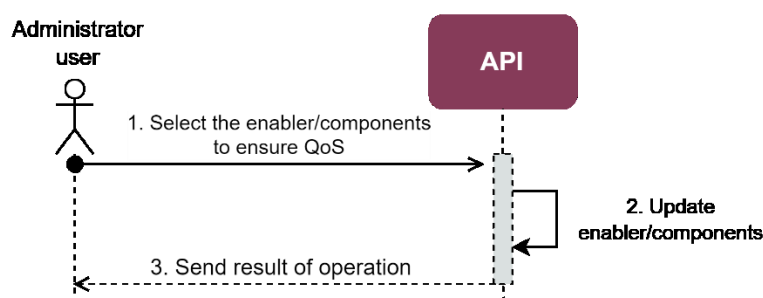


Figure 8. Resource provisioning enabler ES2 (update enablers and components to get inferred)

**STEP 1:** User send the updated list with the enablers and components selected to be inferred to ensure QoS.

**STEP 2:** API updates the enabler and components chosen to be inferred.

**STEP 3:** API send to user the result of the operation.

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

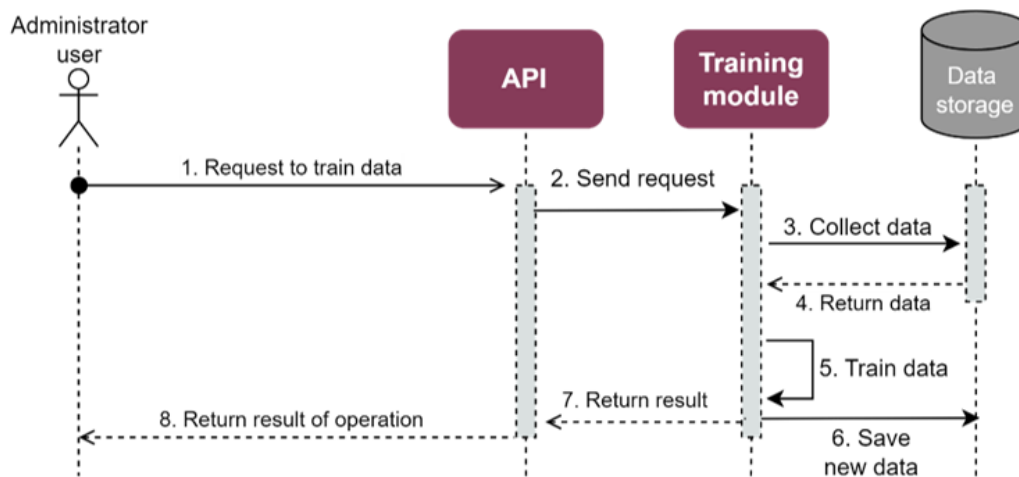


Figure 9. Resource provisioning enabler ES3 (perform training)

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

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

**STEP 3:** The train module component collects the necessary data from the data storage.

**STEP 4:** The data storage returns the raw data.

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

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

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

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

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

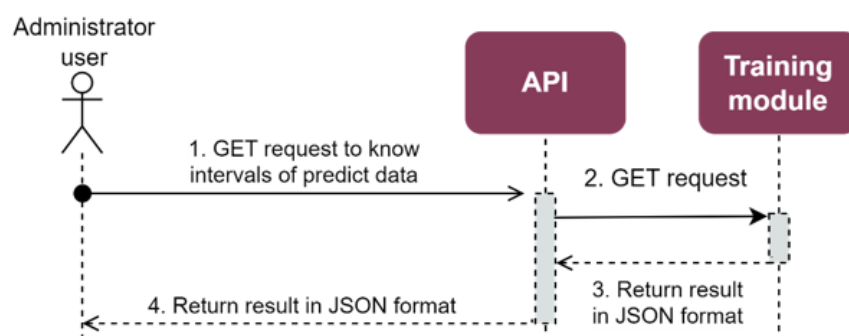


Figure 10. Resource provisioning enabler ES4 (get interval range for training)

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

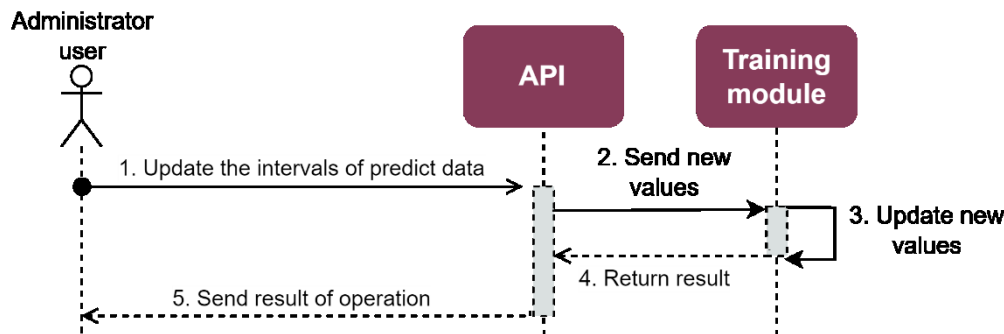
**STEP 2:** The enabler receives the request and redirects to the training module.

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

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



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



*Figure 11. Resource provisioning enabler ES5 (update data interval for training)*

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

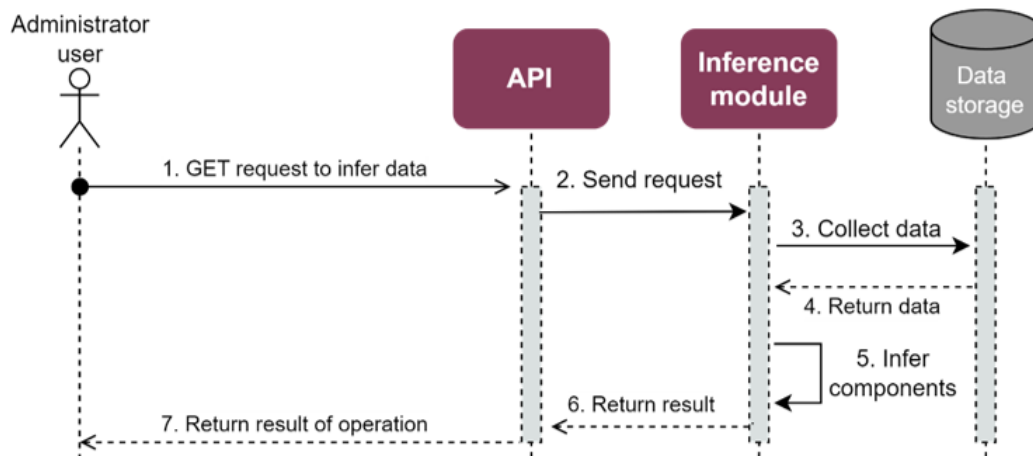
**STEP 2:** The enabler receives the values and sends them to the training module.

**STEP 3:** The training module updates the values.

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

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

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



*Figure 12. Resource provisioning enabler ES6 (perform inference)*

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

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

**STEP 3:** The inference module component collects the data needed for training from the data storage.

**STEP 4:** The data storage returns the raw data.

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

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

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

### 3.1.2.5. Implementation information

*Table 12. Implementation status of Resource provisioning enabler*

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/resource_provisioning_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/resource_provisioning_enabler.html</a>
Potential features	Ensure QoS and autoscaling.
Encapsulation readiness	All components are encapsulated in a helm chart ready to deploy the enabler.
Integration with other enablers	This enabler is ready to integrate with other enablers correctly configured.

### 3.1.3. Monitoring and notifying enabler

#### 3.1.3.1. General specifications and features

*Table 13. General information of the Monitoring and notifying enabler*

Enabler	Monitoring and notifying enabler
Id	T51E4
Owner and support	CERTH
Description and main functionalities	<p>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.</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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).</li> </ul>
Key features	Self-* (Autonomy)
Vertical, related capabilities and features	Self-*
Plane/s involved	<ul style="list-style-type: none"> <li>• Device and Edge Plane – Devices (smart, IoT) are part of the use cases for various actions (e.g. monitor data).</li> <li>• Application and Services Plane – Monitoring will have to include an interface for an end user to interact. Notification is similar to the monitoring.</li> <li>• Data Management Plane – Monitoring and notification is based on data (real-time with streaming, historical as reports). Depending on the case, the need for streaming or historical data may arise.</li> <li>• Communication (Network with SDN)</li> </ul>
Requirements mapping	<ul style="list-style-type: none"> <li>• R-P1-1 (CHE location services)</li> <li>• R-P1-2 (CHE location availability)</li> <li>• R-P1-10 (CHE identification)</li> <li>• R-P2-1 (Personal location tracking)</li> <li>• R-P2-3 (Smart wristband for construction workers)</li> <li>• R-P2-2 (Construction plant location tracking)</li> </ul>

Enabler	Monitoring and notifying enabler
	<ul style="list-style-type: none"> <li>• R-P2-4 (Continuous authentication for wristband)</li> <li>• R-P2-7 (Monitoring the weather conditions at the construction site)</li> <li>• R-P2-10 (Motion Pattern Monitoring and Analysis)</li> <li>• R-P2-12 (Alerts and notifications minimization)</li> <li>• R-P3A-10 (Vehicle Dashboard Notifications)</li> <li>• R-P3B-19 (Critical Damage Identification Time)</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P1-1 (Asset location management)</li> <li>• UC-P1-2 (CHE location tracking)</li> <li>• UC-P1-4 (RTG-truck identification and authentication)</li> <li>• UC-P1-5 - RTG-Truck alignment; RTGs and Trucks needs to notify each other about their positions</li> <li>• UC-P2-1 - Workers' health and safety assurance; It is required in this UC that after breaching some threshold values we need to send notifications across various components of the system (for example notifying OSH manager or to some components that takes action when unauthorized access was detected)</li> <li>• UC-P2-2 - Geofencing boundaries enforcement; When you breach fence the notification will be sent</li> <li>• UC-P2-3 - Danger zone restrictions enforcement; similar to the above</li> <li>• UC-P2-4 (Construction site access control)</li> <li>• UC-P2-5 (Near-miss fall from height detection)</li> <li>• UC-P2-7 (Health and safety inspection support)</li> <li>• UC-P3A-1 (Fleet in-service emissions verification)</li> <li>• UC-P3A-2 (Vehicle's non-conformance causes identification)</li> <li>• UC-P3B-1 (Vehicle's exterior condition documentation)</li> </ul>
Internal components	Communication Interface, Database, Message queue, Registry, Module for implementing the logic

### 3.1.3.2. Structure, components and implementation technologies

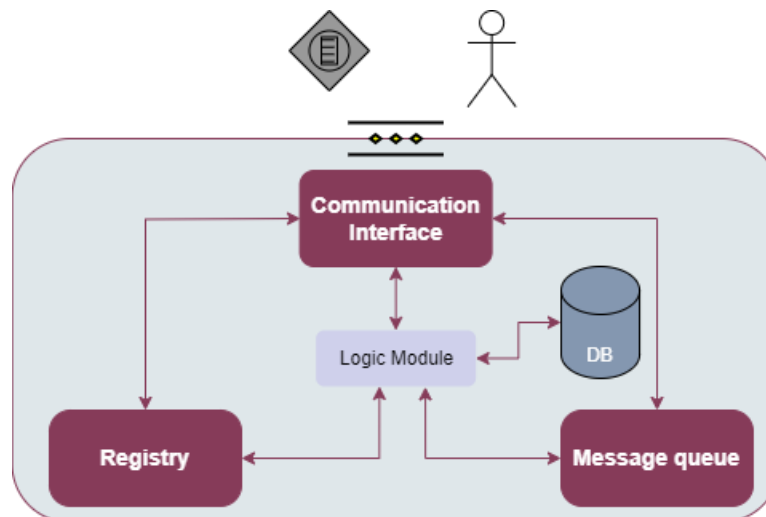


Figure 13. High-level diagram of the Monitoring and notifying enabler

Specifically, a description of each one of the components depicted is provided in the table below, along with the technologies used for implementing them:

Table 14. Components and implementation of the Monitoring and notifying enabler

Component	Description	Technology/s
<b>Registry</b>	The registry serves the role of saving the list of devices and gateways that are registered in each use case. It is the place that the Logic Module searches to query and check the health of devices	Java
<b>Database</b>	Database is the component that enables to store internally critical events.	MongoDB
<b>Communication Interface</b>	The API is the gateway to the internal components of the enabler. It allows devices to be subscribed to a topic in order to send data, and it allows a user to interact with the registry and the message queue through http requests or topic subscriptions.	Kafka, NodeJS, ExpressJS
<b>Monitoring and notifying module</b>	This is the component is the all the logic is implemented. It is responsible for creating the notifications, sending the critical data to the database, registering new devices, updating their statuses and passing the data to the message queue.	Kafka, Java
<b>Message queue</b>	Message queue is the component that the end user can visually monitor the health status of devices.	Kafka, Java

### 3.1.3.3. Communication interfaces

Table 15. API of the Monitoring and notifying enabler

Method	Endpoint	Description
GET	/ping/device	Ping all devices in registry.
GET	/ping/gateway	Ping all gateways in registry.
GET	/endpoints/device	Get all devices in json array format.
GET	/endpoints/gateway	Get all gateways in the registry in json array format.
GET	/endpoints/alive/device	Get all alive devices in the registry in json array format.
GET	/endpoints/alive/gateway	Get all alive gateways in the registry in json array format.
POST	/endpoints	Add new device/gateway in the registry.
DELETE	/endpoints/device/\$device_id	Remove a device from the registry.
DELETE	/endpoints/gateway/\$gateway_id	Remove a gateway from the registry.

### 3.1.3.4. Enabler stories

The first enabler story involves an IoT device which stops receiving data from its integrated sensor.

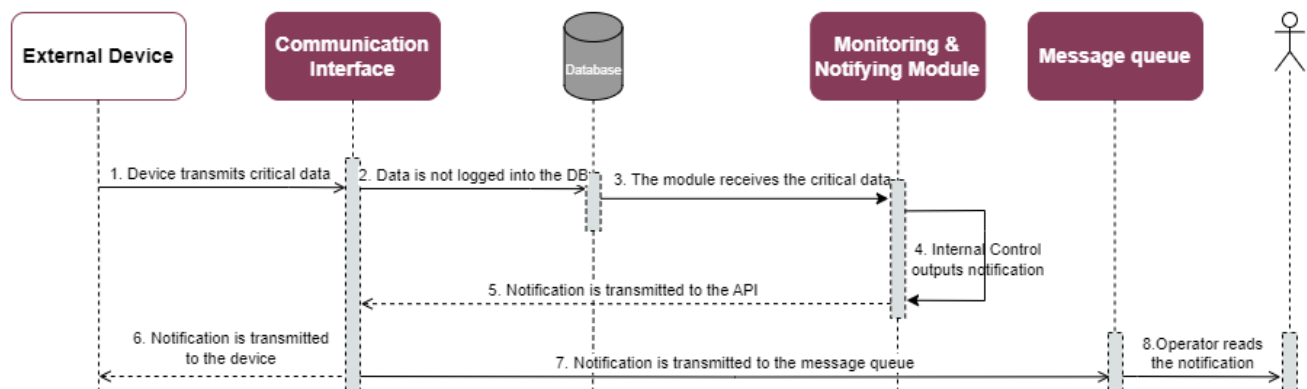


Figure 14. Monitoring and notifying enabler ES1 (IoT device not receiving data)

**STEP 1:** The communication 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 clear 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 occurrence 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 occurrence and has to act accordingly.

The **second enabler story** involves an **IoT edge device**, which transmits critical data (e.g., measurement exceeding a predefined threshold).

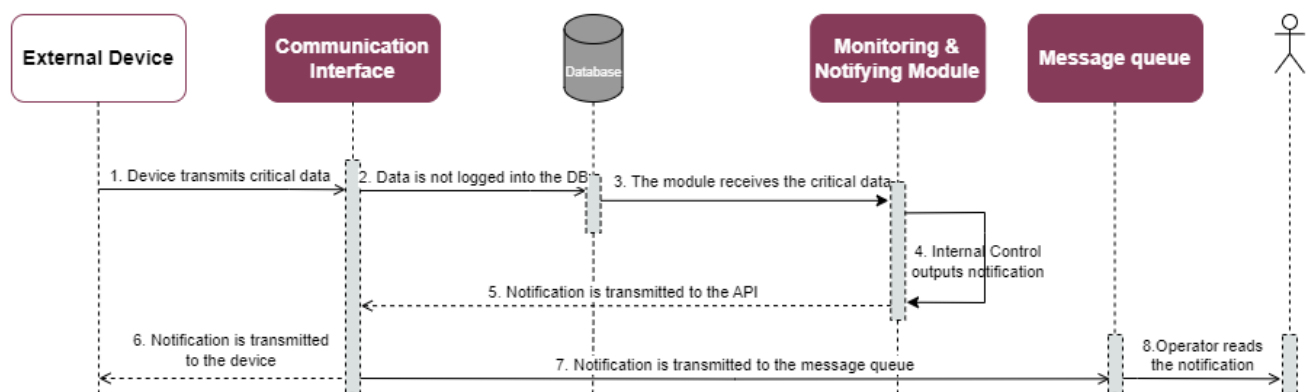


Figure 15. Monitoring and notifying enabler ES2

**STEP 1:** The device transmits critical data.

**STEP 2:** Critical data is logged into the database.

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

**STEP 4:** Since the monitoring module identifies that the data it receives is critical, it has to notify both the device and the operator for the incident.

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

**STEP 6:** The notification is transmitted to the device, indicating that the data is indeed critical.

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

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

The **third enabler story** is related to an end user **querying historical data** and it takes place under the assumption that there is constant monitoring of the devices' metrics:

**STEP 1:** An end user requests a report on the historical data gathered.

**STEP 2:** The request is passed to the database.

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

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

**STEP 5:** The report is sent to the API.

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

**STEP 7:** User reads the report he requested.

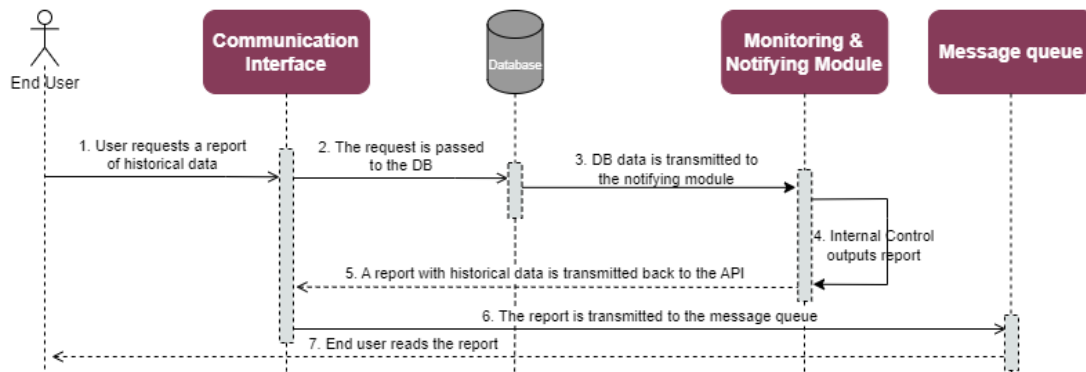


Figure 16. Monitoring and notifying enabler ES3

The **fourth enabler story** involves a user that wants to **check the health of devices or gateways** in order to see if they are connected and communicating with each other.

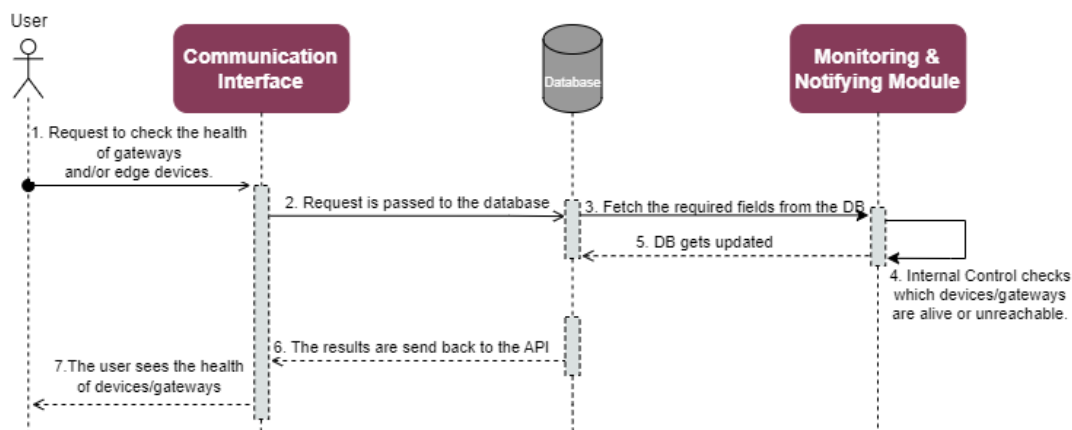


Figure 17. Monitoring and notifying enabler ES4 (checking health of devices or gateways)

**STEP 1:** The user sends the request to the enabler’s API to check the health of devices and/or gateways.

**STEP 2:** The request is passed to the database.

**STEP 3:** The database returns the required fields to the monitoring and notifying module which implements the logic.

**STEP 4:** The module’s internal control system checks which devices and/or gateways are alive or unreachable.

**STEP 5:** The result is send to the DB and it gets updated.

**STEP 6:** The results are sent back to the API for user consumption.

**STEP 7:** The user checks the results and acts accordingly.

### 3.1.3.5. Implementation information

Table 16. Implementation status of the Monitoring and notifying enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/monitoring_and_notifying_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/monitoring_and_notifying_enabler.html</a>
Potential features	-
Encapsulation readiness	The original version of the enabler is fully encapsulated with helm charts ready. Pending to encapsulate the recent component that is described in enabler story four.
Integration with other enablers	It works in a standalone fashion. It has been integrated with Logging & auditing and Distributed broker enabler for the project purposes.

### 3.1.4. Location tracking enabler

The location tracker enabler consists of a location tracking by means of Ultra Wide Band (UWB) technology which is quite new to the market and thus not mature yet in sense of standardization or available hardware.

#### 3.1.4.1. General specifications and features

The current system used in Assist-IoT is based on a Qorvo module DWM1001C. The system developed meets the following specifications:

<b>Accuracy</b>	10-20 cm in a 2D plane (X, Y axes) The accuracy in the third dimension (Z-plane) is slightly lower around 40-50cm.
<b>Range</b>	The range depends greatly on the surroundings. In an open field or open structure the range coverage is around 25m. However, as we have seen during Pilot 2 once the UWB is placed in a building with many smaller rooms and thus walls between tag and anchor the range drops. The DWM1001C system uses only 1 UWB band which limits the ability to penetrate walls. Currently, the system is able to penetrate 1 wall. When the tag moves to a location with 2 walls in between the accuracy drops and is not useful anymore.
<b>UWB band</b>	Channel 5
<b>Refresh rate</b>	Typical 1 Hz to cover a normal size network, however speed can be adapted theoretically up to 1000Hz. However, with current hardware this is not feasible. For the current setup we managed to push the speed up to 4Hz.
<b>Supply</b>	The tag is powered by a battery cell. The anchors are preferably powered by a continuous source but can be adapted to be battery powered.

#### 3.1.4.2. Structure, components and implementation technologies

The UWB system is build around the Qorvo MDEK 1001 development kit. The software provided by Qorvo is very limited and far from operational. For the pilots we enhanced the software allowing both reliable localization as well as communication via UWB technology.

#### 3.1.4.3. Communication interface

The UWB system itself communicates via UWB bursts according protocol IEEE 802.15.4 which allows small packages of data to be sent together with the localization burst. The UWB master anchor communicates via USB with the GWEN where the UWB localization software runs in a Docker container.

#### 3.1.4.4. Implementation information

For implementation a lab setup was constructed. In the setup each tag and anchor had to be reachable via the USB port for (re)programming purpose. Hence a network was build to both test the UWB network as well as being able to (re)program the anchors and tags. The main issue faced was getting the timing of the system properly such that more than 1 hop could be made. This required quite some restructuring and updating of the original code from Qorvo.

### 3.1.5. Location processing enabler

#### 3.1.5.1. General specifications and features

*Table 17. General information of the Location processing enabler*

Enabler	Location processing enabler
<b>Id</b>	T51E3-B
<b>Owner and support</b>	SRIPAS
<b>Description and main functionalities</b>	The Location Processing enabler provides highly configurable and flexible geofencing capabilities based on location data. It runs user-defined queries against the data storage.



Enabler	Location processing enabler
	The enabler handles data updates and queries using both the HTTP-based request-response and streaming approach.
Key features	Geolocation data transformations, storage, and retrieval; MQTT streaming interface; HTTP request-response interface
Vertical, related capabilities and features	Self-*
Plane/s involved	Data management plane – it provides location awareness through location data.
Requirements mapping	<ul style="list-style-type: none"> <li>• R-P1-1: CHE location services</li> <li>• R-P2-1: Personal location tracking</li> <li>• R-P2-2: Construction plant location tracking</li> <li>• R-P2-11: Geofencing</li> <li>• R-C-5: Local Processing Capabilities</li> <li>• R-C-18: Support for autonomous processing</li> <li>• R-C-19: Support for self-aware systems</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P2-1: Workers' health and safety assurance</li> <li>• UC-P2-2: Geofencing boundaries enforcement</li> <li>• UC-P2-4: Construction site access control</li> </ul>
Internal components	Application, Database

### 3.1.5.2. Structure, components and implementation technologies

The enabler delivers features provided by a geolocation storage. It consists of two components - an application and a database. The application is written with the Akka framework to provide scalability and durability. It runs user-defined SQL queries against the database. The queries can be parametrized using the specialized syntax. In effect, the enabler is highly customizable for different scenarios. The incoming data is collected from input streams or HTTP requests; it allows for streaming the query results. The transferred data is in JSON format. The behaviour of the application is configurable through an HTTP interface. The application streaming capabilities are compatible with the MQTT protocol. The database is shipped with the PostGIS extension. It stores the geolocation data and the application configuration.

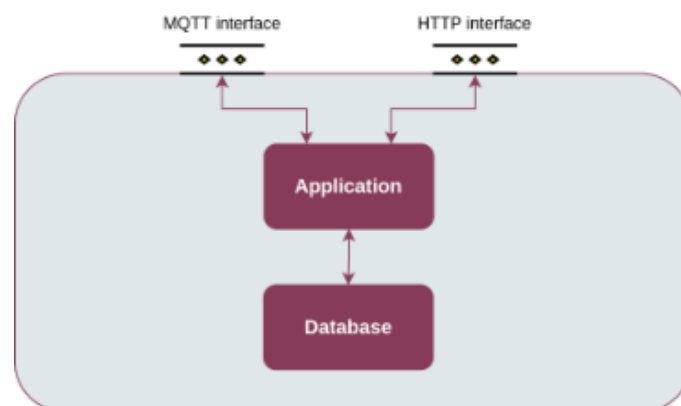


Figure 18. High-level diagram of the Location processing enabler

Specifically, a description of each one of the components depicted is provided in the table below, along with the technologies used for implementing them:

Table 18. Components and implementation of the Location processing enabler

Component	Description	Technology/s
Application	It contains logic to manage and execute user-defined geolocation queries.	Scala, Akka
Database	It stores the geolocation data and the application configuration.	Postgres, PostGIS

### 3.1.5.3. Communication interfaces

Table 19. API of the Location processing enabler

Method	Endpoint	Description
GET	/v1/queries	Returns all queries.
GET	/v1/queries/<name>	Returns a specific query.
POST	/v1/queries	Creates a new query with specified configuration.
POST	/v1/queries/<name>/input	Sends input data to a specific query.
PUT	/v1/queries/<name>	Updates a query.
DELETE	/v1/queries/<name>	Deletes a query.
GET	/v1/queries	Returns all queries.

### 3.1.5.4. Enabler stories

The **first enabler story** is related to the **configuration of a query** via the HTTP interface.

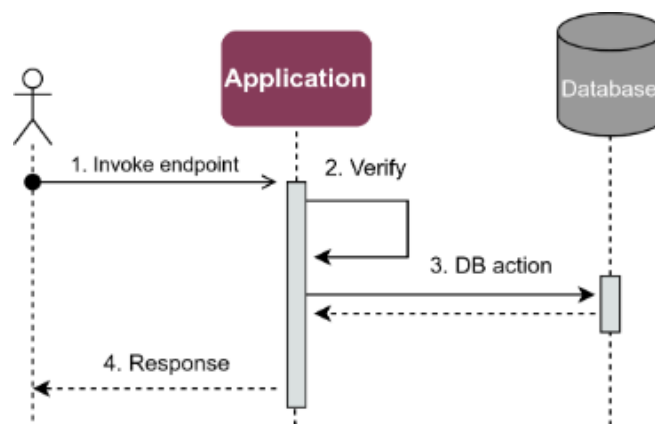


Figure 19. Location processing enabler ES1 (query configuration)

**STEP 1:** The client invokes the /v1/queries API endpoint, passing the configuration details.

**STEP 2:** The application verifies the request data.

**STEP 3:** The application communicates with the database that stores query configurations to perform the requested action.

**STEP 4:** The client receives the confirmation message.

The **second enabler story** involves **passing the input data to a running query** via HTTP interface.

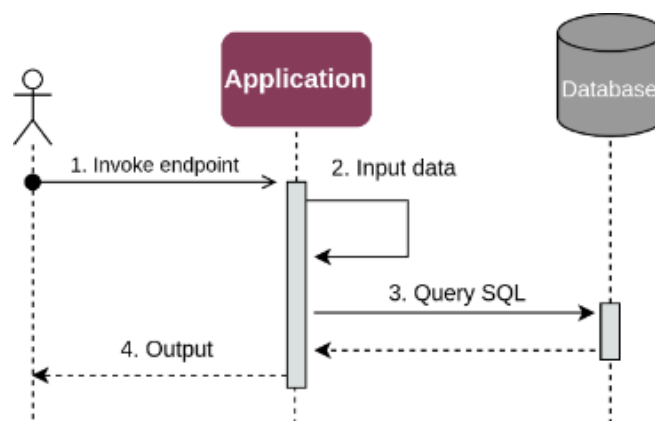


Figure 20. Location processing enabler ES2 (running query input)

**STEP 1:** The client invokes the `/v1/queries/<name>/input` endpoint and passes the input data in the request body.

**STEP 2:** The application communicates with the called query and inputs the received data.

**STEP 3:** Query runs an SQL statement.

**STEP 4:** The application responds with the query output data.

### 3.1.5.5. Implementation information

Table 20. Implementation status of the Location processing enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/location_process_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/location_process_enabler.html</a>
Potential features	Improved scalability mechanisms.
Encapsulation readiness	All components are encapsulated in a Helm chart.
Integration with other enablers	It is used closely with the Location tracking enabler; however, it can operate in a standalone fashion.

### 3.1.6. Automated configuration enabler

#### 3.1.6.1. General specifications and features

Table 21. General information of the Automated configuration enabler

Enabler	Automated configuration enabler
Id	T51E5
Owner and support	SRIPAS
Description and main functionalities	<p>This enabler aims to allow users to abstractly define requirements for functionalities and then to check whether those requirements are met by registered resources. The enabler can also automatically react to external actions based on predefined rules.</p> <ul style="list-style-type: none"> <li>• <i>Administrator</i> can define <i>configuration</i> of a system. The <i>configuration</i> describes what <i>resources</i> are required by a specific <i>functionality</i>.</li> <li>• Administrator can define <i>reactions</i> - rules on how the system should behave when a specific action has happened. <i>Reactions</i> can modify existing <i>configuration</i> and emit notifications.</li> <li>• <i>Self-Configurator</i> can autonomically decide which <i>functionalities</i> will be kept in the event of limited available <i>resources</i>.</li> </ul> <p><i>Resources</i> can notify the Automated configuration enabler about going live and going down.</p>
Key features	Automated configuration management; Kafka streaming interface; HTTP request-response interface
Vertical, related capabilities and features	Interoperability, Self-* (autonomy)
Plane/s involved	<ul style="list-style-type: none"> <li>• Device and edge plane – the enabler will be used for configuring devices</li> </ul> <p>Data management plane – the automated configuration enabler will provide/support interoperability mechanisms for heterogeneous environments</p>
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-2: Data governance</li> <li>• R-C-5: Local processing capabilities</li> <li>• R-C-6: Data persistence and trust</li> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-18: Support for autonomous processing</li> </ul>

Enabler	Automated configuration enabler
	<ul style="list-style-type: none"> <li>• R-C-19: Support for self-aware systems</li> <li>• R-C-28: Distributed Configuration</li> <li>• R-P1-16: Open/accessible remote capabilities</li> <li>• R-P3A-5: Data Storage</li> <li>• R-P3A-12: Edge connectivity</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P1-1: Asset location management</li> <li>• UC-P2-1: Worker's health and safety assurance</li> <li>• UC-P2-6: Safe navigation instructions</li> <li>• UC-P3A-3: Sending new configuration to PCM</li> </ul>
Internal components	Self-configurator, Eventstore, Connector

### 3.1.6.2. Structure, components and implementation technologies

The Automated configuration enabler (AC) keeps heterogeneous devices and services synchronised with their configurations. User can update configuration and define fallback configurations in case of errors.

Internally, the AC represents system configuration as a (possibly disconnected) acyclic directed graph (DAG) with two kinds of vertices: resource and functionality, and edges representing the relation “requires to function”. Edges can exist between pairs: (functionality, resource) and (functionality, functionality). Additionally, different “labels” can be associated with each vertex, allowing to categorise and group vertices, without changing the overall structure of the graph. Numerical values, associated with the functionalities, called weights, are used by the ACs to autonomously decide which functionalities should be maintained in the event of an error in any of the system components.

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

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

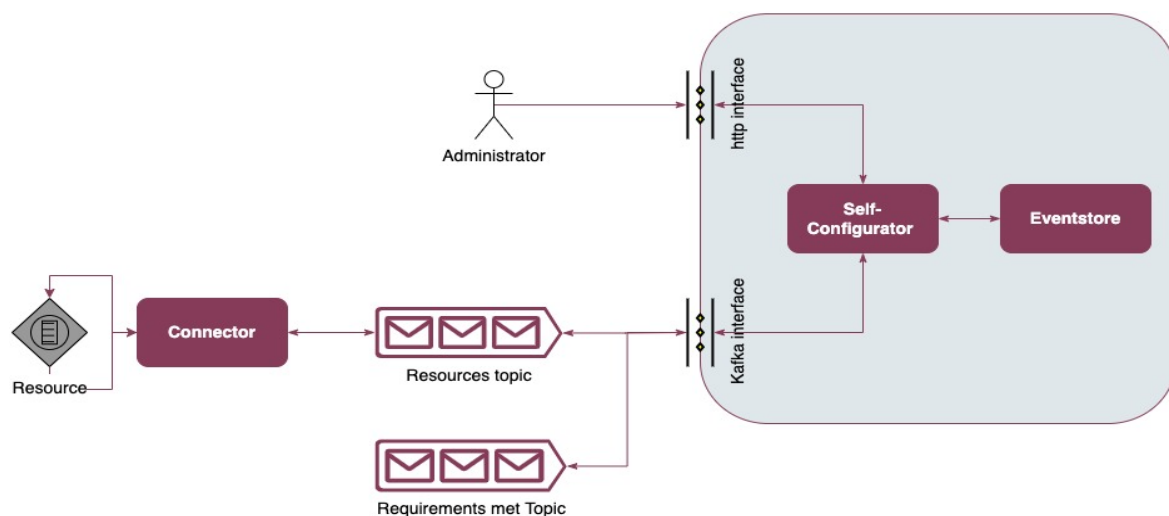


Figure 21. High-level diagram of the Automated configuration enabler

Table 22. Components and implementation of the Automated configuration enabler

Component	Description	Technology/s
<b>Self-configurator</b>	The main component of the AC enabler, responsible for handling resource abstract state representation management. It communicates with resources via connectors.	Scala, Akka, Akka Persistence
<b>Eventstore</b>	The component responsible for persisting the resource specific/generated events that provide input for the Self-configurator.	EventStore DB
<b>Connector</b>	A resource specific component performing direct manipulations on the resource and informing the AC enabler about the state changes of the AC-managed resource.	Kafka

### 3.1.6.3. Communication interfaces

Table 23. Communication interfaces of the Automated configuration enabler

Method	Endpoint	Description
<b>POST</b>	<b>/v1/requirements-model</b>	Creates or updates requirements model
<b>DELETE</b>	<b>/v1/requirements-model/{id}</b>	Deletes requirements model with specific id
<b>POST</b>	<b>/v1/reaction-model</b>	Creates or updates reaction model
<b>DELETE</b>	<b>/v1/reaction-model/{id}</b>	Deletes reaction model with specific id
<b>RegisterResource</b>	<b>Resources topic</b>	Message with which resources register to Automated Configuration enabler
<b>DeregisterResource</b>	<b>Resources topic</b>	Message with which resources deregister from the Automated Configuration enabler
<b>CustomMessage</b>	<b>Resources topic</b>	A custom message that can trigger reactions
<b>RequirementsMet</b>	<b>Requirements met topic</b>	Message sent if all requirements are met with available resources
<b>RequirementsNotMet</b>	<b>Requirements met topic</b>	Message sent if not all requirements are met with available resources

### 3.1.6.4. Enabler stories

The **first enabler story** is related to the **creation/upload of a requirements/reaction model** via the HTTP interface.

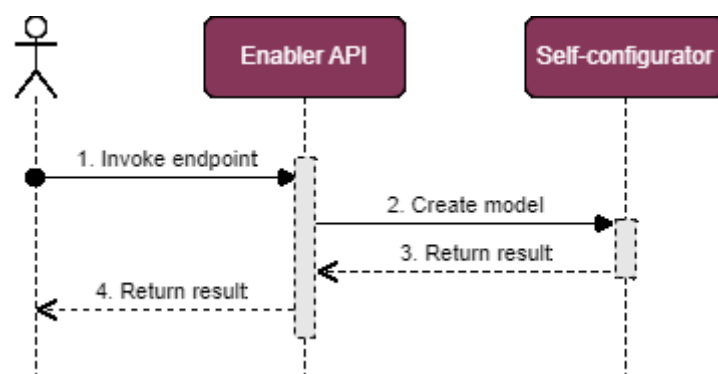


Figure 22. Automated configuration enabler ES1 (model creation)

**STEP 1:** The client invokes the /v1/requirements-model (or /v1/reaction-model) API endpoint, passing the configuration details.

**STEP 2:** The Self-configurator component verifies the request data and on success creates the appropriate model.

**STEP 3:** The Self-configurator communicates the resulting status through the enabler's REST API.

**STEP 4:** The client receives the model creation status message.

The **second enabler story** is related to the **deletion/removal of a requirements/reaction model** via the HTTP interface.

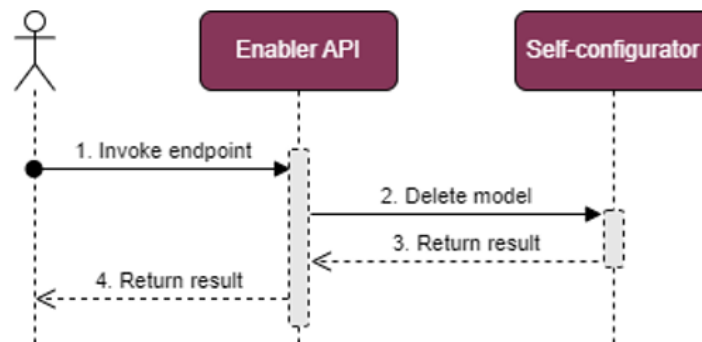


Figure 23. Automated configuration enabler ES2 (model deletion)

**STEP 1:** The client invokes the /v1/requirements-model (or /v1/reaction-model) API endpoint with the DELETE method, passing the model ID.

**STEP 2:** The Self-configurator component verifies the request data and on success removes the appropriate model.

**STEP 3:** The Self-configurator communicates the resulting status through the enabler’s REST API.

**STEP 4:** The client receives the model deletion status message.

The **third and central enabler story** for the AC enabler concerns **updating the resource configuration**.

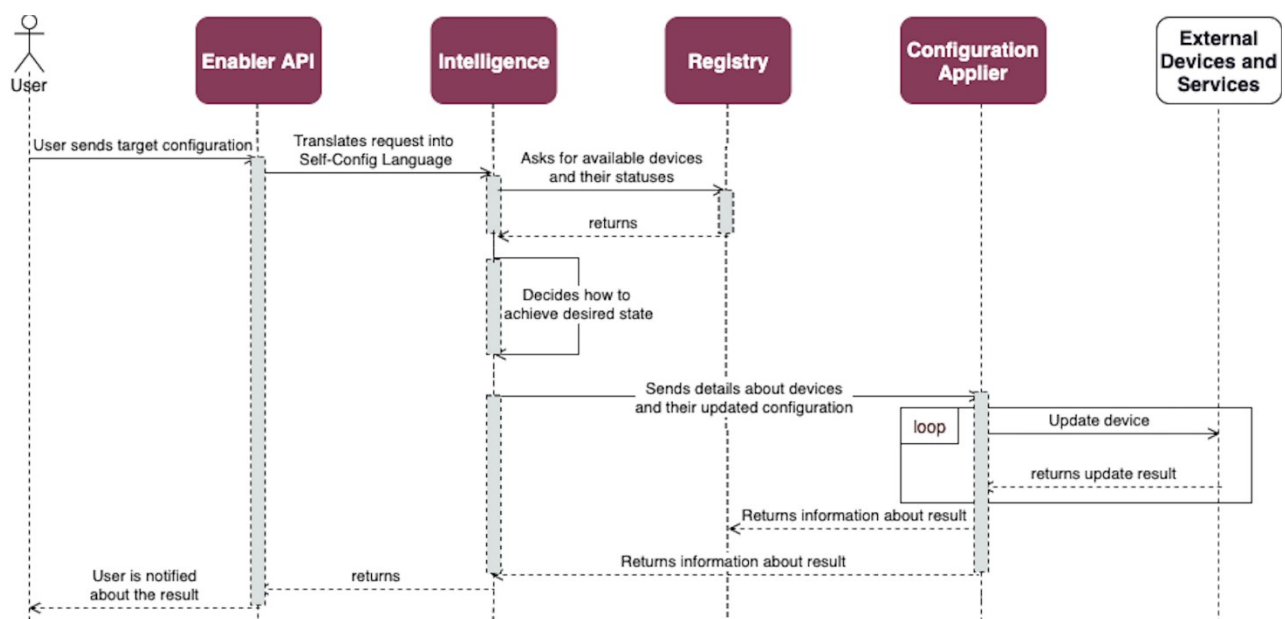


Figure 24. Automated configuration enabler ES3 (updating configuration)

**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 module (an internal part of the Self-configurator component) communicates with Registry module (another internal part of the Self-configurator component) to check the current status of the devices/resources.

**STEP 3:** Intelligence module checks how to achieve *target configuration* using available devices/resources. This requires understanding *actions* a particular device/resource can perform and what are the results of those actions. Based on that, Intelligence module creates series of device/resource updates.

**STEP 4:** Intelligence module sends request to Configuration Applier module (yet another internal part of the Self-configurator component).

**STEP 5:** Configuration Applier module updates all devices/resources involved.

**STEP 6:** Configuration Applier module sends results to both Registry and Intelligence modules.

**STEP 7:** Intelligence module returns result status to the user via the Automated configuration enabler API.

### 3.1.6.5. Implementation status

*Table 24. Implementation status of the Automated configuration enabler*

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/automated_configuration_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/self/automated_configuration_enabler.html</a>
Potential features	Further extensions to the abstract configuration description format/language.
Encapsulation readiness	All components are encapsulated in a Helm chart.
Integration with other enablers	The AC enabler functionality does not depend on any of the other enablers.

## 3.2. Federated learning enablers

### 3.2.1. FL Orchestrator

#### 3.2.1.1. General specifications and features

*Table 25. General information of the FL Orchestrator*

Enabler	FL Orchestrator enabler
Id	T52E1
Owner and support	PRODEVELOP
Description and main 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.
Key features	Training process orchestration
Vertical, related capabilities and features	Privacy, Scalability
Plane/s involved	Smart Network and Control Plane - a network interface should be managed in the communication between parties – mediators (if any) - masters
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-3: Compliance with legal requirements on data protection</li> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-10: Transmission security</li> <li>• R-C-21: Reduction of computing demands for AI training</li> <li>• R-C-22: Support for data privacy during the training process</li> <li>• R-C-23: Multi-model FL support</li> <li>• R-C-24: Cooperative ML training support</li> <li>• R-C-25: Holistic security/privacy approach</li> <li>• R-C-28: Distributed configuration</li> <li>• R-P3A-9: Edge intelligence</li> <li>• R-P3B-13: Automatic defect detection</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P1-7: Target visualization during RTG operation</li> <li>• UC-P2-1: Worker’s health and safety assurance</li> <li>• UC-P3A-2: Vehicle non-conformance causes identification</li> <li>• UC-P3B-1: Vehicle’s exterior condition documentation</li> </ul>
Internal components	Element-based GUI, FLS API server, FL Websocket, FL DB



### 3.2.1.2. Structure, components, and implementation technologies

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

1. Job scheduling
2. Manage the learning lifecycle (including number of connected parties, or evaluation threshold achieved).
3. Selecting and delivering initial version(s) of the shared algorithm

The following depicts the high-level overview of the FL orchestrator components.

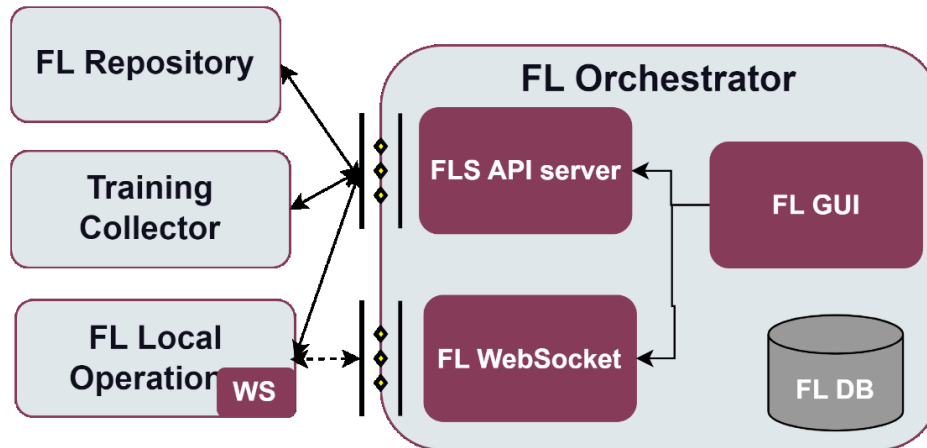


Figure 25. High-level diagram of FL Orchestrator

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

Table 26. Components and implementation of the FL Orchestrator

Component	Description	Technology/s
<b>FLS API server</b>	Offers a REST API to allow the communication and interaction with the rest of FL enablers, allowing to e.g., retrieve/send information from/to FL Local Operations, FL Training Collector, and FL Repository	Flask API server
<b>FL WebSocket</b>	Provides simultaneous two-way communication channel with FL Local Operations (which in turn have an FL client) through TCP connection for monitoring how many of them are available and running.	Websocket Python library
<b>FLS GUI</b>	This component is in charge of providing a graphical user interface that allow FL users and practitioners to set up and track an FL training incarnation. It specifies the initial configuration (e.g., minimum number of FL Local Operations needed, number of FL training rounds, the initial shared ML algorithm, encryption mechanism, FL aggregation strategy, or the evaluation metric and value). It is also in charge of the backend logic to guarantee the lifecycle management (e.g., monitoring the number of FL Local Operations connected, the number of training rounds finished provided by the FL Training Collector, or the required evaluation metric achieved).	Vue.js (Element)
<b>FL DB</b>	It serves as an in-between storage service for the FL Orchestrator initial configuration.	MongoDB

### 3.2.1.3. Communication interface

Table 27. API of the FL Orchestrator

Method	Endpoint	Description
<b>POST</b>	<b>/ShowConfigurationModel/</b>	Recover the current or previous FL configuration files stored into the internal FL Orchestrator database

GET	/AddModelData/<id>	Store into the internal FL Orchestrator database, the retrieved FL algorithm from the FL Repository
GET	/ModelData/<id>	Append FL training configuration with the FL algorithm retrieved from the FL repository of the /appModelData endpoint
POST	/StoreConfigurationModel	Store into the internal FL Orchestrator database, the complete FL algorithm + training configuration (i.e., store the resulting JSON from /ModelData endpoint)
GET	/ConfigurationsbyModel/<id>	Recover configurations of model <id> collected in the FL Orchestrator database
GET	/GetConfigurations	Recover all FL algorithms plus training configurations stored in the FL Orchestrator database
POST	/UpdateConfigurationModel	Update training configuration of the FL algorithm to be used for training
POST	/TrainingModel/<id>	Start FL training with the configuration <id> defined in the WebUI
GET	/RecoverStatusfromEnablers	Receive the status of all the FL Local Operations involved in the training
POST	/FLTrainingRound	Recover last finalized training round from the FL Training Collector
GET	/RecoverTrainingEpochs/<epochs>/<total_epochs>	Get the current epoch being trained and the total number of epochs to be trained
POST	/FL_early_end	Notification about the early ending of the FL training due to the required evaluation metric achieved before last specified round

### 3.2.1.4. Enabler stories

There are 3 enabler stories that apply to this enabler.

The **first enabler story** is related to the **initial setup/configuration of the FL training process** via the customized web application foreseen for the ASSIST-IoT FL system. Its diagram and involved steps are the following:

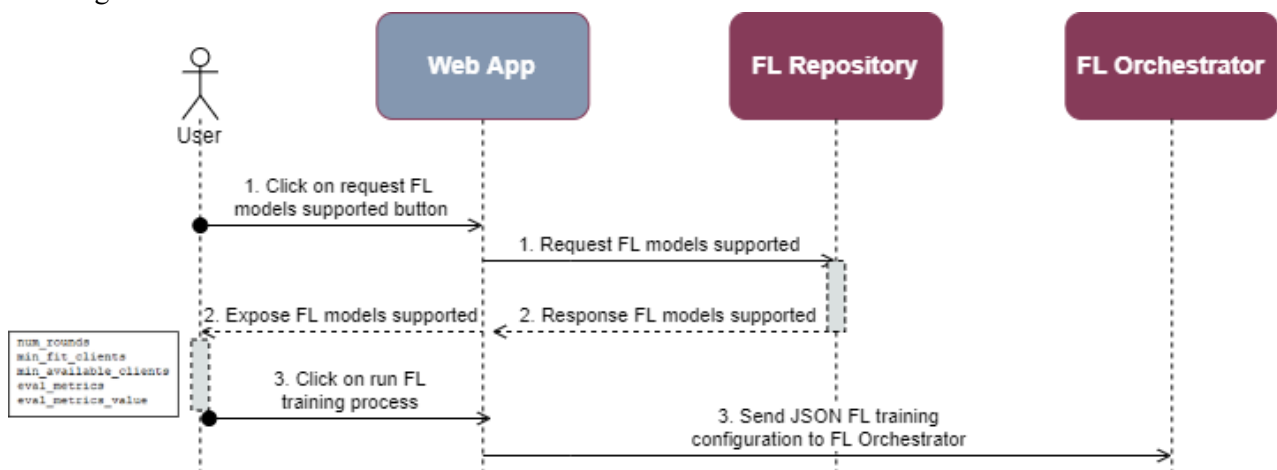


Figure 26. FL Orchestrator ES1 (user configures FL training)

**STEP 1:** User opens the webpage containing the FL system and clicks on the “Start new training” button. Then, clicks on the request FL models supported button, which communicates directly with the FL repository that exposes the supported ones.

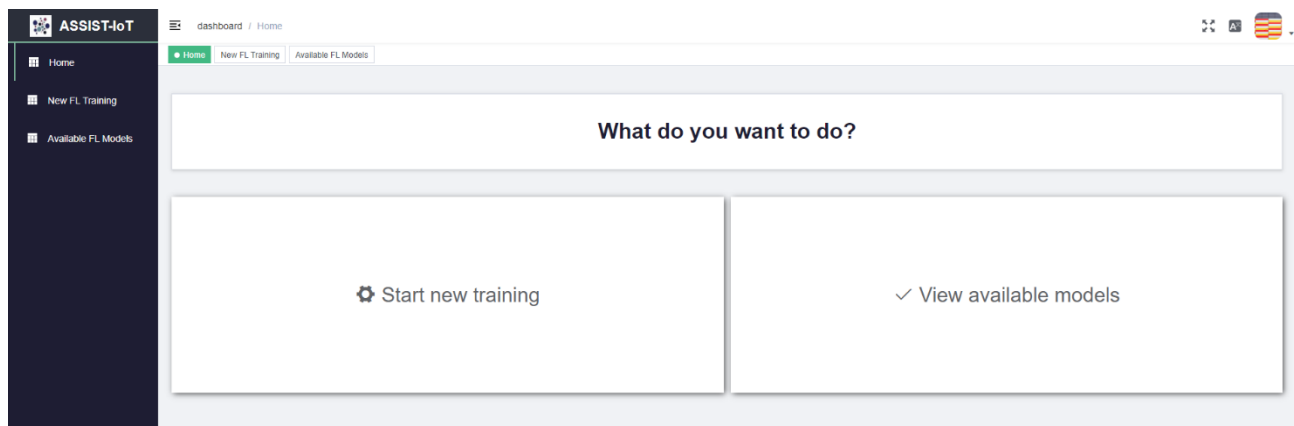


Figure 27. FL WebUI homepage.

**STEP 2:** The user modifies according to their preferences/interests the configuration of the provided models supported by the system for starting a new training (e.g., *num\_rounds*, *min\_fit\_clients*, *eval\_metrics*, *eval\_metrics\_value*).

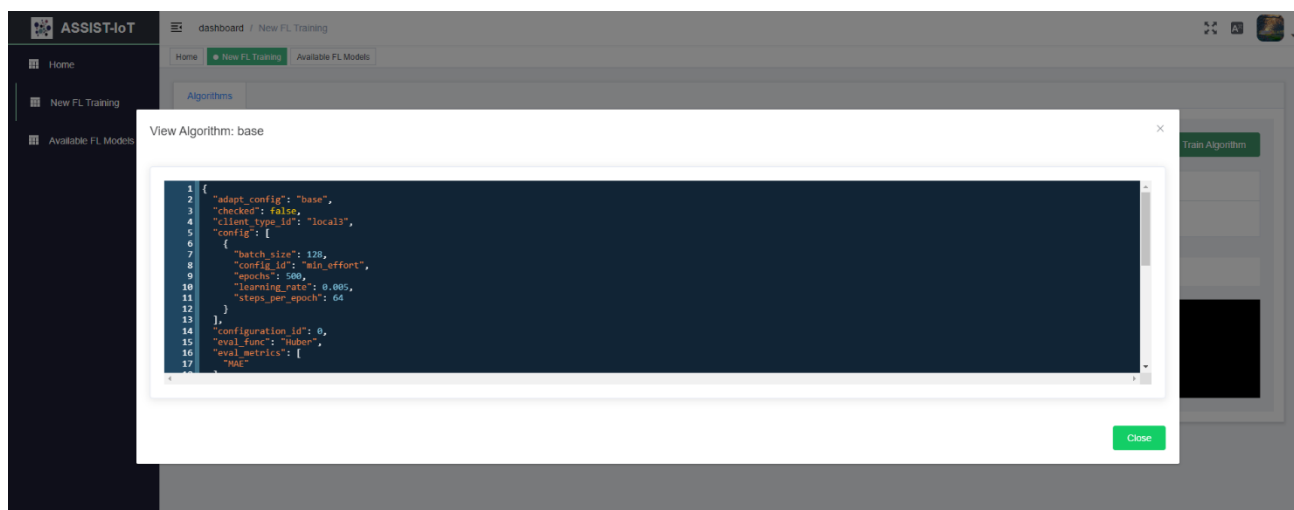


Figure 28. Screenshot of FL WebApp initial FL algorithm parameters

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

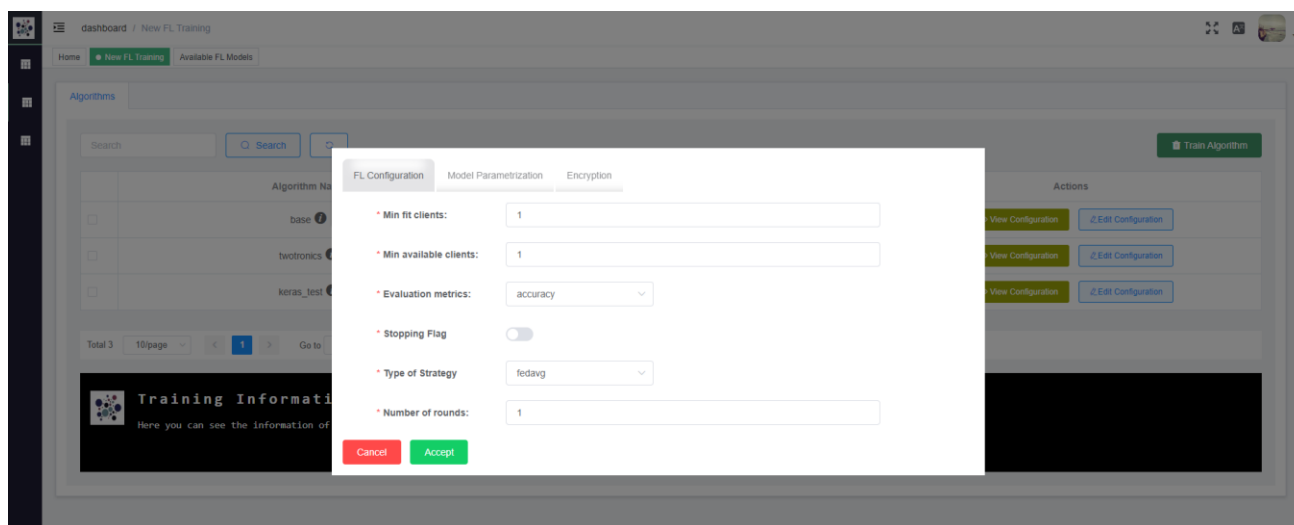


Figure 29. Screenshot of FL WebApp final configuration setup

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

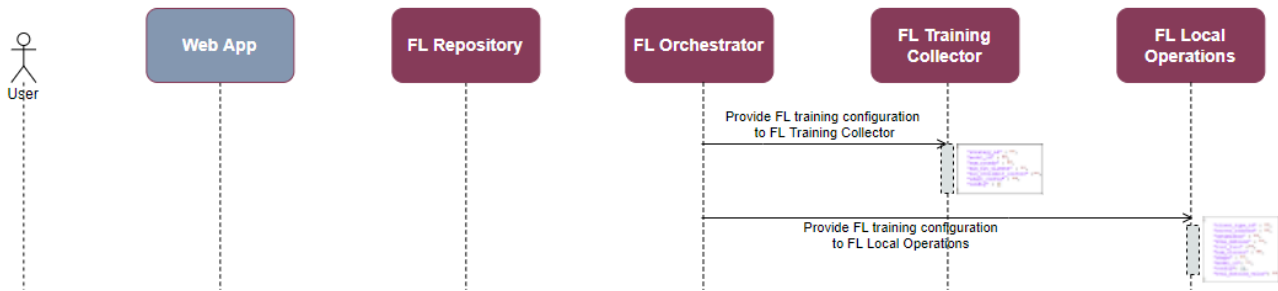


Figure 30. FL Orchestrator ES2 (initial setup of FL architecture)

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

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

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

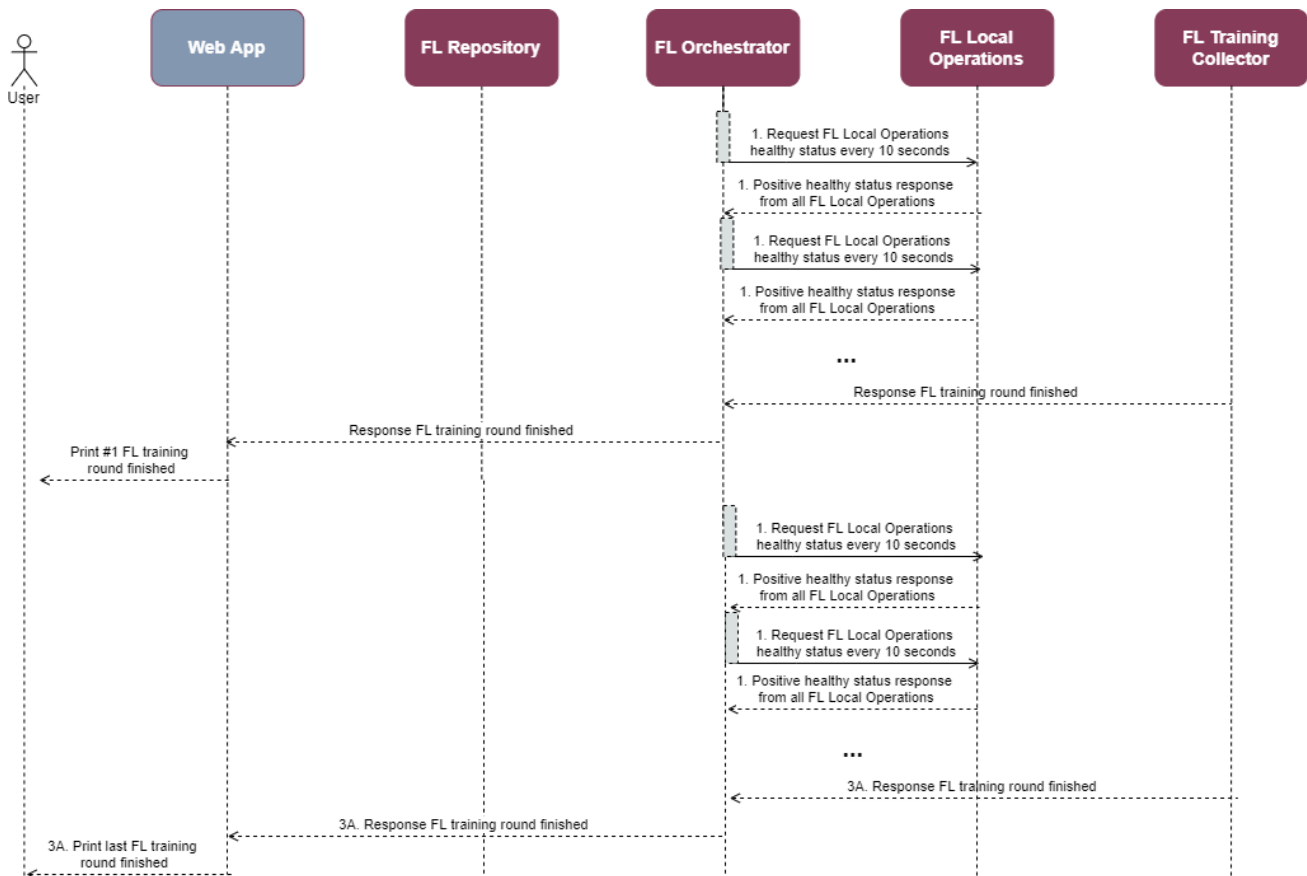


Figure 31. FL Orchestrator ES3 (lifecycle management of FL Training collector – Option A: happy path)

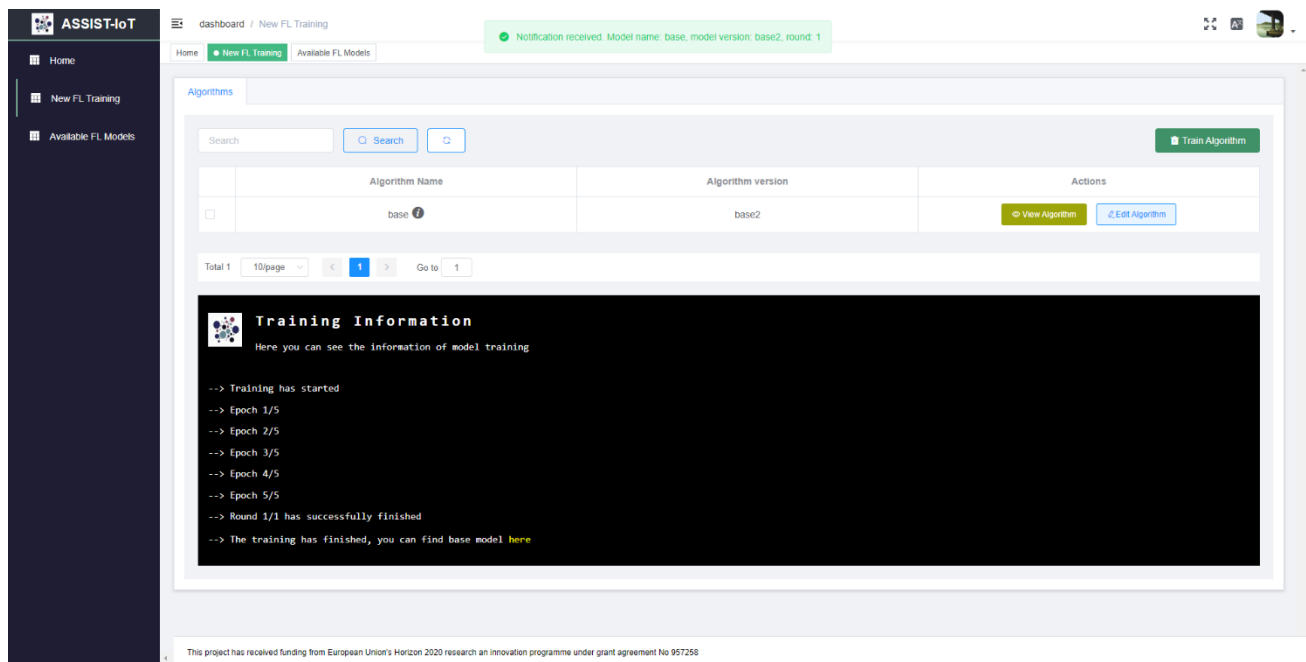


Figure 32. Screenshot of FL WebApp lifecycle management for the Option A: Happy Path

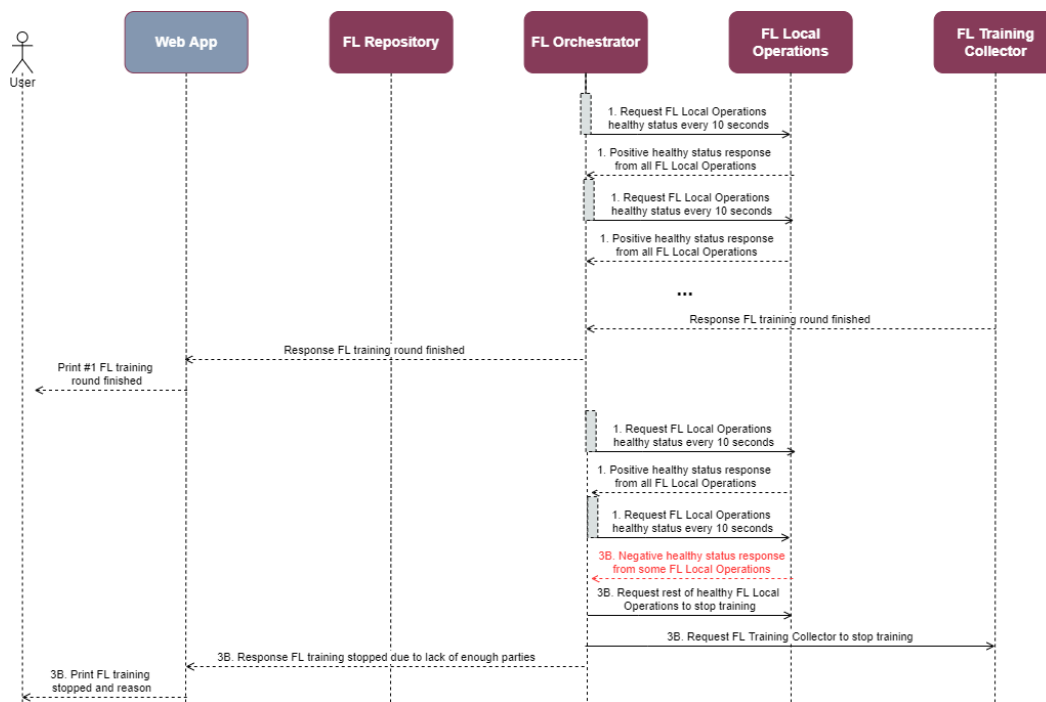


Figure 33. FL Orchestrator ES3 (lifecycle management of FL Training collector – Option B: FL local operations below minimum)

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

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

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

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

### 3.2.1.5. Implementation information

Table 28. Implementation status of the FL Orchestrator

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_orchestrator.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_orchestrator.html</a>
Potential features	Support for the configuration and monitoring of the whole FL Training lifecycle by means of a human-friendly GUI.
Encapsulation readiness	Enabler has a fully functional Helm package ready.
Integration with other enablers	The enabler has to be deployed along FL Repository, FL Training Collector and FL Local Operations for a fully functional FL System.

## 3.2.2. FL Training collector

### 3.2.2.1. General specifications and features

Table 29. General information of the FL Training collector enabler

Enabler	FL Training collector
Id	T52E2
Owner and support	SRIPAS, PRO
Description and main functionalities	<p>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 is tackled by the FL Training Collector, which is also in charge of sending the final results of the training along with the updated model weights to FL Repository for safe storage. In the context of this enabler, some of the most important aspects of the FL process had to be additionally supported through the use of dedicated modules, such as:</p> <p>A variety of available aggregation strategies (e.g., FedSGD, FedAvg) offered by the combiner.</p> <p>Supplemental privacy mechanisms in the form of differential privacy and homomorphic encryption,</p> <p>The possibility to extend the range of possible topologies through a combination of flexible aggregation strategy mechanisms and custom client behaviours.</p> <p>The FL Training Collector will consist of two components: (i) the combiner, responsible of providing updates with respect to the shared averaged model, and (ii) the I/O component, which will carry out the input and output communications of the enabler.</p>
Key features	<ul style="list-style-type: none"> <li>• Training process orchestration performed according to one on of many possible aggregation strategies,</li> <li>• The inclusion of additional, experimental privacy mechanisms, such as homomorphic encryption and differential privacy,</li> <li>• Automatic training results storage.</li> </ul>
Vertical, related capabilities and features	Privacy, Scalability
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-3: Compliance with legal requirements on data protection</li> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-10: Transmission security</li> </ul>

Enabler	FL Training collector
	<ul style="list-style-type: none"> <li>• R-C-21: Reduction of computing demands for AI training</li> <li>• R-C-22: Support for data privacy during the training process</li> <li>• R-C-23: Multi-model FL support</li> <li>• R-C-24: Cooperative ML training support</li> <li>• R-C-25: Holistic security/privacy approach</li> <li>• R-C-28: Distributed configuration</li> <li>• R-P3A-9: Edge intelligence</li> <li>• R-P3B-13: Automatic defect detection</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P2-1: Worker's health and safety assurance</li> <li>• UC-P3B-1: Vehicle's exterior condition documentation</li> </ul>
Internal components	FLTC I/O, FLTC Combiner

### 3.2.2.2. Structure, components and implementation technologies

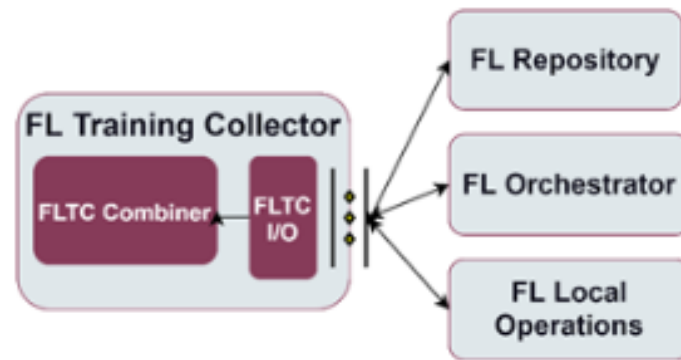


Figure 34. High-level diagram of the FL Training collector

The FL Training Collector fulfills the functionality of a classic Federated Learning server. When it receives the proper configuration from the FL Orchestrator enabler, it moves into a waiting state until a sufficient number of FL Local Operations instances connects to it. Then it manages the proper aggregation of model parameters from all of the instances each round, along with communicating the current state of the training process to the FL Orchestrator. Aside from sending out the new, updated weights after each epoch, the FL Training Collector is responsible for complying with the appropriate privacy mechanisms (homomorphic encryption and/or differential privacy) as well. After the training process is finished, FL Training Collector sends its results (metrics along with the final model weights) to the FL Repository for further storage.

Table 30. Components and implementation of the FL Training collector

Component	Description	Technology/s
<b>FLTC I/O</b>	It manages the communication between the FL Training Collector and the FL Orchestrator and FL Repository. It is therefore necessary for the proper initiation, monitoring and storing the results of the FL training process.	Python, FastAPI
<b>FLTC Combiner</b>	Performs the periodic FL model parameter and metrics aggregation according to a given FL algorithm, all the while adhering to the selected privacy techniques (homomorphic encryption and/or differential privacy).	Python, Flower, TenSEAL

### 3.2.2.3. Communication interfaces

Aside from the REST API presented in the table below, the FL Training Collector enabler uses the gRPC protocol to maintain the communication with FL Local Operations, which is handled in conformity with the Flower framework.



Table 31. API of the FL Training collector

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

### 3.2.2.4. Enabler stories

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

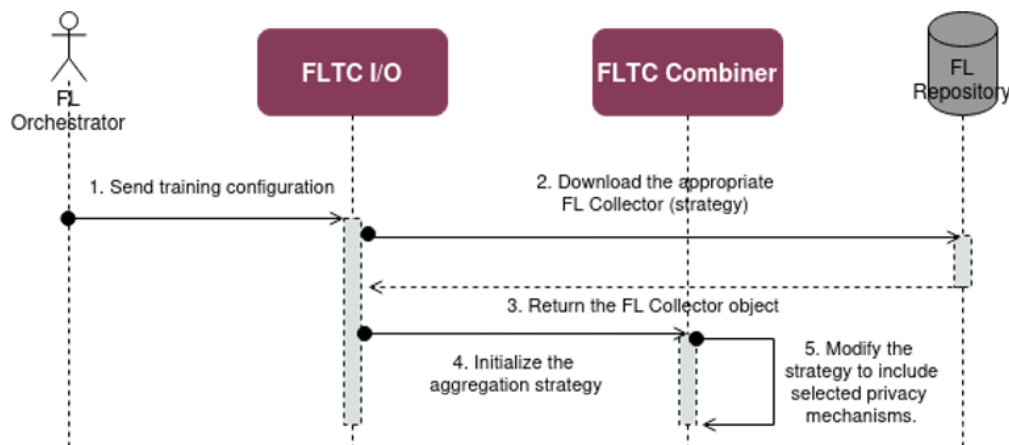


Figure 35. FL Training collector ES1 (training mechanism instantiation)

The involved steps are:

**STEP 1:** Receive configuration information from FL Orchestrator.

**STEP 2:** Retrieve from FL Repository the appropriate FL Collector (aggregation strategy).

**STEP 3:** Initialize aggregation strategy e.g. FedAvg, FedSGD.

**STEP 4:** Modify the strategy to include the right privacy mechanisms (homomorphic encryption, differential privacy) through the privacy manager.

The **second enabler story** is **combining local updates to the model to obtain an updated final model**.

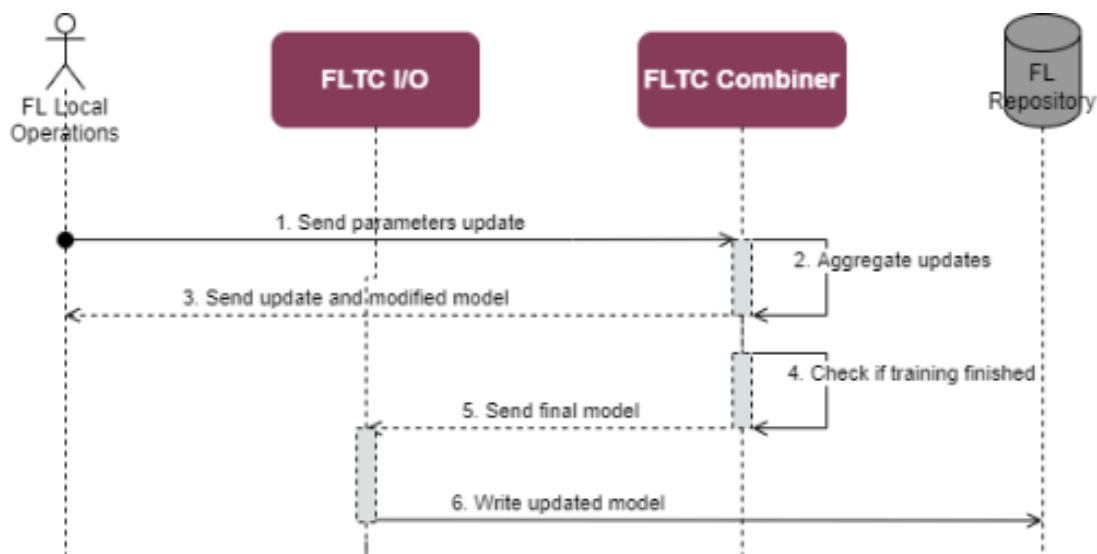


Figure 36. FL Training collector ES2 (local results aggregation)

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

**STEP 2:** FLTC Combiner combines local results to deliver new a shared model version. Depending on the privacy configuration, the process is conducted in a way compatible with homomorphic encryption and/or differential privacy.

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

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

**STEP 5-6:** If the training process is finished, FLTC Combiner sends the final model weights, metrics and configuration to FLTC I/O, which forwards in to FL Repository enabler to be stored.

### 3.2.2.5. Implementation information

*Table 32. Implementation status of the FL Training collector*

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_training_collector.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_training_collector.html</a>
Potential features	Support for the definition of a fully custom server training loop (that would extend beyond the aggregation strategy).
Encapsulation readiness	Enabler has a fully functional Helm package ready.
Integration with other enablers	The enabler has to be deployed along FL Repository, FL Orchestrator and FL Local Operations for a fully functional FL System.

## 3.2.3. FL Repository

### 3.2.3.1. General specifications and features

*Table 33. General information of the FL Repository*

Enabler	FL Repository
Id	T52E3
Owner and support	SRIPAS, PRO
Description and main functionalities	<p>One of the key application aspects of Federated Learning in IoT ecosystems is making it persistent and configurable. For this purpose, the FL Repository enabler was proposed. This repository stores (and delivers upon request/need) the ML aggregation algorithms, ML models and the results of ML training.</p> <p>It consists of four main components: ML Algorithms component (that gathers ML models in its first stage, i.e., without involving any modelling associated with a particular training data set), ML Models component (intermediary or final versions of ML model parameters, once they have been already trained with a particular data set, along with the metrics and configurations gathered throughout training), FL Collectors (aggregation strategies to be used in the FL training process in order to coalesce the learning results from multiple clients), and Auxiliary components (for any needed additional module along with the data transformations utilized for local data preprocessing). The FL Repository is a set of different collections in a NoSQL database.</p>
Key features	External API, Data Retrieval and Storage
Vertical, related capabilities and features	Privacy, Scalability
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-3: Compliance with legal requirements on data protection</li> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-10: Transmission security</li> </ul>

Enabler	FL Repository
	<ul style="list-style-type: none"> <li>• R-C-21: Reduction of computing demands for AI training</li> <li>• R-C-22: Support for data privacy during the training process</li> <li>• R-C-23: Multi-model FL support</li> <li>• R-C-24: Cooperative ML training support</li> <li>• R-C-25: Holistic security/privacy approach</li> <li>• R-C-28: Distributed configuration</li> <li>• R-P3A-9: Edge intelligence</li> <li>• R-P3B-13: Automatic defect detection</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P2-1: Worker's health and safety assurance</li> <li>• UC-P3B-1: Vehicle's exterior condition documentation</li> </ul>
Internal components	ML Algorithms Libraries, FL Collectors, Auxiliary, ML Model Libraries

### 3.2.3.2. Structure, components and implementation technologies

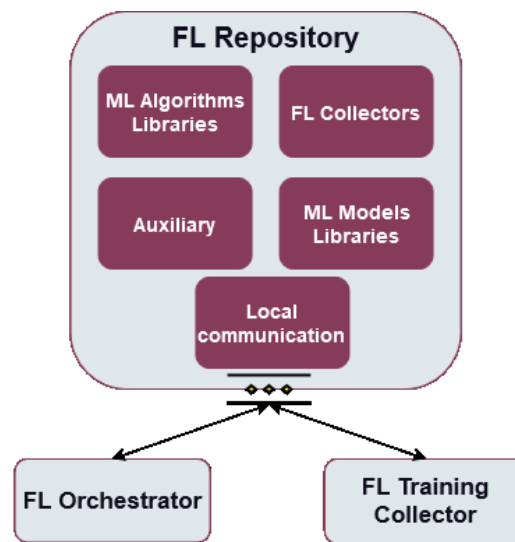


Figure 37. High-level diagram of the FL Repository

The FL Repository is responsible for the storage of all the most crucial reusable components for the FL training process. It stores the initial, fully configured versions of the ML solutions in the ML Algorithms component, along with the data transformations held in the Auxiliary component. Both are then downloaded by the FL Local Operations instances at the beginning of the training process. The FL Training Collector instances, on the other hand, download the FL Collectors for periodic parameter aggregation when starting the training and store the training results in the FL Repository at the end of it. Those results can be then accessed through the FL Orchestrator.

Table 34. Components and implementation of the FL Repository

Component	Description	Technology/s
<b>Local communication</b>	Provides an easy-to-use API for accessing the FL Repository, whether in order to download or upload a given object.	Python, FastAPI
<b>ML Algorithms Libraries</b>	Stores the models used in FL training in their initial form, which includes metadata about the library they should be used with, their architecture and their specific weights.	MongoDB, GridFS
<b>ML Model Libraries</b>	Stores the weights of the models trained throughout an FL training process along with the relevant metadata, including the evaluation and performance metrics in addition to training configuration.	MongoDB, GridFS

Component	Description	Technology/s
<b>FL Collectors</b>	Stores the various aggregation strategies, which can be used to specify the flow of a given training process and define a method of averaging the client weights.	MongoDB, GridFS
<b>Auxiliary</b>	Stores other, auxiliary components, including data transformations, which are employed for data preprocessing in the FL Local Operations inference and training processes.	MongoDB, GridFS

### 3.2.3.3. Communication interfaces

Table 35. API of the FL Repository

Method	Endpoint	Description
POST	/model	Adds the metadata of a new ML Algorithm (initial model) to the library.
PUT	/model/{name}/{version}	For the given ML Algorithm name and version, the binary file of the model is created or updated.
PUT	/model/meta/{name}/{version}	For the given ML Algorithm name and version, the metadata of the model is updated.
GET	/model	Return the list encompassing the metadata of all available ML Algorithms.
GET	/model/meta	Return the metadata of the ML Algorithm with a given name and version.
GET	/model/{name}/{version}	Return the binary file containing the ML Algorithm model weights and structure.
DELETE	/model/{name}/{version}	Delete the metadata and binary file of an ML Algorithm with a given name and version.
GET	/models/available	Return the list encompassing the metadata of all available ML Algorithms, sorted by their upload date.
GET	/models/download/shell/{filename}	Download the binary files containing the ML Algorithm weights and structure for all available ML Algorithms, sorted by their upload date.
POST	/training-results	Create a new ML Model (training results including final model weights and metadata containing aggregated metrics and configuration).
GET	/training-results	Get the list with the metadata of all ML Model positions available in this FL Repository instance.
GET	/training-results/{name}/{version}	Get the list with the metadata of all ML Model positions available in this FL Repository instance for a given model name and version.
PUT	/training-results/{name}/{version}/{training-id}/{configuration-id}	Update the final training weights of a given ML Model.
GET	/training-results/weights/{name}/{version}/{training-id}	Get the final training weights of a selected ML Model.
DELETE	/training-results/{name}/{version}/{training-id}	Delete a selected ML Model (the weights along with the metadata).
POST	/strategy	Create a new FL Collector (aggregation strategy) with the specified metadata.
PUT	/strategy/{name}	Update the object file for the FL Collector with a given name.
PUT	/strategy/meta/{name}	Update the metadata for the FL Collector with a given name.

Method	Endpoint	Description
GET	/strategy	Get the metadata of all available FL Collectors in the form of a list.
GET	/strategy/{name}	Get the object file of the FL Collector with a given name.
DELETE	/strategy/{name}	Delete the FL Collector (the metadata and the object file) of a given name.
POST	/transformation	Create a new data transformation (hosted in the Auxiliary component) with the specified metadata.
PUT	/transformation/{id}	Update the object file for a given data transformation.
PUT	/transformation/meta/{id}	Update the metadata of a given data transformation.
GET	/transformation	Get the list with the metadata of all data transformations available in this FL Repository instance.
GET	/transformation/{id}	Get the object file of a data transformation with a given id.
DELETE	/transformation/{id}	Delete the metadata and the object file of a data transformation with a given id.

### 3.2.3.4. Enabler stories

The **first enabler story** shows the flow of action when **FL Orchestrator** retrieves **ML Algorithm instance** that is available in the library. Assumption is that the requester knows specific parameters of the model to retrieve.

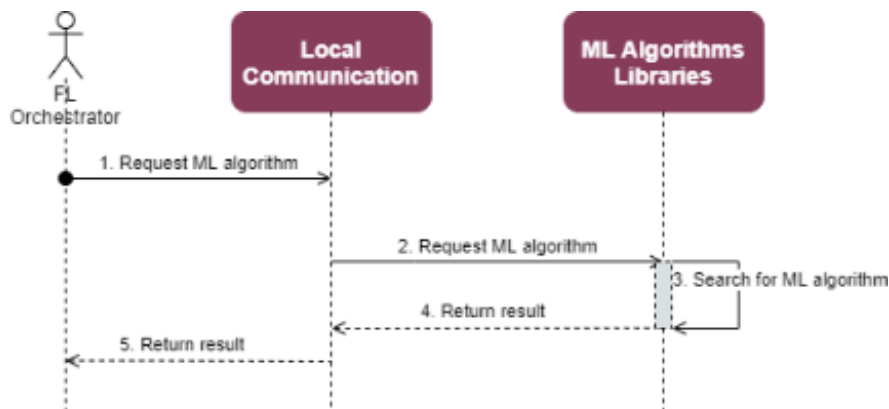


Figure 38. FL Repository ES1 (retrieve algorithm)

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

**STEP 2:** If the request is correct, it is forwarded to ML Algorithms component.

**STEP 3:** The ML Algorithms component searches for an instance with the given name and version.

**STEP 4:** If the ML Algorithm was found, it is returned to the Local Communication. If the ML Algorithm was not found, then the appropriate information is returned to the Local Communication component.

**STEPS 5:** The Local Communication forwards the response to the requester.

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

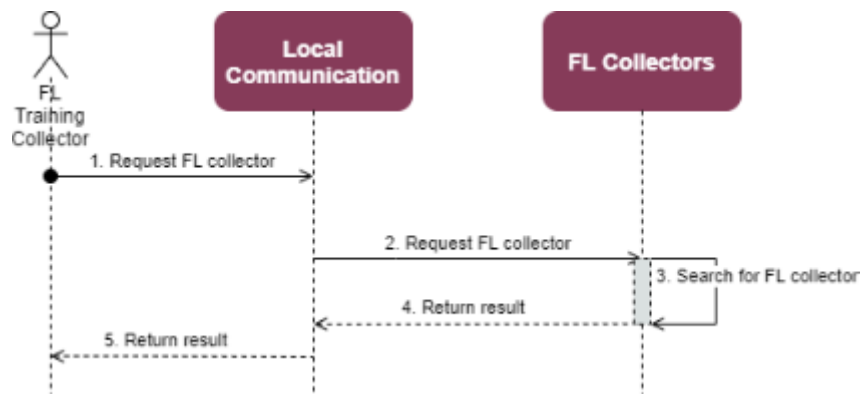


Figure 39. FL Repository ES2 (retrieve ML collector)

**STEP 1:** The FL Training Collector sends a 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 aggregation algorithm with a given name.

**STEP 4:** If the aggregation algorithm is found, it is returned to the Local Communication. If algorithm was not found, then the appropriate information is returned to Local Communication component.

**STEP 5:** Local Communication forwards response to the requester.

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

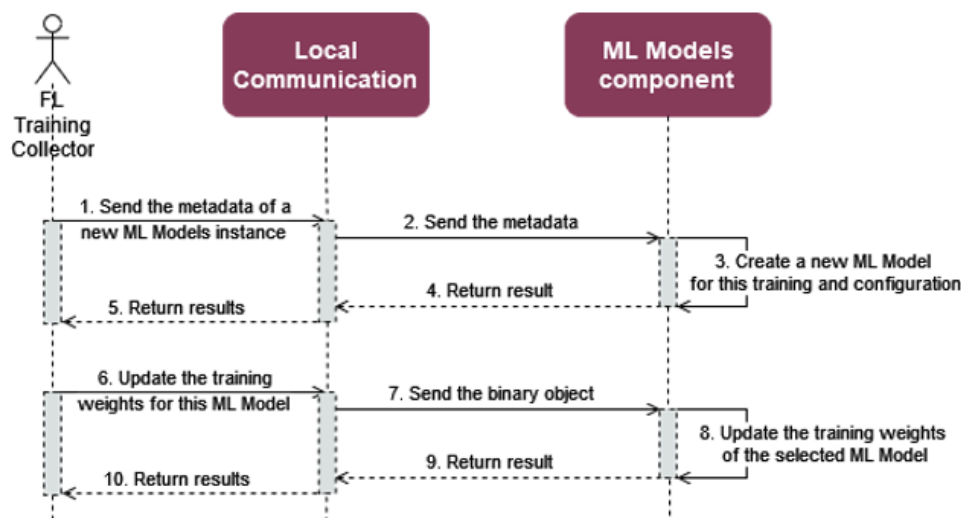


Figure 40. FL Repository ES3 (send updated model results)

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

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

**STEP 3:** ML Models Libraries creates a new instance from the received metadata.

**STEP 4:** If operation is successful confirmation is returned 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.

**STEP 6:** After receiving the correct response, the FL Training Collector sends the binary object containing the trained weights of the model to the Local Communication component.

**STEP 7:** If the request is correct, it is forwarded to ML Models Libraries component.

**STEP 8:** ML Models Libraries updates the trained weights of a selected ML Model to the received object.

**STEP 9:** If operation is successful confirmation is returned to the Local Communication. If any problem occurs, then respective information is returned to Local Communication component.

**STEP 10:** Local Communication forwards response to the requester.

The **fourth enabler story** describes a situation, in which a user wants to **download the final FL model weights produced through an FL training process**. Its diagram and flow are the following:

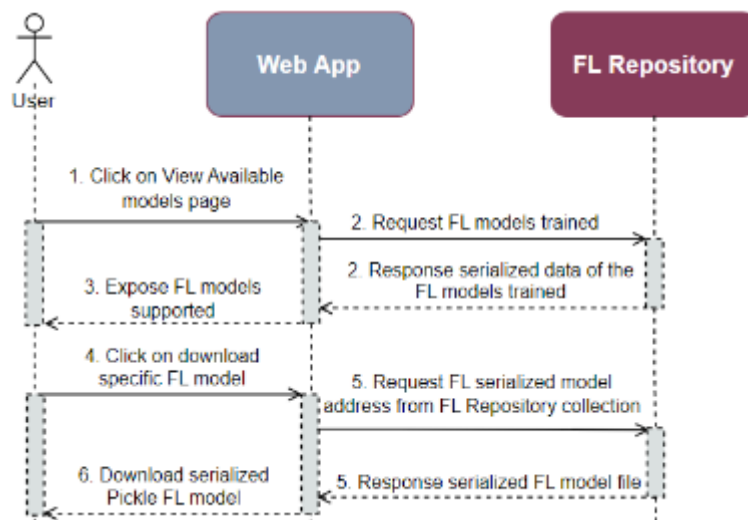


Figure 41. FL Repository ES4 (download FL serialised Pickle file)

**STEP 1:** User opens the second tab of the FL system webpage.

**STEP 2:** Automatically, the FL Orchestrator perform an API request to the FL repository in order to visualise the already stored and finished FL models.

**STEP 3:** These models are presented in a table format to the user, which can download them as serialised pickle files by clicking on the download button.

**STEP 4:** The user decides to download a FL model.

**STEP 5:** A request is made to the repository, which provides the serialised model.

**STEP 6:** The user obtains the model.

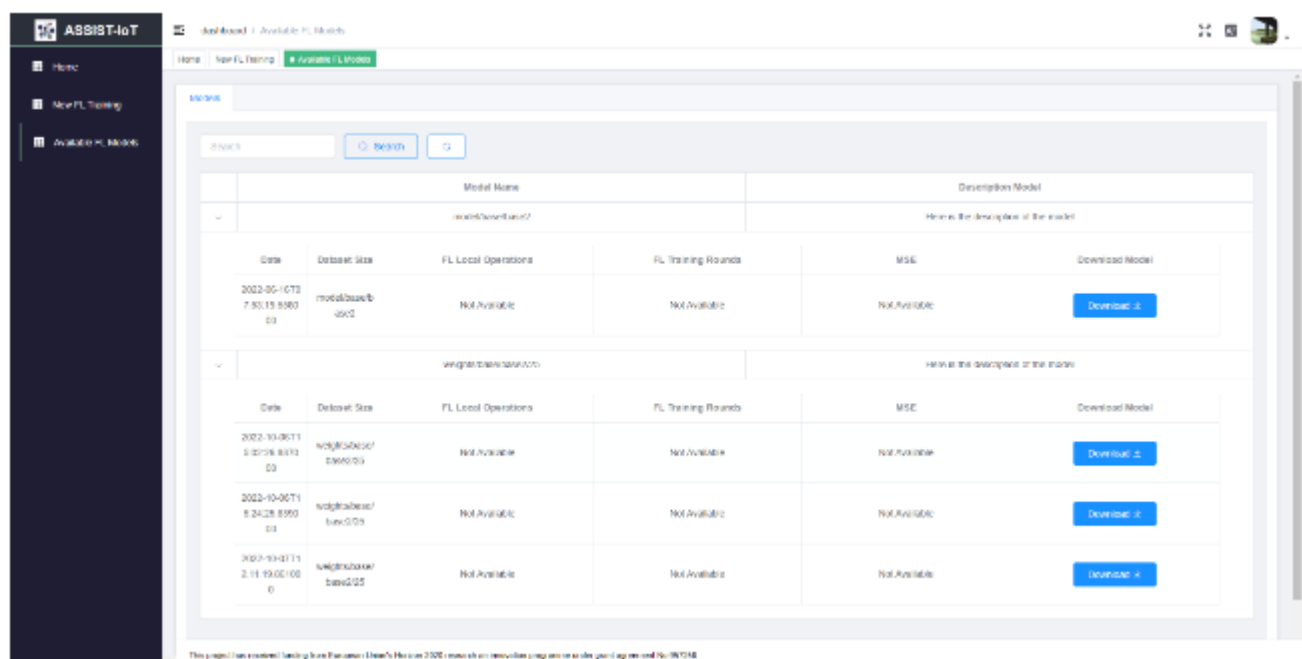


Figure 42. Screenshot of the FL WebApp download page



The final, **fifth enabler story** describes the situation, in which an **FL Local Operations instance does not have access to a given data transformation locally** at the start of its training or inference process, and therefore attempts to download it from the FL Repository.

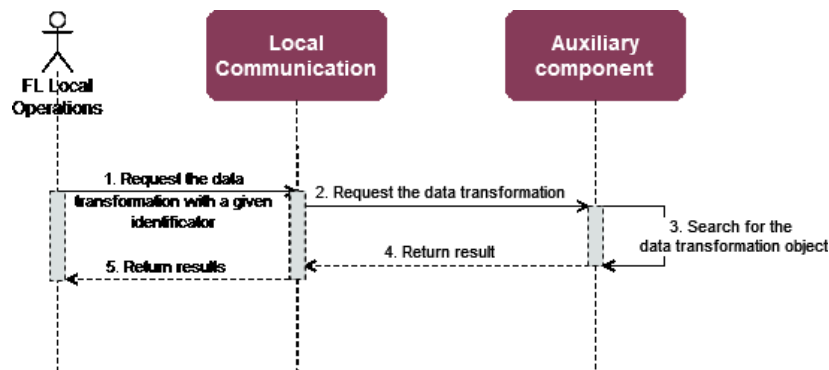


Figure 43. FL Repository ES5 (data transformation)

**STEP 1:** The FL Local Operations instance sends a 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 Auxiliary component.

**STEP 3:** The Auxiliary component searches for a data transformation with a given identification.

**STEP 4:** If the data transformation is found, it is returned to the Local Communication. If the data transformation object was not found, then the appropriate information is returned to Local Communication component.

**STEP 5:** Local Communication forwards response to the requester.

### 3.2.3.5. Implementation information

Table 36. Implementation status of the FL Repository

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_repository.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_repository.html</a>
Potential features	Additional mechanisms increasing the scalability of the solution and the efficiency of large file downloads and uploads.
Encapsulation readiness	Enabler has a full functional Helm package ready
Integration with other enablers	Works in a standalone fashion.

## 3.2.4. FL Local operations

### 3.2.4.1. General specifications and features

Table 37. General information of the FL Local operations

Enabler	FL Local operations
Id	T52E4
Owner and support	SRIPAS, PRO
Description and main functionalities	One of the key goals of FL is to assure protection of privacy of the data owned by individual stakeholders. Therefore, data is expected to be used only locally, to train local version of the shared model, and only the local parameter updates of the ML algorithm are shared with the FL servers or other participants. Additionally, local training has to be compliant with the privacy techniques selected by the user (homomorphic encryption and/or differential privacy).

Enabler	FL Local operations
	<p>When the FL training process has concluded, the final shared ML model is stored in the FL Repository enabler. It can be also later used for local inference through the Local Model Inferencer. The Local Model Inferencer allows for pluggable modification through the Local Data Transformer and supports fast inference using the gRPC protocol.</p> <p>Both operations (model training and model inference) involve access to private data. This means that it is crucial to “encapsulate” local processes within a single “node” (that is controlled by data owner). However, it should be noticed that the data that is being used in both FL training processes has to be preprocessed to reach the same format compliant with the ML model. In order to flexibly data to be used both for training and inference, the Local Data Transformer component is used.</p> <p>In order to carry out with all these local operations, the FL Local Operations enabler was proposed. FL Local Operations enabler is an embedded enabler within each FL involved party/device of the FL systems.</p>
Key features	<ul style="list-style-type: none"> <li>• Remains in constant communication with the FL Training Collector and FL Orchestrator through gRPC, websocket and REST API.</li> <li>• Has local model training capabilities with the flexible ML model download from FL Repository.</li> <li>• Supports the use of privacy mechanisms such as homomorphic encryption and/or differential privacy.</li> <li>• Supports flexible and easy-to-develop data transformations for training and inference data preprocessing, as well as basic format verification.</li> <li>• Includes a Local Model Inferencer component with fast gRPC inference.</li> </ul>
Vertical, related capabilities and features	Privacy, Scalability
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-3: Compliance with legal requirements on data protection</li> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-10: Transmission security</li> <li>• R-C-21: Reduction of computing demands for AI training</li> <li>• R-C-22: Support for data privacy during the training process</li> <li>• R-C-23: Multi-model FL support</li> <li>• R-C-24: Cooperative ML training support</li> <li>• R-C-25: Holistic security/privacy approach</li> <li>• R-C-28: Distributed configuration</li> <li>• R-P3A-9: Edge intelligence</li> <li>• R-P3B-13: Automatic defect detection</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P2-1: Worker’s health and safety assurance</li> <li>• UC-P3B-1: Vehicle’s exterior condition documentation</li> <li>• UC-P3B-2: Exterior defects detection support</li> </ul>
Internal components	Local Data Transformer (that is charge of guaranteeing that data is appropriately formatted for the FL model in use), Local Model Trainer, Local Model Inferencer, Privacy

### 3.2.4.2. Structure, components and implementation technologies

The FL Local Operations enabler offers the functionalities of an FL client through its connection to the FL Orchestrator and FL Training Collector. When prompted by the FL Orchestrator to start training, the FL Local Operations uses its data transformation module to flexibly load and preprocess the data, as well as the client algorithms and the desired model. The client algorithm is also modified to be compliant with the selected privacy techniques. The FL Local Operations maintains a connection with the FL Training Collector in order to maintain the training process. Aside from training, the FL Local Operations can also be configured to provide lightweight inference using a selected model.

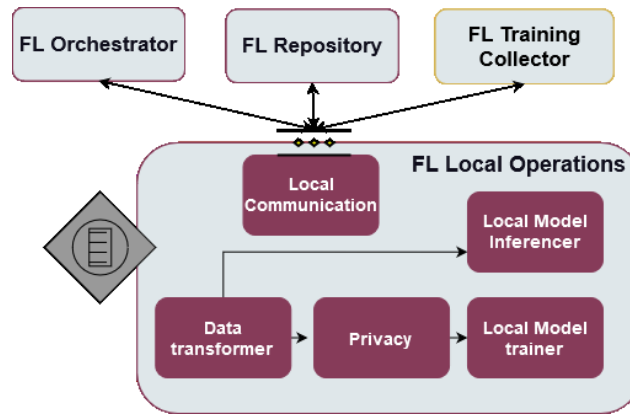


Figure 44. High-level diagram of the FL Local operations

The FL Local Operations enabler is built out of five components:

Table 38. Components and implementation of the FL Local operations

Component	Description	Technology/s
<b>Local Communication</b>	Enables the configuration and monitoring of the training process through the maintenance of regular communication with the FL Training Collector and FL Orchestrator components. Additionally, it provides the ability for necessary components to be downloaded from the FL Repository.	FastAPI, Python
<b>Data Transformer</b>	Supports a wide range of data preprocessing, data loading and training methods through the use of serializable modules. Those modules can be developed and deployed by the users on the go through Kubernetes volumes. Additionally, the Data Transformer enables format negotiation and verification through relevant configuration files.	Python
<b>Privacy</b>	Facilitates the use of two privacy-preserving algorithms dedicated to the FL use case: differential privacy and homomorphic encryption.	Python, Flower, TenSEAL
<b>Local Model Trainer</b>	Offers the functionalities of an FL client, including the local training and evaluation of models from two popular ML libraries.	Python, Flower, Tensorflow, PyTorch
<b>Local Model Inferencer</b>	Designed for fast and lightweight communication, Local Model Inferencer provides predictions of a selected model through a gRPC interface.	gRPC, Python, Tensorflow Lite, Tensorflow

### 3.2.4.3. Communication interfaces

Aside from RESTful interfaces, communication between FL Local Operations and the FL Training Collector throughout the training process is maintained using the gRPC protocol (as in Flower framework). Additionally, gRPC is also used by the FL Local Operations to offer lightweight inference. Finally, websockets are employed to maintain regular communication between the FL Local Operations and the FL Orchestrator enablers.

Table 39. API of the FL Local operations

Method	Endpoint	Description
POST	/job/config/{id}	Receive the configuration for a new training job.
POST	/model/	Receive new training model metadata for local storage
PUT	/model/{name}/{version}	Update the weights and structure of the locally stored training model.
GET	/job/status	Get the statuses of the current jobs.
GET	/job/total	Get the number of currently running jobs.
GET	/capabilities	Get the computational capabilities of the machine that FL Local Operations is running on.
GET	/format	Get the format of the data that a given FL Local Operations instance has currently access to.

### 3.2.4.4. Enabler stories

The **first enabler story** describes the performance of the full FL training process from the point of view of FL Local Operations.

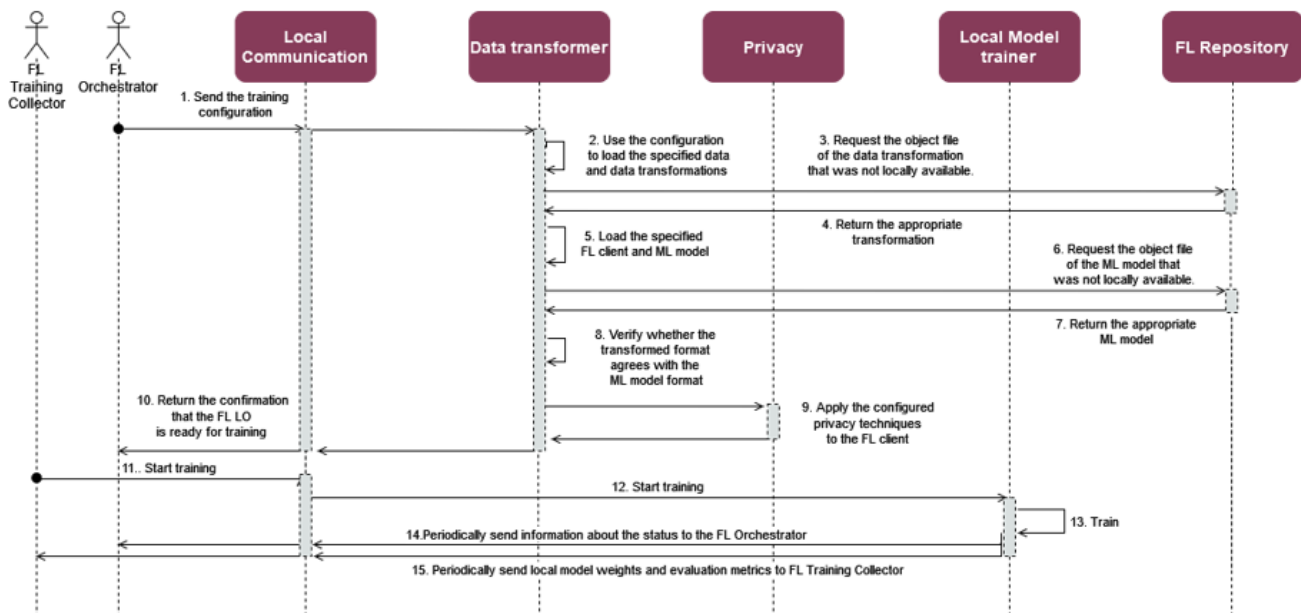


Figure 45. FL Local operations ES1 (FL training)

**STEP 1:** The training process begins with the FL Orchestrator sending to the FL Local Operations the training configuration.

**STEP 2:** The training configuration, along with other local configurations, is then used to deserialize the right data transformations and loaders.

**STEP 3:** In cases where a given data transformation can not be found locally, the Data transformer component attempts to download the specified transformation from the FL Repository.

**STEP 4:** The appropriate data transformation is returned.

**STEP 5:** The Data transformer component loads the specified ML model and sets up the appropriate FL client.

**STEP 6:** In cases where a given ML model can not be found locally, the Data transformer component attempts to download the specified ML model from the FL Repository.

**STEP 7:** The appropriate ML model is returned.

**STEP 8:** Based on the configurations describing the initial format of the data and how the application of a specific data transformation influences the format, the description of the format of data after transformations is obtained. It is then compared with the format accepted by the model to verify whether the pipeline is ready for training.

**STEP 9:** The FL client is configured to use the privacy techniques described in the configuration.

**STEP 10:** The information that the FL Local Operations is ready is returned to the FL Orchestrator.

**STEPS 11-12:** After establishing contact with a sufficient number of FL Local Operations instances, the FL Training Collector starts the training by contacting the Local Communication component.

**STEP 13:** The FL Local Operations starts the training process.

**STEP 14:** After each round of training, the FL Local Operations sends the information about the number of epochs completed to the FL Orchestrator.

The **second enabler story** illustrates the process in which an **FL Local Operations instance prepares the inference and delivers inference capabilities to the user.**

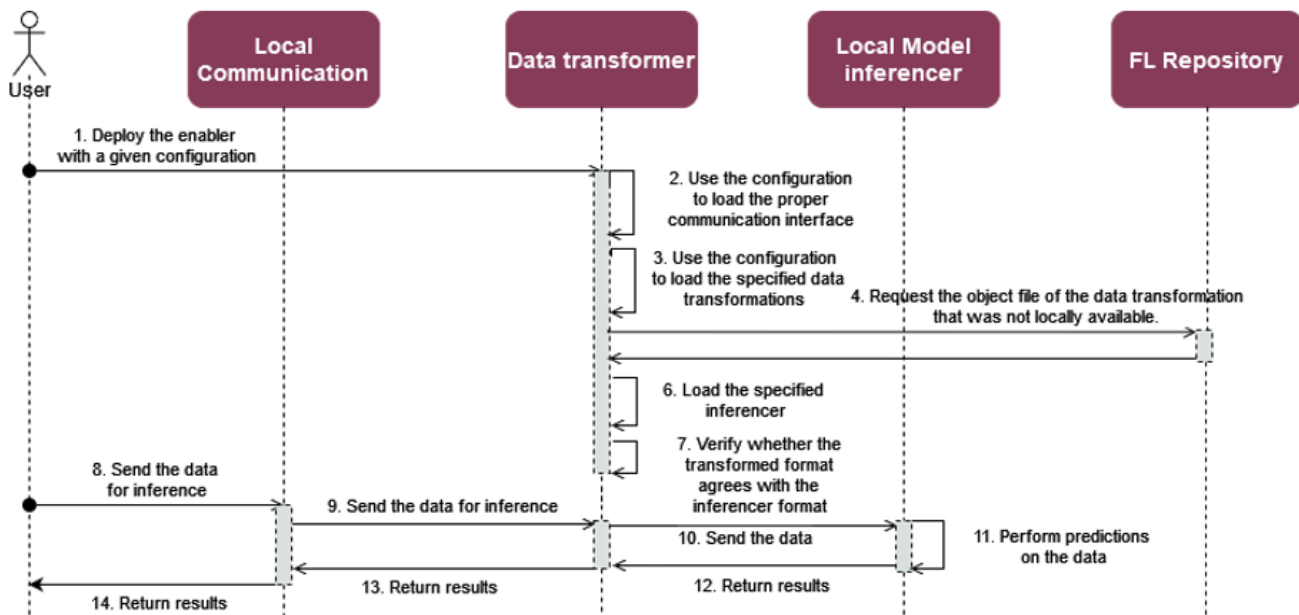


Figure 46. FL Local operations ES2 (inference)

**STEP 1:** The user deploys the FL Local Operations instance with a given inference configuration.

**STEP 2:** The Data transformer component obtains the configuration and uses it to load the specified communication interface.

**STEP 3:** The Data transformer component deserializes and initializes the data transformations described in the configuration.

**STEP 4:** In cases where a given data transformation can not be found locally, it attempts to download the specified transformation from the FL Repository.

**STEP 5:** The appropriate data transformation is returned.

**STEP 6:** The Data transformer component deserializes and initializes the specified ML model as the inference.

**STEP 7:** Based on the configurations describing the initial format of the data and how the application of a specific data transformation influences the format, the description of the format of data after transformations is obtained. It is then compared with the format accepted by the model to verify, whether the pipeline is ready for inference.

**STEP 8:** The user sends the data for inference to the Local Communication component.

**STEP 9:** The Local Communication component passes the data to the Data transformer.

**STEP 10:** The Data transformer applies the specifies transformations to the data and passes it to the ML model.

**STEP 11:** The ML model returns predictions based on the data.

**STEPS 12-14:** The resulting predictions are returned to the user.

### 3.2.4.5. Implementation information

Table 40. Implementation status of the FL Local operations

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_local_operations.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/federated/fl_local_operations.html</a>
Potential features	Extend the functionalities of the Data Transformer to offer more robust and scalable pluggability. Provide a more lightweight and optimized version of homomorphic encryption.
Encapsulation readiness	Enabler has a fully functional Helm package ready.

Category	Status
<b>Integration with other enablers</b>	The standalone enabler offers functionalities limited to those of the Data Transformer and Local Model Inferencer. In order to use the full range of the functionalities offered by the FL Local operations, one has to deploy it along the FL Training collector, FL Orchestrator and FL Repository.

### 3.3. Cybersecurity enablers

#### 3.3.1. Identity manager enabler

##### 3.3.1.1. General specifications and features

Table 41. General information of the Identity manager enabler

Enabler	Identity manager enabler (IdM)
<b>Id</b>	T53E2
<b>Owner and support</b>	S21Sec
<b>Description and main functionalities</b>	<p>The IdM performs the authentication phase of access control process, that is, it is responsible for managing identities on the access control process. Identity manager processes and validates the identity, for later control of the access to the resources by the authorisation enabler. The main goal of identity management is to ensure that only authenticated entities are granted access to the specific resource (applications, systems, or IT environments) for which they are authorised. This includes control over entities (i.e., user provisioning, or entities provisioning) and the process of onboarding new entities (i.e., users, systems, etc.). IdM enabler relies on OAuth2 protocol.</p> <p>The IdM provides a central user database and management console. It is also able to work federated with remote user databases, unifying remote user stores. It provides Single-Sign-On capabilities through OAuth2 protocol. The IdM integrates with the Authorisation enabler in order to offer a common authorisation and authentication process.</p>
<b>Vertical, related capabilities and features</b>	Security, privacy and trust
<b>Plane/s involved</b>	All planes – as it can provide identity management for accessing configuration and/or services offered by enablers of any plane.
<b>Requirements mapping</b>	<ul style="list-style-type: none"> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-25: Holistic security/privacy approach</li> </ul>
<b>Use case mapping</b>	<ul style="list-style-type: none"> <li>• UC-P1-4: RTG-Truck identification and authentication</li> <li>• UC-P1-5: RTG-Truck alignment</li> <li>• UC-P2-5: Safe navigation instruction</li> <li>• UC-P2-6: Health and safety inspection support</li> <li>• UC-P3A-3: Updating the diagnostics methods pool</li> <li>• UC-P3B-2: Exterior defects detection support</li> </ul>
<b>Internal components</b>	IdM administration module, IdM authentication module, Local user database, remote user database (optional)

##### 3.3.1.2. Structure, components and implementation technologies

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

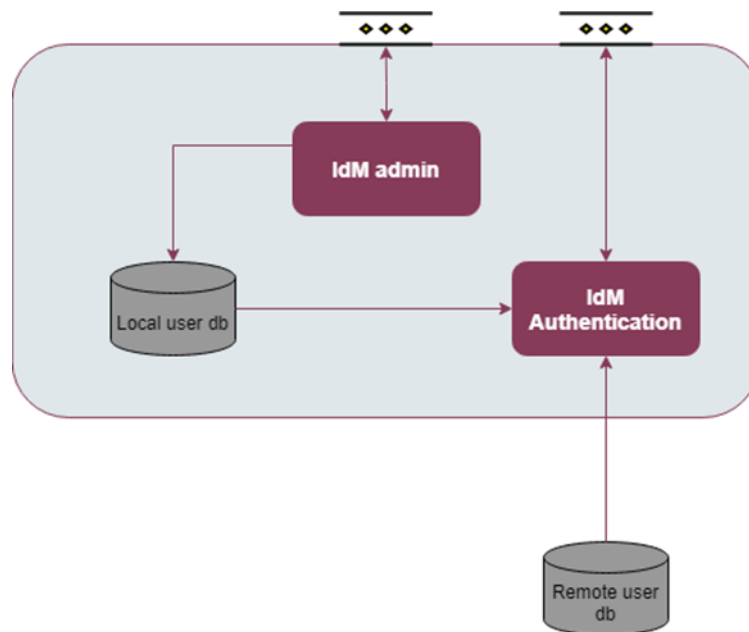


Figure 47. High-level diagram of the Identity Manager enabler

The implementation technologies of the enabler have not changed since the deliverable D5.3. It has been extended to work over machines with low resources, based on ARM64 architectures:

Table 42. Components and implementation of the Identity manager enabler

Component	Description	Technology/s
<b>IdM admin</b>	Enables and administrator to configure and populate the user's database: username, password, role,...	Web interface
<b>IdM Authentication</b>	Enables a user to login himself and after a successful login provides an access token to the user.	Keycloak, OAuth2
<b>Local user DB</b>	Enables an administrator to save all the user's database and the IdM authentication to check if the login parameters are correct	Web interface
<b>Remote user DB</b>	If the Local user dB has no info about the user's parameters, this DB can be checked from IdM authentication to verify user's login.	LDAP connector

### 3.3.1.3. Communication interfaces

Table 43. API of the Identity manager enabler

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

### 3.3.1.4. Enabler stories

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

**STEP 1:** An administrator populates user database.

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

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



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

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

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

**STEP 2.5.1:** Token is validated against the IdM.

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

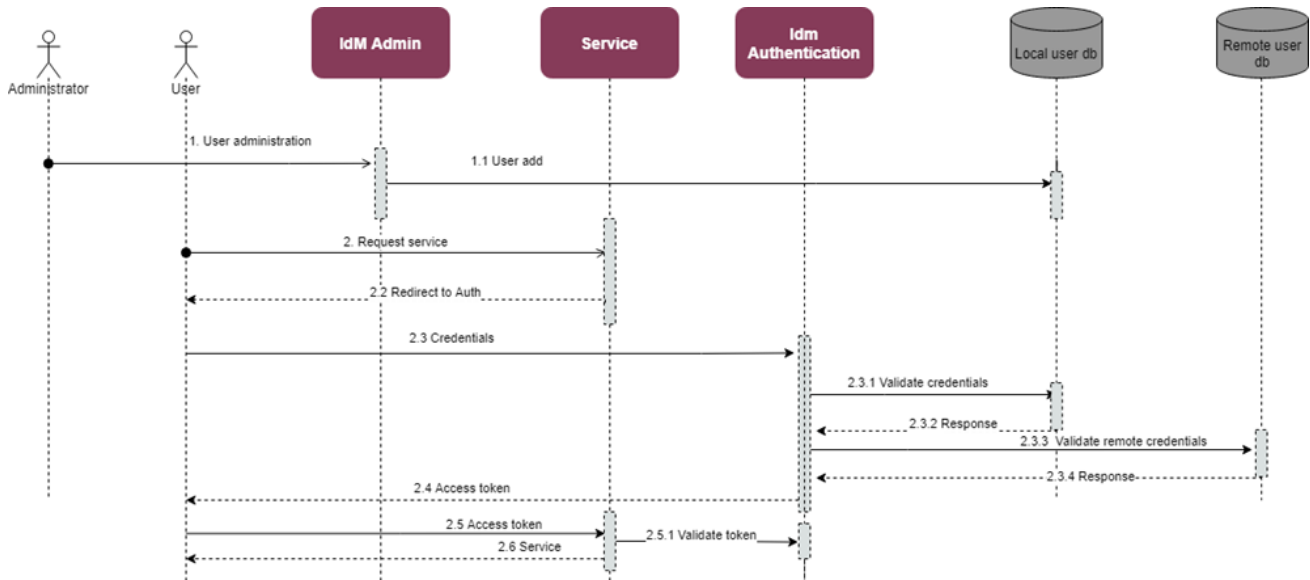


Figure 48. Identity manager enabler ES1 and ES2 (add user, authenticate user)

### 3.3.1.5. Implementation information

Table 44. Implementation status of the Identity manager enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/identity_manager_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/identity_manager_enabler.html</a>
Potential features	<ul style="list-style-type: none"> <li>One feature to increase the security is to use multifactor authentication, so a second code (password, auto-generated-code send via email or phone) would be needed. This would have entailed some modifications in the configurations of the applications and the user experience</li> <li>Another feature would be the Multi-role based user, so a user could be part of different groups. This would have entailed modifications in that applications and Authz enabler to perform different rights access depending the role and the system acceded.</li> </ul>
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler is installed jointly with the Tactile dashboard enabler and the Authorization enabler.

## 3.3.2. Authorisation enabler

### 3.3.2.1. General specifications and features

Table 45. General information of the Authorisation enabler

Enabler	Authorisation enabler
Id	T53E1
Owner and support	S21Sec
Description and main functionalities	Authorisation enabler is responsible for authorisation phase in the access control process. Authorisation is a process of granting, or automatically verifying, permission to an entity

Enabler	Authorisation enabler
	<p>(computer, application, or person) to access requested information after the entity has been authenticated. The enabler is based on XACML standard security policies, results on obligations actions to be deployed after the evaluation process.</p> <p>There are two different modes of deploying the same enabler, it can function as federated server, autonomous edge service or interact between both. In ASSIST-IoT a federated Authorisation enabler distributes a security policy from cloud to the edge to be locally evaluated and enforced.</p> <p>The authorisation enabler provides a Web administrator (PAP) to create and deploy the security policy to the different devices. In the service side that wants to use the authorisation service, there is an enforcement point (PEP) to make request to the authorisation server, this is, asking whether the access should be granted or not, through a REST interface available to receive the request and orchestrate the process. It is also possible to add Information Points (PIP) to generate the context for the request and obtain any data that external provider can offer to be incorporated to the request. Finally, the Authorisation enabler evaluates the requests against the policy, that is locally stored in the policy repository, and makes the decision (PDP) about the authorisation. It is also possible to invoke an obligation server to launch external requests to perform the derived actions (obligations) obtained as a result of the policy decision, through a REST requests.</p>
Vertical, related capabilities and features	Security, privacy and trust
Plane/s involved	All planes – as it can implement authorisation schemas for deciding who (user, enabler, external service) can configure and/or consume enablers offered from any plane
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-25: Holistic security/privacy approach</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P1-4: RTG-Truck identification and authentication</li> <li>• UC-P1-5: RTG-Truck alignment</li> <li>• UC-P2-5: Safe navigation instruction</li> <li>• UC-P2-6: Health and safety inspection support</li> <li>• UC-P3A-3: Updating the diagnostics methods pool</li> <li>• UC-P3B-2: Exterior defects detection support</li> </ul>
Internal components	PAP (Policy Administration Point), PDP (Policy Decision Point), PEP (Policy Enforcement Point), PIP (Policy Information Point)

### 3.3.2.2. Structure, components and implementation technologies

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

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

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

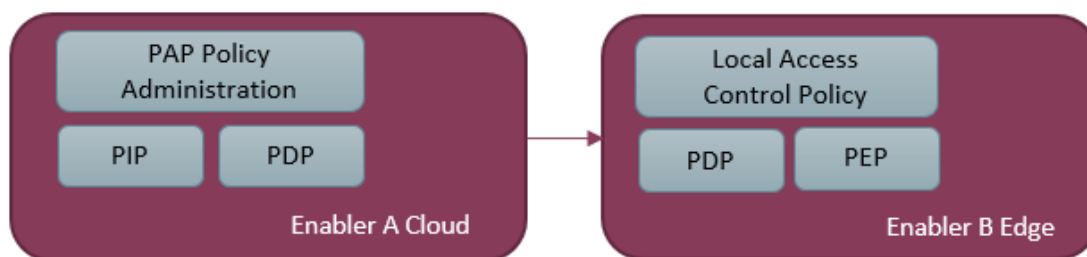


Figure 49. High-level diagram of the Authorization enabler

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

- PAP will provide a Web administrator to create and deploy the security policy to the different devices.
- A service that wants to use the authorization service will have a PEP, enforcement point to make request to the authorization server, this is asking whether the access should be granted or not.
- PDP provides a REST interface available to the PEP to receive the request and orchestrate the process.
- PIP will be responsible of generating the context for the request and obtaining any data that external provider can offer to be incorporated to the request.
- The Policy repository will store locally to the PDP the policy to be applied.
- PDP will evaluate the request against the policy and will respond with the response.
- Obligation server launches external requests to perform the derived actions (obligations) obtained as a result of the policy decision. This will have the form of REST requests.

Table 46. Components and implementation of the Authorisation enabler

Component	Description	Technology/s
PAP	Enables an administrator to create and deploy the security policy to the different devices.	XACML, REST interfaces
PIP	Generates the context for the request and obtaining any data that external provider can offer to be incorporated to the request	XACML
PDP	Enables a REST interface to the PEP to receive the request and orchestrate the process.	XACML, REST interfaces, MQTT
PEP	Enables an enforcement point to make request to the authorization server	XACML

The implementation technologies of the enabler have not changed since the deliverable D5.3. It has been extended to work over machines with low resources, based on ARM64 architectures.

### 3.3.2.3. Communication interfaces

Table 47. API of the Authorisation enabler

Method	Endpoint	Description
GET	/evaluate?resource=<domain>@<resource>&action=<action>&code=<id>	Evaluates a request performed and authorises it or not depending on the stored policies.
GET	/evaluate/trace?resource=<domain>@<resource>&action=<action>&code=<id>	Provides detailed info of evaluation data, such as, Context, XACML REQUEST, XACML RESPONSE and JSON RESULT of the XACML Evaluation.
GET	/rest/fedprov/<pip_name>/<in_value>	Enables to create a local PIP (Policy Information Point), with the aim of consulting any information should be needed for the authorization decision.

Method	Endpoint	Description
POST	/rest/polddbimport	Enables the exchange of security policies between two Authorization servers.

### 3.3.2.4. Enabler stories

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

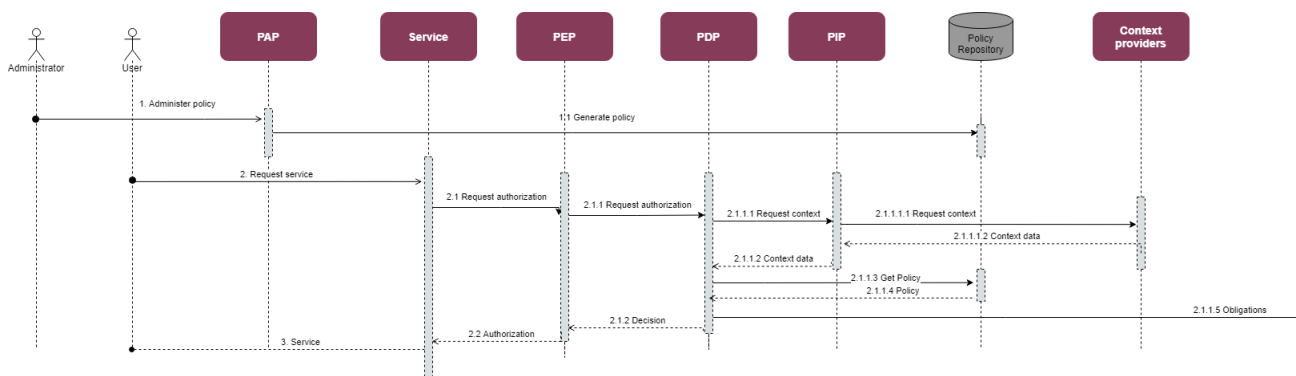


Figure 50. Authorization enabler ES1 (authorization flow)

**STEP 0:** A policy defined in the Authorisation enabler in the Cloud is exported to and received by the Authorisation enabler in the Edge, to be evaluated locally in the Edge when corresponds.

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

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

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

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

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

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

### 3.3.2.5. Implementation information

Table 48. Implementation status of the Authorization enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/authorization_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/authorization_enabler.html</a>
Potential features	<ul style="list-style-type: none"> <li>One feature to improve the authorization enabler would be to change the response from the enabler and go further permit and deny, so more actions could be performed. This would have entailed some modifications in the chart or the enabler, programming of other applications and would be a need of changing the configurations of the rules.</li> <li>Another feature to improve would be the remote PDP management. Now It can only be deployed under one domain, and a more granular management could help. This would have entailed code modifications in the chart and source code.</li> </ul>
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler is installed jointly with the Tactile dashboard enabler and the Identity manager enabler.

### 3.3.3. Cybersecurity monitoring enabler

#### 3.3.3.1. General specifications and features

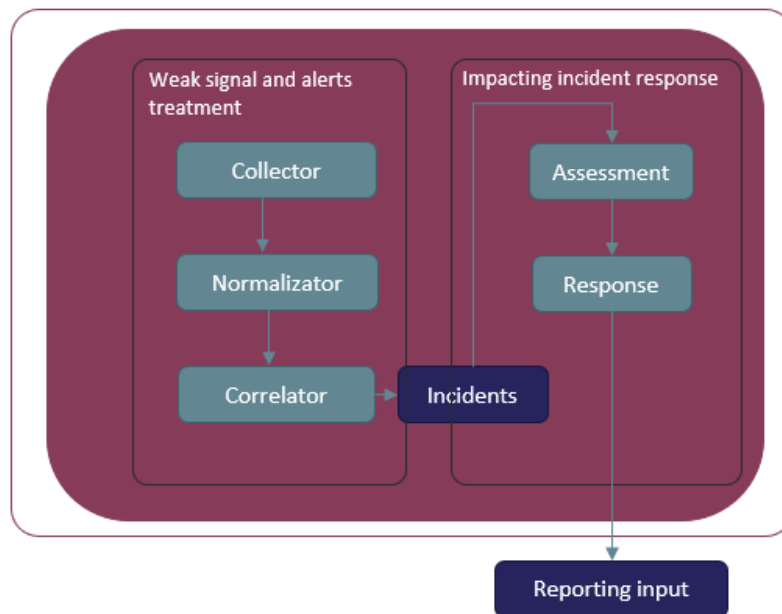
Table 49. General information of the Cybersecurity monitoring enabler

Enabler	Cybersecurity monitoring enabler
<b>Id</b>	T53E3
<b>Owner and support</b>	S21Sec
<b>Description and main functionalities</b>	<p>Cybersecurity monitoring enabler provides cyber security awareness and visibility on cybersecurity objectives and provides infrastructure cybersecurity monitoring. The enabler is responsible of collecting, processing, and analysing the incoming logs and information from the infrastructure under study. It decodes the information and applies security active rules from an existing ruleset to this information. If there is a match, it generates an alert. It normalises and enriches the alert to provide a consolidated output that provides cybersecurity monitoring information related to different events and facilitates the assignment of the risk level of the incident and the response actions to be done.</p> <p>The cybersecurity enabler can do predefined actions for incident mitigation depending on the incident, such as to communicate with the agent so that it performs an action, send an email or send the incident to a ticketing system. This enabler will update information on a graphical interface, so that an admin user can see the status of the alert/incident information, as well as the status of the related monitoring agents.</p>
<b>Vertical, related capabilities and features</b>	Security, privacy and trust
<b>Plane/s involved</b>	All planes – as it can monitor all layers, from device threats to network, services and application threats and vulnerabilities. Responses to certain events can be implemented and automated
<b>Requirements mapping</b>	<ul style="list-style-type: none"> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-18: Support for autonomous processing</li> <li>• R-C-19: Support for self-aware systems</li> <li>• R-C-25: Holistic security/privacy approach</li> <li>• R-C-26: Optimised Security notification</li> <li>• R-C-28: Distributed Configuration</li> </ul>
<b>Use case mapping</b>	<ul style="list-style-type: none"> <li>• UC-P1-4: RTG-Truck identification and authentication</li> <li>• UC-P1-5: RTG-Truck alignment</li> <li>• UC-P2-5: Safe navigation instruction</li> <li>• UC-P2-6: Health and safety inspection support</li> <li>• UC-P3A-3: Updating the diagnostics methods pool</li> <li>• UC-P3B-2: Exterior defects detection support</li> </ul>
<b>Internal components</b>	Alert treatment module, Incident response module, GUI

#### 3.3.3.2. Structure, components and implementation technologies

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

The cybersecurity monitoring server is responsible of collecting, processing, and analysing the incoming information from the infrastructure under study. It consolidates an output that provides cybersecurity monitoring information related to different events. Figure 51 describes cybersecurity monitoring components and describes how cybersecurity monitoring output will be alerts resulted from the processing of security events using rules.



*Figure 51. High-level diagram of the Cybersecurity monitoring enabler*

The functionalities of the Cybersecurity Monitoring enabler are presented below:

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

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

The implementation technologies of the enabler have not changed since the deliverable D5.3. It has been extended to work over machines with low resources, based on ARM64 architectures.

*Table 50. Components and implementation of the Cybersecurity monitoring enabler*

Component	Description	Technology/s
<b>Decoder</b>	Decodes the log, identify the log type and extract some useful fields	Wazuh Server
<b>Rule engine</b>	Is a ruleset that Decoders match the extracted data from the received logs	Wazuh Server
<b>Correlator</b>	Applies the active rules to the received log, and if there is a match, it generates an alert	Wazuh Server
<b>Assesment</b>	Normalizes the alert event and correlate until it determines if it is only a simple alert or a real incident	The Hive
<b>Response</b>	Provides an active responses to perform various countermeasures to address active threats	The Hive
<b>Visualization and Data Storage</b>	Enables long-term data storage and a user-friendly dashboard for data visualization.	Elasticsearch, Filebeat, Logstash and Kibana
<b>External enrichment</b>	Enriches the incident with useful information, to facilitate the assignment of the risk level of the incident and the response actions to be done	Cortex and MISP

### 3.3.3.3. Communication interfaces

Table 51. API of the Cybersecurity monitoring enabler

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

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

### 3.3.3.4. Enabler stories

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

**STEP 1:** Agent detects 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.

Additional enabler stories associated to cybersecurity monitoring server are the following:

- 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.



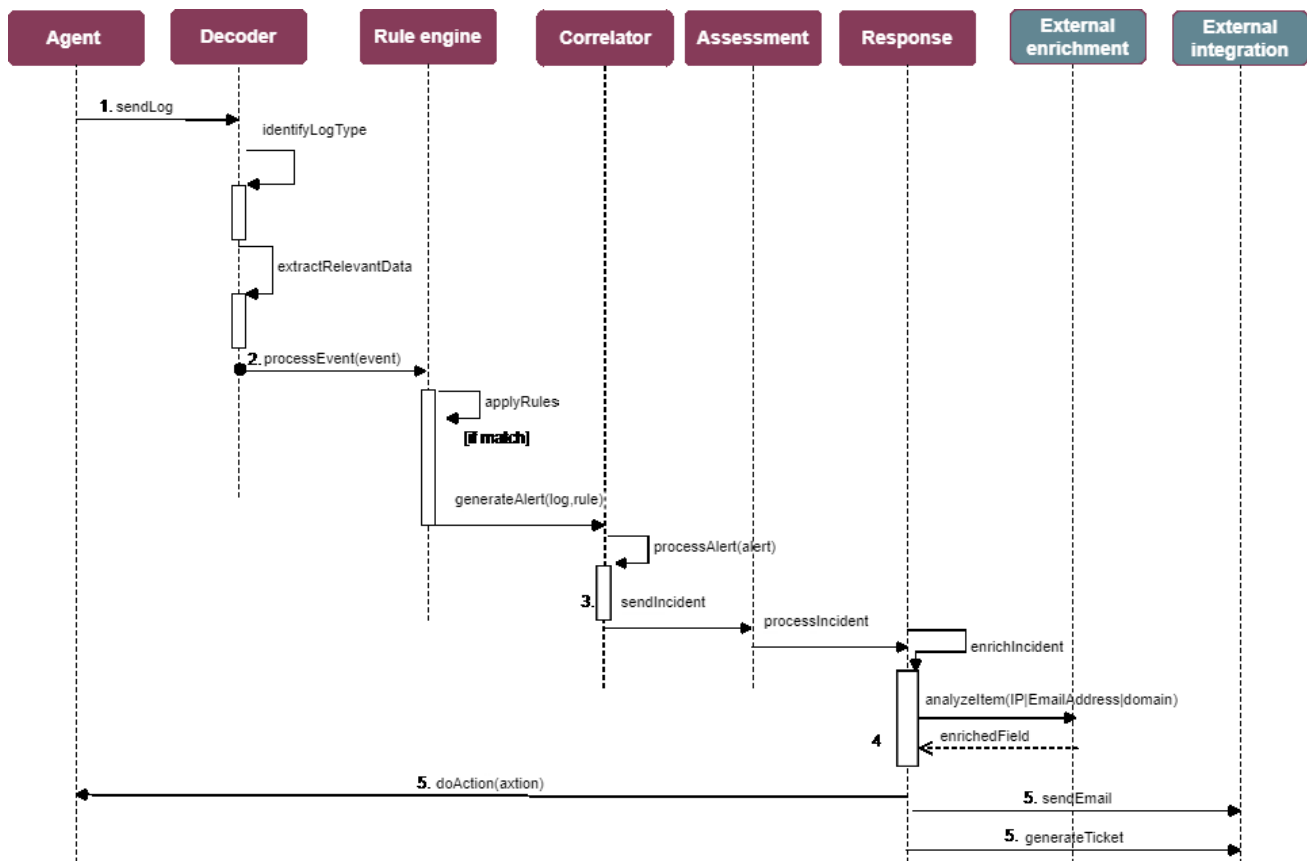


Figure 52. Cybersecurity Monitoring enabler ES1 (cyberthreats protection)

### 3.3.3.5. Implementation information

Table 52. Implementation status of the Cybersecurity monitoring enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/cybersecurity_monitoring_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/cybersecurity_monitoring_enabler.html</a>
Potential features	<ul style="list-style-type: none"> <li>One feature to increase the security of the project would be to integrate the enabler with third-party HW and SW solutions, so it could get events and implement responses in other sources. This would have entailed some modifications in the charts of the enabler.</li> </ul>
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler is integrated with Cybersecurity monitoring agent enabler.

### 3.3.4. Cybersecurity monitoring agent enabler

#### 3.3.4.1. General specifications and features

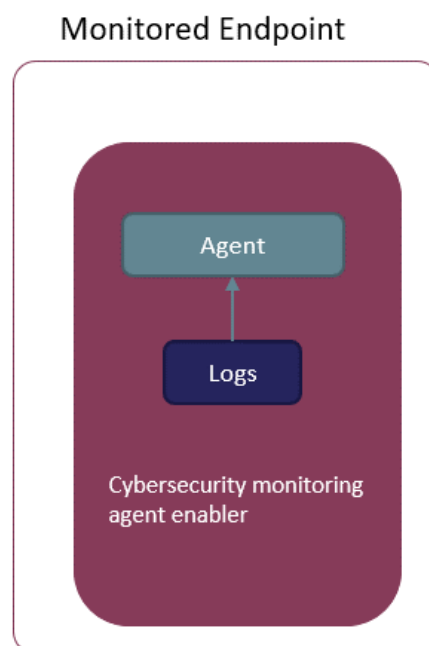
Table 53. General information of the Cybersecurity monitoring agent enabler

Enabler	Cybersecurity monitoring agent enabler
Id	T53E4
Owner and support	S21Sec
Description and main functionalities	Cybersecurity monitoring agent enabler performs functions of an endpoint detection and response system, monitoring and collecting activity from end points that could indicate a cybersecurity threat. The Cybersecurity monitoring agent enables the execution of

Enabler	Cybersecurity monitoring agent enabler
	<p>processes on the system target under study to collect relevant information if a cybersecurity breach is produced. It reports to the Cybersecurity monitoring enabler.</p> <p>The Cybersecurity monitoring agent collects and processes system events and system log messages. It monitors file integrity of critical files and audit data of the system. It also monitors the security of the Docker engine API and the container at runtime. It is also able to perform some actions such as blocking network connection or stopping running processes if the Cybersecurity monitoring enabler requests it.</p>
Vertical, related capabilities and features	Security, privacy and trust
Plane/s involved	<p>Device and edge plane – installed at the monitored hosts, this agent will collect all the necessary logs and data to be processed and analysed by the Cybersecurity monitoring enabler</p> <p>Rest of the Planes – logs and data belong to enablers/services/interfaces belonging to them</p>
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-5: Local Processing Capabilities</li> <li>• R-C-25: Holistic security/privacy approach</li> <li>• R-C-26: Optimised Security notification</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P1-4: RTG-Truck identification and authentication</li> <li>• UC-P1-5: RTG-Truck alignment</li> <li>• UC-P2-5: Safe navigation instruction</li> <li>• UC-P2-6: Health and safety inspection support</li> <li>• UC-P3A-3: Updating the diagnostics methods pool</li> <li>• UC-P3B-2: Exterior defects detection support</li> </ul>
Internal components	The agent module

### 3.3.4.2. Structure, components and implementation technologies

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



*Figure 53. High-level diagram of Cybersecurity monitoring agent enabler*

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

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

The monitoring agent requires to be linked to the monitoring server as well as a registering process to this component. Due to the ephemeral behaviour of containerised solutions, the register of the agents to the server would be lost if the agent container disappears. Hence, it could be considered as an exception to run agents in an encapsulated environment, like a containerised solution. Also, security agent enabler might need to monitor host services and interfaces that might not be reachable if deployed as a container. The proposed solution for overcoming this exception is to deploy the cybersecurity monitoring agent as a service in the underlying host system running the containerised solution, and then evaluating the way to redirect the information that the agent needs to collect from the monitored components running encapsulated.

*Table 54. Components and implementation of the Cybersecurity monitoring agent enabler*

Component	Description	Technology/s
Agent	Collect and processes the system events and system log messages, monitors file integrity of critical files and audit data of the system, monitors the security of the docker engine API and the container at runtime and It is able to perform some actions such as blocking network connection or stopping running processes if the Cybersecurity monitoring enabler requests it	Wazuh agent, Rsyslog

### 3.3.4.3. Communication interfaces

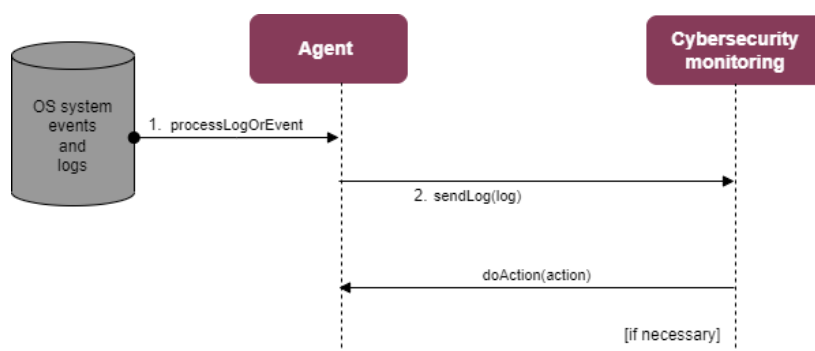
The communication interfaces of the enabler have not changed since the deliverable D5.3.

*Table 55. Communication interfaces (TCP/UDP) of the Cybersecurity monitoring agent enabler*

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

### 3.3.4.4. Enabler stories

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



*Figure 54. Cybersecurity monitoring agent enabler ES1 (send collected data and actuate)*

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

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

Additional enabler stories associated to Cybersecurity monitoring agent enabler are the following:

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

Agent detects unauthorised changed of configuration files and report to the server.

### 3.3.4.5. Implementation information

Table 56. Implementation status of the Cybersecurity monitoring agent enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/cybersecurity_monitoring_agent_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/cybersecurity/cybersecurity_monitoring_agent_enabler.html</a>
Potential features	N/A
Encapsulation readiness	This enabler is an encapsulation exception, as the monitoring of security logs should be monitored outside the virtualized layer. More information can be found at D3.7.
Integration with other enablers	This enabler is integrated with Cybersecurity monitoring enabler.

## 3.4. DLT-based enablers

### 3.4.1. Logging and auditing enabler

#### 3.4.1.1. General specifications and features

Table 57. General information of the Logging and auditing enabler

Enabler	Logging and auditing enabler
Id	T54E1
Owner and support	CERTH
Description and main functionalities	This enabler logs 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 also logs resource requests and identified security events to help to provide digital evidence and resolve conflicts between stakeholders, when applicable.
Key features	DLT
Vertical, related capabilities and features	Security, privacy and trust
Plane/s involved	Device and edge plane – critical events are stored in the ledger Rest of the Planes – any critical action of the rest of the planes can be stored
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-27: Automated accountability</li> <li>• R-P1-1: CHE location services</li> <li>• R-P2-1: Personal location tracking</li> <li>• R-P2-2: Construction plant location tracking</li> <li>• R-P2-7: Monitoring the weather conditions at the construction site</li> <li>• R-P2-9: Assessment of Personal Exposure to UV Radiation capability</li> <li>• R-P3A-6: Active monitoring mode initiation by the OEM software engineer</li> <li>• R-P3B-19: Critical Damage Identification Time</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P1-1: Asset location management</li> </ul>

Enabler	<i>Logging and auditing enabler</i>
	<ul style="list-style-type: none"> <li>• UC-P1-2: CHE location tracking</li> <li>• UC-P2-1: Workers' health and safety assurance</li> <li>• UC-P2-2: Geofencing boundaries enforcement</li> <li>• UC-P2-3: Danger zone restrictions enforcement</li> <li>• UC-P2-4: Construction site access control</li> <li>• UC-P2-5: Near-miss fall from height detection</li> <li>• UC-P2-7: Health and safety inspection support</li> <li>• UC-P3A-1: Fleet in-service emissions verification</li> <li>• UC-P3B-1: Vehicle's exterior condition documentation</li> </ul>
<b>Internal components</b>	Logging and auditing business logic (Smart Contracts), Hyperledger Fabric peers and orderers, Certificate Authorities (CAs), REST API

### 3.4.1.2. Structure, components and implementation technologies

This enabler logs 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 also logs resource requests and identified security events to help to provide digital evidence and resolve conflicts between stakeholders, when applicable. The structure and the components of the enabler is the following:

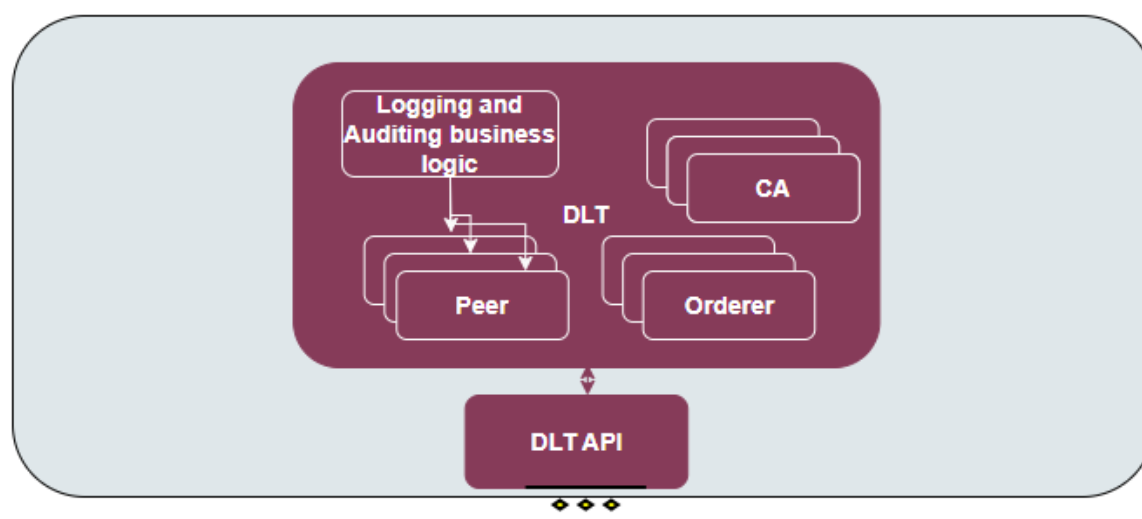


Figure 55. High-level diagram of the Logging and auditing enabler

Particularly, the Logging and auditing enabler is composed of 4 main components:

Table 58. Components and implementation of the Logging and auditing enabler

Component	Description	Technology/s
<b>Logging and Auditing business logic</b>	Contains the logic for storing the filtered logs in the DLT (within the state database)	Hyperledger Chaincode      Fabric (Smart Contracts)
<b>Peers and orderers</b>	Peers maintain a copy of the ledger, validate and endorse transactions, participate in the consensus, and execute chaincode. Orderers manage the ordering of transactions in the network.	Hyperledger peers, orderers      Fabric
<b>Certification Authorities (CAs)</b>	CAs manage identities of the network by issuing and managing digital certificates for organisations, peers, administrators, and clients	Hyperledger Certificate Authority (CA)      Fabric Authority
<b>DLT API</b>	This is a middleware between the clients and the blockchain network that forwards filtered logs to the DLT for storage.	REST API written with node.js and express.js

### 3.4.1.3. Communication interfaces

Table 59. API of the Logging and auditing enabler

Method	Endpoint	Description
POST	{ip}/api/logging/insert	Create logs
GET	{ip}/api/logging/gets	Get list of logs
POST	{ip}/api/logging/getbyid	Get log by specific id
POST	{ip}/api/logging/inserts	Create logs for location with information for: tag_id, lat, lon
GET	{ip}/api/logging/getlist	Get list of logs involving location
POST	{ip}/api/logging/getid	Get log by specific id( from the list of logs involving location)

### 3.4.1.4. Enabler stories

The **first enabler story** where this enabler is used, is the **immutable logging** of the configurations, the logging of security incidents and the logging of the resources access. The enabler story is depicted in the following diagram.

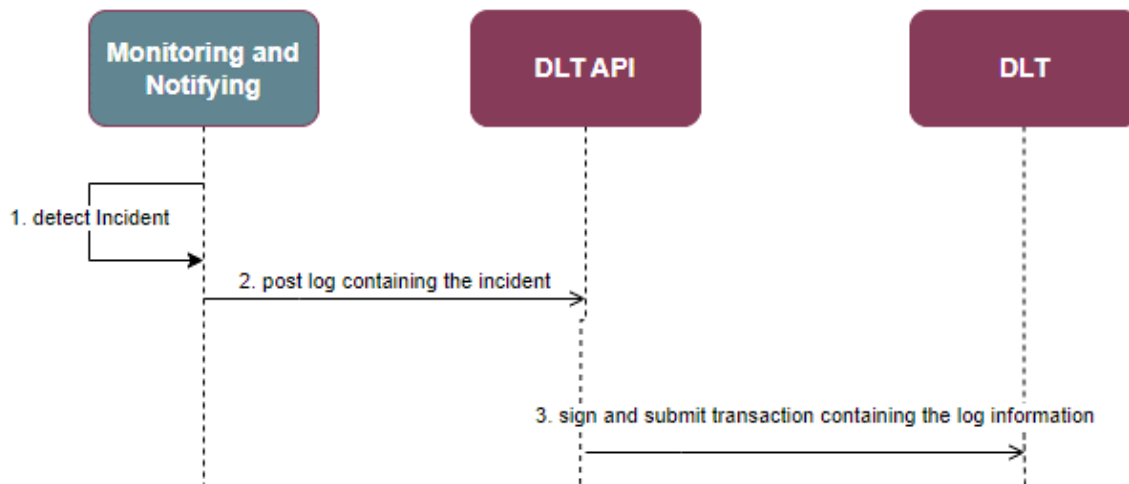


Figure 56. Logging and auditing enabler ESI (logging)

**STEP 1:** The interacting enabler (Monitoring and Notifying enabler) detects internally an incident.

**STEP 2:** The 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. Implementation information

Table 60. Implementation status of the Logging and auditing enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/logging_and_auditing_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/logging_and_auditing_enabler.html</a>
Potential features	-
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler is integrated with the Monitoring and notifying enabler

### 3.4.2. Data integrity verification enabler

#### 3.4.2.1. General specifications and features

Table 61. General information of the Logging and auditing enabler

Enabler	Data integrity verification enabler
<b>Id</b>	T54E2
<b>Owner and support</b>	CERTH
<b>Description and main functionalities</b>	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 contract (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.
<b>Key features</b>	DLT
<b>Vertical, related capabilities and features</b>	Security, privacy and trust
<b>Plane/s involved</b>	All planes – verification is performed specifically over data generated or processed at the different planes
<b>Requirements mapping</b>	<ul style="list-style-type: none"> <li>• R-C-6: Data Persistence and Trust</li> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-P3A-9: Edge Intelligence</li> </ul>
<b>Use case mapping</b>	<ul style="list-style-type: none"> <li>• UC-P3A-1: Fleet in-service emissions verification</li> </ul>
<b>Internal components</b>	Data integrity verification business logic (Smart Contracts), Hyperledger Fabric peers and orderers, Certificate Authorities (CAs), REST API

#### 3.4.2.2. Structure, components and implementation technologies

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. The structure and the components of the enabler appear in the figure below:

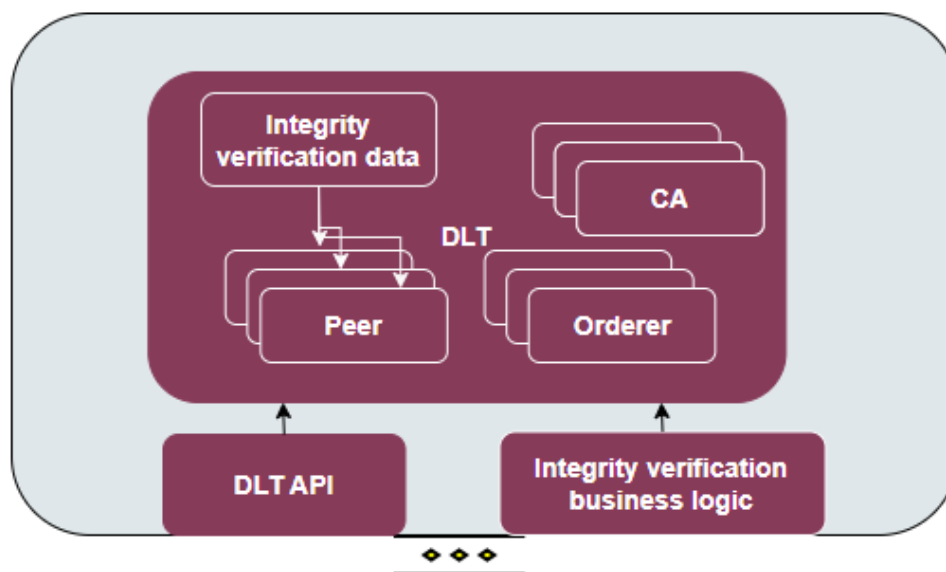


Figure 57. High-level diagram of the Data integrity verification enabler

Particularly, the integrity verification enabler is composed of 4 main components:



Table 62. API of the Data integrity verification enabler

Component	Description	Technology/s
<b>Integrity business logic</b>	Contains the logic for storing the hashed data in the DLT (within the state database)	Hyperledger Fabric Chaincode (Smart Contracts)
<b>Peers and orderers</b>	Peers maintain a copy of the ledger, validate and endorse transactions, participate in the consensus, and execute chaincode. Orderers manage the ordering of transactions in the network.	Hyperledger Fabric peers, orderers
<b>Certification Authorities (CAs)</b>	CAs manage identities of the network by issuing and managing digital certificates for organisations, peers, administrators, and clients	Hyperledger Fabric Certificate Authority (CA)
<b>DLT API</b>	This is a middleware between the clients and the blockchain network that prepares hashed data and forwards them to the DLT for storage. It also contains the integrity verification logic, i.e. provides a service that a client can use to compare hashed data at question against the hashed data that are stored in the DLT.	REST API written with node.js and express.js

### 3.4.2.3. Communication interfaces

Table 63. API of the Data integrity verification enabler

Method	Endpoint	Description
POST	{ip}/insert	Stores hashed data
POST	{ip}/getbyid	Returns if the hashes data exist in the ledger or not
POST	{ip}/images/insert	Stores hashed data of an image
POST	{ip}/images/getbyid	Returns the hash of the image with a specific id

### 3.4.2.4. Enabler stories

The enabler story where this enabler is used, is the **integrity verification of the data**, to detect their alteration; this enabler is used for storing on-chain data whose the integrity needs to be ensured. The enabler story is depicted in the following diagram.

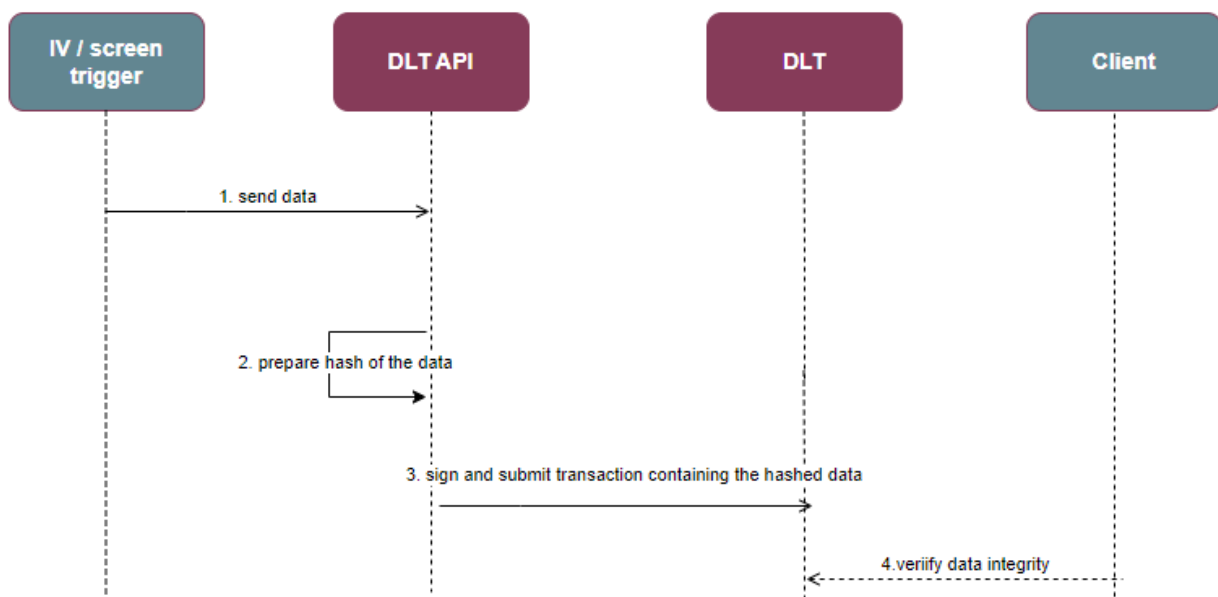


Figure 58. Data Integrity Verification enabler ES1 (data integrity check)

**STEP 1:** The interacting enabler (e.g. the IV enabler, the screen trigger enabler) sends to the DLT API the data.

**STEP 2:** The DLT API prepares the hash of the data.

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

**STEP 4:** The client queries the DLT API to verify the integrity of the data.

### 3.4.2.5. Implementation information

*Table 64. Implementation status of the Data integrity verification enabler*

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/data_integrity_verification_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/data_integrity_verification_enabler.html</a>
Potential features	-
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler works in a standalone mode

## 3.4.3. Distributed broker enabler

### 3.4.3.1. General specifications and features

*Table 65. General information of the Distributed broker enabler*

Enabler	<i>Distributed broker enabler</i>
Id	<i>T54E3</i>
Owner and support	<i>CERTH</i>
Description and main functionalities	This enabler provides a mechanism that facilitates data sharing among different heterogeneous IoT devices of the architecture. It deals with data source metadata management and provide trustable, findable, and retrievable metadata for the data sources.
Key features	<i>DLT</i>
Vertical, related capabilities and features	Security, privacy and trust
Plane/s involved	Device and edge plane – critical events are stored in the ledger Rest of the Planes – any critical action of the rest of the planes can be stored
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-P3A-1: Monitored Data channels*</li> <li>• R-P2-15: BIM data models and interoperability compliance</li> </ul>
Use case mapping	<ul style="list-style-type: none"> <li>• UC-P2-2: Geofencing boundaries enforcement</li> <li>• UC-P2-3: Danger zone restrictions enforcement</li> <li>• UC-P2-6: Safe navigation instructions</li> <li>• UC-P2-7: Health and safety inspection support</li> <li>• UC-P3A-1: Fleet in-service emissions verification*</li> <li>• UC-P3A-2: Vehicle non-conformance causes identification*</li> <li>• UC-P3A-3: Updating the diagnostics methods pool*</li> </ul>
Internal components	Distributed broker business logic (Smart Contracts), Hyperledger Fabric peers and orderers, Certificate Authorities (CAs), REST API

### 3.4.3.2. Structure, components and implementation technologies

This enabler provides a mechanism that facilitates data sharing among different heterogeneous IoT devices of the architecture. It deals with data source metadata management and provides trustable, findable, and retrievable metadata for the data sources.

Another functionality for the distributed broker enabler is the data source registration. The enabler serves as a trusted registry of the ASSIST-IoT devices that act as data producers. Indexing and querying services facilitates the efficient retrievability of the stored metadata of the registered producers by consumers in compliance with the FAIR principles. The structure and the components of the enabler appear in the figure below.

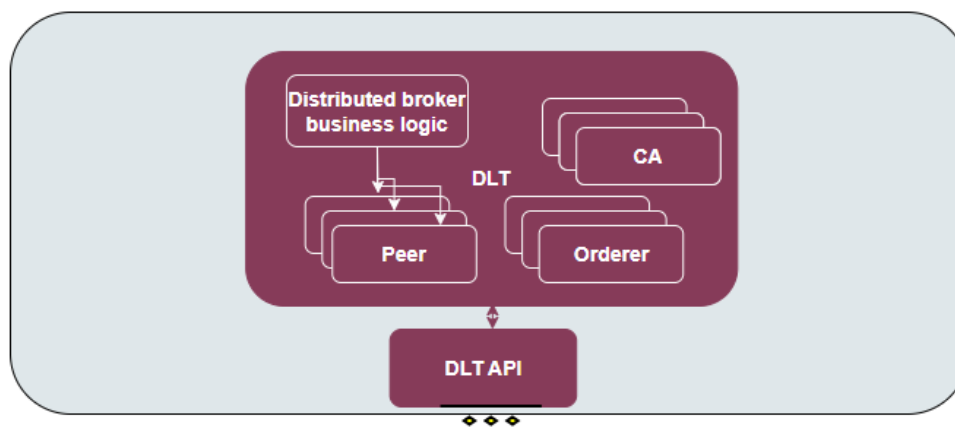


Figure 59. High-level diagram of the Distributed broker enabler

Particularly, the Distributed broker enabler is composed of 4 main components:

Table 66. Components and implementation of the Distributed broker enabler

Component	Description	Technology/s
<b>Distributed Broker business logic</b>	Contains the logic for storing the IoT device (data source/data producer) metadata in the DLT (within the state database)	Hyperledger Fabric Chaincode (Smart Contracts)
<b>Peers and orderers</b>	Peers maintain a copy of the ledger, validate and endorse transactions, participate in the consensus, and execute chaincode. Orderers manage the ordering of transactions in the network.	Hyperledger Fabric peers, orderers
<b>Certification Authorities (CAs)</b>	CAs manage identities of the network by issuing and managing digital certificates for organisations, peers, administrators, and clients	Hyperledger Fabric Certificate Authority (CA)
<b>DLT API</b>	This is a middleware between the clients and the blockchain network that signs and submits to the Blockchain/DLT transactions about incidents referring to IoT device (data source/data producer) metadata.	REST API written with node.js and express.js

### 3.4.3.3. Communication interfaces

Table 67. API of the Distributed broker enabler

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

### 3.4.3.4. Enabler stories

The **main enabler story** where this enabler is the **immutable logging of data source metadata**. It is depicted considering the following steps and diagram:

**STEP 1:** The interacting enabler (Monitoring and Notifying enabler) detects internally an incident referring to an IoT device (data source/data producer) metadata.

**STEP 2:** The Monitoring and Notifying enabler posts the log containing the incident information to the DLT API.

**STEP 3:** The DLT API (Blockchain Client) signs and submits to the Blockchain/DLT a transaction containing the log information

**STEP 4:** A data consumer can query the DLT for the device metadata before initiating the data exchange.

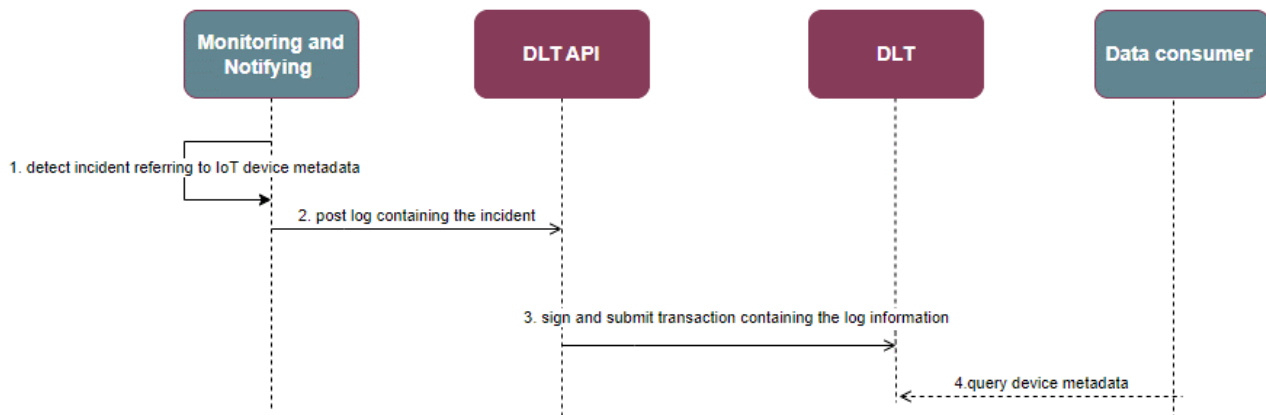


Figure 60. Distributed broker enabler ES1 (immutable metadata logging)

### 3.4.3.5. Implementation information

Table 68. Implementation status of the Distributed broker enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/distributed_broker_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/distributed_broker_enabler.html</a>
Potential features	-
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler is integrated with the Monitoring and notifying enabler

## 3.4.4. DLT-based FL enabler

### 3.4.4.1. General specifications and features

Table 69. General information of the DLT-based FL enabler

Enabler	DLT-based FL enabler
Id	T54E4
Owner and support	CERTH
Description and main functionalities	The DLT-based FL enabler is a system that provides a secure reputation mechanism for all local operators in a federated learning (FL) system. The reputation mechanism serves as a safeguard against free-riders and malicious adversaries, ensuring that only reputable local operators can contribute to the global model.
Key features	DLT
Vertical, related capabilities and features	Security, privacy and trust
Plane/s involved	<ul style="list-style-type: none"> <li>Device and edge plane – metadata and reputation scores are stored in the ledger</li> <li>Data management plane – metadata and reputation scores are stored in the ledger</li> </ul>
Requirements mapping	<ul style="list-style-type: none"> <li>R-C-7: Edge-oriented deployment</li> <li>R-C-22: Support for data privacy during the training process</li> <li>R-P3A-9: Edge Intelligence</li> <li>R-P3A-12: Edge Connectivity</li> </ul>

Enabler	DLT-based FL enabler
Use case mapping	<ul style="list-style-type: none"> <li>UC-P3A-2: Vehicle's non-conformance causes identification</li> <li>UC-P3B-1: Vehicle's exterior condition documentation</li> </ul>
Internal components	DLT-base FL business logic (Smart Contracts), Hyperledger Fabric peers and orderers, Certificate Authorities (CAs), REST API

### 3.4.4.2. Structure, components and implementation technologies

The DLT-based FL enabler is a system that provides a secure reputation mechanism for all local operators in a federated learning (FL) system. The reputation mechanism serves as a safeguard against free-riders and malicious adversaries, ensuring that only reputable local operators can contribute to the global model.

The integration of the DLT enabler with the FL baseline system involves the calculation of reputation scores for each FL local operator instance, which is stored on a permissioned blockchain network that allows only authorized users to access the scores and participate in the consensus algorithm that updates them. The final reputation score for each local operator is calculated using the cosine similarity between the weights of FL local operations and the aggregated weight. The FL training collector can query the reputation scores to decide on penalties or incentives for the FL local operations.

The DLT-based FL enabler consists of three components: the DLT communicator, the reputation score calculator, and the DLT storage. The DLT communicator is a RESTful API that receives weights from the training collector and fetches reputation scores from the DLT storage. The reputation score calculator applies the reputation mechanism and calculates scores for each local operator in each training round. The DLT storage component stores the reputation scores to the Distributed Ledger.

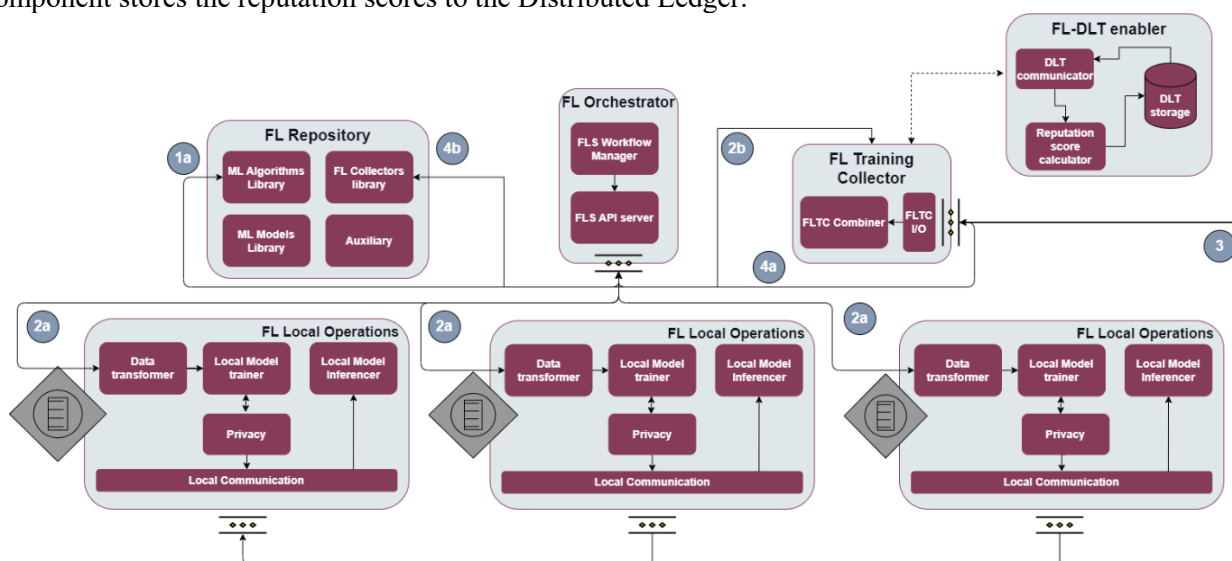


Figure 61. High-level diagram of DLT-based FL enabler

Particularly, the DLT-FL enabler is composed of 4 main components:

Table 70. Components and implementation of the DLT-based FL enabler

Component	Description	Technology/s
<b>Hyperledger Fabric Chaincode (Smart Contracts)</b>	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.	DLT-FL business logic
<b>Hyperledger Fabric peers, orderers</b>		Hyperledger Fabric peers and orderers
<b>Hyperledger Fabric Certificate Authority (CA)</b>		Certification Authorities (CAs)
<b>REST (Enabler's API)</b>	A popular web microframework written in JavaScript, FastAPI is known for being both robust and high performing. It is based on OpenAPI (previously Swagger) standards.	DLT communicator

### 3.4.4.3. Communication interfaces

Table 71. API of the DLT-based FL enabler

Method	Endpoint	Description
POST	{ip}/upload	Upload files with weights, aggregated and per client
GET	{ip}/FLDLT /gets	Get list of all the entries of each training round
POST	{ip}/FLDLT /getbyid	Get client list by specific id
GET	{ip}/FLDLT /getclients	Get list of all clients with their scores
POST	{ip}/FLDLT /getbyidclient	Get client's score by specific client id

### 3.4.4.4. Enabler stories

The **main enabler story** for which this enabler is used is the enforcement of a **secure reputation mechanism** for detecting malicious local operators. The enabler story is depicted in the following diagram.

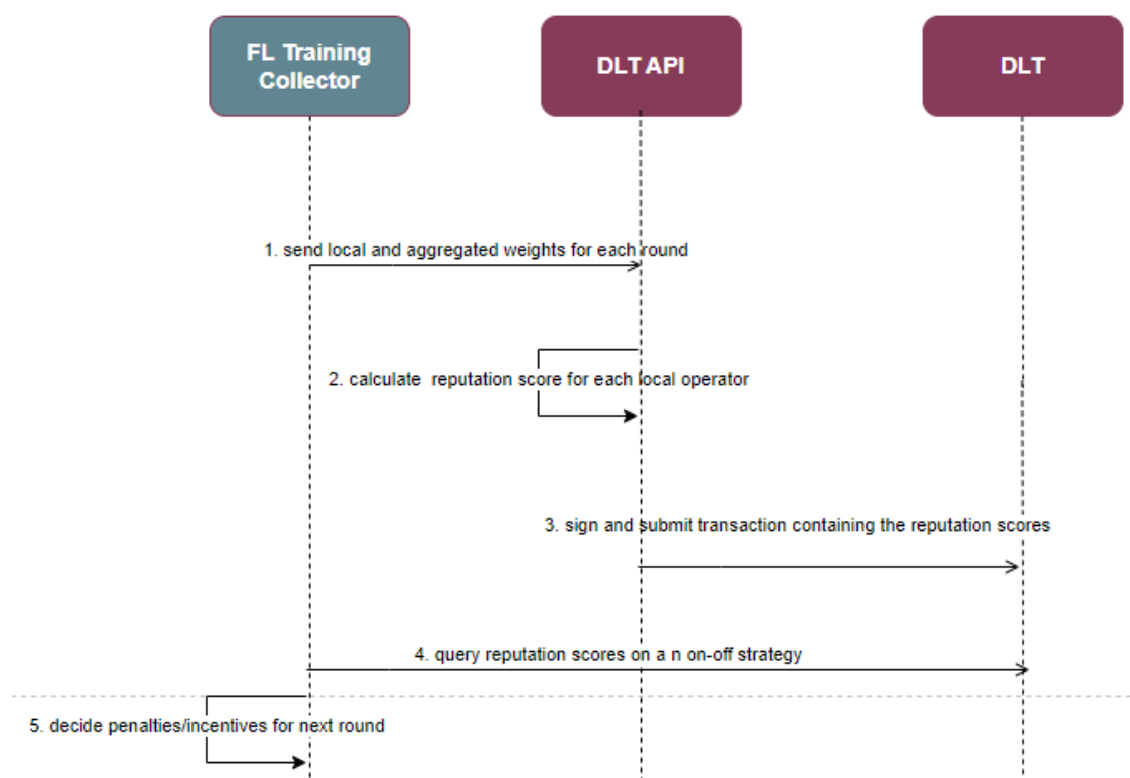


Figure 62. DLT-based FL enabler ES1 (secure reputation)

The steps of the flow are described below:

**STEP 1:** FL Training Collector sends the weights from the FL Local Operations and the weights from the global model to the Distributed Ledger (DLT).

**STEP 2:** The reputation score for each FL Local Operations is calculated using the cosine similarity between the weights of FL Local Operations and the aggregated weight.

**STEP 3:** The reputation score for each local operator is stored in the DLT along with the reputation set that contains the FL Local Operations who would be considered reputable in this round.

**STEP 4:** The FL Training Collector queries the DLT on an on-off strategy for the reputation scores and reputation set, so that further decisions on the penalties or incentives for the FL Local operations may be taken.

### 3.4.4.5. Implementation information

Table 72. Implementation status of the DLT-based FL enabler

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/dlt_based_fl_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/dlt/dlt_based_fl_enabler.html</a>
Potential features	-
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler is integrated with the FL Training Collector

## 3.5. Manageability enablers

### 3.5.1. Enablers manager

#### 3.5.1.1. General specifications and features

Table 73. General information of the Enablers manager

Enabler	Enablers manager
Id	T55E1
Owner and support	UPV
Description and main functionalities	This enabler acts as a central manager for the enablers of a specific deployment. In particular, it allows (i) registering, updating and deleting enabler repositories; (ii) deploying an enabler from a registered repository, as well as terminating, re-instantiating and eliminating them; and (iii) retrieving a list of currently-running enablers, with access to dedicated graphical management interfaces (when existing) as well as logs (if the enabler with log collection capabilities is in place).
Key features	<ul style="list-style-type: none"> <li>• Offers user-friendly access to all the features provided by the smart orchestrator.</li> <li>• Supports several kinds of repositories, from public to private, from GitHub to GitLab and self-hosted.</li> <li>• Allows keeping the persistent volume claims of terminated enablers, to keep data in case of re-deployment.</li> <li>• Provides a text-based form to specify the Helm values that need to be tailored before a specific enabler implementation.</li> </ul>
Vertical, related capabilities and features	Manageability
Plane/s involved	All planes – as it can manage enablers belonging to any of them.
Requirements mapping	<ul style="list-style-type: none"> <li>• R-C-7: Edge-oriented deployment</li> <li>• R-C-9: Workload placement</li> <li>• R-C-28: Distributed Configuration</li> </ul>
Use case mapping	All use cases will require of this enabler to be fulfilled
Internal components	User interfaces, Backend, Database

#### 3.5.1.2. Structure, components and implementation technologies

The Enablers manager is deployed along with the Tactile dashboard. As well as the rest of manageability enablers, it has been implemented using the same baseline frontend and backend technologies (see table below), for the sake of simplicity and realization. In this way, a single webserver is needed for host them, not four different ones.



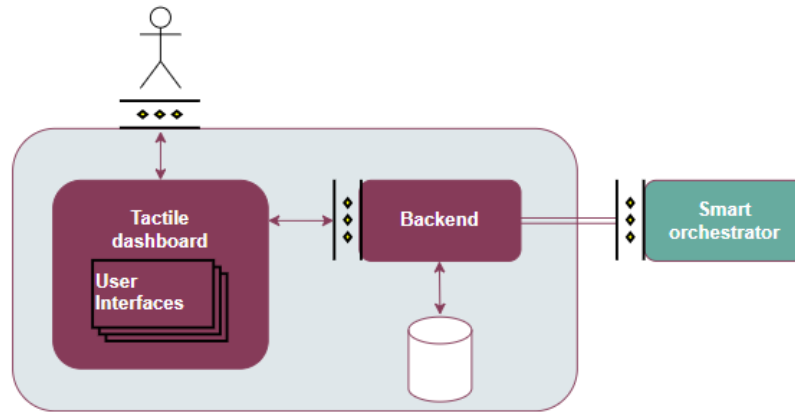


Figure 63. High-level diagram of the Enablers manager

Specifically, a description of each one of the components depicted is provided in the table below, along with the technologies used for implementing them:

Table 74. Components and implementation of the Enablers manager

Component	Description	Technology/s
<b>User interfaces (Frontend)</b>	Two graphical interfaces are part this enabler. One for managing enabler repositories (register, updating, deleting) and another for managing the lifecycle of the rest of enablers.	PUI9, Vue.js
<b>Backend</b>	The backend acts as an interface between the graphical interfaces and other services of a deployment (mainly, the smart orchestrator). It formats the received data to make requests in a suitable way.	PUI9, Java, Spring framework
<b>Database</b>	Manages the persistence of the backend's data.	PUI9, PostgreSQL

### 3.5.1.3. Communication interfaces

Being a webserver, users interact directly with the frontend. To load the pages, two endpoints are used by the frontend.

Table 75. API of the Enablers manager

Method	Endpoint	Description
GET	/dashboard/enabler	Enablers view of the frontend
GET	/dashboard/ helmrepository	Repositories view of the frontend

### 3.5.1.4. Enabler stories

The enabler stories of this enablers have suffered some modifications from the previous iterations. First, the flows for terminating and deleting an enabler have been merged into a single one. The reason to this separation initially was to have the possibility of launching again a terminated enabler which data was persisted, however, this has been changed to having the possibility of deleting or not its data when removing it. In case that the enabler is normally deployed again, it will reuse it. Secondly, the enabler story added in D5.4 does not longer make sense with the previous change, hence it has been removed. An additional story related to enabler upgrade to new version has been added. Also, the one about logs has been deleted as their status can be seen from the enablers list (detailed information can be consulted with the PUD enabler). And finally, four enabler stories have been added to represent the possibility of registering, listing, updating and deleting a Helm repository. Except the update operation, these flows were implemented previously, but have not been reported until now, having thus a total of **8 enabler stories**. The **first enabler story** is to **retrieve the list of registered Helm repositories**:

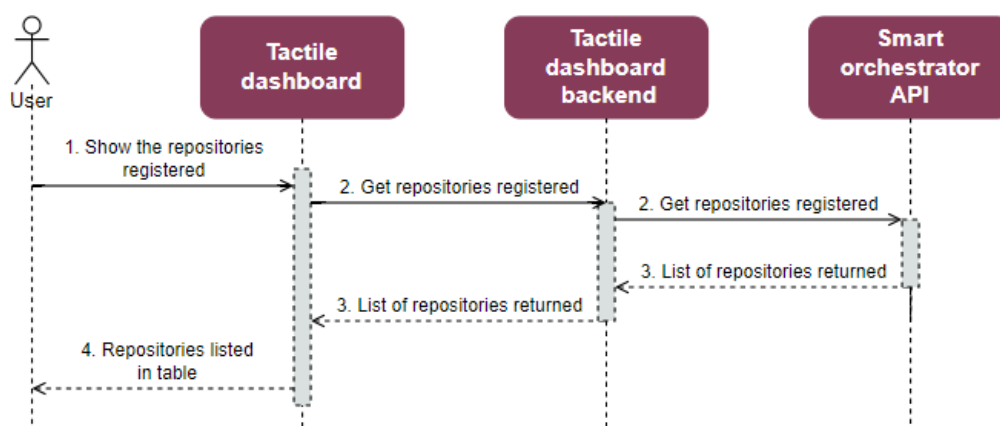


Figure 64. Enablers manager ES1 (list repositories)

**STEP 1:** The user interacts with the tactile dashboard and selects the repositories view on the menu, which shows the dedicated manageability interface.

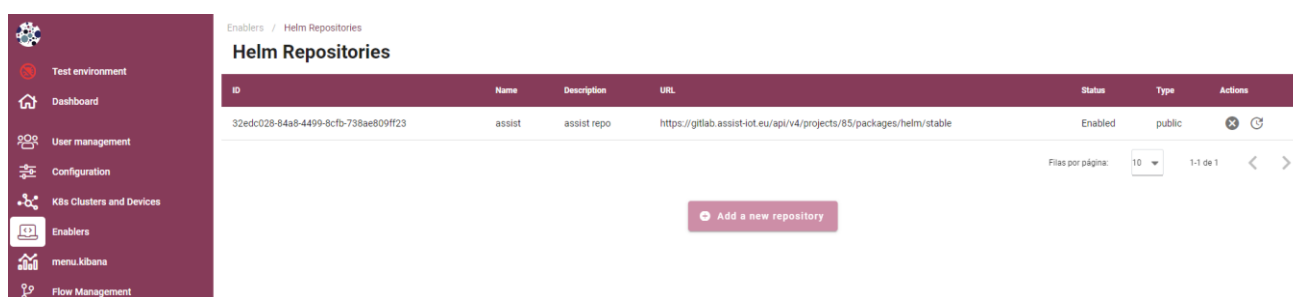


Figure 65. Repositories view of the enablers manager

**STEP 2:** The dashboard sends an HTTP GET request to its backend to obtain the list of registered repositories, which formats and forwards it to the Smart orchestrator API.

**STEP 3-4:** The orchestrator sends back an object with the registered repositories, which are properly shown by the manageability interface.

The **second one** occurs when a user **registers a Helm repository** (public and private options available). It consists of the following steps:

**STEP 1:** In the repositories view, the user clicks on the “Add new repository” button and fills in the “new repository” form.

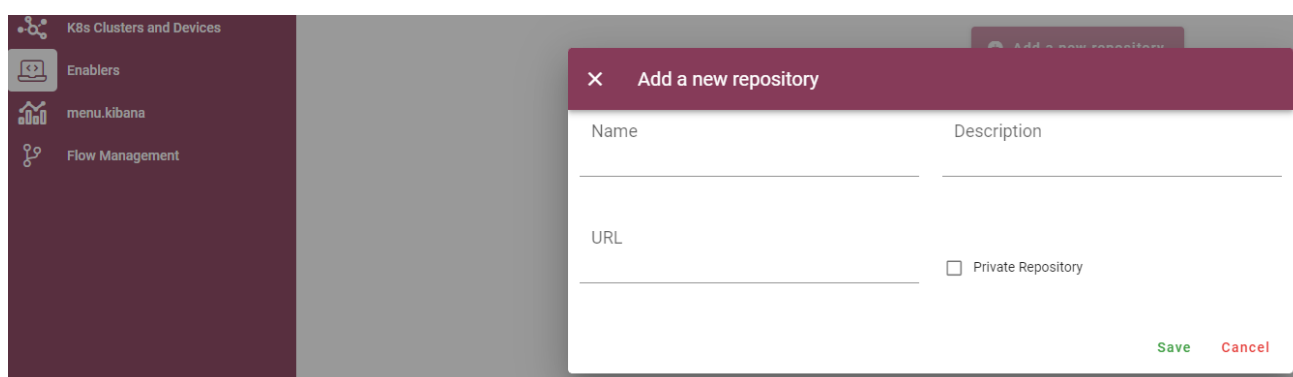


Figure 66. Form to add new repositories

**STEP 2:** The dashboard sends an HTTP POST request to its backend to register the new repository, which formats and forwards the request to the Smart orchestrator API.

**STEP 3-4:** If the repository has been registered successfully, the API confirms the operation to the backend and the dashboard shows to the user the updated list of repositories.

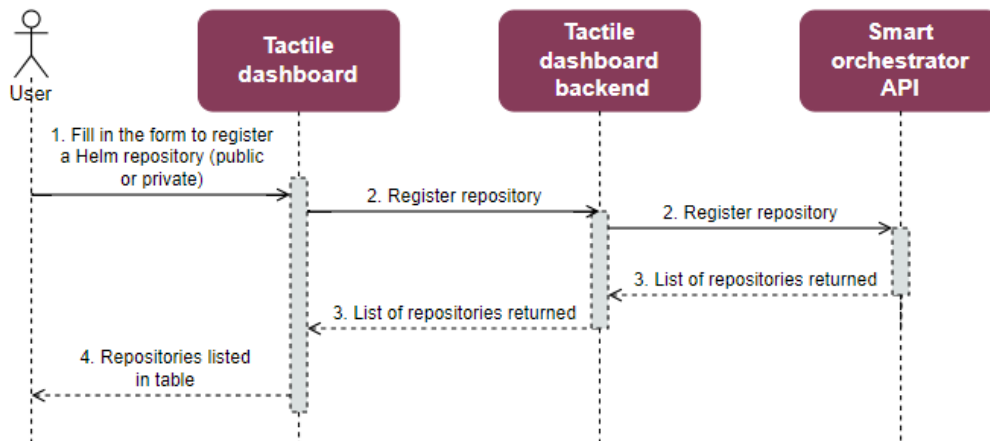



Figure 67. Enablers manager ES2 (register repository)

The **third enabler story** happens when a user wants to **update the list of Helm repositories**, needed for having the possibility of installing new enablers or newer versions of them. The diagram and involved steps are the following:

**STEP 1:** In the repositories view, the user clicks on the “Update repository” button, starting the flow. In this version, pushing that button on any repository updates all of them.

URL	Status	Type	Actions
<a href="https://gitlab.assist-iot.eu/api/v4/projects/85/packages/helm/stable">https://gitlab.assist-iot.eu/api/v4/projects/85/packages/helm/stable</a>	Enabled	public	

Filas por página: 10 1-1 de 1

Figure 68. Update repository button

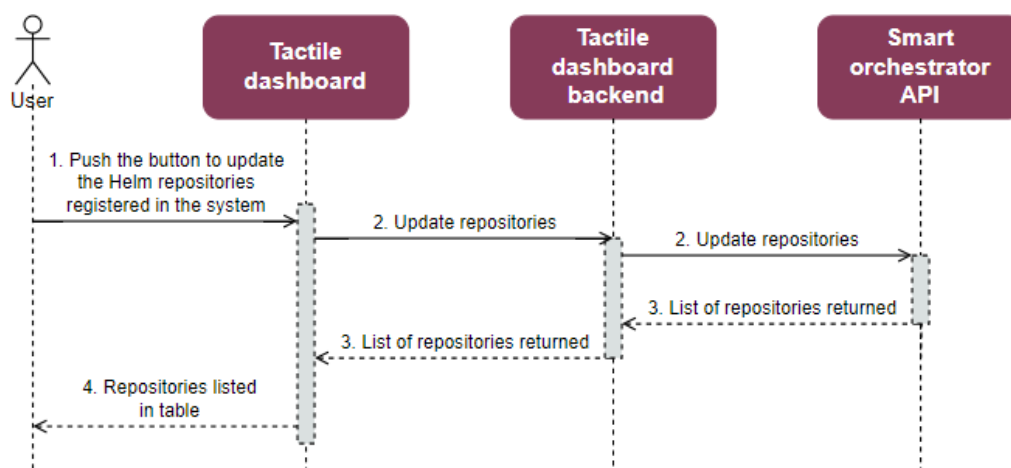




Figure 69. Enablers manager ES3 (update repositories)

**STEP 2:** The dashboard sends an HTTP POST request to its backend to update the repositories, which formats and forwards the request to the Smart orchestrator API.

**STEP 3-4:** If the repositories have been registered successfully, the API confirms the operation to the backend. The list should not suffer any change because of this operation.

The **fourth enabler story** is to **remove a Helm repository** from the registered ones. The diagram of the story is the following:

**STEP 1:** In the repositories view, the user clicks on the “Delete repository” button in the action column of the one that they want to remove, starting the flow.

URL	Status	Type	Actions
<a href="https://gitlab.assist-iot.eu/api/v4/projects/85/packages/helm/stable">https://gitlab.assist-iot.eu/api/v4/projects/85/packages/helm/stable</a>	Enabled	public	 

Filas por página: 10 1-1 de 1

Figure 70. Remove repository button

**STEP 2:** The dashboard sends an HTTP DELETE request to its backend to remove the selected repository, which formats and forwards the request to the Smart orchestrator API.

**STEP 3-4:** If the repository has been removed successfully, the API confirms the operation to the backend and the dashboard shows to the user the updated list of repositories (one less should appear).

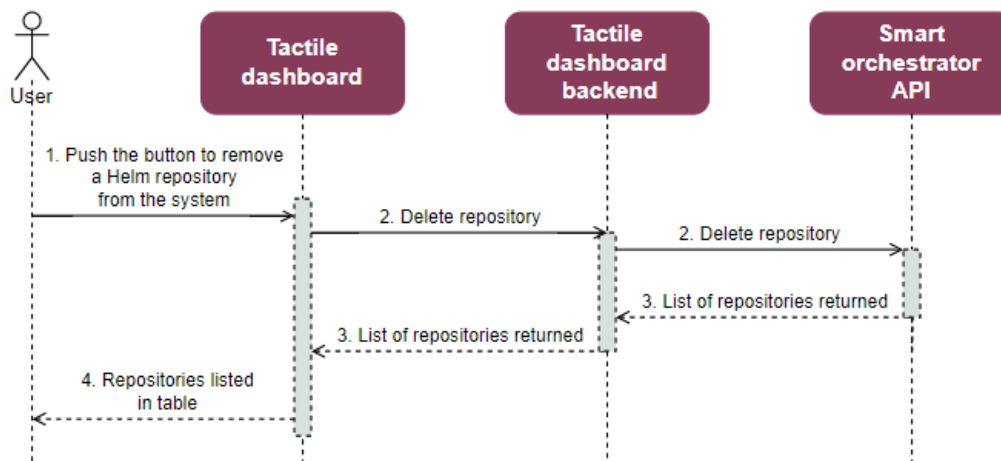


Figure 71. Enablers manager ES4 (delete repository)

The **fifth enabler story** is to **show the list of the deployed enablers in a table**. The diagram and related steps are the following:

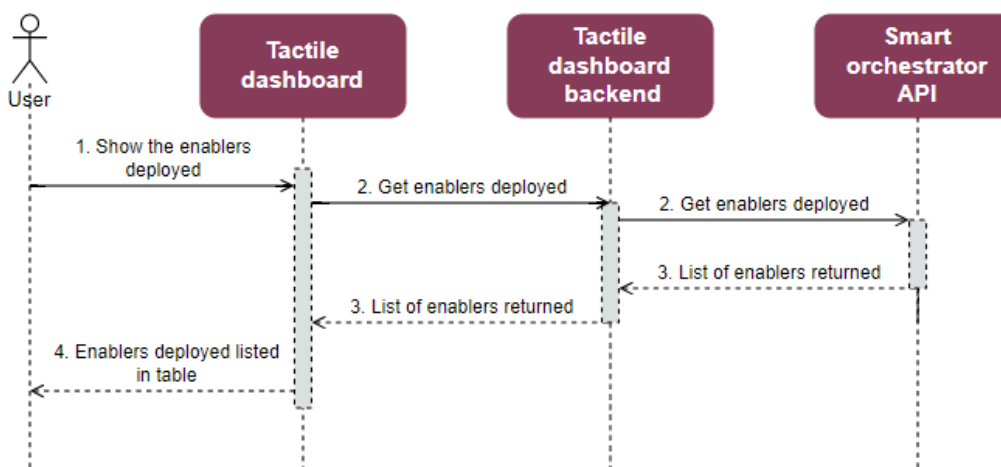


Figure 72. Enablers manager ES5 (list enablers)

**STEP 1:** The user interacts with the tactile dashboard and selects the enablers view on the menu, which shows the dedicated manageability interface.

Enablers / Enabler List

**Enabler List**

ID	Name	Helm Chart	Version	K8s Cluster	Enabler Status	Detailed Status	Actions
6b224397-c47f-4884-b0c1-c68e5f710f55	test	assist/vpn	1.1.0	bce538ab-e7fc-4448-9903-6f5bbb444b5d	Enabled		
0b387c07-326c-479b-8ce1-6a630916034d	test2	assist/idm	0.1.0	bce538ab-e7fc-4448-9903-6f5bbb444b5d	Enabled		
b24378b8-aa6f-41d3-9da3-8d7b2a04a5e0	test3	assist/edgedatabroker	0.1.2	fb94bd7d-021b-4d0d-b306-be5490a5033f	Pending		

Filas por página: 10 1-3 de 3 < >

[Add a new enabler](#)

Figure 73. Enablers view of the enablers manager

**STEP 2:** The dashboard sends an HTTP GET request to its backend to obtain the list of deployed enablers, which formats and forwards it to the Smart orchestrator API.

**STEP 3-4:** The orchestrator sends back an object with the deployed enablers, which are properly shown by the manageability interface.

The **sixth enabler story** is to **deploy a new enabler**. The diagram and involved steps are the following:

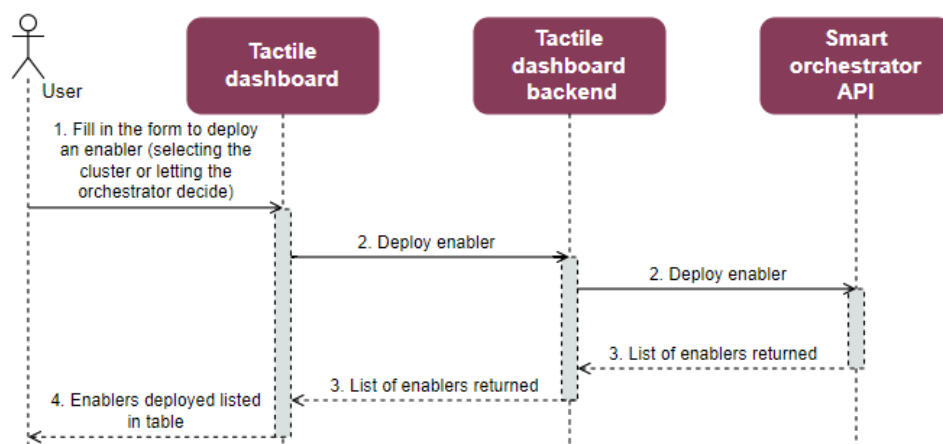


Figure 74. Enablers manager ES6 (deploy enabler)

**STEP 1:** In the enablers view, the user clicks on the “Add new enabler” button and fills in the “new enabler” form.

**Add a new enabler**

**General info**

Name: test3 ☐ Override Full Name

**Enabler selection**

Helm Chart Repository: assist Enabler: edgedatabroker

Enabler Versions: 0.1.2

**Deployment configuration**

☐ Automatic Cluster Selection K8s Cluster: edge

Timeout: 60

Additional Parameters

Figure 75. Form to add new enablers

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

**STEP 3-4:** If the enabler has been deployed successfully, the API confirms the operation to the backend and the dashboard shows to the user the updated list of enablers.

The **seventh enabler story** comes into play when a user aims at **upgrading an enabler**:

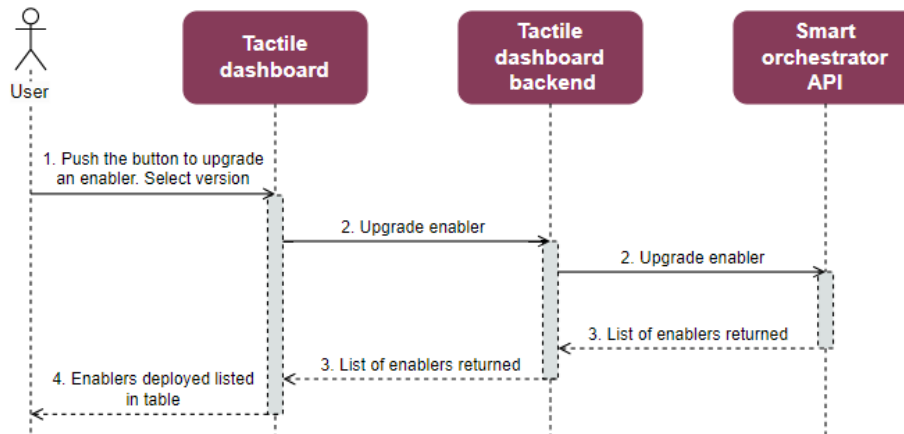


Figure 76. Enablers manager ES7 (upgrade enabler)

**STEP 1:** In the enablers view, the user clicks on the “Upgrade enabler” button in the action column of the enabler to update. A form appears, and after selecting the desired version, the flow starts.







K8s Cluster	Enabler Status	Detailed Status	Actions
bce538ab-e7fc-4448-9903-6f5bbb444b5d	Enabled		 
bce538ab-e7fc-4448-9903-6f5bbb444b5d	Enabled		 
fb94bd7d-021b-4d0d-b306-be5490a5033f	Pending		 

Figure 77. Upgrade enabler button

**STEP 2:** The dashboard sends an HTTP POST request to its backend to upgrade the enabler to the desired version, which formats and forwards the request to the Smart orchestrator API.

**STEP 3-4:** If the enabler has been upgraded successfully, the API confirms the operation to the backend. The only change that should happen in the list is the version of the enabler in the respective column.

The **eight enabler story** is to **delete a deployed enabler**. The diagram and involved steps are the following:

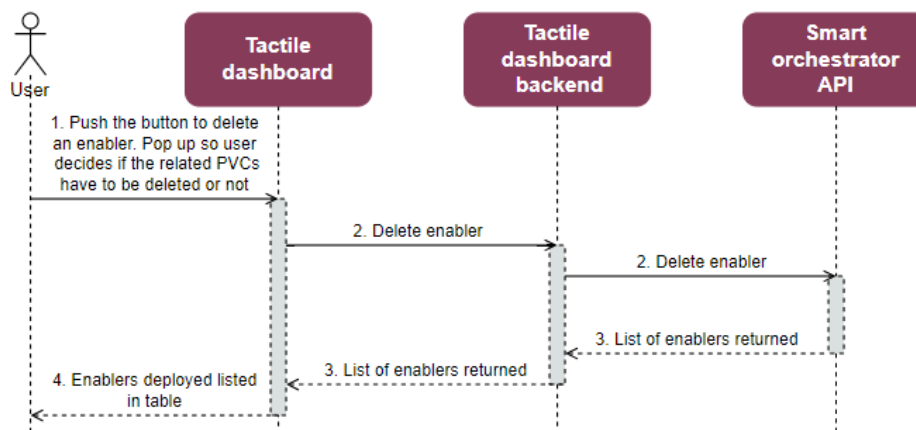








Figure 78. Enablers manager ES8 (delete enabler)

**STEP 1:** In the repositories view, the user clicks on the “Delete enabler” button in the action column of the one that they want to remove, starting the flow.

K8s Cluster	Enabler Status	Detailed Status	Actions
bce538ab-e7fc-4448-9903-6f5bbb444b5d	Enabled		 
bce538ab-e7fc-4448-9903-6f5bbb444b5d	Enabled		 
fb94bd7d-021b-4d0d-b306-be5490a5033f	Pending		 

Filas por página: 10 1-3 de 3 < >

Figure 79. Delete enabler button

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

**STEP 3-4:** If the enabler has been deleted successfully, the API confirms the operation to the backend and the dashboard shows to the user the updated list of enablers (one less should appear).

### 3.5.1.5. Implementation information

Table 76. Implementation status of the Enablers manager

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/registration_and_status_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/registration_and_status_enabler.html</a>
Potential features	<ul style="list-style-type: none"> <li>One feature to increase use experience would have been that the field for modifying the Helm values (the additional parameters section) would have been a form rather than a textbox in which put a JSON object. This would have entailed some modifications in the charts of all the enablers of the project.</li> <li>Another feature that will be included in time is the possibility to deploy a specific enabler in all the registered clusters, so that one does not have to go one by one for those enablers that may be deployed everywhere.</li> </ul>
Encapsulation readiness	Full functional Helm package ready
Integration with other enablers	This enabler is installed jointly with the Tactile dashboard enabler, and require at least of a previous deployment of the Smart orchestrator enabler.

## 3.5.2. Composite services manager

### 3.5.2.1. General specifications and features

Table 77. General information of the Composite services manager

Enabler	Composite services manager
Id	T55E3
Owner and support	UPV
Description and main functionalities	This enabler provides a graphical environment where ASSIST-IoT administrators can connect different enablers to compose a chain, or composite service, easing the realisation of data pipelines. To do so, this enabler is able to provision basic agents to ease data movement among other enablers, making the required protocol translations.
Key features	<ul style="list-style-type: none"> <li>Bridging protocols (HTTP &amp; MQTT)</li> <li>Offering graphical configuration possibilities</li> <li>Deploying these agents in the right spot of the computing continuum</li> </ul>



Enabler	Composite services manager
	<ul style="list-style-type: none"> <li>Abstracting IP addresses and port-related information of the involved service</li> </ul>
Vertical, related capabilities and features	Manageability
Plane/s involved	Data management plane – it will primarily work with enablers that manage data. Some enablers belonging to the Device and edge plane might also be involved.
Requirements mapping	<ul style="list-style-type: none"> <li>R-C-7: Edge-oriented deployment</li> <li>R-C-9: Workload placement</li> <li>R-C-28: Distributed Configuration</li> <li>R-C-29: Configurable data flows</li> </ul>
Use case mapping	This enabler will be deployed in all pilots, however, only some use cases will make active use of it. Particularly, UC-P2-1 to 4 in Pilot 2 and UC-P3A-1 in Pilot 1.
Internal components	Frontend, Backend

### 3.5.2.2. Structure, components and implementation technologies

The Composite services manager is deployed along with the Tactile dashboard. As well as the rest of manageability enablers, it has been implemented using the same baseline frontend and backend technologies (see table below), for the sake of simplicity and realization. In this way, a single webserver is needed for host them, not four different ones.

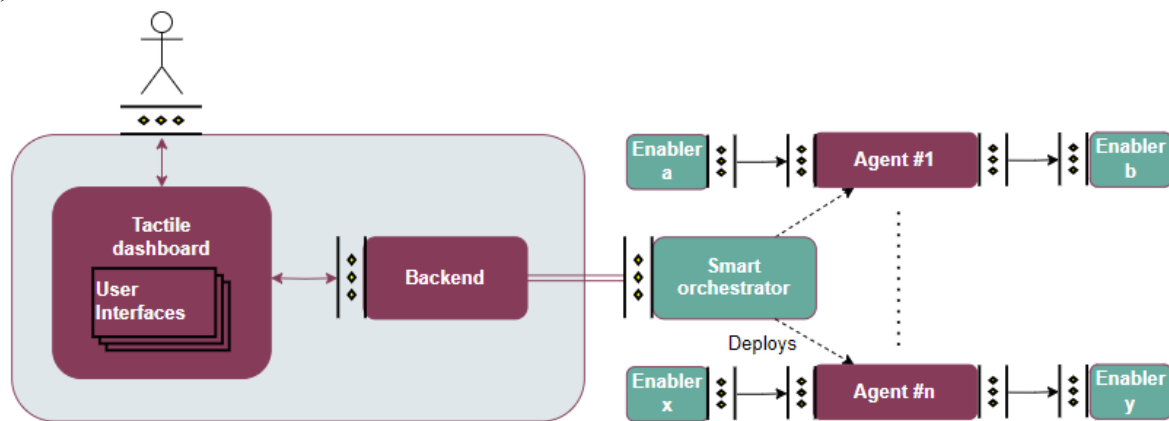


Figure 80. High-level diagram of the Composite services manager

Specifically, a description of each one of the components depicted is provided in the table below, along with the technologies used for implementing them:

Table 78. Components and implementation of the Composite services manager

Component	Description	Technology/s
Frontend	This component will be used by users to graphically setup the agents to interconnect services, generating a JSON file with data related to the involved services, protocols and payload transformations, if any. The dashboard will send the generated file to the backend, which will carry the proper actions.	PUI9, Vue.js, Node-RED
Backend	The backend will provision the needed agents, deploying them in the right spot of the continuum via the Smart orchestrator and configuring it based on the user's input.	PUI9, NodeJS, Java, Spring framework
Agent/s	Rather than components, they can be considered full-fledged enablers, as they are deployed as Helm charts, pre-configured by the backend. They are included here for the sake of simplicity, which the main goal of acting as middleware for changing data protocols (MQTT to HTTP, HTTP to MQTT, event-based or periodical HTTP to HTTP, etc.).	NodeJS

### 3.5.2.3. Communication interfaces

Being a webserver, users interact directly with the frontend. To load the page, one endpoint is used by the frontend. Regarding agents, they make use of the interfaces (MQTT, HTTP) of the enablers involved.

Table 79. API of the Composite services manager

Method	Endpoint	Description
GET	/composite-services-manager	Node-RED-based view of the frontend

### 3.5.2.4. Enabler stories

The enabler stories remain the same as in the previous deliverable version. Four stories have been identified, being the **first one listing the existing pipelines** present in the system. From this list, a user can add pipelines with new agents, modify or delete existing ones.

**STEP 1:** The user interacts with the tactile dashboard and selects the composite services view in the menu, which shows the dedicated manageability interface.

**STEP 2:** The dashboard sends an HTTP POST request to its backend to get the current list of active pipelines.

**STEP 3-4:** Then, the backend returns the information about the active pipelines to the Smart Orchestrator API.

**STEP 4:** The answer from the Orchestrator is sent to the backend, which is properly shown by the manageability interface.

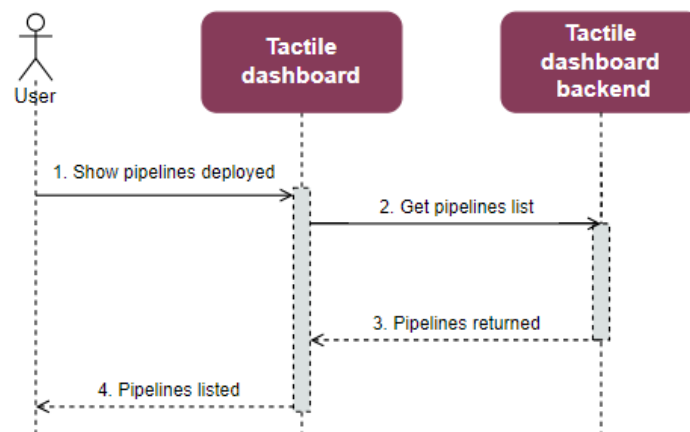


Figure 81. Composite services manager ES1 (list pipelines)

The **second enabler story** is related to the **deployment of new pipelines**, and in this case the interaction is the following:

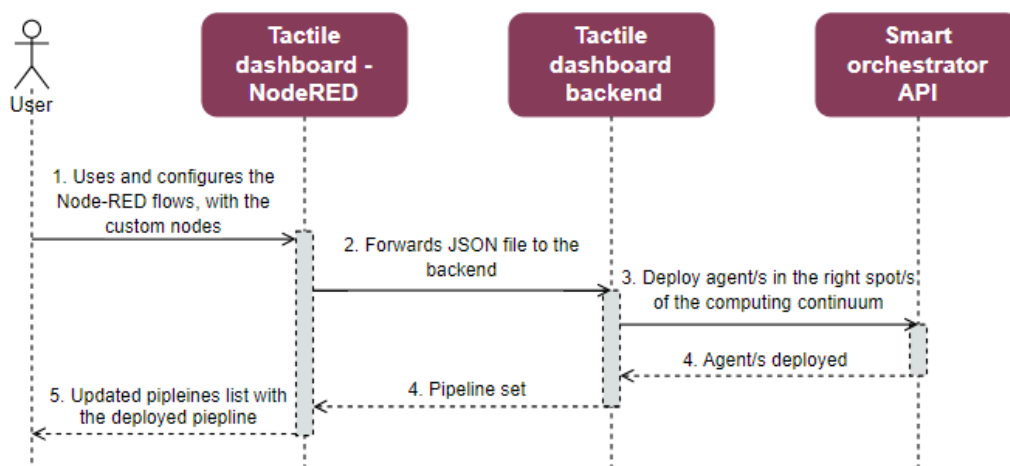


Figure 82. Composite services manager ES2 (deploy pipeline)

**STEP 1:** The user interacts with the tactile dashboard, specifically with the graphical Node-RED interface, and sets up a pipeline (selecting the nodes, the involved input-output protocols, topics and endpoints, when needed).

**STEP 2:** The dashboard sends an HTTP POST request to its backend to deploy the required agents. To that end, the backend interprets the JSON file received from the frontend and translates it to commands that the orchestrator can manage.

**STEP 3:** The backend asks to deploy the required agent/s to the orchestrator.

**STEP 4:** Once these are deployed, a confirmation message is sent to the backend, and the pipeline is correctly set.

**STEP 5:** If the pipeline has been deployed successfully, the dashboard shows to the user the updated list of pipelines.

The **third enabler story** comes to play when a **pipeline** is to be **deleted**. In this case, the flow and involved steps are the ones indicated below:

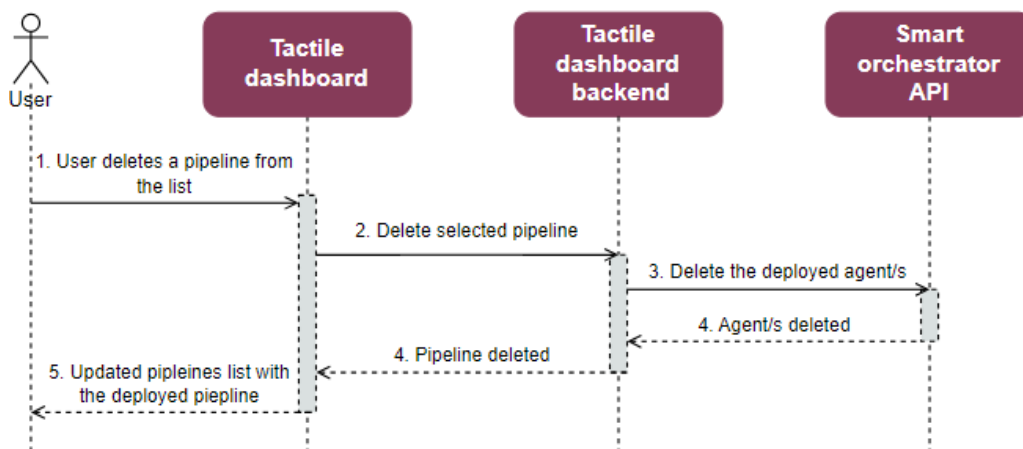


Figure 83. Composite services manager ES3 (delete pipeline)

**STEP 1:** The user interacts with the tactile dashboard, specifically with the list of deployed pipelines, to delete one of them.

**STEP 2:** The dashboard sends an HTTP POST request to its backend to remove the involved agent/s. Then, the backend recovers the data related to this pipeline.

**STEP 3:** The backend asks to delete the required agent/s to the orchestrator.

**STEP 4:** Once these are deleted, a confirmation message is sent to the backend, and the pipeline is correctly removed.

**STEP 5:** If the pipeline has been deleted successfully, the manageability interface shows to the user the updated list of pipelines. It should not include the recently-removed pipeline.

The **fourth** and last **enabler story** corresponds to the **modification** of an **existing pipeline**. In this moment, the flow is still the same as in D5.4, consisting in a deletion of the previous version (ES#3) and followed by the creation of a new pipeline (ES#2). It has been decided that in case of (graphical) modification of the pipeline, it is easier and more robust to follow this steps than to implement logic to decide when and how to reconfigure and reallocate the deployed agents, which could have many ramified implications.

### 3.5.2.5. Implementation information

Table 80. Implementation status of the Composite services manager

Category	Status
Link to ReadtheDocs	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/management_of_services_and_enablers.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/management_of_services_and_enablers.html</a>

Category	Status
<b>Potential features</b>	<p>This enabler has room for improvement, as the number of agents could be largely increased. In essence:</p> <ul style="list-style-type: none"> <li>• Only MQTT and HTTP protocols are managed. Agents for dedicated technologies (like Kafka) could have been developed, as some project enablers make use of it.</li> <li>• Finally, the enabler's agents do not provide any kind of payload transformation. In case this is needed, semantic enablers could be included in the pipeline, requiring of work to defining the needed data models / ontologies.</li> </ul>
<b>Encapsulation readiness</b>	Full functional Helm package ready
<b>Integration with other enablers</b>	This enabler is installed jointly with the Tactile dashboard enabler, and require at least of a previous deployment of the Smart orchestrator enabler. Agents will require that the enablers they connect are properly deployed (generally, those from the Data management plane, e.g., LTSE, EDBE).

### 3.5.3. Clusters and topology manager

#### 3.5.3.1. General specifications and features

Table 81. General information of the Cluster and topology manager

Enabler	Cluster and topology manager
<b>Id</b>	T5534
<b>Owner and support</b>	UPV
<b>Description and main functionalities</b>	Integrated in the tactile dashboard, the main functionality of this enabler is to register new clusters of computing nodes (or a single computing node) in an ASISST-IoT deployment, and present an overview of its topology. These nodes require to have a functional and accessible K8s deployment.
<b>Key features</b>	<ul style="list-style-type: none"> <li>• It allows monitoring any registered node in the deployment, including its status (i.e., availability and used resources) and current instantiated enablers.</li> </ul>
<b>Vertical, related capabilities and features</b>	Manageability
<b>Plane/s involved</b>	<ul style="list-style-type: none"> <li>• Device and edge plane – addition of computing node to an ASSIST-IoT deployment.</li> <li>• Smart network and control plane – manageability actions related to K8s networking and topology management.</li> <li>• Application and services plane – as a service for system administrators, it will allow deploying enablers in specific computing nodes.</li> </ul>
<b>Requirements mapping</b>	<ul style="list-style-type: none"> <li>• RC7: Edge-oriented deployment</li> </ul>
<b>Use case mapping</b>	This enabler will be present at all use cases, for administration purposes
<b>Internal components</b>	User interfaces, Backend, Database

#### 3.5.3.2. Structure, components and implementation technologies

The Clusters and topology manager is deployed along with the Tactile dashboard. As well as the rest of manageability enablers, it has been implemented using the same baseline frontend and backend technologies (see table below), for the sake of simplicity and realization. In this way, a single webserver is needed for host them, not four different ones.

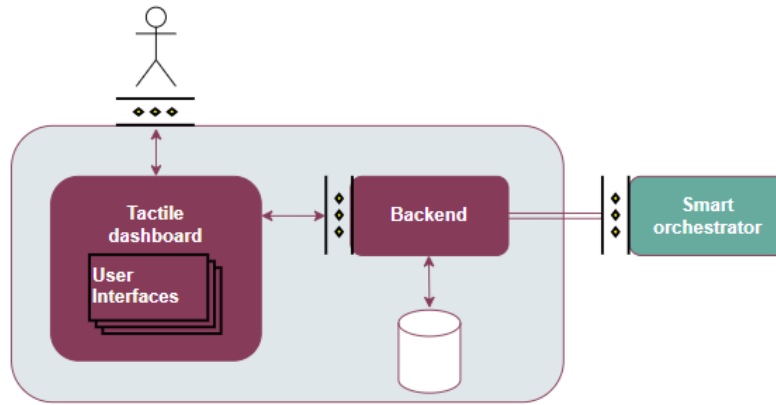


Figure 84. High-level diagram of the Cluster and topology manager

Specifically, a description of each one of the components depicted is provided in the table below, along with the technologies used for implementing them:

Table 82. Components and implementation of the Cluster and topology manager

Component	Description	Technology/s
<b>User interfaces (Frontend)</b>	Two graphical interfaces are part this enabler. One for managing the clusters (register, deleting) and another for presenting the overall topology, with the possibility of seeing the enablers deployed on each registered cluster.	PUI9, Vue.js
<b>Backend</b>	The backend acts as an interface between the graphical interfaces and other services of a deployment (mainly, the smart orchestrator). It formats the received data to make requests in a suitable way.	PUI9, Java, Spring framework
<b>Database</b>	Manages the persistence of the backend's data.	PUI9, PostgreSQL

### 3.5.3.3. Communication interfaces

Being a webserver, users interact directly with the frontend. To load the pages, two endpoints are used by the frontend.

Table 83. API of the Cluster and topology manager

Method	Endpoint	Description
GET	/k8scluster	K8s clusters view of the dashboard
GET	/clustertopology	K8s clusters topology view of the dashboard

### 3.5.3.4. Enabler stories

As in the previous version, there are 5 enabler stories that apply to this enabler. There have been some modifications in the code to optimise them, but overall the flow remains the same from a user perspective. The **first enabler story** is to **list the registered K8s clusters**. The diagram and the involved steps are the following:

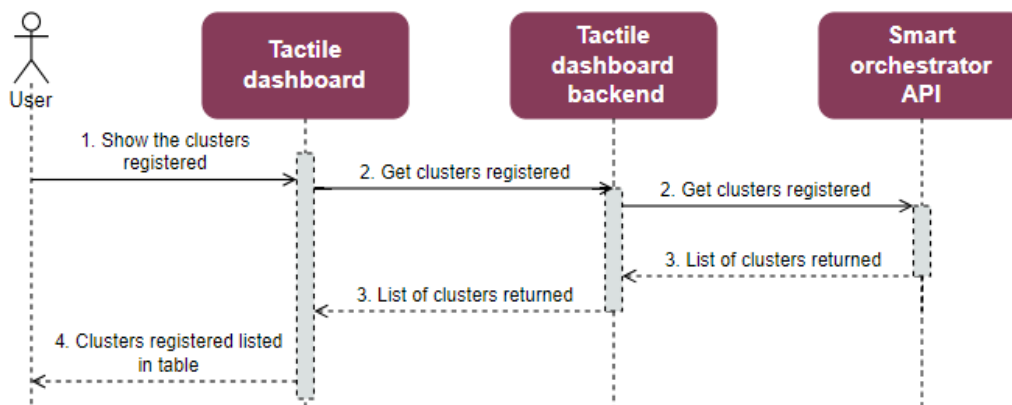


Figure 85. Clusters and topology manager ES1 (list clusters)

**STEP 1:** The user interacts with the tactile dashboard and selects the K8s clusters view on the menu, which shows the dedicated manageability interface.

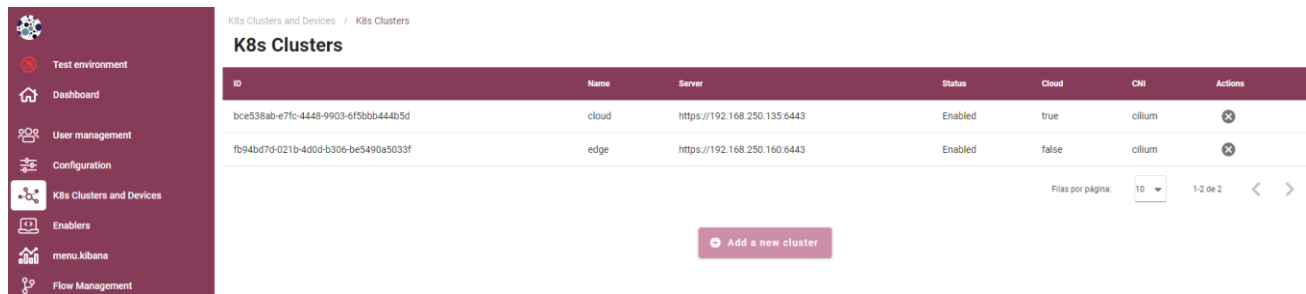


Figure 86. K8s clusters view of the clusters and topology manager

**STEP 2:** The dashboard sends an HTTP GET request to its backend to obtain the list of registered clusters, which formats and forwards it to the Smart orchestrator API.

**STEP 3-4:** The orchestrator sends back an object with the registered clusters, which are properly shown by the manageability interface.

The **second enabler story** takes place when a user aims at **registering a K8s cluster** in the system. The diagram and related steps are the following:

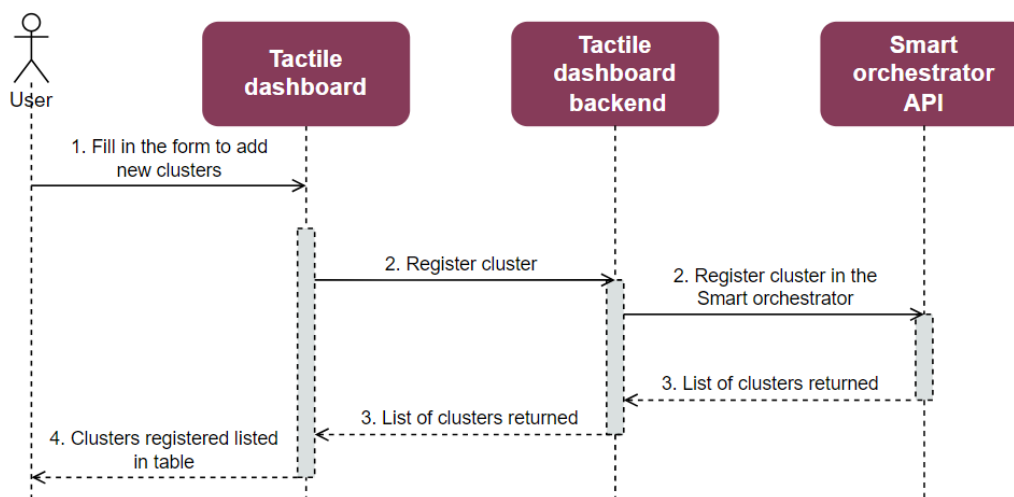


Figure 87. Clusters and topology manager ES2 (register cluster)

**STEP 1:** In the repositories view, the user clicks on the “Add new cluster” button and fills in the “new cluster” form.

**STEP 2:** The dashboard sends an HTTP POST request to its backend to register the new cluster, which formats and forwards the request to the Smart orchestrator API.

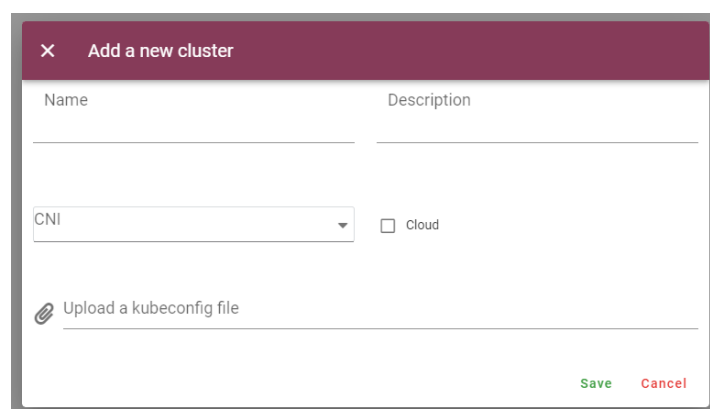


Figure 88. Form to register new clusters

**STEP 3-4:** If the cluster has been registered successfully, the API confirms the operation to the backend and the dashboard shows to the user the updated list of clusters.

The **third enabler story** is to **delete a registered K8s cluster**. The diagram and related steps are the following:

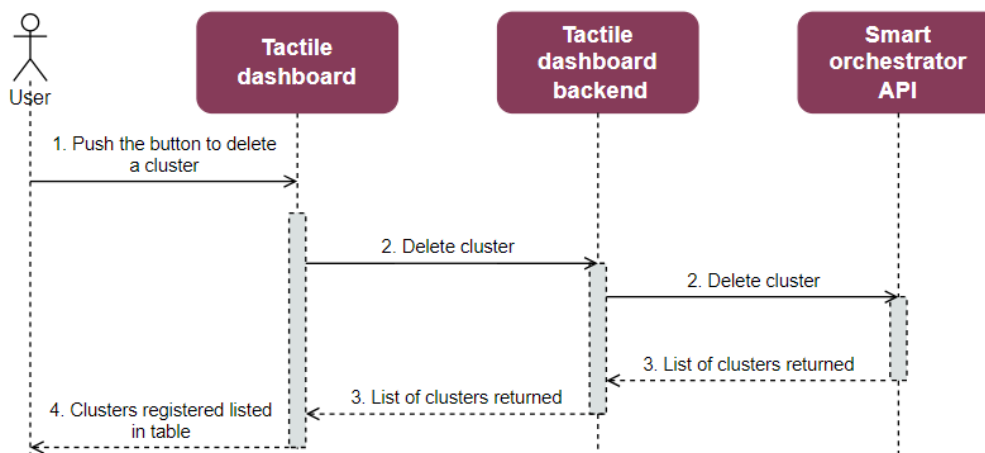


Figure 89. Clusters and topology manager ES3 (delete cluster)

**STEP 1:** In the clusters view, the user clicks on the “Delete cluster” button in the action column of the one that they want to remove, starting the flow.

Status	Cloud	CNI	Actions
Enabled	true	cilium	
Enabled	false	cilium	

Filas por página: 10 1-2 de 2 < >

Figure 90. Delete cluster button

**STEP 2:** The dashboard sends an HTTP DELETE request to its backend to remove the selected clusters, which formats and forwards the request to the Smart orchestrator API. This can only be done if the cluster does not have any enabler deployed (unless forced by the user).

**STEP 3-4:** If the cluster has been removed successfully, the API confirms the operation to the backend and the dashboard shows to the user the updated list of clusters (one less should appear).

The **fourth enabler story** consists in **depicting the topology** of the deployment, including the clusters and the nodes managed by the system as well as showing the enablers deployed in each computing node. Its flow and steps are the following ones:

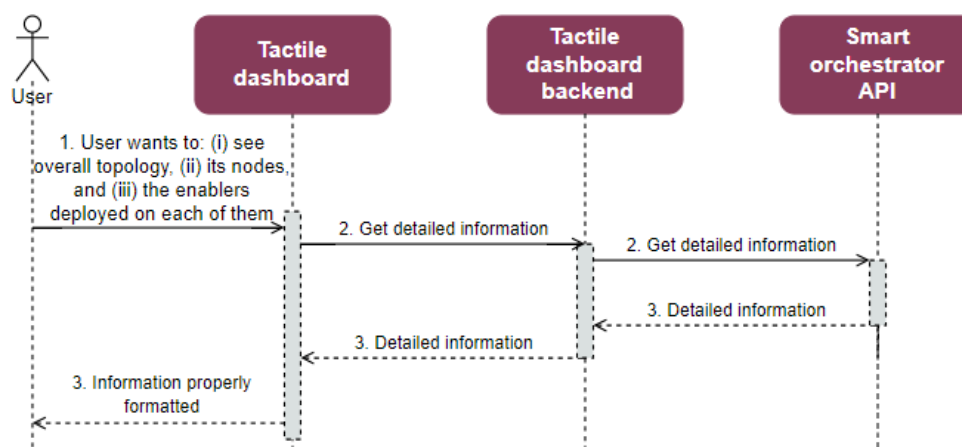


Figure 91. Clusters and topology manager ES4 (depict topology)



**STEP 1:** The user interacts with the tactile dashboard, particularly to the topology view, to get topology information (if clicking on a computing node, the interface will show additional information from it, as the enablers deployed on it). It presents the clusters available (in this case, cloud and edge) and the nodes of each cluster (“smart” node in edge cluster, “smart2” node in the cloud one).

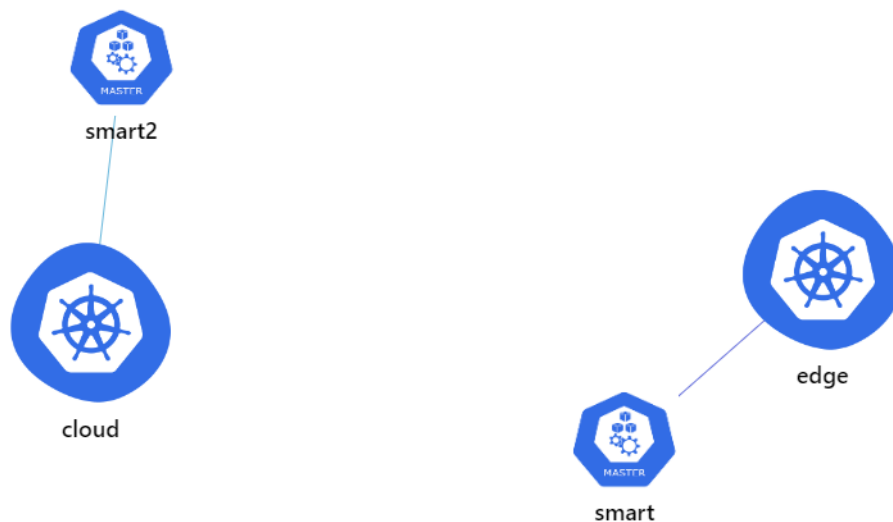


Figure 92. Topology view of the clusters and topology manager

**STEP 2:** The dashboard sends an HTTP POST request to its backend to gather the requested data from the Smart orchestrator.

**STEP 3:** Once the data is available, the dashboard formats it conveniently.

Finally, the **fifth enabler story** consists in the possibility of manually **deploying an enabler in a specific node of the topology**, with the interface of the previous story as starting point. Its flow is as follows:

**STEP 1:** The user interacts with the tactile dashboard, specifically with the topology view, to select a computing node to deploy an enabler on it. The form is simplified with respect to the enabler story #6 from the enablers manager.

**STEP 2:** The form to deploy enablers is shown to the user, but with some fields locked and filled with default information. Then, the enablers manager #ES5 takes over.

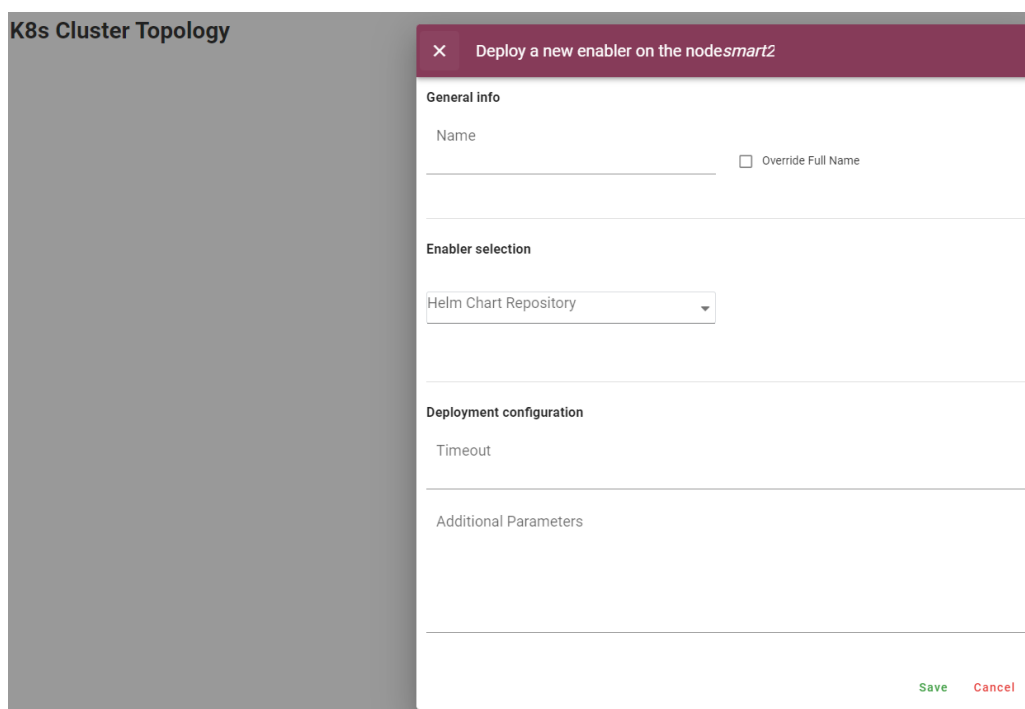


Figure 93. Form to add new enabler in specific node

### 3.5.3.5. Implementation information

*Table 84. Implementation status of the Cluster and topology manager*

Category	Status
<b>Link to ReadtheDocs</b>	<a href="https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/devices_management_enabler.html">https://assist-iot-enablers-documentation.readthedocs.io/en/latest/verticals/manageability/devices_management_enabler.html</a>
<b>Potential features</b>	This enabler has focused only in manually-installed K8s and K3s clusters. Tests and adaptations might be needed (also in the Smart orchestrator) for supporting also other K8s-based distributions.
<b>Encapsulation readiness</b>	Full functional Helm package ready
<b>Integration with other enablers</b>	This enabler is installed jointly with the Tactile dashboard enabler, and require at least of a previous deployment of the Smart orchestrator enabler.

## 4. Enabler's Technical Documentation and Demo Videos

The Technical Documentation (<https://assist-iot-enablers-documentation.readthedocs.io/en/latest/>) for all the aforementioned enablers is available on the Read the Docs platform. This documentation is encapsulated in the final Deliverable [D6.6 – Technical and Support Documentation – Final](#), which represents a comprehensive analysis of the provided documentation (updates are also provided through the final WP6 deliverable D6.8). Its primary objective is to furnish users with essential information concerning the deployment and utilisation of ASSIST-IoT enablers across both the horizontal and vertical facets of the ASSIST-IoT architecture. The Technical Documentation is thoughtfully structured around the overarching ASSIST-IoT architecture, adhering to a general approach encompassing the following key sections: Introduction, Features, Placement within the Architecture, User Guide, Prerequisites, Installation, Configuration Options, Developer Guide, Version Control and Release, Licensing, and Notices.

Selected enablers are also showcased through videos available on the official YouTube channel of the ASSIST-IoT project (<https://www.youtube.com/@assist-iot>). The primary objective of this endeavour is twofold: firstly, to illustrate the accomplishments achieved and to provide external audiences with insights into the technical intricacies implemented within the project at enablers level, thereby expanding our YouTube channel views and subscribers base. Secondly, these videos aim to highlight the Consortium's preparations for the final pilot trials and the practical execution of pilot work, bridging the gap between theory and application. Ultimately, these videos serve as a testament on how these enablers can be integrated into other components, projects or even within a commercial service/product context.

## 5. Conclusions

This document provides an update and extension of the specifications provided in the previous two iterations of this deliverable series constituting a final development of enablers proposed in WP5. This deliverable provides insight about the implementation, outcomes, design and technical information for the transversal enablers, extending concepts introduced in deliverables D5.1, D5.2, D5.3 and D5.4 being the final deliverable in the WP. The software outcomes of WP5 have been integrated and validated in pilots. Specifically, most of them (besides identified exceptions) are containerised, integrated with K8s (manifests ready) and prepared for packaging (in Helm charts). At this stage, the implementation, design and software structure of enablers are finalised. The only modifications can come as a result of further validation and verification. The enablers developed so far allow for continuing efforts related implementing them in pilots (WP7) for further validation and assessment (WP8), either fully or partially packaged.