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

D6.3 - Testing and integration plan - Final

Deliverable No.	D6.3	**Due Date**	30/04/2023
Type	Report	**Dissemination Level**	Public
Version	1.0	**WP**	WP6
Description	Includes testing plan, to be followed for all components belonging. The initial release outlines the initial plan, while second version will include plan update and results.		

Copyright

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

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

Disclaimer

Authors

Name	Partner	e-mail
Alejandro Fornés	P01 UPV	alforlea@upv.es
Rafael Vañó Garcia	P01 UPV	ravagar@upv.es
Raúl Reinosa	P01 UPV	rreisim@upv.es
Juan Gascón	P01 UPV	juagasre@upv.es
Eduardo Garro	P02 PRO	egarro@prodevelop.es
Juan Antonio Pavón	P02 PRO	jpavon@prodevelop.es
Adrián Ramos	P02 PRO	aramos@prodevelop.es
Piotr Sowinski	P03 IBSPAN	psowinski@ibspan.waw.pl
Paweł Szmeja	P03 IBSPAN	pawel.szmeja@ibspan.waw.pl
Karolina Bogacka	P03 IBSPAN	bogacka@ibspan.waw.pl
Evripidis Tzionas	P04 CERTH	tzionasev@iti.gr
Iordanis Papoutsoglou	P04 CERTH	ipapoutsoglou@iti.gr
Anastasia Blitsi	P04 CERTH	akblitsi@iti.gr
Konastninos Flevarakis	P04 CERTH	kostisfl@iti.gr
Aggeliki Papaioannou	P06 INF	apapaioannou@infolysis.gr
Konstantinos Fragkos	P06 INF	cfragkos@infolysis.gr
Thomas Papaioannou	P10 ICCS	thomas.papaioannou@iccs.gr
Konstantinos Routsis	P10 ICCS	konstantinos.routsis@iccs.gr
Rafael Borne Jaular	P13 S21Sec	rborne@s21sec.com
Zbigniew Kopertowski	P15 OPL	zbigniew.kopertowski@orange.com

History

Date	Version	Change
21-Feb-2023	0.1	ToC and task assignments
14-Mar-2023	0.2	ToC updates
28-Mar-2023	0.3	First round of contribution
11-Apr-2023	0.4	Second round of contribution
05-May-2023	0.8	IR Review
09-May-2023	1.0	Submission of Final version after PIC review

Key Data

Keywords	IoT, software, testing, integration, GitLab, DevSecOps
Lead Editor	P04 CERTH – Evripidis Tzionas
Internal Reviewers	P06 INF - Konstantinos Fragkos, P08 MOW

Executive Summary

This deliverable is written in the framework of WP6 – Testing, Integration and Support of **ASSIST-IoT** project under Grant Agreement No. 957258. The deliverable is the second iteration for testing and integration within the ASSIST-IoT project. This deliverable focuses on tools used, report of implemented testing so far, and test plan for the developed components of the project following the DevSecOps methodology.

The document outlines the current tools used by the development team to coordinate their work, including GitLab, GitLab CI/CD, GitLab Runner, Helm Registry, Container Registry, and Kubernetes. These tools serve as the framework for implementing tests in ASSIST-IoT, and detailed instructions for their use are provided. The test strategy is presented in line with the DevSecOps methodology used by the project.

The report also includes a detailed account of the tests implemented in the first three testing phases, with special focus on functional and integration testing which are the most important in the current stage of the project. Moreover, the document offers guidelines for conducting end-to-end, acceptance and performance testing in accordance with WP3 requirements and WP8 KPIs.

A final deliverable will be released in M36, jointly with two other WP6 deliverables, to conclude the testing and integration, packaging and release, and technical support documentation.

.

Table of contents

List of tables

List of figures

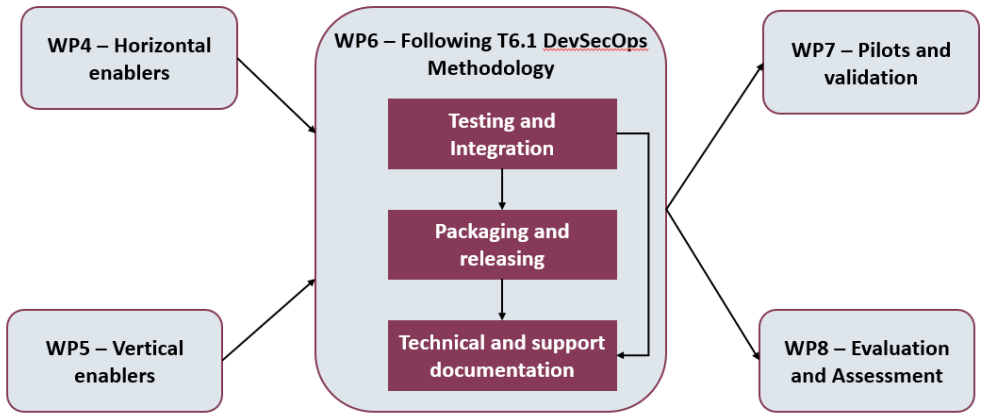
List of acronyms

Acronym	Explanation
AC	Automated Configuration
AD	Active Directory
API	Application Programming Interface
CI	Continuous Integration
CI/CD	Continuous Integration/continuous delivery
DevSecOps	Development Security Operations
DLT	Distributed Ledger Technology
FAT	Factory Acceptance Testing
FL	Federated Learning
GUI	Graphical User Interface
IoT	Internet of Things
IPSec	Internet Protocol Security
JSON	JavaScript Object Notation
K8S	Kubernetes
KPI	Key Performance Indicator
LDAP	Lightweight Directory Access Protocol
LP	Location Processing
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
NG-IoT	Next Generation IoT
OAT	Operational Acceptance Testing
PAP	Policy Administration Point
PUD	Performance and usage diagnosis
RAM	Random Access Memory
REST	Representational State Transfer
SAT	Site Acceptance Testing
SD-WAN	Software-defined wide area network
SSD	Solid State Drive
SSO	Single sign-on
UI	User Interface
URL	Uniform Resource Locator
VPN	Virtual Private Network
Wi-Fi	Wireless Fidelity

1 About this document

The key objective of this document is to extend the initial plan with details about the plan of testing and integration for the project's final phase. The details follow the separate tests in sequence, considering the DevSecOps methodology to allow the project's stakeholders to comply with the methodology. While this is the final deliverable titled testing and integration, there will be another version covering WP6 on M36 due to the extension of the project. This is also the reason that the priority of this deliverable is reporting the implemented tests of the first phases and setting the guidelines for the remaining ones.

1.1 Deliverable context

Keywords	**Lead Editor**
Objectives	**O1:** The deliverable aims to guarantee the architectural structure for NG-IoT and sets tests to facilitate the DevSecOps methodology. **O2 to O5:** Each of the implementations is a subject to testing. **O6:** The final testbed for the project and its validation are pilot sites using the developed solutions.
Work plan	
Milestones	This deliverable does not mark any specific milestone completion. However, it contributes to the MS6 Software structure finished (M24) and MS7 – Integrated solution (M30). The deliverable is the basis for testing methods and integration. In any case, a final review will be made on M36, in the final deliverable of the work package (D6.8).
Deliverables	Task 6.2 – Testing and Integration efforts resulted in this current deliverable. Other concurrent deliverables of WP6 are complementary to this deliverable, namely those devoted to the plan for release and distribution (D6.7 [1.]) and documentation (D6.6 [2.]). It partially draws from the D6.1 [3.] – DevSecOps methodology delivered in M6. The last deliverable of this WP (D6.8) will contain outcomes from all the tasks, although main focus will be on testing and integration rather than packaging, which basis and supporting CI/CD scripts are well-established.

1.2 The rationale behind the structure

This deliverable follows a straightforward approach: Section 2 presents the infrastructure and tools available for the project, aiming to provide a clear understanding of their functionalities, as well as to explain how to use them effectively. Then, in Section 3, the test strategy is elaborated and the progress of enablers' integrations is

presented. In Section 4, a detailed documentation of the implemented tests is provided, including the approach and guidelines for the remaining phases. Finally, conclusions are drawn in Section 5, summarising the outcomes expected in the final iteration of the current deliverable.

1.3 Outcomes of the Deliverable

The main outcome of this deliverable is the documentation of the testing and integration methodology, as well as an overview of the infrastructure and tools used in ASSIST-IoT. The work progress of implemented tests is reported, and the approach for the remaining testing phases is outlined, with the goal of delivering a comprehensive solution to the pilots. This deliverable, along with D6.6 and D6.7, provides a complete solution for testing, integration, and supporting documentation of the project.

The first section presents the GitLab tools utilised in the project for testing and integration purposes, such as GitLab CI/CD, GitLab Runner, Helm, and container registries. The testing environment infrastructure, including VMs, VPN enabler, and K8s clusters, is also outlined with instructions on usage.

The following section elaborates on the testing methodology developed in ASSIST-IoT, the progress made so far, and the updated time plan due to the amendment of the project. The six testing phases agreed upon in the GA of the project are interwoven with the DevSecOps methodology to ensure the software's quality and reliability.

Section 4 reports the project's testing progress, providing details on functional and integration tests implemented, along with guidelines and methodology approach for the remaining testing phases. The successful completion of the testing and integration phases is crucial for the success of the entire project and its impact on the pilot sites.

As the project reaches its final stages, the verification of compliance with the gathered requirements, delivery to the pilots, and determination of acceptance of the final product become paramount.

1.4 Lessons Learnt

During the past months, the Consortium partners have dedicated much efforts to developing the design specifications of the enablers that will realise the ASSIST-IoT solution. Through this work, several important insights have been gained:

- Starting software testing as early as possible during the enabler development phase can significantly improve the efficiency of the testing process, by allowing it to proceed alongside development.
- The deployment of a unified testing environment infrastructure has greatly facilitated the integration of enablers. Given that modern projects often require the combination of microservices to create larger components such as ASSIST-IoT's enablers, implementing a testing environment earlier can speed up integration and component delivery.
- In some cases, the multitude of testing phases may create unnecessary complexity due to overlaps between them. This can make it difficult to distinguish between different testing phases and could be streamlined to improve efficiency.
- In order to ensure a successful testing process, it is crucial that every technical partner contributes to the testing methodology and approach, given that they each have their own tools and unique insights on how to tackle issues that involve everyone.
- Apart from the requirements defined by pilot stakeholders, in order to ensure that the solution meets their needs, it is essential that end users are present in the final phases of testing, to help deliver a more satisfactory and user-friendly end product.

1.5 Deviation and corrective actions

The previous deliverable series provided a framework for implementing the testing and integration plan of the project, however, the original plan for integration testing has deviated. The initial plan for integration testing was to focus on the use cases reported in WP4 and WP5 deliverables and the internal components of the enablers.

While it was possible to partially implement the previous plan, it was not practical as integration testing's focus has to be on integrating different enablers together, rather than a single enabler's internal components. Also, the acceptance testing approach's focus is to verify compliance with the requirements gathered in D3.3. These tests will be initially conducted in a laboratory environment, which will assess the readiness for delivery to the pilots.

2 Integration infrastructure and tools

2.1 GitLab

The previous deliverable version pinpointed GitLab's [5.] suitability in conducting the DevSecOps methodology's objectives. In particular, GitLab's role as a web-based Git repository enables open and private repositories, tracking issues, and wikis. The platform's security features align with ASSIST-IoT's methodology and its objectives. Furthermore, it streamlines the DevSecOps procedures' automation through the established CI pipeline's security testing.

The current deliverable version focuses explicitly on GitLab Runner [8.], Container Registry [9.], Helm Registry [10.], and their respective functionalities. These tools are all part of GitLab's platform and can be used in conjunction with each other to enhance the software development lifecycle. GitLab Runner is used for continuous integration and delivery, while the Container Registry and Helm Registry are used for storing and sharing container images and Helm charts, respectively.

Overall, the GitLab platform is a convenient location to support software development throughout the different phases. In particular, GitLab provides a central location for managing the software development lifecycle from the initial phase of project planning to its final stage with source code management, testing, monitoring, and security. Its features and tools can enhance collaboration between teams, shorten product lifecycles, and boost productivity resulting in greater value for customers.

2.1.1 GitLab CI/CD

GitLab CI/CD [7.] is a tool that allows developers to automate the testing, building, and deployment of their code changes. CI/CD stands for Continuous Integration/Continuous Deployment, which refers to the process of automatically building and testing code changes as they are committed to a repository, and then deploying them to a production environment if they pass all tests. With GitLab CI/CD, developers can define custom pipelines that specify the exact steps needed to test and deploy their code changes. These pipelines can be triggered either automatically by a commit to a specific branch or tagged in a repository, or manually by a repository's user.

By automating the testing and deployment process, GitLab CI/CD helps to ensure that code changes are thoroughly tested and vetted before being deployed to production, reducing the risk of errors or downtime. It also allows for faster deployment of new features and fixes, as well as greater collaboration between developers and operations teams.

To get started with GitLab CI/CD, these general steps are followed:

1. To create a `.gitlab-ci.yml` file in the root directory of your GitLab repository. This file will define the steps and stages of your CI/CD pipeline.
2. To define the stages of your pipeline, such as "build", "test", and "deploy".
3. To define the jobs for each stage. A job is a specific task that needs to be executed within a stage, such as "build Docker image" or "run unit tests".
4. To specify the runner that will execute the jobs. Shared runners provided by GitLab or custom runners can be used.
5. To set up any necessary variables or environment variables needed for your jobs.

This is the main file used as the baseline for the enablers' repositories. The partners' development teams are advised to leverage the current pipeline for consistency, even if they have their own valid pipelines. Generally, the project does not mandate strictly adhering to the above pipeline, as long as the outcomes are analogous (e.g., static code is analysed, containerised as Docker images, packaged as Helm charts, deployed in staging environment, tested – see Section 4.1, and released in the project registries and repositories).

```
# Environmental variables available for all the jobs
variables:
  ENV_VAR_1: "VALUE"
  ENV_VAR_2: "VALUE"
  ENV_VAR_N: "VALUE"
```

```
stages:
  - build
  - test
  - deploy

build-code-job:
  # Environmental variables available only in this job
  variables:
    JOB_ENV_VAR_1: "VALUE"
    JOB_ENV_VAR_2: "VALUE"
    JOB_ENV_VAR_N: "VALUE"
  # Stage in which this job will be executed
  stage: build
  # Optional tag to select the desired runner to execute the job
  tags:
    - gitlab_runner_tag
  script:
    - echo "Build job

test-code-job1:
  stage: test
  script:
    - echo "Test job 1"

test-code-job2:
  stage: test
  script:
    - echo "Test job 2"

deploy-job:
  stage: deploy
  script:
    - echo "Deploy job"
```

6. Commit the `.gitlab-ci.yml` file to your GitLab repository.
7. Once the file has been committed, GitLab will automatically detect changes and start executing the pipeline based on the configuration in the `.gitlab-ci.yml` file.
8. Monitor the pipeline's progress through the GitLab web interface or command line tools.
9. Once the pipeline is complete, review the results and any logs or artifacts generated by the jobs.

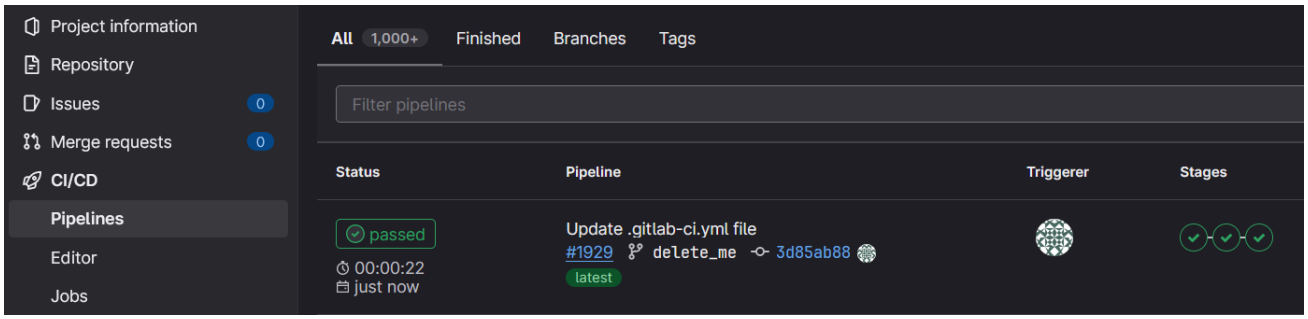


Figure 1. Pipelines of the repository

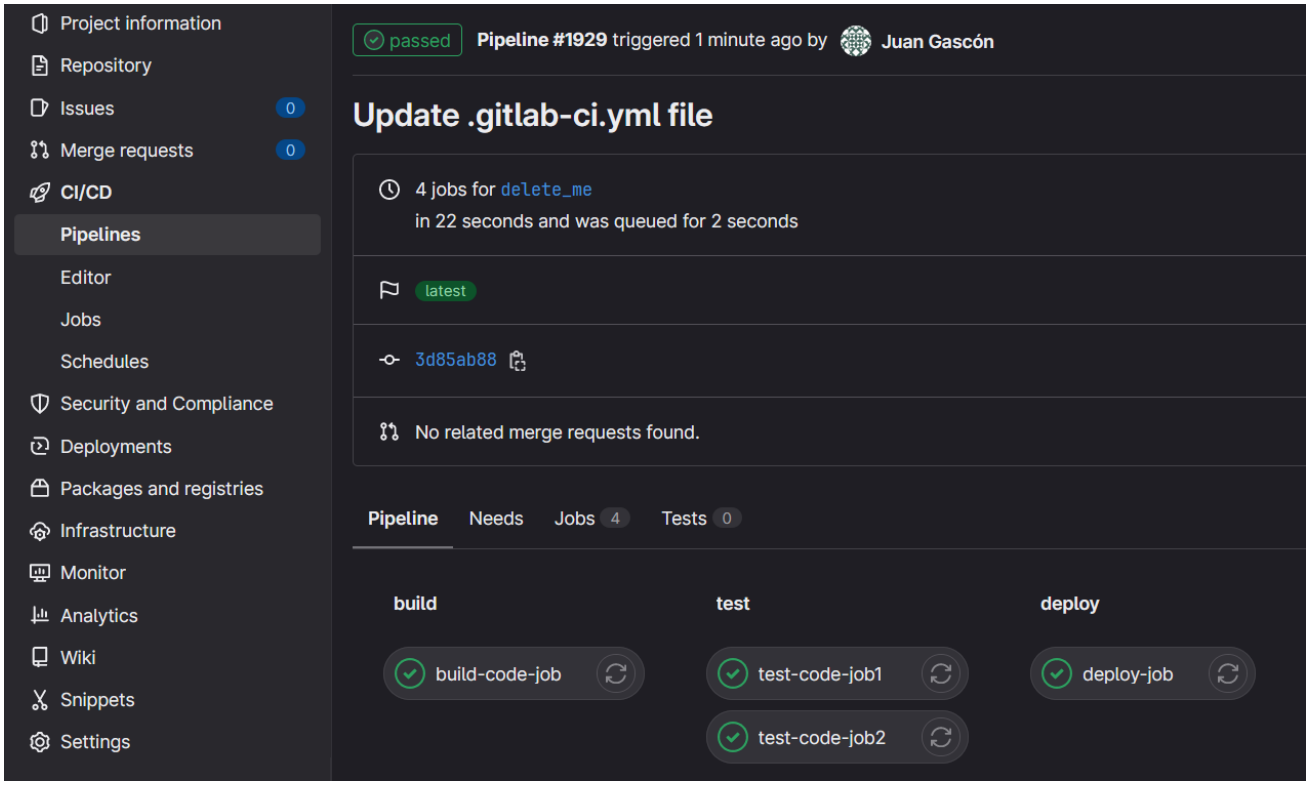


Figure 2. Successful pipeline with the stages and jobs executed

10. If the pipeline is successful, the changes can be merged into the main branch of your repository and deployed to the production environment.

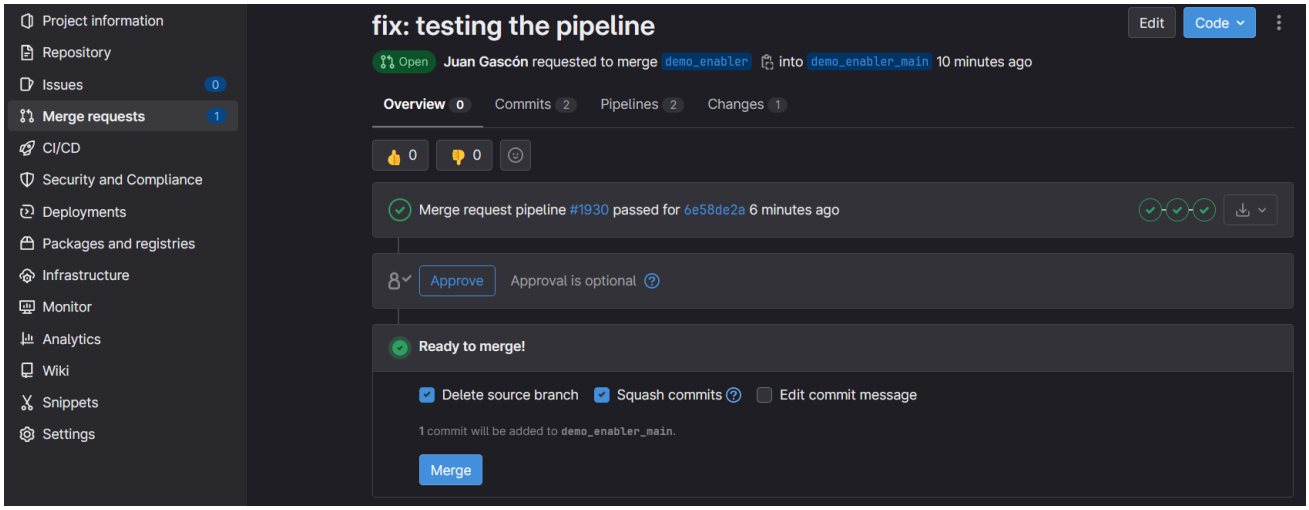


Figure 3. Merge request with successful pipeline execution, awaiting owner/maintainer approval for merging

These steps provide a basic overview of how to set up and use GitLab CI/CD. However, the specific instructions and configuration details will depend on the application and development workflow.

2.1.2 GitLab Runner

GitLab Runner [8.] is defined as an application working with GitLab CI/CD to run jobs in a pipeline and was introduced in detail in the D6.1 DevSecOps Methodology and Tools [3.].

GitLab Runner is open-source and written in Go. It does not rely on any particular programming language and can be run as a standalone binary. It can also run inside a Docker container or be deployed to a Kubernetes cluster and can be installed and used on GNU/Linux, macOS, FreeBSD, and Windows:

- In a container.
- By downloading a binary manually.
- By using a repository for rpm/deb packages.

After the GitLab runner installation, individual runners must be registered (runners are the agents that run the CI/CD jobs that come from GitLab). Once the runner has been registered, the machine set up with the runner will be able to communicate with the ASSIST-IoT GitLab instance. They usually process jobs on the same machine with the GitLab Runners installation. However, it can also be a runner process job in a container, in a Kubernetes cloud or auto-scaled instances in the cloud.

After the runner is registered, an executor must be chosen. This determines the environment each job runs in.

As it is mentioned before, Gitlab runner has been introduced in ASSIST-IoT's D6.1, and the installation instance is the following one:

```
curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash
sudo apt-get install gitlab-runner
sudo gitlab-runner -version
sudo gitlab-runner status
sudo gitlab-runner start
sudo visudo
#Add the gitlab-runner user in sudoers group and set NOPASSWD as shown below
gitlab-runner ALL=(ALL:ALL) ALL
gitlab-runner ALL=(ALL) NOPASSWD: ALL
```

Register the runners in the GitLab server:

The registration of each individual GitLab runner follows the installation described in this section's introduction. The following steps guide the developers in registering their GitLab runner.

Go to Settings - CI/CD – Runner -> Expand

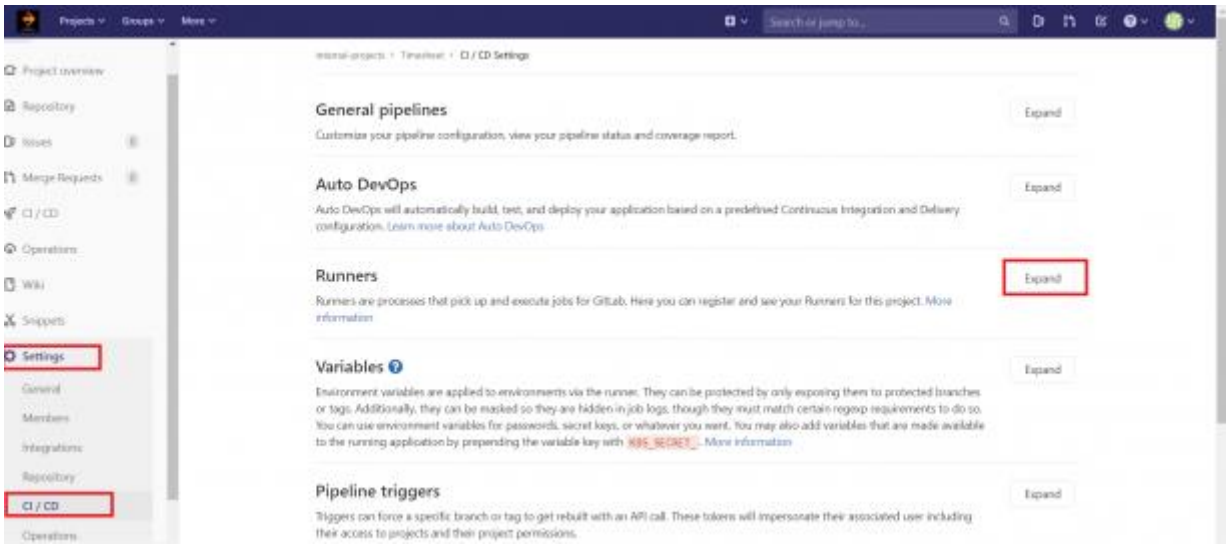


Figure 4. Runners path

After clicking in “Expand”, the screen shown will be similar to the one below (Figure 5).

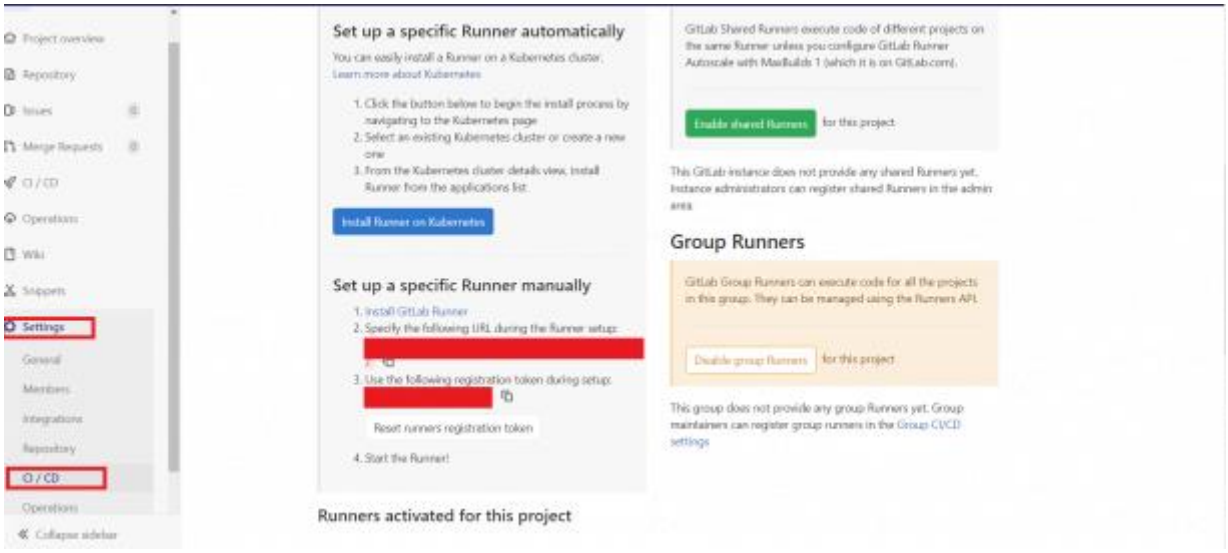


Figure 5. Runners menu

Now in the Runners instance, runner setup must be launched with the specified URL and API key. The following commands show how to launch the runner setup for both options (Docker, Shell):

1) *Specific runner for “docker” tagged steps in CI-CD:*

```
gitlab-runner register \
    --non-interactive \
    --url https://gitlab.assist-iot.eu/ \
    --registration-token GR13489415Ec3MtpvpMS6sef8fyxW\
    --description "s21sec-wp6-docker-runner" \
    --tag-list "docker" \
    --executor docker \
    --docker-image "docker:dind" \
    --docker-volumes /var/run/docker.sock:/var/run/docker.sock, /cache
```

2) *Specific runner for “shell” tagged steps in CI-CD:*

```
gitlab-runner register \
    --non-interactive \
    --url https://gitlab.assist-iot.eu/ \
    --registration-token GR13489415Ec3MtpvpMS6sef8fyxW\
    --description "s21sec-wp6-shell-runner" \
    --tag-list "shell" \
    --executor shell
```

Once all the previously described steps are executed, the service has to be restarted for the changes to take effect in the GitLab.

Restart the service.

```
sudo gitlab-runner restart
```

Also, if further information is needed to customise the runners, the config file is located in the following path:

```
/etc/gitlab-runner
```

And the config file that must be edited is

```
config.toml
```

An example of this config file is shown in the following figures. In this case two runners are shown for WP5 project, one in Docker and the other in shell:

```
[[runners]]
  name = "s21sec-wp5-docker-runner"
  url = "https://gitlab.assist-iot.eu/"
  id = 16
  token = "aB7ytucvBfZYdG7E54_6"
  token_obtained_at = 2022-10-05T15:12:32Z
  token_expires_at = 0001-01-01T00:00:00Z
  executor = "docker"
  [runners.custom_build_dir]
  [runners.cache]
    MaxUploadedArchiveSize = 0
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
  [runners.docker]
    tls_verify = false
    image = "docker:stable"
    privileged = true
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
    shm_size = 0

[[runners]]
  name = "s21sec-shell-wp5-runner"
  url = "https://gitlab.assist-iot.eu/"
  id = 18
  token = "5SaQ7yTUzKMHfNPs7L2s"
  token_obtained_at = 2022-10-05T15:14:27Z
  token_expires_at = 0001-01-01T00:00:00Z
  executor = "shell"
  [runners.custom_build_dir]
  [runners.cache]
    MaxUploadedArchiveSize = 0
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
```

Figure 6. Runners config file

The runners menu (*Settings - CI/CD – Runner -> Expand)* is where you can double-check that everything has been set up correctly. The operation has been completed successfully if the runners configured in the previous step are displayed as assigned project runners (Figure 7).

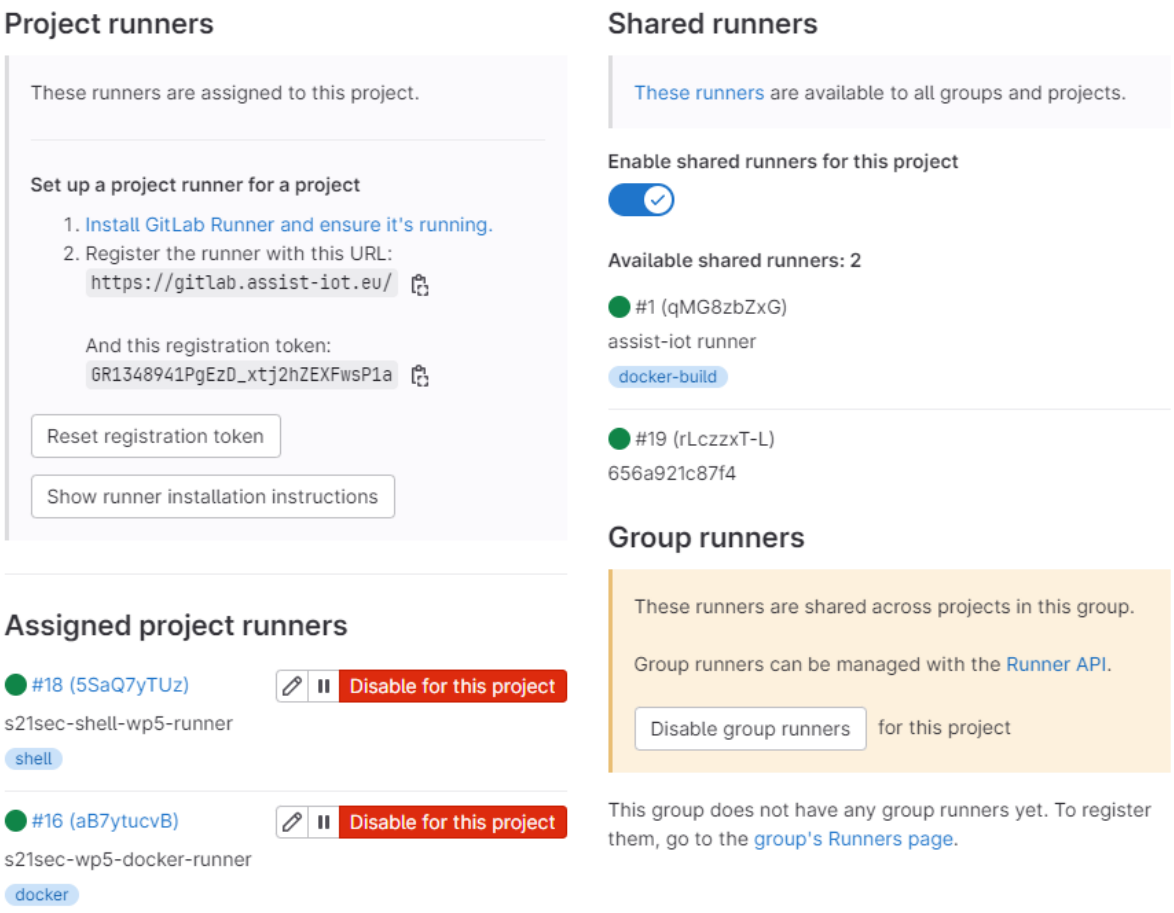


Figure 7. Runners verification screen

Once the pipeline is executed, runners will complete their allocated tasks and report back a “passed” status if everything went as planned. The following Figure 8is a demonstration:

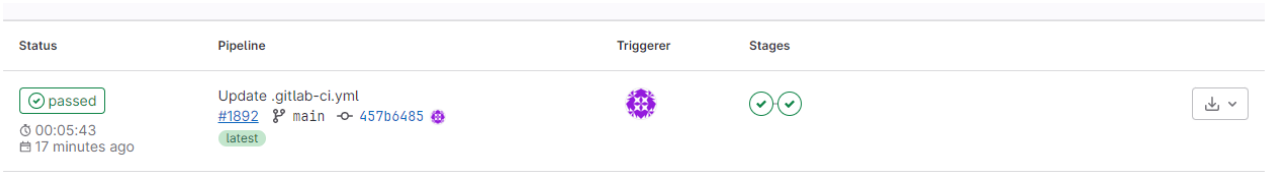


Figure 8. Runner status

The above outcome is verifiable through the project’s GitLab under the jobs section. If the result is different than expected, users can examine any discrepancies by clicking on the runner's status, which will reveal all the steps the runner took to get there. Consequently, users can mitigate the discrepancies in sequential order. The following images serve as clarification by illustrating 3 examples of 3 different jobs (9, 10, & 11):

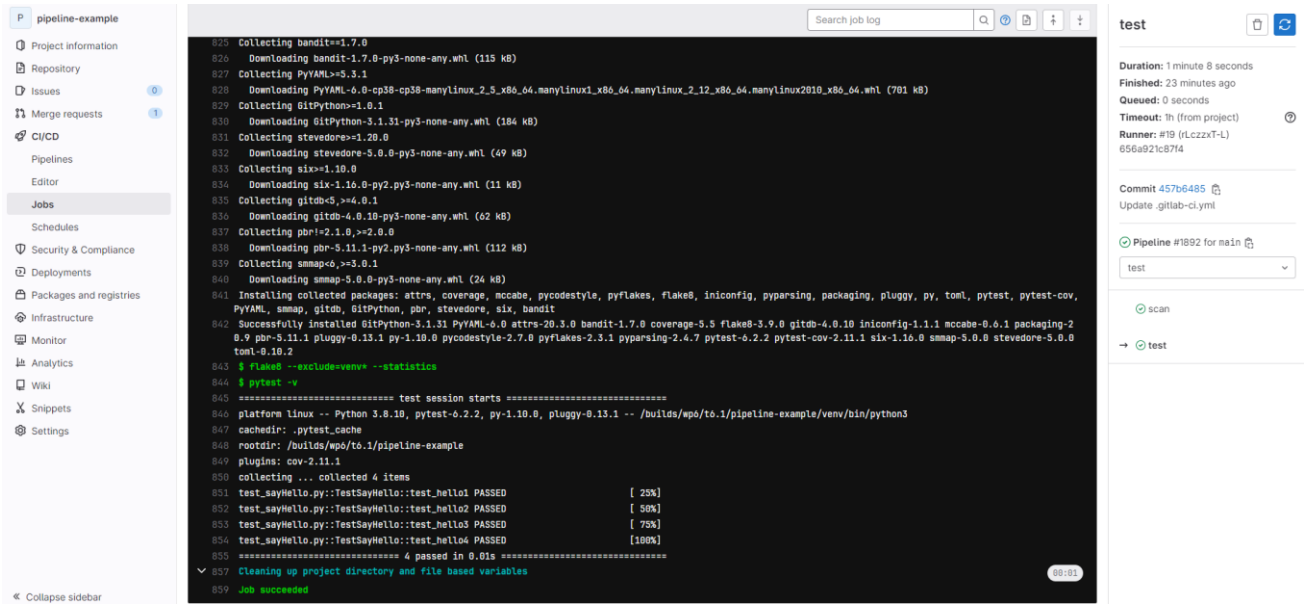


Figure 9. Job running. Example 1

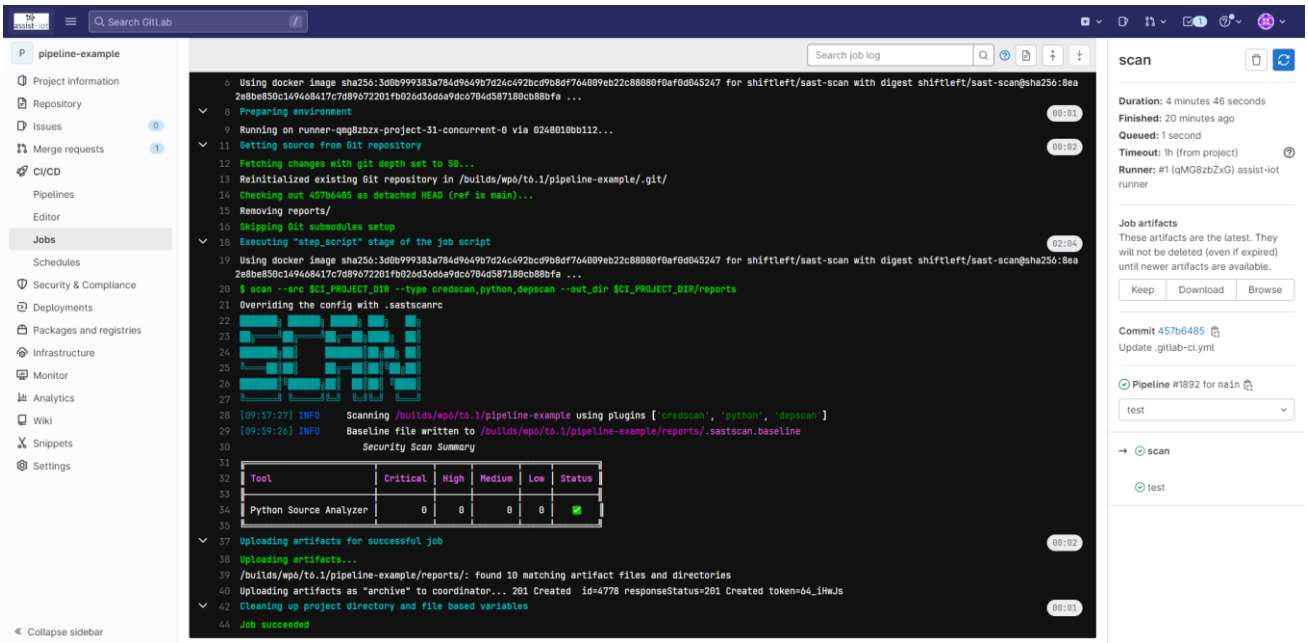


Figure 10. Job running. Example 2

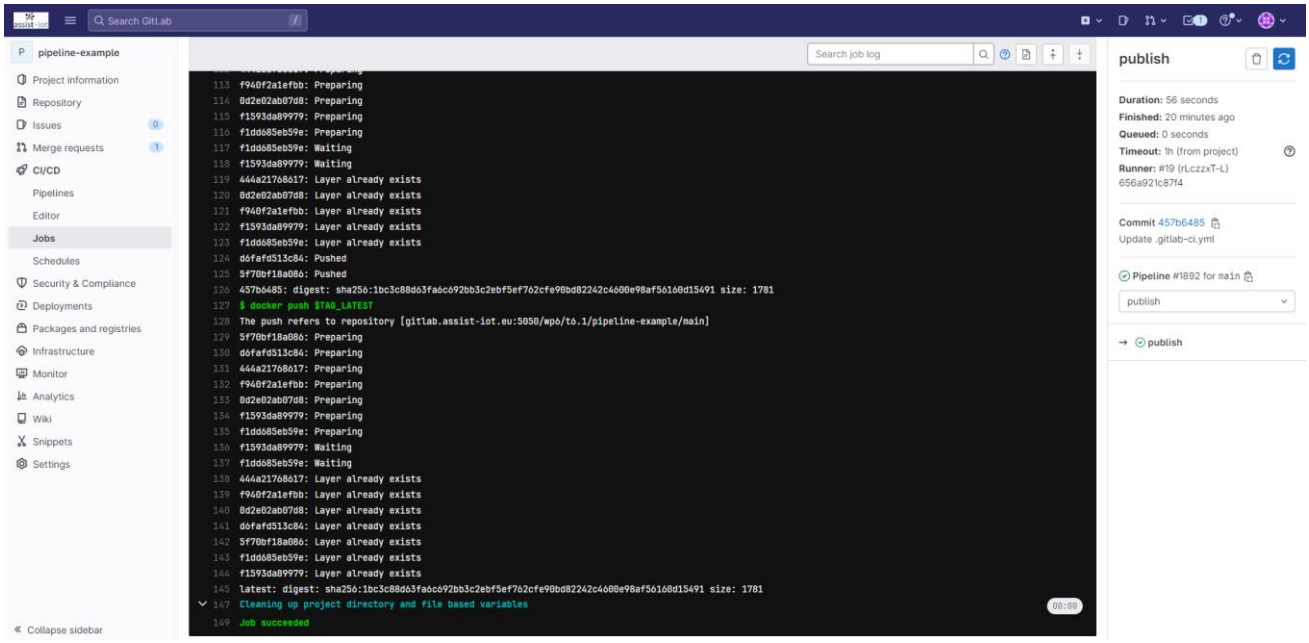


Figure 11. Job running. Example 3

2.1.3 Helm registry

Helm [11.] is a package manager for Kubernetes that simplifies the deployment and management of applications. It uses charts, which are collections of files that describe a set of Kubernetes resources, to package and distribute applications. Charts can be versioned, shared, and reused, which makes it easy to manage the lifecycle of applications in Kubernetes.

To enable the distribution and discovery of Helm charts, Helm registry [10.] is used. It is a tool similar to Docker registry or GitLab container registry but specifically designed for Helm charts [12.]. With Helm registry, developers can store and share charts with their team or the wider community, making it easier to collaborate on building and deploying applications on Kubernetes. The registry allows hosting and distributing charts internally, privately, or publicly. Additionally, developers can define and manage access control policies and manage the lifecycle of the charts. This enhances security and collaboration in the development and deployment of applications on Kubernetes [13.].

Helm chart registries allow developers to version control their Helm charts, which is especially useful if there are multiple environments (such as development, staging, and production). Specifically, developers can easily keep track of different versions of their Helm charts and roll back to previous versions if needed. This ensures consistent deployments across the Kubernetes cluster, as everyone on the team is deploying the same version of a chart with the same configuration.

Helm chart registries provide a centralised location for managing Helm charts, making it easier to share and collaborate on charts across multiple teams. By using a Helm chart registry, developers can save time by reusing existing charts instead of reinventing the wheel every time there is a need for deploying a new application. This can increase productivity and reduce the chance of errors in the deployments.

Access control is also a critical feature of Helm chart registries. Developers can control who has access to the Helm charts by setting up authentication and authorisation for their registry. This ensures that only authorised users can deploy charts, which is especially important for sensitive or production workloads. Additionally, developers can define and manage access control policies to manage the lifecycle of the charts, which enhances security and collaboration in the development and deployment of applications on Kubernetes.

An example flow of using GitLab's Helm registry is presented below:

The first step is to create a GitLab access token, which has access to the container registry, through profile in the upper right corner -> edit profile -> Access tokens. (Figure 12)

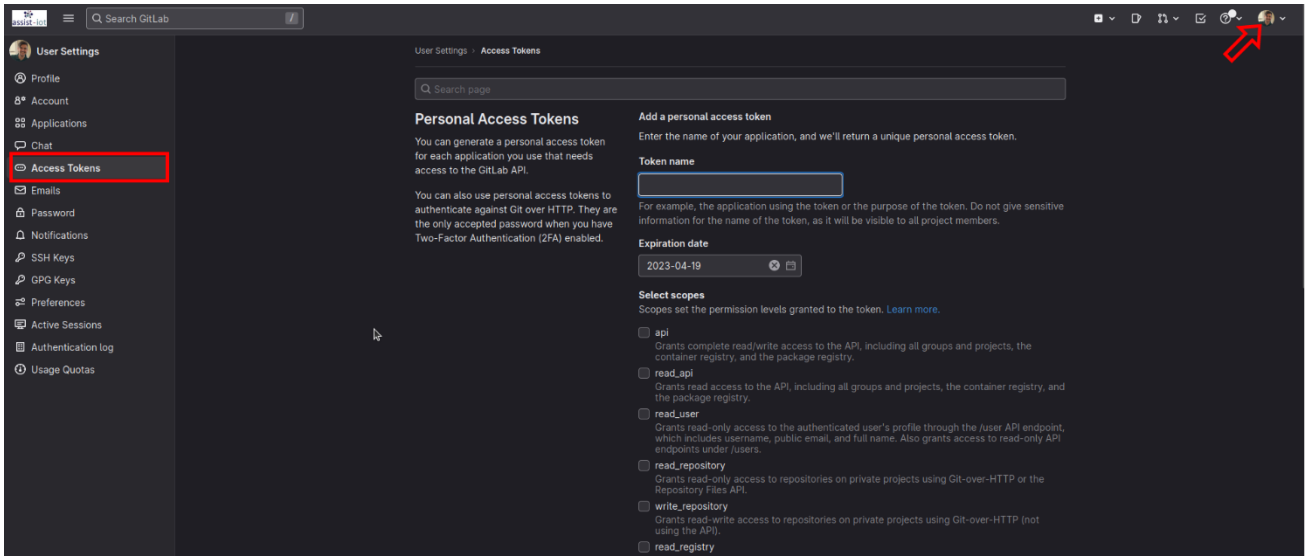


Figure 12. Generating an access token

In order to upload an enabler to the GitLab package registry steps depicted in Figure 13 are followed. Note that because the project's repository is private, the first command should include the token generated in the previous step:

```
$helm repo add --username <username> --password <access_token> assist-public-repo \
https://gitlab.assist-iot.eu/api/v4/projects/85/packages/helm/stable
```

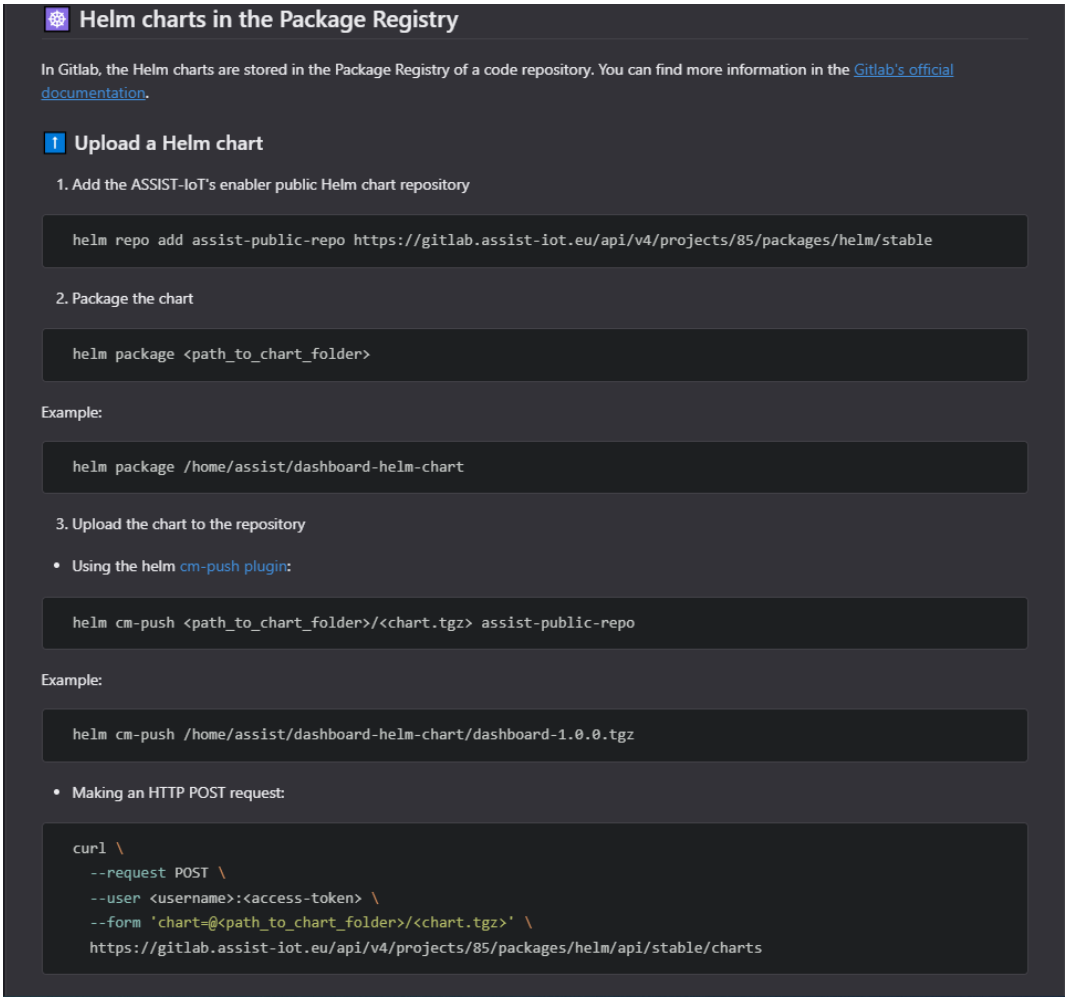


Figure 13. Instruction set for uploading an enabler's helm chart

After uploading the Helm charts to the registry, developers can view them in the project's dedicated public repository at ASSIST-IoT's GitLab (Figure 14).

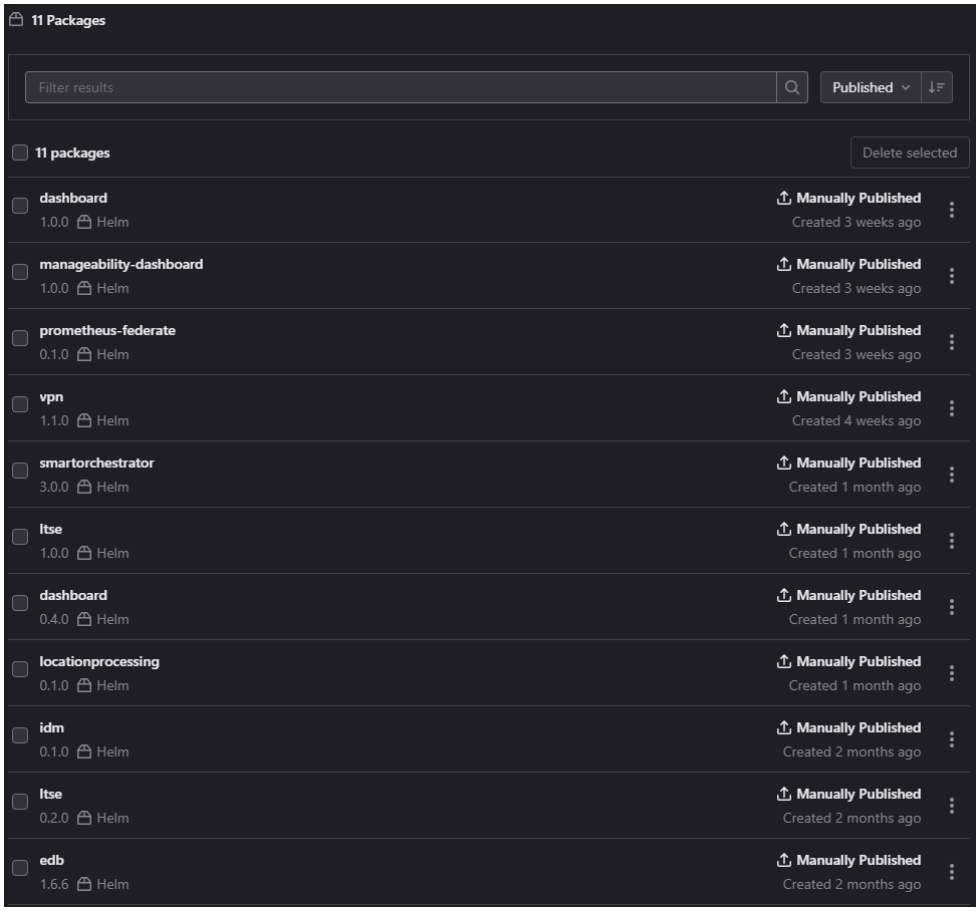


Figure 14. Docker login and creating a secret

The final step presents how to install an enabler, whose Helm chart has been uploaded and stored in the package registry (Figure 15).

Figure 15. Docker login and creating a secret (2)

Further information about how to use Helm registry is available in:

GitLab docs

Our dedicated GitLab repo.

2.1.4 Container registry

GitLab Container Registry [9.] is a powerful tool that offers a secure and private space for storing Docker images. This feature enables teams to create, store and share Docker images that can be easily accessed by other team members or even across different projects. The container registry allows teams to avoid the use of public repositories that may pose a security risk by providing a private and secure space for image storage.

In addition to the private repository, the container registry also allows teams to integrate with the GitLab CI/CD pipeline for automated builds, testing and deployment of Docker images. This integration provides the ability to build, test and deploy Docker images in a single, streamlined workflow, while ensuring that the images are secured and properly authenticated.

Another important aspect of security when using the GitLab Container Registry is the use of secrets in Helm chart deployments. Without the use of secrets, images cannot be pulled from the private repository without providing GitLab credentials, which may not be ideal from a security standpoint. With the use of secrets, however, teams can ensure that their images are only accessible to authorised personnel, while maintaining a high level of security throughout the development and deployment process.

An example flow looks like this:

- Create a gitlab access token, which has access to the container registry, through profile in the upper right corner -> edit profile -> Access tokens. (pic1)
- Login to the gitlab container registry through the token that was created:
  $docker login gitlab.assist-iot.eu:5050 (pic2)
- Build the images that are about to be pushed and tag them properly:
  $docker build -t gitlab.assist-iot.eu:5050/wp4/applications/openapi-enabler/frontend:v1.0.0 .
- Then push them to the registry:
  $docker push gitlab.assist-iot.eu:5050/wp4/applications/openapi-enabler/frontend:v1.0.0
- Create a secret:
  $kubectl create secret docker-registry <your_secret> --docker-server=$docker_server --docker-username=$docker_username --docker-password=$personal_token (pic3)
- Then the tagged container can be viewed inside the repo by accessing the sidebar at Packages and registries -> container registry (pic4 + 5)
- Then the imagePullSecrets are added to the helm chart under the component that uses the image that has been built in values.yaml like (pic6)

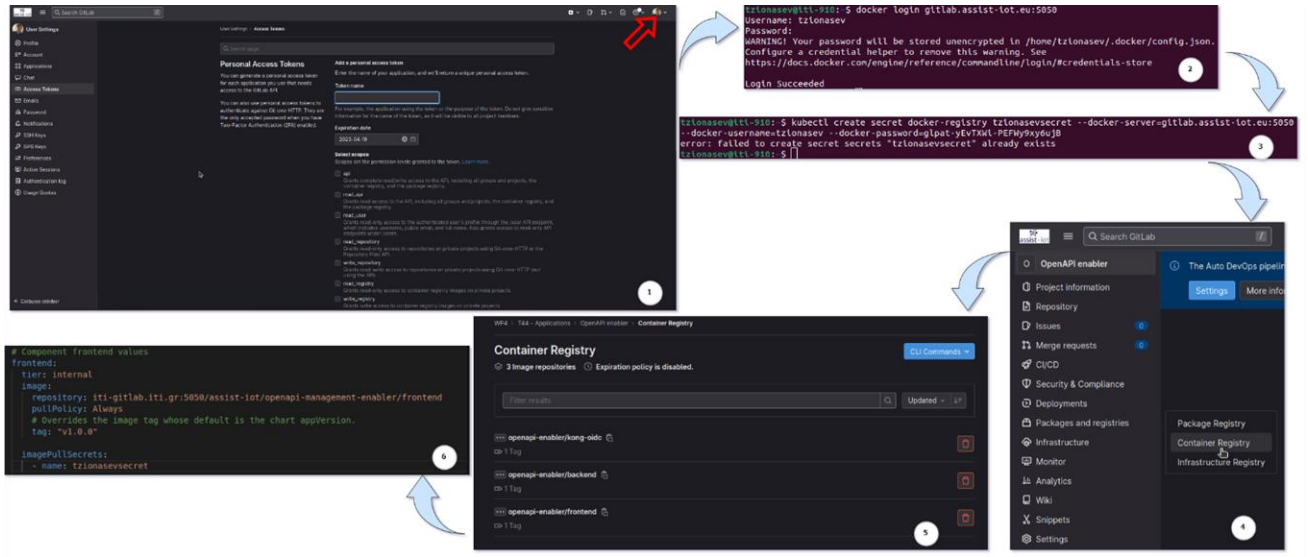


Figure 16. Container registry workflow example

For more information about how to use Container registry is available in:

GitLab docs,

Our dedicated GitLab repo.

2.2 ASSIST-IoT Testing Environment

The testing environment is the logical architecture of hardware and software for serving the purposes of ASSIST-IoT. This section's goal is to provide a detailed explanation of the logical architecture, as well as the criteria outlined by the project's specifications for testing and deployment.

Initially, the specifications for testing and deployment consider the needs of pilot sites and developers. These are the subjects of testing the developed enablers and deploying them on their premises. The computing units to be used by the pilot sites come in a variety of sizes, ranging from smaller and limited capabilities to ones with resources to spare. Additionally, enablers are developed with the vision of diverse environments that can be sufficiently covered by clusters of virtual machines running on separate units.

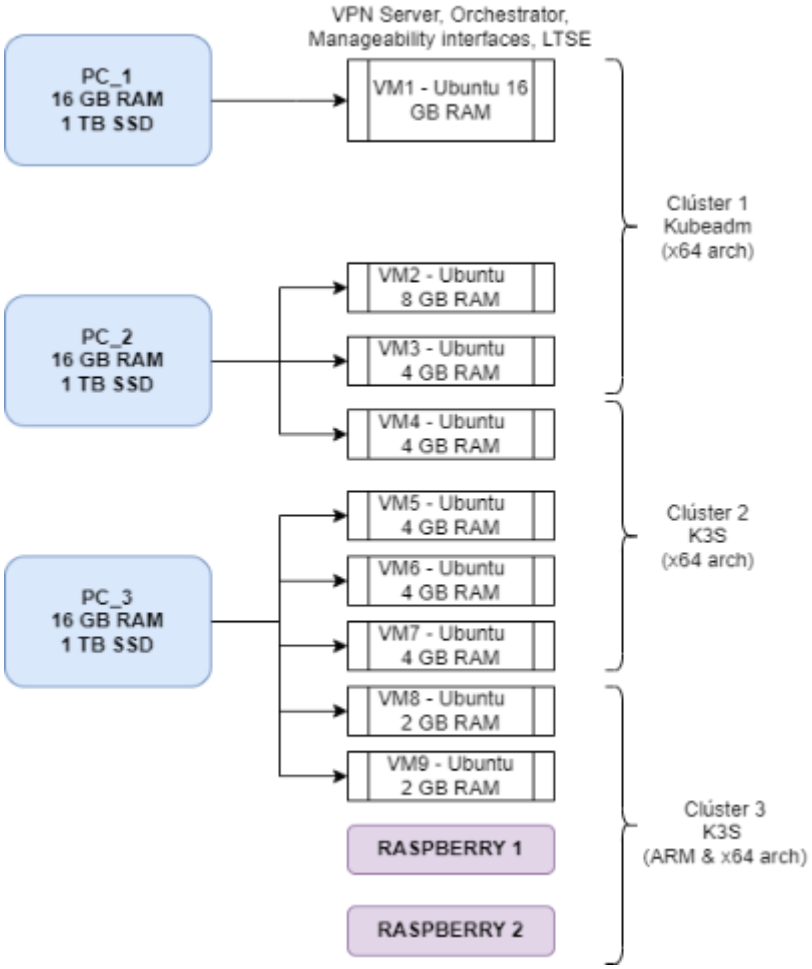


Figure 17. Test environment to simulate pilot site premises

The current infrastructure for testing is hosted on the premises of ASSIST-IoT's partner, CERTH. The infrastructure is composed of five units for fulfilling the project's requirements in terms of testing in an environment analogous to the pilot sites for future deployment. In particular, the infrastructure is composed of three different computers accompanied by two Raspberry Pi microcontrollers that serve as the smaller units. The machines' hardware features are 16GB RAM and 1 TB SSD.

Virtual machines are distributed between the different computers to ensure that there is a running instance for testing despite any inconveniences with the particular computer. Moreover, a computer running a single virtual

machine is considered to provide a computer to carry out more computationally demanding tasks. Other virtual machines vary to the allocated capabilities.

In order to execute tests on enablers, the infrastructure has been designed to be partitioned into three separate clusters according to a logical architecture. The clusters are consisted of virtual machines that function on a variety of different units, replicating computers that are in different locations. The first cluster created with kubeadm is composed of three different virtual machines, representing the cloud and being directly connected with three different K8s nodes (One master, two worker nodes). The second cluster running the K3s distribution operates between four distinct virtual machines and it represents an edge k8s distribution with a one-to-one connection with the K8s nodes (One master, three worker nodes). The final cluster accommodates the smaller processing units and the two Raspberry Pis, also with one master node and three workers. The topology can be viewed graphically through the manageability enablers (Figure 18).

Figure 18. Kubernetes cluster topology

The testing environment is connected through a VPN enabler, which includes all the virtual machines and Raspberry Pis in the network. To access the testing environment, each user is provided with VPN credentials by the network administrator. Once logged in, users can access the virtual machines and Raspberry Pis in the network using SSH from a shell. The IP addresses for each machine range from 10.10.10.2 to 10.10.10.12. To ensure the security of the network, all user connections are monitored continuously.

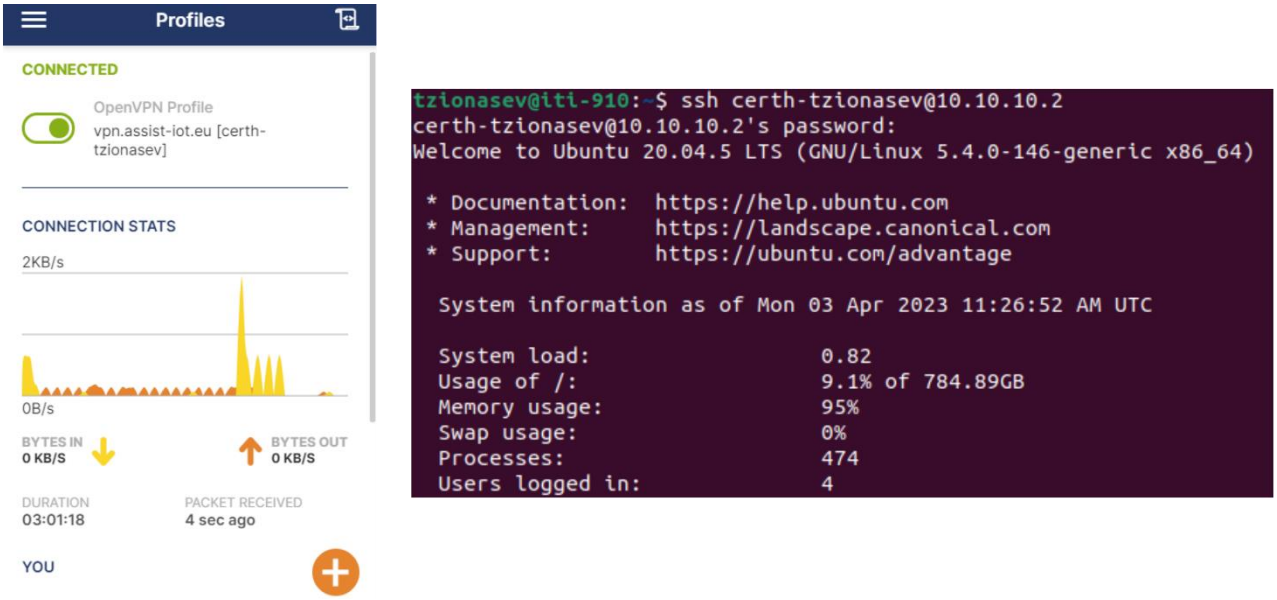


Figure 19. VPN and Kubernetes cluster login

To monitor the Kubernetes cluster and its users, a monitoring tool has been implemented. This tool enables the administrator to monitor the cluster's performance, the number of active users, and other relevant metrics, as shown in Figure 20. By using this monitoring tool, the administrator can ensure that the cluster is performing optimally and address any issues that arise promptly.

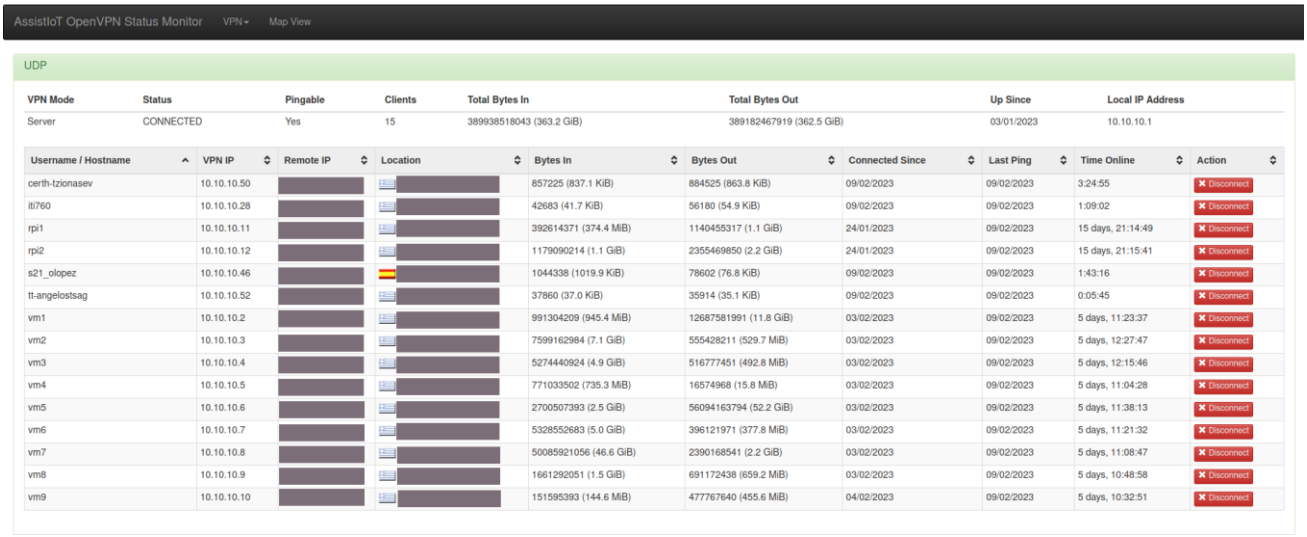


Figure 20. VPN monitoring tool GUI

An overview of some enablers currently working in the cluster can be viewed below in Figure 21, along with their corresponding services in Figure 22.

```
NAME                                                              READY   STATUS    RESTARTS          AGE
pod/dashboard-manageability-dashboard-backend-7bf7bc9dd7-8v5k8    1/1     Running   4 (2d14h ago)     24d
pod/dashboard-manageability-dashboard-db-0                        1/1     Running   4 (2d14h ago)     24d
pod/dashboard-manageability-dashboard-frontend-5ccbb9bcb4-qxghg   1/1     Running   4 (2d14h ago)     24d
pod/edbe-vernemq-0                                                1/1     Running   5 (2d14h ago)     33d
pod/location-processing-test-2-locationprocessing-app-7d584d75xqj8j   1/1     Running   14 (2d12h ago)    9d
pod/location-processing-test-2-locationprocessing-db-0            1/1     Running   0                 2d12h
pod/location-processing-test-2-locationprocessing-pgadmin-86762g62n   1/1     Running   1 (2d14h ago)     9d
pod/ltse-api-6466f8f697-psg8x                                     1/1     Running   5 (2d14h ago)     33d
pod/ltse-elastic-0                                                1/1     Running   7 (2d14h ago)     33d
pod/ltse-postgresql-0                                             1/1     Running   5 (2d14h ago)     33d
pod/ltse-postgrest-66f85b64d-lbkcp                                1/1     Running   5 (2d14h ago)     33d
pod/my-mosquitto-5496966897-vk4bc                                 1/1     Running   1 (2d14h ago)     9d
pod/node-red-b7d486d8b-ztrs8                                      1/1     Running   6 (2d14h ago)     33d
pod/openapi-backend-5485ccc9f8-v9jtc                              1/1     Running   1 (2d14h ago)     9d
pod/openapi-frontend-dcff4d968-6c6ps                              1/1     Running   1 (2d14h ago)     9d
pod/openapi-konga-7f76989584-hkzhw                                1/1     Running   1 (2d14h ago)     9d
pod/openapi-kongdb-0                                              1/1     Running   0                 2d12h
pod/openapi-kongpod-6f78f754f5-xsh5v                              1/1     Running   14 (2d12h ago)    9d
pod/pubsub-56dfc9cd56-x7vsc                                       1/1     Running   26 (2d13h ago)    33d
pod/s21-authz-authorization-authzdatabase-0                       1/1     Running   1 (2d14h ago)     3d22h
pod/s21-authz-authorization-authzweb-5b98794b5c-4f54z             1/1     Running   0                 2d12h
pod/s21-idm-database-0                                            1/1     Running   4                 28d
pod/s21-idm-keycloak-749466b8b7-cg2p9                             1/1     Running   4 (2d14h ago)     28d
pod/s21-siem-elasticsearch-0                                      1/1     Running   4 (2d14h ago)     24d
pod/s21-siem-kibana-687d7646f-ckxbk                               1/1     Running   4 (2d14h ago)     24d
pod/s21-siem-wazuh-0                                              1/1     Running   4 (2d14h ago)     24d
pod/smartorchestrator-mongodb-0                                   1/1     Running   1 (2d14h ago)     4d21h
pod/smartorchestrator-multiclusterservice-bbb57ff9c-q7842         1/1     Running   0                 2d12h
pod/smartorchestrator-smartorchestrator-684d68478b-pwz6t          1/1     Running   0                 2d12h
```

Figure 21. Deployed enablers in ASSIST-IoT's testing environment

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/dashboard-manageability-dashboard-backend	ClusterIP	10.101.218.96	<none>	8080/TCP	24d
service/dashboard-manageability-dashboard-db	ClusterIP	10.98.129.49	<none>	5432/TCP	24d
service/dashboard-manageability-dashboard-db-headless	ClusterIP	None	<none>	5432/TCP	24d
service/dashboard-manageability-dashboard-frontend	NodePort	10.108.202.122	<none>	80:30080/TCP	24d
service/edbe-vernemq	NodePort	10.99.162.94	<none>	1883:1883/TCP	298d
service/edbe-vernemq-headless	ClusterIP	None	<none>	4369/TCP,8888/TCP	298d
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	312d
service/location-processing-test-2-locationprocessing-app	NodePort	10.107.54.134	<none>	80:12271/TCP	12d
service/location-processing-test-2-locationprocessing-db	NodePort	10.98.72.63	<none>	5432:5300/TCP	12d
service/location-processing-test-2-locationprocessing-db-headless	ClusterIP	None	<none>	5432/TCP	12d
service/location-processing-test-2-locationprocessing-pgadmin	NodePort	10.100.13.158	<none>	5433:26383/TCP	12d
service/ltse-apl	NodePort	10.101.77.7	<none>	8080:30801/TCP	75d
service/ltse-elastic	ClusterIP	10.111.51.35	<none>	9200/TCP,9300/TCP	75d
service/ltse-elastic-headless	ClusterIP	None	<none>	9200/TCP	75d
service/ltse-postgresql	ClusterIP	10.97.233.255	<none>	5432/TCP	75d
service/ltse-postgresql-headless	ClusterIP	None	<none>	5432/TCP	75d
service/ltse-postgrest	ClusterIP	10.96.52.214	<none>	3000/TCP	75d
service/my-mosquitto	LoadBalancer	10.106.217.113	192.168.1.201,192.168.1.202,192.168.1.203	1883:10571/TCP	12d
service/node-red	NodePort	10.100.9.174	<none>	1880:4241/TCP	291d
service/openapi-backend	NodePort	10.96.30.126	<none>	9000:30900/TCP	21d
service/openapi-frontend	NodePort	10.111.50.169	<none>	3000:30300/TCP	21d
service/openapi-konga	NodePort	10.108.226.195	<none>	1337:31337/TCP	21d
service/openapi-kongdb	NodePort	10.100.81.88	<none>	5432:13528/TCP	21d
service/openapi-kongdb-headless	ClusterIP	None	<none>	5432/TCP	21d
service/openapi-kongpod	NodePort	10.96.175.97	<none>	8080:30000/TCP,8443:30443/TCP,8001:30001/TCP,8444:30444/TCP	21d
service/s21-authz-authorization-authzdatabase	ClusterIP	10.97.170.26	<none>	3306/TCP	3d22h
service/s21-authz-authorization-authzdatabase-headless	ClusterIP	None	<none>	3306/TCP	3d22h
service/s21-authz-authorization-authzweb	NodePort	10.97.230.115	<none>	8080:30015/TCP	3d22h
service/s21-idm-database	NodePort	10.97.154.43	<none>	5432:31959/TCP	28d
service/s21-idm-database-headless	ClusterIP	None	<none>	5432/TCP	28d
service/s21-idm-keycloak	NodePort	10.111.79.93	<none>	8080:9120/TCP	28d
service/s21-siem-elasticsearch	NodePort	10.106.83.129	<none>	9200:5839/TCP	24d
service/s21-siem-elasticsearch-headless	ClusterIP	None	<none>	9200/TCP	24d
service/s21-siem-kibana	NodePort	10.99.90.191	<none>	5601:14804/TCP	24d
service/s21-siem-wazuh	NodePort	10.106.148.235	<none>	1514:24184/TCP,1515:28042/TCP,514:17245/UDP,55000:15760/TCP	24d
service/s21-siem-wazuh-headless	ClusterIP	None	<none>	1514/TCP,1515/TCP,514/UDP,55000/TCP	24d
service/smartorchestrator-mongodb	ClusterIP	10.110.91.205	<none>	27017/TCP	4d21h
service/smartorchestrator-mongodb-headless	ClusterIP	None	<none>	27017/TCP	4d21h
service/smartorchestrator-smartorchestrator	NodePort	10.99.11.15	<none>	5002:22199/TCP	4d21h

Figure 22. Deployed services in ASSIST-IoT's testing environment

Further information about ASSIST-IoT's infrastructure is available in the following links:

Smart Orchestrator,

Manageability Enablers

3 Acceptance and integration test plan

3.1 Development of the Testing Methodology

In ASSIST-IoT's DevSecOps methodology [14.], the testing phase can be broken down into two separated concepts. The first involves testing software components in the form of micro-services, before they are released, to ensure they function correctly. The second concept involves integrating and testing all software components after release to create pilot trials or use cases in a unified environment. ASSIST-IoT intertwines the phases of the DevSecOps methodology with the testing phases presented in D6.2 [15.]. The planning phase has already been implemented and the process is straightforward.

The code implementation phase involves developing the internal components of an enabler, with a focus on implementing unit tests and functional tests before committing code. No code should be committed without proper testing, although there is some flexibility in terms of deciding when to develop tests for a new component. GitLab and its CI/CD pipelines are used in ASSIST-IoT to embed tests into enabler deployment pipelines.

The build phase follows the code implementation phase and involves creating executables using dependency management tools, to ensure that all necessary libraries and components are included in the build. Security checks can also be incorporated into the build process, and integration tests can be developed to ensure optimal interaction between an enabler's internal components. This phase also involves creating versions of the enablers that are released, in order to incorporate new functionalities.

After building the enablers, ASSIST-IoT uses a unified testing environment to proceed to the testing phase of the DevSecOps lifecycle. Enablers are locally tested and ready to interact with each other to create complete use cases and pilot trials. To simulate a real IoT environment, as described in section 2.2, K8s nodes are used to replicate cloud and edge deployments. This environment allows teams to test enablers, modify and rebuild new versions, and develop end-to-end tests for easier debugging and fine-tuning of interacting components for each set of pilot trials.

Following testing, the acceptance phase begins in parallel with acceptance testing to confirm that requirements and KPIs are met. Factory acceptance testing is currently being conducted in ASSIST-IoT's testing environment, where pilot trials are simulated and tests are performed for validation.

The deployment phase involves delivering the final product to stakeholders and beginning site acceptance testing. ASSIST-IoT has just started this phase, and the deployments that are present in the testing environment will now be deployed in the pilot premises, and tested to ensure that everything is functioning as expected in the real-case scenario.

The final phase is the operation, in which the complete product of the project operates as it should in the pilot premises and is accompanied by performance testing. This phase is crucial in ensuring that the product meets the requirements and KPIs and performs well in the real-world environment.

Overall, the DevSecOps methodology implemented in the ASSIST-IoT project is a systematic and rigorous approach to software development, testing, and deployment. Each phase plays a critical role in ensuring that the final product meets the project goals and stakeholder requirements. All of the aforementioned methodology is a hands-on attempt to interweave DevSecOps methodology (Figure 23) with ASSIST-IoT testing methodology (Table 1).

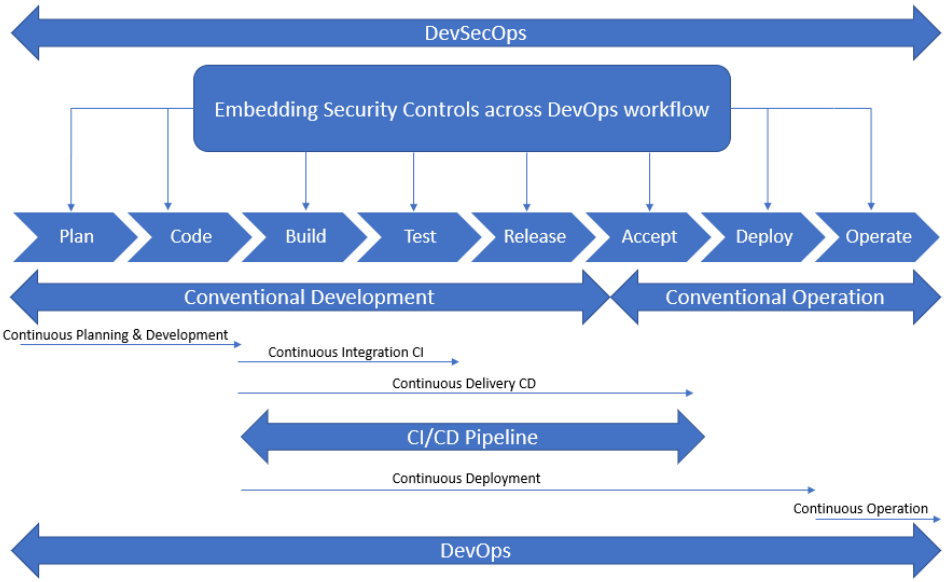


Figure 23. DevSecOps embedded security control

The general activities, frequency and responsibilities for ASSIST-IoT testing and integration methodology is summarised in the table below:

Table 1: Software Test & Integration plan

DevSecOps Phase	Level of Testing	Activities	Test environments	Frequency of testing	Responsible		
					Writing test cases	Providing test data	Running tests
Code	**Unit**	Select test cases. Write automated tests cases	Developer environment Continuous Integration Infrastructure	Create test before / while developing. Automated tests run continuously when the component is built on the CI Infrastructure.	Developer	Developer	Component/Unit provider
	Functional	Select test cases according to requirements. Prepare demos with test data. Run demos.	Developer environment CI Infrastructure	Create tests whenever new functionalities are introduced. Run tests continuously when adding the functionality to the enabler.	Developer	Developer	Developer
Build	**Integration**	Select test cases Manage unit dependencies. Write automated tests. Prepare non automated test cases.	CI Infrastructure	Automated tests run continuously when binding enablers together. Manual testing each time a new enabler is introduced to the pipeline.	Developer	Developer / Integration team	Developer / Integration team
Test	**End-to-end**	Design and prepare tests	CI Infrastructure / ASSIST-IoT	Automated tests run continuously	Developer Integration	Developer / Integration	Integration team

DevSecOps Phase	Level of Testing	Activities	Test environments	Frequency of testing	Responsible		
					Writing test cases	Providing test data	Running tests
		around pilot trials. Automate tests and run them. Design, prepare, and run manual tests.	Testing Environment	when all the enablers of a pilot trial are ready. Manual tests run whenever a new version of an enabler / component is introduced in the trial.	team	team	
Accept	**Factory Acceptance**	Define test cases according to the pilot trials. Prepare test data for real time case scenario. Run the integration tests of the system. Identify the observations and track the issues. Acceptance test review with respect to requirements.	ASSIST-IoT Testing Environment	Tests run on the integrated / production platform which will be used, whenever a trial is validated.	Integration team Pilot site stakeholders	Integration team Pilot site stakeholders	Integration team / End users / vendors / pilot site stakeholders
Deploy	**Site Acceptance**	Execute the tests designed in the previous phase. Update, refine, and align with the real pilot site. Acceptance test review with respect to requirements.	Pilot Site environment	Tess run on the pilot site environment whenever a complete trial is validated and deployed.	Integration team Pilot site stakeholders	Integration team Pilot site stakeholders	Integration team / End users / vendors / pilot site stakeholders
Operate	**Performance**	Design test cases for scalability, stress, load, endurance and extreme unlikely scenarios. Run the tests along with integration team and pilot site stakeholders. Define the boundaries that the trials cannot perform at their best.	Factory / Pilot Site environment	After the application has passed all test levels, validate the scenarios in which the designed application has the desirable performance.	Developers Integration team Pilot site stakeholders	Developers Integration team Pilot site stakeholders	Developers Integration team End users Pilot site stakeholders

Please take note that the term “Integration team” refers to the developers responsible for developing the enablers and integrating them together to deliver the functional final product, as detailed in section 4.3.

3.2 Integration progress in ASSIST-IoT

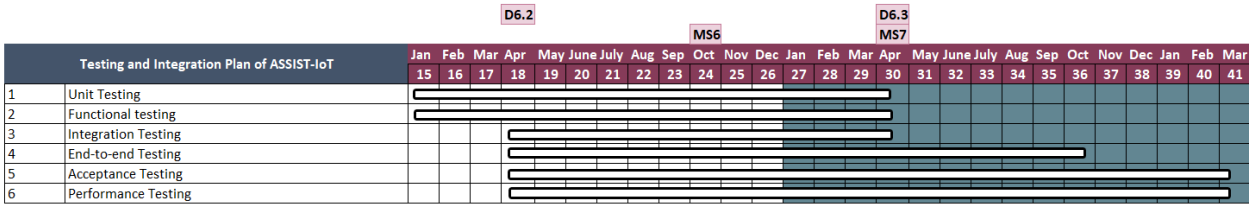
The level of packaging, currently in M30, is completed for the vast majority of the enablers (see D6.7, released jointly with this deliverable). There are a few exceptions already reported, hence this section is mostly for reporting the level of enablers' integration.

To monitor and track the integration progress, a comprehensive table has been developed in section 4.2 that captures essential information. This table enables the project team to have a clear overview of the integration progress, identify potential roadblocks or issues, cooperate and, take necessary actions to ensure the timely delivery of the integrated platform.

The importance of a unified environment has really shined in the last months, leading us to the conclusion that the integration process is not just a mere combination of individual components, but also a complex task that requires a systematic and organized approach. Thanks to the K8s deployment, the teams have been able to fully test and interact with each other's enablers, creating a solid foundation for the pilot trials. Furthermore, GitLab has played a critical role in the integration process, providing a streamlined approach to code management and testing with its CI/CD pipelines, the runners and the registries, allowing for continuous integration and deployment. The project's time plan expects the rate of integration progress to remain stable as the work transitions into establishing end-to-end testing and further enhancing our testing infrastructure.

3.3 Time plan

The time plan from the previous version of the deliverable has to be updated to include the extension period of the project. The current deliverable is the overview of the project's actions at the time of devising the deliverable on M30. The project extension requires the addition of a last deliverable presenting the entirety of the work from WP6. In other words, the last deliverable will include testing and integration along with packaging and documentation.



	Deliverables
D6.2	Testing and Integration Plan - Initial
D6.3	Testing and Integration Plan - Final

	Milestones
MS6	Integrated solution -Final integrated pilots deployed and working
MS7	Software structure finished - Software structure of enablers defined

Figure 24. ASSIST-IoT testing and integration time plan

4 Test Strategy and Results

As documented in D6.2 [15.], the methodology for unit testing in the ASSIST-IoT project has already been defined. In brief, the focus of unit testing is to test individual components within enablers to ensure that they meet the design specification requirements. The smallest testable part of each enabler's component will be defined as the "unit", and unit tests should be automated and written using a unit test framework corresponding to the programming language being used.

As unit testing is a crucial part of the software development process, it is important that all unit tests are completed and passed before committing code into repositories. Specifically, a single test should be implemented for each major method, function, class, or internal API of each enabler's component. Test inputs and outputs should be deterministic, have clear and unambiguous error messages, and have a one-to-one relationship with the functionality being tested. Some universal testing principles also apply, such as:

- Test should be designed to run independently of other tests in a clean environment and before the main code is invoked.
- Tests should be lightweight and easy to set up.
- Tests should be able to run in random order.
- The best strategy is to have the tests automated, without that meaning that manual tests cannot exist.
- Well-designed tests should be able to fail, meaning that by changing the input, the test may not always pass.

While unit testing is a vital step in software development, reporting the results of each individual test can be time-consuming and may make the document unnecessarily long. As the unit testing methodology has already been documented in D6.2, it is not necessary to report the results of unit testing in D6.3. Instead, D6.3 should focus on other types of testing, such as functional, integration, and end-to-end testing, as well as any modifications made to the existing testing methodology.

4.1 Functional testing

Functional testing is part of the testing pipeline to ensure the system functionality by testing on the requirements set by WP4 and WP5. Functional testing is treated as a black box test method where the source code is not the main point of testing, but it is the functionality.

The current version presents updates to the initial deliverable version on the functional tests. During the time from the previous version of the deliverable, enablers development was prone to changes. The current section will update on the tests of enablers. In addition, it is worth noting that in order to ensure the quality and reliability of our releases, all functional tests have to be executed, either manually or automatically, prior to any package release.

4.1.1 Functional Testing of horizontal enablers

4.1.1.1 Smart Network and Control Plane

Smart Orchestrator Enabler

Table 2: Smart Orchestrator enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Login*	A client is authenticated by the smart orchestrator by returning an access token.	A set of queries are executed to the API with right and wrong credentials. Valid returns, or authentication error messages, should be returned depending on the case.	Pass / Fail
2	*Add cluster*	A K8s cluster is attached to the orchestrator to allow deploying enablers on it.	A K8s test cluster is provisioned correctly, and a test enabler is deployed to assess that it is working.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
3	*Get clusters*	The clusters joined are returned as a JSON.	An API call is performed, returning a JSON with the test cluster added, or empty in case it has not been provisioned.	Pass / Fail
4	*Get K8s Nodes by cluster*	The worker nodes joined to a master node are returned.	Upon executing an API call, a JSON object is returned containing information about the cluster's master and worker nodes.	Pass / Fail
5	*Delete cluster*	A K8s cluster is removed from the orchestrator system.	An API call is made to remove the K8s test cluster, and it is not possible to instantiate enablers in it anymore. It can be removed only if any enabler is running in it.	Pass / Fail
6	*Add repository*	A Helm repository is registered in the orchestrator system.	A Helm repository is added, and the test enabler it contains can be instantiated.	Pass / Fail
7	*Get repository*	The Helm repositories registered are returned in JSON format.	An API call is performed, returning a JSON with the test repository added, or empty in case it has not been provisioned.	Pass / Fail
8	*Delete repository*	A Helm repository is removed from the orchestrator system.	An API call is made to remove the test repository, and it is not possible to instantiate the test enabler from it anymore.	Pass / Fail
9	*Add enabler*	An enabler is instantiated in a K8s cluster selected by a user.	An API call is made to deploy a test enabler in a cluster chosen, action that can be checked with calls to the K8s API.	Pass / Fail
10	*Get enablers in a cluster*	The enablers instantiated in an specific cluster.	An API call is executed to get the enablers deployed in a cluster in JSON format.	Pass / Fail
11	*Get enablers*	The enablers deployed and running are returned in JSON format.	An API call is performed, returning a JSON with the test enabler instantiated, or empty if it was not placed and running.	Pass / Fail
12	*Terminate enabler*	An enabler is stopped and prepared to be deleted.	An API call is made to terminate the test enabler, which stops its execution and cannot be accessed to perform any work.	Pass / Fail
13	*Delete Enabler's PV and PVC*	The PV and PVC attached to an Enabler are deleted.	When an API call is executed, the PV (persistent volume) and PVC (persistent volume claim) associated with an Enabler are removed from their respective cluster.	Pass / Fail
14	*Delete enabler*	The terminated enabler is deleted from the system.	An API call is made to delete the test enabler, which is completely removed from the system, leaving no traces in the cluster.	Pass / Fail

Table 3: Smart Orchestrator enabler's functional tests 1-8 results

Enabler	**Smart Orchestrator enabler (test 1-8)**
Description	A functional test is conducted to assess the performance of the cluster and repositories API calls. This initial test focuses on API calls that operate independently of one another, allowing them to be evaluated as separate test blocks.
Approach	The test is partially automated and relies on the Swagger-OpenAPI description. To execute the test, a JSON file containing the required parameters must be provided for retrieval and utilization.
Test tool/s	Any capable software of making HTTP request, for instance: Postman, curl or a python script with the request library.
Pre-test conditions	The Smart Orchestrator must be installed, up and running to be able to execute the API calls. The enabler needs some pre-requirements to make possible its functionality.
Additional information	The JSON file provided to the Docker container must adhere to a specific format, containing all the necessary parameters as previously outlined.

Enabler	Smart Orchestrator enabler (test 1-8)	
Test sequence	Step 1	Generate a POST request for requesting the login token.
	Step 2-a	Generate a POST request for adding a cluster or a repository to the Smart Orchestrator Enabler being identified with the token from the first step. Take the K8s Cluster ID or the K8s repository ID assigned to the cluster/repository joined. The kubeconfig associated with the cloud cluster can be obtained through a dedicated API call for testing purposes.
	Step 2-b	Verify the response returned is a status code of 200. If not, the script throws an error.
	Step 2-c	Check if the schema returned is equal to the schema defined. If it is, the test is validated. If not, the script throws an error.
	Step 3-a	Generate the GET and DELETE requests utilizing the parameters obtained from the POST request and the identifier token.
	Step 3-b	The verification process should be conducted in accordance with steps 1-b and 1-c.
Test verdict	The script verifies the response code and schema to validate the test. The verdict is based in these comparisons -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 4: Smart Orchestrator enabler's functional tests 9-14 results

Enabler	Smart Orchestrator enabler (test 9-14)	
Description	A functional test is conducted to assess the performance of the enabler API calls. This test focuses on API calls that depends on the other calls as the cluster and repository ones for the enabler instantiation.	
Approach	The test is partially automated and relies on the Swagger-OpenAPI description. To execute the test, a JSON file containing the required parameters must be provided for retrieval and utilization.	
Test tool/s	Any capable software of making HTTP request, for instance: Postman, curl or a python script with the request library.	
Pre-test conditions	The Smart Orchestrator must be installed, up and running to be able to execute the API calls. The enabler needs some pre-requirements to make possible its functionality.	
Additional information	The JSON file provided to the Docker container must adhere to a specific format, containing all the necessary parameters as previously outlined.	
Test sequence	Step 1	Generate a POST request for requesting the login token.
	Step 2	Generate a POST request for adding a cluster or a repository to the Smart Orchestrator Enabler being identified with the token from the first step. Take the K8s Cluster ID or the K8s repository ID assigned to the cluster/repository joined. The descriptive JSON should contain the "wait" option to ensure that the K8s Cluster is fully instantiated before proceeding.
	Step 3	Generate a POST request to instantiate an Enabler. From the response, the script should extract the necessary parameters. The descriptive JSON should contain the "wait" option to ensure that the Enabler is fully instantiated before proceeding.
	Step 4	Generate the GET and DELETE requests utilizing the parameters obtained from the POST requests and the identifier token.
	Step x	For each test, the script verifies if the response returned is a status code of 200, the script checks if the schema returned is equal to the schema defined. If it is, the test is validated. If the code is different or the schema is not equal to the schema defined, the script throws an error.
Test verdict	The script verifies the response code and schema to validate the test. The verdict is based in these comparisons -> Passed	
Additional logs/ Report (in case of manual)	N/A	

SDN Controller Enabler

Table 5: SDN Controller enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Network configuration*	Test of topology and network configuration parameters.	A set of commands for network configuration.	Pass / Fail
2	*API usage*	Test of API REST commands for network control function.	A set of REST requests for network configuration.	Pass / Fail
3	*GUI*	Test of GUI interface.	Network topology shown with requested configuration parameters.	Pass / Fail

Table 6: SDN Controller enabler's functional test 1 results

Enabler	**SDN Controller enabler (test 1)**	
Description	Testing of network configuration parameters and network topology with CLI commands.	
Approach	Semi-automatic using developed scripts.	
Test tool/s	No additional tools required.	
Pre-test conditions	Installed ONOS, terminal with CLI and Mininet.	
Additional information	E.g., why a specific tool have been used; if mock tools have been needed and why;	
Test sequence	Step 1	Start Mininet
	Step 2	Start ONOS configured with Mininet.
	Step 3	Start terminal with CLI.
	Step 4	Run scripts with CLI commands for network configuration proposes like: [add-flows] [add-host-intent] [cfg] [device-remove] [device-role] [devices] [flows] [get-stats] [host-remove] [hosts] [intents] [links] [nodes] [paths] [ports] [remove-intent] [resource-allocations] [resource-available] [summary] [topology]
Test verdict	Evaluation of the network configuration with applied network parameters.	
Additional logs/ Report (in case of manual)	Logs from command line interface.	

Table 7: SDN Controller enabler's functional test 2 results

Enabler	**SDN Controller enabler (test 2)**	
Description	Testing of network configuration parameters and network topology with REST API commands.	
Approach	Semi-automatic using developed scripts.	
Test tool/s	No additional tools required.	
Pre-test conditions	Installed ONOS, REST interface and Mininet.	
Additional information	Scripts with REST requests for network configuration and topology collection information.	
Test sequence	Step 1	Start Mininet
	Step 2	Start ONOS configured with Mininet.
	Step 3	Start API REST interface.

Enabler	SDN Controller enabler (test 2)	
	Step 4	Run selected REST commands for network configuration proposes like: GET/POST/PUT/DELETE /devices, links, hosts, flows, intent GET /configuration, /paths, /topology
Test verdict	Evaluation of the network configuration with applied network parameters.	
Additional logs/ Report (in case of manual)	Logs from REST API requests.	

Table 8: SDN Controller enabler's functional test 3 results

Enabler	SDN Controller enabler (test 3)	
Description	Testing of GUI interface.	
Approach	Semi-automatic using developed scripts.	
Test tool/s	No additional tools required.	
Pre-test conditions	Installed ONOS and Mininet.	
Additional information	Scripts with commands for network configuration and topology collection information.	
Test sequence	Step 1	Start Mininet
	Step 2	Start ONOS configured with Mininet.
	Step 3	Start browser with GUI interface.
	Step 4	Run scripts for network configuration and creation.
Test verdict	Evaluation of the network configuration with applied network parameters.	
Additional logs/ Report (in case of manual)	Screenshots from GUI interface.	

Auto-configurable network enabler

Table 9: Auto-configurable network enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Network resources optimizations*	Network resources optimization based on overall throughput.	Network traffic distribution is optimised according to throughput.	Pass / Fail
2	*Network quality parameters optimization*	Network quality parameters optimization based on losses and latency.	Network traffic distribution is optimized according to network QoS parameters.	Pass / Fail

Table 10: Auto-configurable network enabler's functional test 1 results

Enabler	Auto-configurable network enabler (test 1)	
Description	Testing of network resources optimization based on overall throughput.	
Approach	Manual testing for network configuration scenarios.	
Test tool/s	Traffic generator tool.	
Pre-test conditions	Installed SDN controller enabler, Mininet, and traffic generator.	
Additional information	Scripts with commands for network configuration and topology collection information. Sflow-rt monitoring tool installed, AI module installed.	
Test sequence	Step 1	Start Mininet

Enabler	Auto-configurable network enabler (test 1)	
	Step 2	Start SDN controller enabler configured with Mininet (specified network topology).
	Step 3	Start monitoring tool.
	Step 4	Start Auto-configurable network enabler (AI module).
	Step 5	Run scripts for traffic generation.
Test verdict	Evaluation of network resources optimisation (optimal traffic distribution in the network).	
Additional logs/ Report (in case of manual)	Logs from monitoring tool (throughput).	

Table 11: Auto-configurable network enabler's functional test 2 results

Enabler	Auto-configurable network enabler (test 2)	
Description	Testing of optimisation for network quality parameters based on losses and latency.	
Approach	Manual testing for network configuration scenarios.	
Test tool/s	Traffic generator tool.	
Pre-test conditions	Installed SDN controller enabler and Mininet, and traffic generator.	
Additional information	Scripts with commands for network configuration and topology collection information. Sflow-rt monitoring tool installed, AI module installed.	
Test sequence	Step 1	Start Mininet
	Step 2	Start SDN controller enabler configured with Mininet (specified network topology).
	Step 3	Start monitoring tool for losses and latency measurement.
	Step 4	Start Auto-configurable network enabler (AI module).
	Step 5	Run scripts for traffic generation with specified scenarios of generation.
Test verdict	Evaluation of optimal traffic distribution in the network to achieve minimal values of QoS parameters: packet losses and delays.	
Additional logs/ Report (in case of manual)	Logs from monitoring tool (throughput, losses, delays).	

Traffic Classification Enabler

Table 12: Traffic Classification enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Preprocess*	Raw .pcap files with data from specific application and traffic types are preprocessed correctly to be used for training.	After the operation, compressed JSON files for each .pcap file are generated, and a message informing of the success is returned.	Pass / Fail
2	*Create train and test set*	Preprocess data is are separated into training and validation, and further prepared for training	After the operation, parquet files are split for training and testing (80% for training), and parquet files are generated. A message informing of the success is returned.	Pass / Fail
3	*Train model*	With a dedicated model and a database present in the host, the training module will be able to train a model to classify packets.	An API call will be made to return a model with test samples, for CNN and Resnet models, and application and traffic specific types. A message informing of the success is returned.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
4	*Classify packets*	The inference component will classify packets according to different classes.	A set of API calls will be made to validate that the inference operation (for CNN and Resnet models, and application and traffic specific types) work as expected, for a .pcap file with two email packets.	Pass / Fail

Table 13: Traffic Classification enabler's functional tests results

Enabler	**Traffic classification enabler tests**	
Description	Formal description of functional tests. All tests follow the same steps based on the ability to self-manage without dependencies on other enablers.	
Approach	Fully automatic. OpenAPI Swagger file-based pipeline integration.	
Test tool/s	This enabler only requires any software capable of executing REST API calls such as POSTMAN, Gitlab pipelines or a script with curl or similar software.	
Pre-test conditions	Enabler deployed and prerequisites specified in documentation applied. This enabler also requires a set of folder/files present in the staging environment host to test its proper performance (at least, raw .pcap files with correct naming for the pre-processing endpoint).	
Additional information	-	
Test sequence	Step 1	Send an HTTP Request (GET, POST) to each respective endpoint. In case of POST request, it is mandatory to include a custom body specified in the swagger file. Specifically, the involved endpoints are the following, to be executed in this order: 1. /vx/preprocess (for 2xx response, .pcap files must be present) 2. /vx/create-train-test-set 3. /vx/train (with cnn model and app type) 4. /vx/train (with cnn model and traffic type) 5. /vx/train (with resnet model and app type) 6. /vx/train (with resnet model and traffic type) 7. /vx/inference_app_cnn 8. /vx/inference_traffic_cnn 9. /vx/inference_app_resnet 10. /vx/inference_traffic_resnet
	Step 2a-1	Check if an HTTP 2xx or 3xx response code is returned.
	Step 2a-2	Check if response schema matches with the HTTP Request response.
	Step 2a-3	If last 2 steps are successful, it returns the success of the operation.
	Step 2b-1	Check if an HTTP 4xx or 5xx response code is returned.
	Step 2b-2	Response log is returned.
	Step 3	Go to the next endpoint assessed
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Multi-link enabler

Table 14: Multi-link enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Start Multi-link server*	Start the multilink server, bringing up the bridge interface and set up the tunnels.	Successfully bringing up the interface and tunnels of the server.	Pass / Fail
2	*Get server key*	Get the key of the tunnels from the server.	Successfully received the key of the tunnels from the server.	Pass / Fail
3	*Start Multi-link client*	Start the multilink client, bringing up the bond interface and set up the tunnels.	Successfully bringing up the interface and tunnels of the client.	Pass / Fail
4	*Check client-server connection*	Ping between the client and server multilink's interfaces.	The connection between the client and the server is successfully achieved by multilink interfaces/tunnels.	Pass / Fail
5	*Backup interface*	Bring down primary interface, the backup interface will bring up and will be selected by the bond.	Change the active slave in the bond to backup interface and check the client-server connection persists.	Pass / Fail
6	*Reselect primary interface*	. The active slave in the bond is the backup interface, if the primary interface brings up, the bond changes active slave to primary interface.	Check in the bond information that the active slave changes to the primary interface.	Pass / Fail
7	*All interfaces down*	Bring down all the interfaces (primary and backup's)	The connectivity between client and server is lost.	Pass / Fail

Table 15: Multi-link enabler's functional tests results

Enabler	**Multi-link Enabler tests**	
Description	Formal description of functional tests. All tests follow the same steps based on the ability to self-manage without dependencies on other enablers.	
Approach	Semi-automatic. The interfaces can bring up/down with API request, but it is interesting to test losing physical channel connection like unplug Ethernet cable, turn off WiFi AP, etc.	
Test tool/s	Any tool capable of executing HTTP requests. Creating a script with environment variables is a valid option.	
Pre-test conditions	Multilink server and client has to be deployed in different machines and both of them have to be connected between them by at least two different interfaces (Ethernet and WiFi for example).	
Additional information	Check carefully the OpenAPI of the enabler and the documentation provided.	
Test sequence	Step 1	Start server (POST) following the schema.
	Step 2	Start client (POST) following the schema.
	Step 3	Connection test (POST) following the schema.
	Step 4	Bring down* primary interface (GET) following the schema.
	Step 5	Check backup interface as active slave in bond (GET) following the schema
	Step 6	Check connection like in Step 3
	Step 7	Bring down* backup interface (GET) following the schema

Enabler	Multi-link Enabler tests	
	Step 8	Check connection like in Step 3. In this step the connection has to be lost http response (4xx)
	Step 9	Bring up* backup interface (GET) following the schema
	Step 10	Apply steps 5 and 6
	Step 11	Bring up* primary interface (GET) following the schema
	Step 12	Apply steps 5 and 6
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation -> Passed	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

*It is recommended that the up/down shifting of the interfaces was done in the physical connection channel, like unplug the ethernet cable from the interface or disconnecting the WiFI AP.

**This test is valid for the actual implementation but it will be evaluate the implementation of a solution cloud-native approach with respect to the existing (host level)

SD-WAN enabler

All information on acronyms and objects description is available on the official documentation page:

https://assist-iot-enablers-documentation.readthedocs.io/en/latest/horizontal_planes/smart/sd_wan_enabler.html

Table 16: SD-WAN enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Create an overlay*	Register new overlay to manage an environment	Relational information from new overlay is received.	Pass / Fail
2	*Get all overlays*	Get information from all overlays available.	Relational information from all overlays available is received.	Pass / Fail
3	*Get specific overlay*	Get information from a specific overlay.	Relational information from a specific overlay is received.	Pass / Fail
4	*Update already created overlay*	Update information from a specific overlay.	Overlay's information with the new changes is received.	Pass / Fail
5	*Delete an overlay*	Delete an existing overlay.	Output doesn't show any issue related with the deletion operation.	Pass / Fail
6	*Create a proposal*	Register new proposal to define type of communications encryption	Relational information from new proposal is received.	Pass / Fail
7	*Get all proposals*	Get information from all proposals available.	Relational information from all proposals available are received.	Pass / Fail
8	*Get specific proposal*	Get information from a specific proposal.	Relational information from a specific proposal is received.	Pass / Fail
9	*Update already created proposal*	Update information from a specific proposal.	Proposal's information with the new changes is received.	Pass / Fail
10	*Delete a proposal*	Delete an existing proposal.	Output doesn't show any issue related with the deletion operation.	Pass / Fail
11	*Create subnet range for edge virtual IPs.*	Register new IPRange for an existing overlay	Relational information from new IPRange is received	Pass / Fail
12	*Get all IPRanges from an existing Overlay*	Get IPRanges created and available from an existing Overlay.	Relational information from IPRanges is received.	Pass / Fail
13	*Get specific IPRange already created from an existing Overlay*	Get information from a specific IPRange from an existing Overlay.	Relational information from an IPRange object is correctly received.	Pass / Fail
14	*Update already created IPRange from an Overlay*	Update information or subnet range from a specific IPRange linked to an existing Overlay.	IPRange's information with the new changes is correctly received.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
15	*Delete an IPRange from an Overlay.*	Delete an existing IPRange linked to an available Overlay.	Output doesn't show any issue related with the deletion operation.	Pass / Fail
16	*Register new hub*	Configuring the hub environment passing specific information and kubeconfig	Relational information from new hub is correctly received or display some issues relation with bad configuration or bad kubeconfig	Pass / Fail
17	*Get all hubs already registered*	Get information from all hubs deployed and configured correctly.	Relational information from hubs available in a specific Overlay is correctly received.	Pass / Fail
18	*Get specific hub already registered in specific Overlay*	Get information from a specific deployed and configured hub in a specific Overlay.	Relational information from a specific hub available in specific Overlay is correctly received.	Pass / Fail
19	*Update already available hub from a specific Overlay environment.*	Update information (no kubeconfig) about an existed and deployed hub from and available Overlay	Hub's information with the new changes is received.	Pass / Fail
20	*Delete existing hub*	Delete an existing hub linked to and available Overlay.	Output doesn't show any issue related with the deletion operation or problems derivative with the communication with the hub.	Pass / Fail
21	*Register new edge device*	Configuring the edge device environment passing specific information and kubeconfig	Relational information from new edge device is correctly received or display some issues relation with bad configuration or bad kubeconfig	Pass / Fail
22	*Get all edge devices already registered*	Get information from all edge devices deployed and configured correctly.	Relational information from edge devices available in a specific Overlay is correctly received.	Pass / Fail
23	*Get specific edge device already registered in specific Overlay*	Get information from a specific deployed and configured edge device in a specific Overlay.	Relational information from a specific edge device available in specific Overlay is correctly received.	Pass / Fail
24	*Update already available edge from a specific Overlay environment.*	Update information (no kubeconfig) about an existed and deployed edge device from and available Overlay	Edge device's information with the new changes is received.	Pass / Fail
25	*Delete existing edge device*	Delete an existing edge device linked to and available Overlay.	Output doesn't show any issue related with the deletion operation or problems derived with the communication with the edge device.	Pass / Fail
26	*Create a connection between a hub and an edge device*	Set up a new connection between existing and deployed hub and device edge. Create a new virtual IP and corresponding tunnels between them.	Relational information from connection is correctly received.	Pass / Fail
27	*Get all connections between all edge devices in a specific hub from a specific Overlay environment.*	Get information derived by hub's connections with the edge devices with the virtual IPs correctly assigned and the status of connection.	Relational information from connections is correctly received.	Pass / Fail
28	*Delete connection between an existing and deployed hub and device edge.*	Delete the connection already done deleting all the configurations realised previously in the edge device like the virtual IP assigned, etc.	Output doesn't show any issue related the operation or problems derived with the communication with the hub or edge device.	Pass / Fail
29	*Get device connections in an environment currently working.*	Display the edge devices with their virtual IPs assigned in an environment by a specific Overlay.	Relational information from connection between edge devices is correctly received.	Pass / Fail

Table 17: SD-WAN enabler's functional tests 1-15 results

Enabler	**SD-WAN Enabler (test 1-15)**	
Description	The following tests can be run to create a specific environment for interconnecting clusters using SD-WAN technology. For these tests, it is not necessary to have a real environment, but can be considered the preconditions for creating successful connections.	
Approach	Semi-automatic. In fact, this test can be run fully automatically using, for example, a Gitlab pipeline. However, there is little use in automatically conducting this test if there is no clear relation between the environment and a specific future scenario.	
Test tool/s	Any tool capable of executing HTTP requests. Creating a script with environment variables is a valid option.	
Pre-test conditions	For this test it is not mandatory to have the wan-acceleration enabler deployed. In the case of simply testing, having an environment with Kubernetes and helm software is sufficient.	
Additional information	It is recommended to have the wan acceleration enabler previously installed in the environment.	
Test sequence	Step 1-a	Create (POST) an overlay following the schema.
	Step 1-b	Get the list of full created overlays.
	Step 1-c	Get a specific overlay previously created.
	Step 1-d	Update description or some relevant information of overlay.
	Step 1-e	Delete the overlay previously created.
	Step 1-x	For each test 1-(a-e) check the success of the operation by observing the response code and the displayed output.
	Step 2	Recreate an overlay like ‘Step 1-a’
	Step 2-a	Create (POST) a proposal following the schema.
	Step 2-b	Get the list of full created proposals.
	Step 2-c	Get a specific proposal previously created.
	Step 2-d	Update description or some relevant information of overlay.
	Step 2-e	Delete the proposal previously created.
	Step 2-x	For each test 2-(a-e) check the success of the operation by observing the response code and the displayed output.
	Step 3	Recreate a proposal like ‘Step 2-a’.
	Step 3-a	Create (POST) an IPRange following the schema.
	Step 3-b	Get the list of full created IPRanges.
	Step 3-c	Get a specific IPRange previously created.
	Step 3-d	Update description or some relevant information of IPRange.
	Step 3-e	Delete the IPRange previously created
	Step 3-f	Repeat ‘Step 3-a’
	Step 3-x	For each test 3-(a-f) check the success of the operation by observing the response code and the displayed output.
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation. -> Pass	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

Table 18. SD-WAN enabler's functional tests 16-20 results

Enabler	**SD-WAN Enabler (test 16-20)**
Description	For realise this test it is needed a cluster acting as a hub. Central nodus is created to manage and redirect SD-WAN communications.

Approach	Fully manual. All pertinent cluster information (IPs, Kubeconfig, etc.) must be known in advance for this test.	
Test tool/s	Any tool capable of executing HTTP requests.	
Pre-test conditions	• Cluster with Calico CNI installed. • Kubernetes and helm software deployed. • WAN-Acceleration Enabler deployed and configured with its respective networks. • Test 1-15 passed and Overlay, proposal and IPRange successfully created.	
Additional information	It is highly recommended to have read all the relevant documentation to execute this test.	
Test sequence	Step 1	Get cluster Kubeconfig and encrypt in base64.
	Step 2	Get public IPs of the CNF (generated by WAN-Acceleration Enabler).
	Step 3	Test connection between SD-WAN environment and the future HUB Cluster.
	Step 4-a	Create (POST) a HUB registration following the schema.
	Step 4-b	Get the list of full created and configured HUBs.
	Step 4-c	Get a specific information about registered HUB.
	Step 4-d	Update description or some secondary data of registered HUB
	Step 4-e	Delete HUB configuration previously created.
	Step 4-x	For each test 4-(a-e) check the success of the operation by observing the response code and the displayed output.
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation. -> Passed	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

Table 19. SD-WAN enabler's functional tests 21-25 results

Enabler	**SD-WAN Enabler (test 21-25)**	
Description	For realise this test it is needed a cluster acting as an edge node. Device edge node is created to be managed and having SD-WAN communications.	
Approach	Fully manual. All pertinent cluster information (IPs, Kubeconfig, etc.) must be known in advance for this test.	
Test tool/s	Any tool capable of executing HTTP requests.	
Pre-test conditions	• Cluster with Calico CNI installed. • Kubernetes and helm software deployed. • WAN-Acceleration Enabler deployed and configured with its respective networks. • Test 1-15 passed and Overlay, proposal and IPRange successfully created.	
Additional information	It is highly recommended to have read all the relevant documentation to execute this test.	
Test sequence	Step 1	Get cluster kubeconfig and encrypt in base64.
	Step 2	Get public IPs of the CNF (generated by WAN-Acceleration Enabler).
	Step 3	Test connection between SD-WAN environment and the future Edge Cluster.
	Step 4-a	Create (POST) a Device registration following the schema.
	Step 4-b	Get the list of full created and configured Devices.
	Step 4-c	Get a specific information about registered Device.
	Step 4-d	Update description or some secondary data of registered Device
	Step 4-e	Delete Device configuration previously created.

	Step 4-x	For each test 4-(a-e) check the success of the operation by observing the response code and the displayed output.
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation. -> Passed	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

Table 20. SD-WAN enabler's functional tests 26-29 results

Enabler	**SD-WAN Enabler (test 26-29)**	
Description	Once the information about the overlays, proposals, IPRange and the clusters that will act as HUB and those that will act as Edge nodes or Devices have been configured. The connection that will create the SD-WAN communication environment will be made.	
Approach	Fully manual. In order to conduct this test, the devices and hubs to communicate with one another must be defined.	
Test tool/s	Any tool capable of executing HTTP requests.	
Pre-test conditions	• Cluster with Calico CNI installed. • Kubernetes and helm software deployed. • WAN-Acceleration Enabler deployed and configured with its respective networks. • Test 1-15 passed and Overlay, proposal and IPRange successfully created. • Test 16-20 passed and successfully created HUB configuration. • Test 21-25 passed and successfully created Devices configurations.	
Additional information	It is highly recommended to have read all the relevant documentation to execute this test.	
Test sequence	Step 1	Once the HUB has been determined, the devices requiring the interconnection between them will be assigned. The hub will act as a central node.
	Step 2	Create (POST) a HUB-Device registration following the schema.
	Step 3	Test connection between HUB and Device/edge by new OVN network. This test can be done by ICMP requests. This step is repeated by all the devices.
	Step 4	Get the list of full connections in a HUB cluster.
	Step 5	Test connection between DEVICES using the new OVN network. This test can be done by ICMP request.
	Step 6	Check that the 'ipsecsite' manifests in the HUB cluster and the 'ipsechost' manifests in the DEVICE clusters have been successfully created. It is also possible to check this within the CNF in the IPSEC section.
	Step 7	Delete connections between HUB and DEVICEs. All the configurations should be eliminated.
	Step X	For each test 1-7 check the success of the operation by observing the response code and the displayed output.
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation. -> Passed	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

WAN Acceleration enabler

All information on acronyms and objects description is available on the official documentation page for WAN Acceleration enabler's Readthedocs.

Table 21: WAN Acceleration enabler's functional tests

Nº	**Test**	**Description**	**Evaluation criteria**	**Results**
1	*Create firewall zone*	Configure new firewall zone in an existing edge device	Relational information from new firewall zone created is correctly received.	Pass/Fail

Nº	Test	Description	Evaluation criteria	Results
2	*Get firewalls zones*	Display all firewall zones applied in an existing edge device	Relational information from firewall zones applied is correctly received.	Pass / Fail
3	*Get specific firewall zone*	Display a specific firewall zone in an existing edge device	Relational information from a firewall zone applied is correctly received.	Pass / Fail
4	*Delete firewall zone*	Delete existing firewall zone in an existing edge device	Output doesn't show any issue related the operation or problems derived with the communication with the edge device chosen.	Pass / Fail
5	*Create firewall SNAT*	Configure new firewall SNAT in an existing edge device	Relational information from new firewall SNAT created is correctly received.	Pass / Fail
6	*Get firewalls SNATs*	Display all firewall SNAT applied in an existing edge device	Relational information from firewall SNATs applied is correctly received.	Pass / Fail
7	*Get specific firewall SNAT*	Display a specific firewall SNAT applied in an existing edge device	Relational information from a specific firewall SNAT applied is correctly received	Pass / Fail
8	*Delete firewall SNAT*	Delete existing firewall SNAT in an existing edge device	Output doesn't show any issue related the operation or problems derived with the communication with the edge device chosen.	Pass / Fail
9	*Create firewall DNAT*	Configure new firewall DNAT in an existing edge device	Relational information from new firewall DNAT created is correctly received.	Pass / Fail
10	*Get firewalls DNATs*	Display all firewall DNAT applied in an existing edge device	Relational information from firewall DNATs applied is correctly received.	Pass / Fail
11	*Get specific firewall DNAT*	Display a specific firewall DNAT applied in an existing edge device	Relational information from a specific firewall DNAT	Pass / Fail
12	*Delete firewall DNAT*	Delete existing firewall DNAT in an existing edge device	Output doesn't show any issue related the operation or problems derived with the communication with the edge device chosen.	Pass / Fail
13	*Create firewall forwarding*	Configure new firewall forwarding in an existing edge device	Relational information from new firewall forwarding created is correctly received.	Pass / Fail
14	*Get firewalls forwarding*	Display all firewall forwarding applied in an existing edge device	Relational information from firewall forwarding applied is correctly received.	Pass / Fail
15	*Get specific firewall forwarding*	Display a specific firewall forwarding applied in an existing edge device	Relational information from a specific firewall forwarding is correctly received.	Pass / Fail
16	*Delete firewall forwarding*	Delete existing firewall forwarding in an existing edge device	Output doesn't show any issue related the operation or problems derived with the communication with the edge device chosen.	Pass / Fail
17	*Create firewall rule*	Configure new firewall rule in an existing edge device	Relational information from new firewall rule created is correctly received.	Pass / Fail
18	*Get firewall rules*	Display all firewall rule applied in an existing edge device	Relational information from firewall rules applied is correctly received.	Pass / Fail
19	*Get specific firewall rule*	Display a specific firewall rule applied in an existing edge device	Relational information from a specific firewall rule is correctly received.	Pass / Fail
20	*Delete firewall rule*	Delete existing firewall rule in an existing edge device	Output doesn't show any issue related the operation or problems derived with the communication with the edge device chosen.	Pass / Fail
21	*Create mwan3 policy*	Configure new mwan3 policy in an existing edge device	Relational information from new mwan3 policy created is correctly received.	Pass / Fail
22	*Get mwan3 policies*	Display all mwan3 policy applied in an existing edge device	Relational information from mwan3 policies applied is correctly received.	Pass / Fail
23	*Get specific mwan3 policy*	Display existing mwan3 policy applied in an existing edge device	Relational information from a specific mwan3 policy applied is correctly received	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
24	*Delete mwan3 policy*	Delete existing mwan3 policy in an existing edge device	Output doesn't show any issue related the operation or problems derived with the communication with the edge device chosen.	Pass / Fail
25	*Create mwan3 rule*	Configure new mwan3 rule in an existing edge device	Relational information from new mwan3 rule created is correctly received.	Pass / Fail
26	*Get mwan3 rules*	Display all mwan3 rule applied in an existing edge device	Relational information from mwan3 rules applied is correctly received.	Pass / Fail
27	*Get specific mwan3 rule*	Display existing mwan3 rule in an existing edge device	Relational information from a specific mwan3 rule is correctly received.	Pass / Fail
28	*Delete mwan3 rule*	Delete existing mwan3 rules in an existing edge device	Output doesn't show any issue related the operation or problems derived with the communication with the edge device chosen.	Pass / Fail
39	*Get version*	Display enabler version	The correct version string is correctly display	Pass / Fail
30	*Get health*	Display health status of the environment	The current enabler environment is healthy or not.	Pass / Fail

Table 22. WAN Acceleration enabler's functional tests 1-20 results

Enabler	**WAN Acceleration Enabler (test 1-20)**	
Description	The following test can be run to create/configure a specific environment to implement firewall rules applied in the CNF for each cluster. This test checks the connection and behaviour between the cluster components. In addition, it allows outputs and inputs connections of other components to other clusters via SD-WAN communications.	
Approach	Fully manual, for this test, we need to create some manifests via API REST and test the success of the operation in creation and implementation with specific components not integrated in the own enabler (like nginx or httpbin component/pod).	
Test tool/s	Any tool capable of executing HTTP requests.	
Pre-test conditions	• Cluster with Calico CNI installed. • Kubernetes and helm software deployed. • WAN-Acceleration Enabler deployed and configured with its respective networks and prerequisites. • Some test components capable to interact with the CNF via OVN networks.	
Additional information	It is highly recommended to have read all the relevant documentation to execute this test.	
Test sequence	Step 1	Get information about networks already created
	Step 2-a	Create (POST) firewall zone following the schema.
	Step 2-b	Get the specific firewall zone previously created.
	Step 2-c	Delete firewall zone
	Step 2-d	Recreate firewall zone like 'Step 2-1'
	Step 2-x	For each test 2-(a-d) check the success of the operation by observing the response code, displayed output and firewall zone CRDS successfully created and deleted.
	Step 3-a	Create (POST) firewall SNAT following the schema with the specific firewall zone and virtual IP assigned for CNF.
	Step 3-b	Get the specific firewall SNAT previously created.
	Step 3-c	Delete firewall SNAT
	Step 3-x	For each test 3-(a-c) check the success of the operation by observing the response code, displayed output and firewall SNAT CRDS successfully created and deleted.
	Step 4	Get information about components already created and configured with OVN networks.
	Step 5-a	Create (POST) firewall DNAT following the schema with the specific firewall zone, virtual IP assigned for CNF and OVN network assigned to the specific

Enabler	WAN Acceleration Enabler (test 1-20)	
		component.
	Step 5-b	Get the specific firewall DNAT previously created.
	Step 5-c	Delete firewall DNAT
	Step 5-x	For each test 5-(a-c) check the success of the operation by observing the response code, displayed output and firewall DNAT CRDS successfully created and deleted.
	Step 6-a	With the information obtained in 'Step 4', create (POST) firewall forwarding following the schema with the specific firewall zone, virtual IP assigned for CNF and port desired.
	Step 6-b	Get the specific firewall forwarding previously created.
	Step 6-c	Delete firewall forwarding
	Step 6-x	For each test 6-(a-c) check success of the operation by observing the response code, displayed output and firewall forwarding CRDS successfully created and deleted.
	Step 7-a	With the information obtained in 'Step 4', create (POST) firewall rule following the schema with the specific firewall zone.
	Step 7-b	Get the specific firewall rule previously created.
	Step 7-c	Delete firewall rule
	Step 7-x	For each test 7-(a-c) check success of the operation by observing the response code, displayed output and firewall rules CRDS successfully created and deleted.
	Step 8	Delete firewall zone.
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation -> Passed	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

Table 23. WAN Acceleration enabler's functional tests 21-28 results

Enabler	WAN Acceleration Enabler (test 21-28)	
Description	The following test can be run to create/configure a specific environment to implement mwan3 policy and rules applied in the CNF for each cluster. This test checks the connection and behaviour between the cluster components. In addition, it allows outputs and inputs connections of other components to other clusters via SD-WAN communications.	
Approach	Fully manual, for this test, we need to create some manifests via API REST and test the success of the operation in creation and implementation.	
Test tool/s	Any tool capable of executing HTTP requests.	
Pre-test conditions	• Cluster with Calico CNI installed. • Kubernetes and helm software deployed. • WAN-Acceleration Enabler deployed and configured with its respective networks and prerequisites. • Some test components capable to interact with the CNF via OVN networks.	
Additional information	It is highly recommended to have read all the relevant documentation to execute this test.	
Test sequence	Step 1	Get information about networks already created
	Step 2-a	Create (POST) mwan3 policy following the schema.
	Step 2-b	Get the specific mwan3 policy previously created.
	Step 2-c	Delete mwan3 policy
	Step 2-d	Recreate mwan3 policy like 'Step 2-a'
	Step 2-x	For each test 2-(a-d) check the success of the operation by observing the response code, displayed output and mwan3 policy CRDS successfully created and deleted.

Enabler	WAN Acceleration Enabler (test 21-28)	
	Step 3-a	Create (POST) mwan3 rule following the schema with the specific mwan3 policy for each network created.
	Step 3-b	Get the specific mwan3 rule previously created.
	Step 3-c	Delete mwan3 rule.
	Step 3-x	For each test 3-(a-c) check the success of the operation by observing the response code, displayed output and mwan3 rule CRDS successfully created and deleted.
	Step 4	Delete mwan3 policy.
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation -> Passed	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

Table 24. WAN Acceleration enabler's functional tests 28-30 results

Enabler	WAN Acceleration Enabler (test 28-30)	
Description	Get version and health (common endpoint) of WAN Acceleration Enabler.	
Approach	Fully automatic. In fact, these two endpoints are the only ones you can test fully automatic in this enabler.	
Test tool/s	Any tool capable of executing HTTP requests. GitLab pipeline functional testing is also possible.	
Pre-test conditions	• Cluster with Calico CNI installed. • Kubernetes and helm software deployed. • WAN-Acceleration Enabler deployed and configured with its respective networks and prerequisites.	
Additional information	It is highly recommended to have read all the relevant documentation to execute this test.	
Test sequence	Step 1	Get version
	Step 2	Get health
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation -> Passed	
Additional logs/ Report (in case of manual)	In error case, the logs are showed in display output line.	

VPN Enabler

The tests related to the management of VPN clients (generation of keys, provisioning, enabling, disabling and deleting them) are those stated for the VPN enabler, and have to be passed also under the scope of this enabler as the underlying technology is different. Also, the following tests have to be passed:

Table 25: VPN enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Network Interface info*	The enabler returns the information about the network interface of the VPN server	The information about the network interface of the VPN server successfully obtained and is not empty.	Pass / Fail
2	*Generate keys*	The enabler generates the needed keys (public, private and pre-shared) to create a new VPN client.	The generated keys are successfully generated and are obtained in JSON format.	Pass / Fail
3	*Create client*	The enabler creates a new VPN client.	The client is listed in the information about the network interface of the VPN server and a VPN	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
			connection can be stablished using the generated client (test #7).	
4	*Delete client*	The enabler deletes a VPN client.	The client is not listed in the information about the network interface of the VPN server and a VPN connection cannot be stablished using the generated client (test #7).	Pass / Fail
5	*Enable client*	A VPN client is enabled (that was previously disabled).	The client is listed in the information about the network interface of the VPN server and a VPN connection can be stablished using the enabled client (test #7).	Pass / Fail
6	*Disable client*	A VPN client is disabled (not eliminated).	The client is not listed in the information about the network interface of the VPN server and a VPN connection cannot be stablished using the disabled client (test #7).	Pass / Fail
7	*Connect to VPN*	A user connects to the VPN using a VPN client program configured with a previously created client.	Make a ping to the IP address of the VPN server network interface and, depending on the VPN network configuration, to other hosts and services that are only accessible via the VPN. Furthermore, the VPN client program provides information about the VPN connection status.	Pass / Fail

Table 26: VPN enabler's functional tests 1 and 2 results

Enabler	**VPN Enabler (tests 1-2)**	
Description	Functional tests 1 and 2 for the VPN enabler that only involve the API component of the enabler.	
Approach	Semi-automatic	
Test tool/s	• REST API client (e.g., Postman or cURL)	
Pre-test conditions	The enabler itself must be deployed.	
Additional information	N/A	
Test sequence	Step 1	Send an HTTP GET request to the */info*, */info/conf* and */keys* endpoints
	Step 2	Check if an HTTP 200 code, the expected information about the network interface of the VPN server and the generated keys are returned respectively.
Test verdict	The test only passes if it is returned an HTTP 200 code and the expected information about the network interface of the VPN. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 27: VPN enabler's functional test 3 result

Enabler	**VPN Enabler (test 3)**
Description	Functional test 3 for the VPN enabler that involves the VPN Server and API components of the enabler.
Approach	Fully manual because the connection to the VPN must be performed manually from a machine located in another network where the VPN enabler is deployed.
Test tool/s	• REST API client (e.g., Postman or cURL) • WireGuard VPN client (TunSafe for Windows and WireGuard CLI for Linux)
Pre-test conditions	The enabler itself must be deployed and the VPN client keys must be previously generated (public, private and preshared). These keys can be obtained during the test #1.

Enabler	VPN Enabler (test 3)	
Additional information	For testing the VPN client in a machine using Windows, use TunSafe instead of the official WireGuard client because the latter doesn't work on specific VPN network configurations.	
Test sequence	Step 1	Send an HTTP POST request to the */client* endpoint with a body including the previously generated public and preshared keys.
	Step 2	Check if the information about the created client is returned along with an HTTP 200 code.
	Step 3	Perform the test #1 (without the keys generation part) to check if the client is listed in the network interface information.
	Step 4	Perform the test #9 to check if the connection to the VPN sever is possible using the new client's credentials.
Test verdict	The test only passes if both the new client is listed in the network interface information and the connection to the VPN server can be performed using the new client's credentials. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 28: VPN enabler's functional test 4 results

Enabler	VPN Enabler (test 4)	
Description	Functional test 4 for the VPN enabler that involves the VPN Server and API components of the enabler.	
Approach	Fully manual because the connection to the VPN must be performed manually from a machine located in another network where the VPN enabler is deployed.	
Test tool/s	• REST API client (e.g., Postman or cURL) • WireGuard VPN client (TunSafe for Windows and WireGuard CLI for Linux)	
Pre-test conditions	Enabler deployed and a client created (test #3).	
Additional information	For testing the VPN client in a machine using Windows, use TunSafe instead of the official WireGuard client because the latter doesn't work on specific VPN network configurations.	
Test sequence	Step 1	Send an HTTP DELETE request to the */client* endpoint with a body including the public key of a previously created client.
	Step 2	Check if an HTTP 200 code is returned.
	Step 3	Perform the test #1 (without the keys generation part) to check if the client is listed in the network interface information.
	Step 4	Perform the test #9 to check if the connection to the VPN server is possible using the credentials of the deleted client.
Test verdict	The test only passes if both the deleted client is not listed in the network interface information and the connection to the VPN server cannot be performed using the deleted client's credentials. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 29: VPN enabler's functional test 5 results

Enabler	VPN Enabler (test 5)
Description	Functional test 5 for the VPN enabler that involves the VPN Server and API components of the enabler.
Approach	Fully manual because the connection to the VPN must be performed manually from a machine located in another network where the VPN enabler is deployed.
Test tool/s	• REST API client (e.g., Postman or cURL) • WireGuard VPN client (TunSafe for Windows and WireGuard CLI for Linux)
Pre-test conditions	Enabler deploye and a client created and disabled.
Additional information	For testing the VPN client in a machine using Windows, use TunSafe instead of the official WireGuard client because the latter doesn't work on specific VPN network configurations.

Enabler	VPN Enabler (test 5)	
Test sequence	Step 1	Send an HTTP PUT request to the */client/enabled* endpoint with a body including the public key of a previously disabled client.
	Step 2	Check if an HTTP 204 code is returned.
	Step 3	Perform the test #1 (without the keys generation part) to check if the client is listed in the network interface information.
	Step 4	Perform the test #9 to check if the connection to the VPN server is possible using the credentials of the enabled client.
Test verdict	The test only passes if both the enabled client is listed in the network interface information and the connection to the VPN server can be performed using the enabled client's credentials. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 30: VPN enabler's functional test 6 results

Enabler	VPN Enabler (test 6)	
Description	Functional test 6 for the VPN enabler that involve the VPN Server and API components of the enabler.	
Approach	Fully manual because the connection to the VPN must be performed manually from a machine located in another network where the VPN enabler is deployed.	
Test tool/s	• REST API client (e.g., Postman or cURL) • WireGuard VPN client (TunSafe for Windows and WireGuard CLI for Linux)	
Pre-test conditions	Enabler deployed and a client created and enabled.	
Additional information	For testing the VPN client in a machine using Windows, use TunSafe instead of the official WireGuard client because the latter doesn't work on specific VPN network configurations.	
Test sequence	Step 1	Send an HTTP PUT request to the */client/disable* endpoint with a body including the public key of a previously created and enabled client.
	Step 2	Check if an HTTP 204 code is returned.
	Step 3	Perform the test #1 (without the keys generation part) to check if the client is listed in the network interface information.
	Step 4	Perform the test #9 to check if the connection to the VPN server is possible using the credentials of the disabled client.
Test verdict	The test only passes if both the disabled client is not listed in the network interface information and the connection to the VPN server cannot be performed using the disabled client's credentials. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 31: VPN enabler's functional test 7 results

Enabler	VPN Enabler (test 7)	
Description	Functional test 7 for the VPN enabler that only involve the VPN Server component of the enabler.	
Approach	Fully manual because the connection to the VPN must be performed manually from a machine located in another network where the VPN enabler is deployed.	
Test tool/s	• WireGuard VPN client (WireGuard CLI, TunSafe, …) • Networking tools: ping and cURL	
Pre-test conditions	Enabler deployed, a VPN client created and its related information.	
Additional information	For testing the VPN client in a machine using Windows, use TunSafe instead of the official WireGuard client because the latter doesn't work on specific VPN network configurations.	
Test sequence	Step 1	Create a WireGuard configuration file (file with a *.conf* extension) using the VPN client's previously obtained configuration
	Step 2	Use the WireGuard VPN client program to connect to the VPN

Enabler	VPN Enabler (test 7)	
	Step 3	Perform a ping to the IP address of the VPN server network interface and, depending on the VPN network configuration, to other hosts and then try to access to services that are only accessible via the VPN using cURL.
Test verdict	The test only passes if the networking tests performed in step 3 are successful. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

4.1.1.2 Data management Plane

Semantic Repository enabler

Table 32: Semantic Repository enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Add namespace*	An empty namespace is created.	A namespace is created. This request should be rejected if the requested namespace already exists.	Pass / Fail
2	*Get namespaces*	Retrieve the list of existing namespaces.	An API call is performed, returning a JSON with all existing namespaces.	Pass / Fail
3	*Add model (default)*	A model is added with default options, to a namespace.	A model is added to an empty existing namespace, is assigned the default metadata, and the 'latest' version tag is pointed at it.	Pass / Fail
4	*Add model (with metadata)*	A model is added to a namespace.	A model is added to a namespace, under the declared version tag, and with attached metadata. Overwriting existing versioned model should be possible only, if the 'force overwrite' parameter is set.	Pass / Fail
5	*Get models in a namespace*	Retrieve all models with versions under a given namespace	An API call is performed, returning a JSON with the list of all models and their metadata, under the given namespace.	Pass / Fail
6	*Get model*	A model is retrieved	An API call is performed, returning a model file, provided, that a model with given namespace name, name and version exists. Using the 'latest' version tag should return the same model, as explicitly using the version tag pointed to by the 'latest' tag.	Pass / Fail
7	*Remove model*	A model is removed	A model is removed by namespace, name, and version. This should be possible only if the 'allow removal' parameter is set. The call should be rejected, if the model with given IDs does not exist, or if the version tag is 'latest' (version tags must be explicit when removing models).	Pass / Fail
8	*Remove namespace*	A namespace is removed	Remove a namespace by name. Removing an existing namespace should be possible only, if the 'allow removal' parameter is set, and the namespace does not contain any models. Otherwise, the request should be rejected.	Pass / Fail
9	*Upload documentation*	Model documentation is uploaded for an existing model	The uploaded documentation source files should be automatically transformed by the Semantic Repository enabler into human-	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
			readable HTML pages. The pages should be well-formatted and properly linked.	

Table 33: Semantic Repository enabler's functional tests results

Enabler	**Semantic Repository enabler (tests 1–9)**	
Description	Functional tests for the Semantic Repository enabler.	
Approach	Fully automatic (integrated in a pipeline)	
Test tool/s	• ScalaTest library • Akka HTTP TestKit • Java Virtual Machine • GitLab CI	
Pre-test conditions	• Deployment of all components of the enabler in a test environment.	
Additional information	The full functional test suite consists of hundreds of test cases and covers all functionalities of the enabler. Only a simplified selection of the tests is presented in this deliverable. The full list of test cases can be examined in the enabler's source code and CI logs.	
Test sequence	Step 1	The components of the enabler are set up by GitLab CI in a containerised environment.
	Step 2	In each test (managed by ScalaTest), Akka HTTP TestKit simulates an HTTP request to the enabler.
	Step 3	Semantic Repository enabler performs the requested action.
	Step 4	ScalaTest checks if the enabler behaved as expected and reports the result.
Test verdict	Pass	

Additional logs	Example logs from the pipeline (only the summary):
	
	`[info] Run completed in 26 seconds, 242 milliseconds.`
	`[info] Total number of tests run: 765`
	`[info] Suites: completed 14, aborted 0`
	`[info] Tests: succeeded 765, failed 0, canceled 0, ignored 0, pending 0`
	`[info] **All tests passed.**`
	`[success] Total time: 42 s, completed Mar 10, 2023, 3:26:58 PM`	

Semantic Translation enabler

Table 34: Semantic Translation enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Add alignment*	An alignment is loaded into internal persistent storage.	User uploads an alignment file. The request should be rejected, if the alignment file contents are not correct (wrong format, not enough metadata, no alignment cells), or if the alignment with given metadata already exists in the internal persistent storage.	Pass / Fail
2	*Get alignment*	An alignment is retrieved.	An API call is performed, returning an alignment file, provided that an alignment with given ID was previously uploaded.	Pass / Fail
3	*Delete alignment*	An alignment is removed from internal persistent storage.	Alignment is removed by ID, provided that it exists, and there are no active translation channels, that use the alignment.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
4	*Add translation channel*	A translation channel is created.	A translation channel with the given pair of alignments (input and output alignment) is created, and input and output topics are exposed. Clients should be able to write to the input topic and receive data at the output topic.	Pass / Fail
5	*Remove translation channel*	A translation channel is destroyed.	A translation channel stops accepting new messages and shuts down after a configured timeout to allow flushing of messages that are being process at the time, when the shutdown request comes. After or before the timeout, the channel should no longer exist.	Pass / Fail
6	*Translate batch data*	One-time translation is performed.	An API call is made to translate attached payload using a chain of alignments specified by ID, provided, that the alignments were uploaded previously. The returned payload should be equivalent to streaming translation through channels that use the same alignments.	Pass / Fail
7	*Send data through translation channel*	Data is translated in a stream.	Send a message to an input topic of a translation channel. The message should be processed (semantically translated) and pushed to the output topic of the translation channel.	Pass / Fail

Table 35: Semantic Translation enabler's functional tests results

Enabler	**Semantic Translation enabler (tests 1-7)**	
Description	Functional tests for the Semantic Translation enabler	
Approach	Fully automatic (integrated in a pipeline)	
Test tool/s	• ScalaTest library • Akka TestKit • Java Virtual Machine	
Pre-test conditions	Enabler deployed in a test environment.	
Additional information	The functional test suite for the Semantic Translation Enabler consists of nearly 300 test cases, covering alignment compilation, validation, and application. Additionally, message translation tests along a selected set of predefined alignments are also performed.	
Test sequence	Step 1	Each test is managed by the ScalaTest testing environment
	Step 2	Semantic Translation enabler performs the requested alignment-handling- or message-translation-related operation.
	Step 3	ScalaTest verifies the outcome of the operation
Test verdict	Pass	

Additional logs/ Report (in case of manual)	`[info] Run completed in 8 seconds, 581 milliseconds.`
	`[info] Total number of tests run: 276`
	`[info] Suites: completed 4, aborted 0`
	`[info] Tests: succeeded 276, failed 0, canceled 0, ignored 0, pending 0`
	`[info] All tests passed.`	

Semantic Annotation Enabler

Table 36: Semantic Annotation enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Convert YARRML to RML*	Annotation formats are converted.	Using the web GUI user converts YARRML into RML, provided that the YARRML is syntactically correct.	Pass / Fail
2	*Test RML*	Test data is annotated.	Using the web GUI user declares some data and annotation file contents in RML. The data is annotated using provided RML and displayed back to the user.	Pass / Fail
3	*One-time annotation*	Data is annotated using RML.	A one-time API call is made with payload, that contains both data to be annotated, and annotation rules in RML. Annotation result is returned to the user.	Pass / Fail
4	*Add streaming annotation file*	An annotation file is loaded into internal persistent storage.	User uploads an annotation file with given metadata and received auto-generated ID.	Pass / Fail
5	*Get streaming annotation file*	An annotation file is retrieved.	An API call is performed, returning an annotation file, provided that an annotation with given ID was previously uploaded.	Pass / Fail
6	*Delete streaming annotation file*	An annotation file is removed from internal persistent storage.	Annotation file is removed by ID, provided that it exists, and there are no active annotation channels, that use the annotation file.	Pass / Fail
7	*Add streaming annotation channel*	An annotation channel is created.	An annotation channel using the given annotation file is created, and input and output topics are exposed. Clients should be able to write to the input topic and receive data at the output topic.	Pass / Fail
8	*Remove annotation channel*	An annotation channel is destroyed.	An annotation channel stops accepting new messages and shuts down after a configured timeout to allow flushing of messages that are being process at the time, when the shutdown request comes. After or before the timeout, the channel should no longer exist.	Pass / Fail
9	*Send data through annotation channel*	Data is annotated in a stream.	Send a message to an input topic of an annotation channel. The message should be processed (semantically annotated) and pushed to the output topic of the annotation channel.	Pass / Fail

Table 37: Semantic Annotation enabler's functional tests 1-2 results

Enabler	**Sematic Annotation enabler (tests 1-2)**
Description	Web GUI operations – format conversion and test data annotation
Approach	Fully manual – web GUI tests were not automated, because of limited functionality and relatively large effort and number of additional test tools required to cover such small amount of features to be tested.
Test tool/s	Web browser – tested on Chromium and Firefox
Pre-test conditions	Enabler deployed with GUI

Enabler	Sematic Annotation enabler (tests 1-2)	
Additional information	The GUI comes pre-loaded with example YARRML, RML, and data to be annotated.	
Test sequence	Step 1	Load YARRML or use the pre-loaded files
	Step 2	Click on “to RML”
	Step 3	Click on “Annotate”
Test verdict	Pass	
Additional logs/ Report (in case of manual)	`RML conversion done` `Annotation done`	

Table 38: Semantic Annotation enabler's functional tests 3-9 results

Enabler	Sematic Annotation enabler (tests 3-9)	
Description	Web GUI operations – format conversion and test data annotation	
Approach	Functional tests of the streaming component and annotation backend.	
Test tool/s	Fully automatic	
Pre-test conditions	• ScalaTest library • Akka HTTP TestKit • Java Virtual Machine	
Additional information	Enabler deployed with an MQTT broker and Kafka broker (both can be auto-deployed with the enabler)	
Test sequence	Step 1	Tests are granular and automated with the ScalaTest library. Entire test suite can be run at once.
Test verdict	Pass	

Additional logs/ Report (in case of manual)	`[info] Run completed in 59 seconds, 101 milliseconds.`
	`[info] Total number of tests run: 50`
	`[info] Suites: completed 3, aborted 0`
	`[info] Tests: succeeded 50, failed 0, cancelled 0, ignored 0, pending 0`
	`[info] All tests passed.`
	`[success] Total time: 93 s, completed Nov 10, 2022, 1:00:12 PM`	

Edge Data Broker enabler

Table 39: Edge Data Broker enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Send and receive Raw Data*	Subscribe to a test topic (two clients, one publisher and one consumer). The publisher sends raw data to the topic and the consumer receives the data.	The consumer receives the data.	Pass / Fail
2	*Send data and filter them (not passing the filter)*	A publisher subscribes to a test topic and a consumer subscribes to the filtered test topic. The publisher sends raw data that does not pass the filter threshold to the test topic, and the consumer does not receive the data.	The consumer does not receive any data.	Pass / Fail
3	*Send data and filter them (passing the filter*	A publisher subscribes to a test topic and a consumer subscribes to the filtered test topic. The publisher sends raw data, that pass the threshold of the filter, to the test topic and the consumer receives the data.	The consumer receives the data.	Pass / Fail
4	*Create an alert with a*	A rule is created on the rule engine that specifies two test topics (topic1 and topic2). One publisher client subscribes to topic1 and one publisher client	The consumer receives the alert.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
	preconfigured rule	subscribes to topic2. One consumer client subscribes to the test alert topic topic3. The publisher clients send data that trigger the rules. The rule engine create an alert to the topic3. The consumer client receives the alert.		
5	*Receive Data to a second edbe cluster*	Create consumer(s) that subscribe at the same topics created for the first edbe cluster and check if they receive all the data published in the beforementioned cluster.	The consumer receives the data.	Pass / Fail

Table 40: Edge Data Broker enabler's functional tests results

Enabler	**Edge Data Broker Enabler (tests 1-5)**	
Description	Functional tests for the Edge Data Broker Enabler	
Approach	Fully manual. Each separate test requires changes regarding the client's subscribed topic and type and content of the messages' payload as well as manually examining the messages generated by fr_script.	
Test tool/s	• Paho-mqtt library • Postman	
Pre-test conditions	• One Edbe cluster with two vernemq instances and fr_script deployed on a two node kubernetes cluster (EDBE_A). • Another Edbe cluster with one vernemq instance deployed in a different kubernetes cluster connected via mqtt bridge with the beforementioned (EDBE_B).	
Additional information	The full functional test consists of more test cases that covers "#" and "+" mqtt topic wildcards as well as variation in fr_script's statements, conditions, payloads and logic.	
Test sequence	Step 1	Create a mqtt client connected to the EDBE_A and publish raw data to a test topic.
	Step 2	Create a mqtt client connected to the EDBE_A and subscribe to the test topic.
	Step 3	Check that the consumer receives the messages.
	Step 4	Repeat steps 1, 2, 3 for the EDBE_B cluster.
	Step 6	Post filters and rules of fr_script with Postman.
	Step 7	Create a mqtt client connected to EDBE_A and publish json formatted data to a test topic.
	Step 8	Create two mqtt clients one connected to the EDBE_A and subscribed to the test subtopic, and one connected to the EDBE_B and subscribed to the test subtopic.
	Step 9	Check that both consumers received the filtered messages.
	Step 10	Create a second mqtt client connected to the EDBE_A and publish json formatted data to a new test topic.
Test verdict	Pass	
Additional logs/ Report (in case of manual)	**Logs from EDBE_A :** % Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left Speed 100 48370 0 48370 0 0 324k 0 --:--:-- --:--:-- --:--:-- 325k /usr/sbin/start_vernemq: line 54: warning: command substitution: ignored null byte in input % Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left Speed 100 48370 0 48370 0 0 555k 0 --:--:-- --:--:-- --:--:-- 562k Will join an existing Kubernetes cluster with discovery node at edbe-1.edbe-headless.default.svc.cluster.local config is OK 08:51:38.752 [info] cluster node 'VerneMQ@edbe-1.edbe-headless.default.svc.cluster.local' UP 08:51:48.340 [info] successfully connected to cluster node 'VerneMQ@edbe-1.edbe-headless.default.svc.cluster.local'	

Enabler	Edge Data Broker Enabler (tests 1-5)
	Logs from EDBE_B : % Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left Speed 100 25147 0 25147 0 0 139k 0 --:--:-- --:--:-- --:--:-- 153k /usr/sbin/start_vernemq: line 54: warning: command substitution: ignored null byte in input % Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left Speed 100 25147 0 25147 0 0 153k 0 --:--:-- --:--:-- --:--:-- 162k config is OK 08:57:26.284 [info] Bridge br0 connected to 10.43.0.1:31883. 08:57:26.289 [info] Bridge Pid <0.414.0> is subscribing to Topics: [{[<<"#">>],0}] **Logs from EDBE_A fr_script :** edbe.default.svc.cluster.local 1883 Connected to MQTT Broker! INFO: Started server process [9] INFO: Waiting for application startup. INFO: Application startup complete. INFO: Uvicorn running on http://0.0.0.0:8000

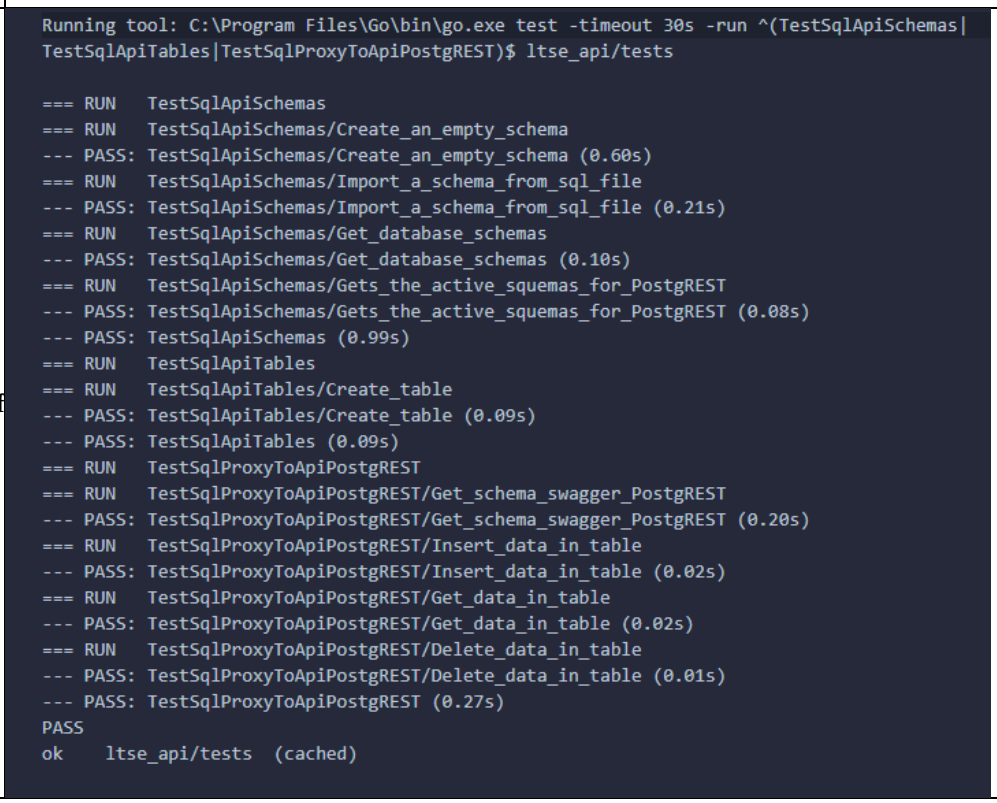
Long-Term Storage Enabler

Table 41: Long-Term Storage enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Manage SQL server*	The historical relational data managed by ASSIST-IoT deployments can be stored in SQL schemas, and tables	The success of the operation can be checked by exploring the existence of the SQL tables that have been managed through LTSE API.	Pass / Fail
2	*Ingest Raw SQL Data*	The relational data is ingested in the corresponding SQL table defined by the system.	The SQL raw data is collected into the corresponding LTSE SQL table through the LTSE API.	Pass / Fail
3	*Retrieves filtered SQL data*	Some filtered SQL data from the LTSE through the LTSE API should be provided.	The range of requested data is successfully obtained through the LTSE API.	Pass / Fail
4	*Manage noSQL cluster*	Create the noSQL indices for storing corresponding noSQL information from authorised enablers via LTSE API	The success of the operation can be checked by exploring the existence of the noSQL indices through LTSE API.	Pass / Fail
5	*Ingest Raw noSQL Data*	The non-relational data of the ASSIST-IoT system can be ingested in its corresponding noSQL index defined by the system.	The noSQL raw data is collected into the corresponding LTSE noSQL index through LTSE API.	Pass / Fail
6	*Retrieves filtered noSQL data*	Some filtered noSQL data from the LTSE through the LTSE API should be provided..	The range of requested noSQL data is successfully obtained through the LTSE API.	Pass / Fail

Table 42: Long-Term Storage enabler's functional tests 1-3 results

Enabler	LTSE (tests 1-3)
Description	Functional test of the LTSE SQL component

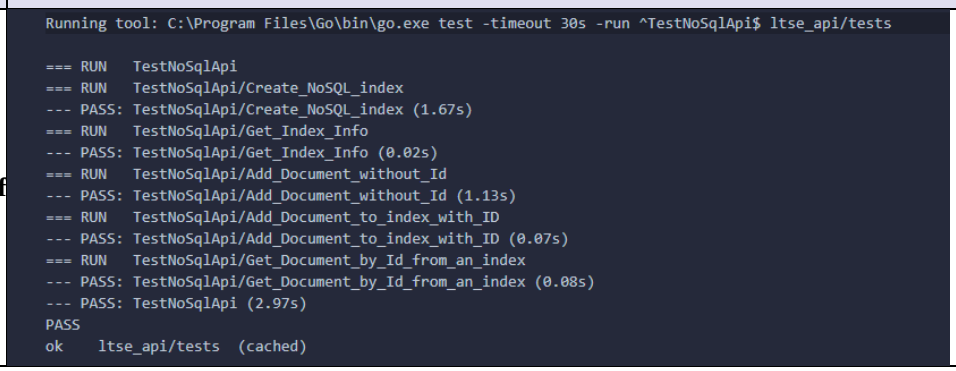
Enabler	LTSE (tests 1-3)	
Approach	Fully automatic, via a GO script that executes every SQL-based API endpoint of the LTSE	
Test tool/s	• Any CMD debugging console • GO test run library	
Pre-test conditions	• GO installed • LTSE helm chart deployed (and for running tests locally, to port-forward the LTSE services)	
Test sequence	Step 1	Generate different GET, POST and PUT requests for create SQL schemas, SQL tables, ingest SQL data, retrieve SQL data, delete SQL data.
	Step 2	Receive OK API responses depending on the request (mainly 200 for getting data and 201 for successful creation).
Test verdict	Pass	
Additional logs/ Report (in case of manual)		

```
Running tool: C:\Program Files\Go\bin\go.exe test -timeout 30s -run ^(TestSqlApiSchemas|
TestSqlApiTables|TestSqlProxyToApiPostgREST)$ ltse_api/tests

=== RUN   TestSqlApiSchemas
=== RUN   TestSqlApiSchemas/Create_an_empty_schema
--- PASS: TestSqlApiSchemas/Create_an_empty_schema (0.60s)
=== RUN   TestSqlApiSchemas/Import_a_schema_from_sql_file
--- PASS: TestSqlApiSchemas/Import_a_schema_from_sql_file (0.21s)
=== RUN   TestSqlApiSchemas/Get_database_schemas
--- PASS: TestSqlApiSchemas/Get_database_schemas (0.10s)
=== RUN   TestSqlApiSchemas/Gets_the_active_squemas_for_PostgREST
--- PASS: TestSqlApiSchemas/Gets_the_active_squemas_for_PostgREST (0.08s)
--- PASS: TestSqlApiSchemas (0.99s)
=== RUN   TestSqlApiTables
=== RUN   TestSqlApiTables/Create_table
--- PASS: TestSqlApiTables/Create_table (0.09s)
--- PASS: TestSqlApiTables (0.09s)
=== RUN   TestSqlProxyToApiPostgREST
=== RUN   TestSqlProxyToApiPostgREST/Get_schema_swagger_PostgREST
--- PASS: TestSqlProxyToApiPostgREST/Get_schema_swagger_PostgREST (0.20s)
=== RUN   TestSqlProxyToApiPostgREST/Insert_data_in_table
--- PASS: TestSqlProxyToApiPostgREST/Insert_data_in_table (0.02s)
=== RUN   TestSqlProxyToApiPostgREST/Get_data_in_table
--- PASS: TestSqlProxyToApiPostgREST/Get_data_in_table (0.02s)
=== RUN   TestSqlProxyToApiPostgREST/Delete_data_in_table
--- PASS: TestSqlProxyToApiPostgREST/Delete_data_in_table (0.01s)
--- PASS: TestSqlProxyToApiPostgREST (0.27s)
PASS
ok    ltse_api/tests  (cached)
```

Table 43: Long-Term Storage enabler's functional tests 4-6 results

Enabler	LTSE (tests 4-6)	
Description	Functional test of the LTSE noSQL component	
Approach	Fully automatic, via a GO script that executes every API endpoint of the LTSE	
Test tool/s	• Any CMD debugging console • GO test run library	
Pre-test conditions	• GO installed • LTSE helm chart deployed (and for running tests locally, to port-forward the LTSE services)	
Test sequence	Step 1	Generate different GET and PUT requests for create noSQL indices, as well as ingest, and retrieve noSQL data
	Step 2	Receive OK API responses depending on the request (mainly 200 for getting data and 201 for successful creation).
Test verdict	Pass	

Enabler	LTSE (tests 4-6)
Additional logs/ Report (in case of manual)	

```
Running tool: C:\Program Files\Go\bin\go.exe test -timeout 30s -run ^TestNoSqlApi$ ltse_api/tests

=== RUN   TestNoSqlApi
=== RUN   TestNoSqlApi/Create_NoSQL_index
--- PASS: TestNoSqlApi/Create_NoSQL_index (1.67s)
=== RUN   TestNoSqlApi/Get_Index_Info
--- PASS: TestNoSqlApi/Get_Index_Info (0.02s)
=== RUN   TestNoSqlApi/Add_Document_without_Id
--- PASS: TestNoSqlApi/Add_Document_without_Id (1.13s)
=== RUN   TestNoSqlApi/Add_Document_to_index_with_ID
--- PASS: TestNoSqlApi/Add_Document_to_index_with_ID (0.07s)
=== RUN   TestNoSqlApi/Get_Document_by_Id_from_an_index
--- PASS: TestNoSqlApi/Get_Document_by_Id_from_an_index (0.08s)
--- PASS: TestNoSqlApi (2.97s)
PASS
ok    ltse_api/tests  (cached)
```

4.1.1.3 Application and Services Plane

Tactile dashboard

Table 44: Tactile Dashboard enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Client test*	The client of the dashboard provides a visual tool for the user, based on the logic run in the server and the data stored in the database.	User requests are sent correctly to the dashboard database, and output at the client side is displayed correctly (i.e., when proper log-in process, the main dashboard page is shown – if invalid login, an unauthorised alert is prompted and does not allow to get into the webpage).	Pass / Fail
2	*Database test*	The dashboard data should be stored in its database.	All application requests and queries from the client to the database should be properly managed through the dashboard API	Pass / Fail
3	*Cookies test*	Cookies are used for speeding up some frequent actions, such as login sessions, but should be removed depending on user policies or cache size.	Testing cookies (sessions) are deleted either when cache is cleared, or when they reach their expiry	Pass / Fail

Table 45: Tactile Dashboard enabler's functional test 1 results

Enabler	Tactile dashboard (test 1)	
Description	Functional test of the client component from the tactile dashboard (login process)	
Approach	Fully automatic, via a set of different python unit-test scripts run over selenium framework	
Test tool/s	• Any CMD debugging console • Python unit-test library • Selenium framework	
Pre-test conditions	• Python v3.11 installed • node.js and npm installed. • Tactile dashboard helm chart deployed with default parameters and data included in the dashboard database	
Test sequence	Step 1	The script runs a webbrowser and goes to the IP address of the dashboard
	Step 2	The configured user/password data are automatically inserted into their corresponding

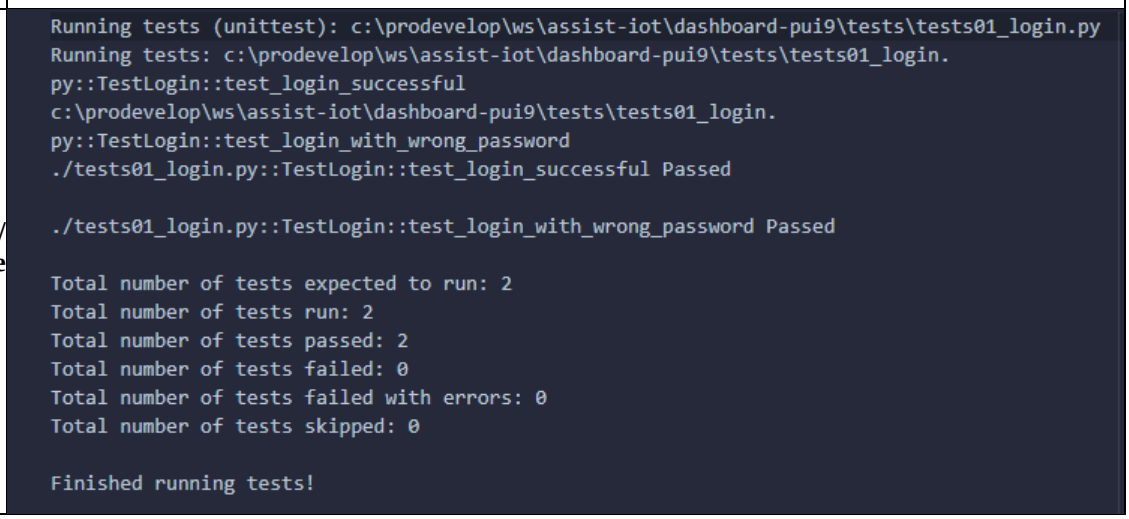
Enabler	Tactile dashboard (test 1)	
		fields on the login page of the dashboard
	Step 3	The Login button of the login pages is automatically pressed
	Step 4	If the user is already included and the password matches with the stored one in the dashboard database, the main page of the dashboard is shown
	Step 5	If the user is either not included yet, or if it is included, but the typed password is wrong, an unauthorized alert is prompted and the dashboard remains in the login page
Test verdict	Pass	
Additional logs/ Report (in case of manual)		

Table 46: Tactile Dashboard enabler's functional test 2 results

Enabler	Tactile dashboard (test 2)	
Description	Functional test of the API and database component from the tactile dashboard	
Approach	Fully automatic, via a set of different python unit-test scripts run over selenium framework	
Test tool/s	• Any CMD debugging console • Python unit-test library • Selenium framework	
Pre-test conditions	• Python v3.11 installed • node.js and npm installed. • Tactile dashboard helm chart deployed with default parameters and data included in the dashboard database	
Test sequence	Step 1	The script sends a GET request towards the Dashboard API
	Step 2	The Dashboard API consults to the dashboard database
		If the dashboard database, the dashboard API response with a 200 OK message
Test verdict	Pass	

Enabler	Tactile dashboard (test 2)
Additional logs/ Report (in case of manual)	Running tests (unittest): c:\prodevelop\ws\assist-iot\dashboard-pui9\tests\tests03_api.py Running tests: c:\prodevelop\ws\assist-iot\dashboard-pui9\tests\tests03_api. py::TestAPI::test1_connection_api ./tests03_api.py::TestAPI::test1_connection_api Passed Total number of tests expected to run: 1 Total number of tests run: 1 Total number of tests passed: 1 Total number of tests failed: 0 Total number of tests failed with errors: 0 Total number of tests skipped: 0 Finished running tests!

Table 47: Tactile Dashboard enabler's functional test 3 results

Enabler	Tactile dashboard (test 3)	
Description	Functional test of the cookies handling by the tactile dashboard	
Approach	Fully automatic, via a set of different python unit-test scripts run over selenium framework	
Test tool/s	• Any CMD debugging console • Python unit-test library • Selenium framework	
Pre-test conditions	• Python v3.11 installed • node.js and npm installed. • Tactile dashboard helm chart deployed with default parameters and data included in the dashboard database	
Test sequence	Step 1	The script runs a web browser and goes to the IP address of the dashboard
	Step 2	The configured user/password data are automatically inserted into their corresponding fields on the login page of the dashboard
	Step 3	The Login button of the login page is automatically pressed
	Step 4	If the user is already included and the password matches with the stored one in the dashboard database, the main page of the dashboard is shown
	Step 5	After a predefined timer, the cached cookies of the user credentials are automatically removed from the webbrowser
	Step 6	After Step 5 is executed, a navigation button of the webpage is automatically pressed by the script, and the dashboard pushes the user out to the login page again
Test verdict	Pass	
Additional logs/ Report (in case of manual)	Running tests (unittest): c:\prodevelop\ws\assist-iot\dashboard-pui9\tests\tests02_cookies.py Running tests: c:\prodevelop\ws\assist-iot\dashboard-pui9\tests\tests02_cookies.py::TestCookies::test_exist_cookies ./tests02_cookies.py::TestCookies::test_exist_cookies Passed Total number of tests expected to run: 1 Total number of tests run: 1 Total number of tests passed: 1 Total number of tests failed: 0 Total number of tests failed with errors: 0 Total number of tests skipped: 0 Finished running tests!	

Business KPI Reporting enabler

Table 48: Business KPI Reporting enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Create a graphical space*	The enabler should permit to configure different graphical spaces depending on the deployments needs	The BKPI enabler API should support the creation, retrieval, update, and deletion of graphical spaces	Pass / Fail
1	*Create graph chart*	The Business KPI enabler is a tool that provides functionalities for generating graphs and charts from data stored in the LTSE.	Through its REST API, the Business KPI enabler will create a sample graph with data stored in LTSE NoSQL component	Pass / Fail

Table 49: Business KPI Reporting enabler's functional test 1 results

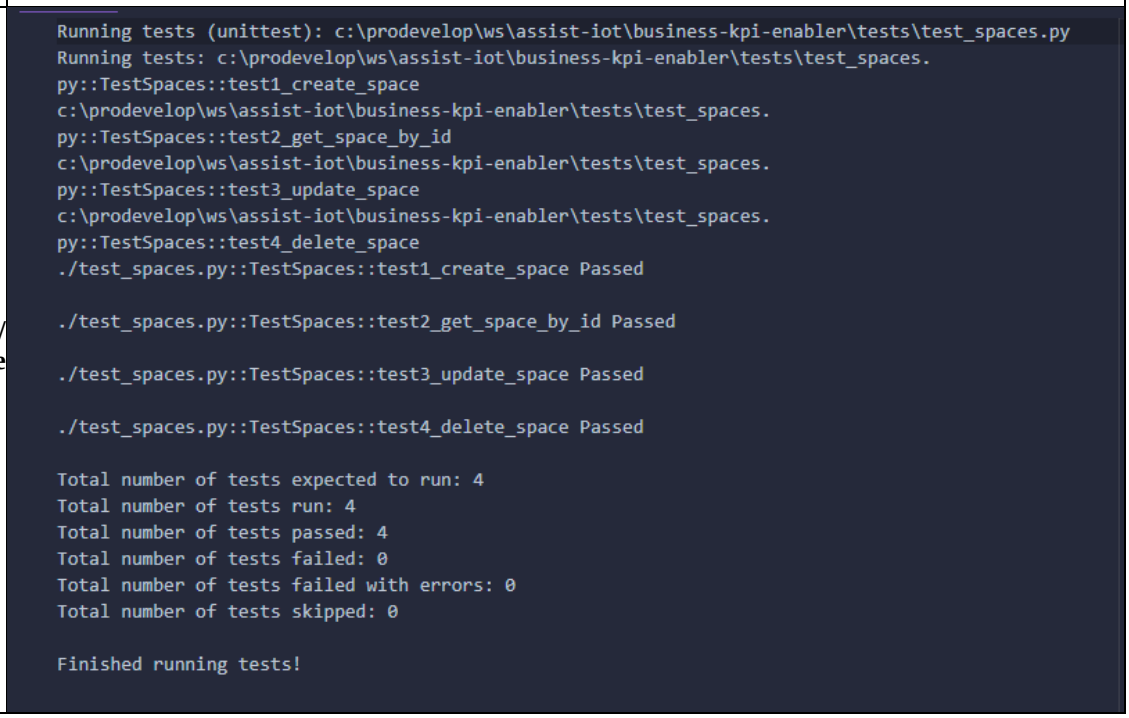
Enabler	**Business KPI reporting enabler (test 1)**	
Description	Functional test for the creation of graphical spaces in the BKPI enabler	
Approach	Fully automatic, with a Python script	
Test tool/s	• Any CMD debugging console • Python unit-test library	
Pre-test conditions	• Python v3.11 installed • LTSE helm chart deployed • BKPI helm chart deployed	
Test sequence	Step 1	Generate different GET, POST, PUT, DELETE requests to create, update, get, and delete graphical spaces.
	Step 2	Receive OK API responses depending on the request (200 for getting spaces data, 204/404 for successful deletion).
Test verdict	Pass	
Additional logs/ Report (in case of manual)		

Table 50: Business KPI Reporting enabler's functional test 2 results

Enabler	Business KPI reporting enabler (test 2)	
Description	Functional test for the creation of graphical charts in the BKPI enabler	
Approach	Fully automatic, with a Python script	
Test tool/s	• Any CMD debugging console • Python unit-test library	
Pre-test conditions	• Python v3.11 installed • LTSE helm chart deployed • BKPI helm chart deployed	
Test sequence	Step 1	Generate different GET, POST, PUT, DELETE requests to create, update, get, and delete graphical charts from LTSE NoSQL data.
	Step 2	Receive OK API responses depending on the request (mainly 200 for creation of graphical charts, and 204 for successful creation/deletion).
Test verdict	Pass	
Additional logs/ Report (in case of manual)		

Performance and usage diagnosis (PUD) enabler

Table 51: PUD enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Monitoring other enablers*	Enablers' metrics should be collected and stored in Prometheus time series database.	Other enablers that should be monitored, such as Edge data broker, should appear as a target with its state as "UP" on the Prometheus UI and its metrics should be collected, stored in	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
			Prometheus time series database and be accessible through its UI.	
2	*Monitoring kubernetes cluster*	Kube state metrics is a listening service that generates metrics about the state of Kubernetes objects. Those metrics should be collected and stored in Prometheus time series database.	Kube state metrics should appear as a target with its state as "UP" on the Prometheus UI and its metrics should be collected, stored in Prometheus time series database and be accessible through its UI.	Pass / Fail
3	*Monitoring the host system*	Node_exporter is a Prometheus exporter for hardware and OS metrics exposed by *NIX kernels. Those metrics should be collected and stored in Prometheus time series database	Node exporter metrics should appear as a target with its state as "UP" on the Prometheus UI and its metrics should be collected, stored in Prometheus time series database and be accessible through its UI.	Pass / Fail
4	*Elasticsearch as persistent storage for Prometheus metrics*	Elasticsearch should be able to receive and store the same metrics stored in Prometheus time series database.	Metrics that are stored in Prometheus time series database and appear in its UI should be permanently stored in elasticsearch cluster and appear in Kibanas UI as well.	Pass / Fail

Table 52: PUD enabler's functional tests results

Enabler	**PUD enabler (test 1-4)**	
Enabler	**Performance and Usage Diagnosis Enabler (tests 1-4)**	
Description	Functional tests for the Performance and Usage Diagnosis Enabler	
Approach	Fully manual	
Test tool/s	Web Browser	
Pre-test conditions	• A two node kubernetes cluster with edge data broker and kube state metrics installed. Node exporter installed as system service • A kubernetes cluster with elasticsearch and kibana installed.	
Test sequence	Step 1	**Test sequence**
Test verdict	Pass	
Additional logs/ Report (in case of manual)	While performing the tests, the exporters/targets in the scope of scraping could be accessed by their corresponding URL endpoints and their state in PUD's Prometheus target page was "UP" for all of them. By selecting and plotting metrics in metrics explorer, the gathered and stored metric values are available in Prometheus time series database. Lastly, the same metric values are stored and could be accessed from Elasticsearch and can be visualised in Kibana UI as well.	

OpenAPI Management Enabler

Table 53: OpenAPI Management enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Add Service*	Creates a new service that is pointing to an OpenAPI	An new service is created through the OpenAPI Manager given its URL where the service listens for requests	Pass / Fail
2	*Add Route*	Creates a new route in order for the service to be accessible through the OpenAPI Gateway	If the OpenAPI gateway receives a (http/https) request that matches the route's path it sends it back to the URL/path address	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
3	*Add plugin*	A plugin is added to an existed service that can provide authentication, security, monitoring etc.	The generated plugin is attached to the service in order to authenticate and secure the API	Pass / Fail
4	*Add Consumers*	Consumers develop the applications that use APIs	With an authenticated API, it is necessary to generate apikey before calling API. Routes with GET method will be assigned for READER consumers and routes with POST/DELETE/PATH method will be assigned for EDITOR consumers	Pass / Fail
5	*Interact through SwaggerUI*	Developer interacts with the uploaded documentation by using SwaggerUI through the OpenAPI Portal	Developer interacts with the OpenAPI Definition that has been uploaded to the portal by using SwaggerUI and can check if the all endpoints are functioning correctly behind the Open API Gateway.	Pass / Fail
6	*Inspect Functionality*	OpenAPI manager GUI displays basic information about the Gateway instance	An Admin user can obtain details about the performance of the API gateway by accessing Dashboard menu	Pass / Fail
7	*Backup*	Administrator user backup, restore and move OpenAPI configuration across different nodes	Through the OpenAPI manager GUI an Admin chose to backup, restore and save gateway's configuration	Pass / Fail

Table 54: OpenAPI Management enabler's functional tests 1-5 results

Enabler	**OpenAPI enabler (tests 1-5)**	
Description	The following tests will check if a developer of an Assist IoT enabler can expose the endpoints of the enabler by using the OpenAPI Portal.	
Approach	Semi-automatic	
Test tool/s	Any tool capable of executing HTTP requests.	
Pre-test conditions	• Keycloak IdM to enable authentication with Keycloak • Kubernetes environment	
Additional information	To register an OpenAPI definition it needs to be of a version OpenAPI 3.0.0 and later. It is recommended have Keycloak IdM manager deployed	
Test sequence	Step 1-a	Connect to OpenAPI Portal through a browser using credentials from Keycloak
	Step 1-b	Upload the OpenAPI definition file to automatically register the enabler to Kong Gateway
	Step 1-c	Alternative use ingress to regsister the service through Kubernetes as and ingress resource
	Step 2	Check if routes are accessible behind the Gateway by using the SwaggeUI through the OpenAPI Portal or any http tool
	Step 3-a	Add Kong Plugins through the OpenAPI definition.yaml file or through the ingress.yaml file to the endpoints
	Step 3-b	Add OIDC plugin for authentication to integrate with Keycloak IdM
	Step 3-c	Make a authenticated request with a token provided by Keycloak IdM manager to access your resource
	Step 4	Add specific consumers to a service if needed
	Step 5	Interact with the created service through the OpenAPI Portal
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation.	
Additional logs/ Report (in case of	In error cases , the logs are showed as HTTP responses	

Enabler	OpenAPI enabler (tests 1-5)
manual)	

Table 55: OpenAPI Management enabler's functional tests 6-7 results

Enabler	OpenAPI enabler (tests 6-7)	
Description	The following test scenario will use the Konga Manager to inspect functionality in the Kong Gateway.	
Approach	Semi-automatic	
Test tool/s	Konga GUI	
Pre-test conditions	• Kong Gateway deployed in a Kubernetes cluster	
Additional information	To register an OpenAPI definition it needs to be of a version OpenAPI 3.0.0 and later. It is recommended have Keycloak IdM manager deployed	
Test sequence	Step 6-a	Connect to Konga GUI through a browser
	Step 6-b	Inspect traffic and info from the logs provided by Konga GUI
	Step 7	Back up Kong configuration from the manager
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation.	
Additional logs/ Report (in case of manual)	Logs from the Konga GUI	

Video Augmentation enabler

Table 56: Video Augmentation enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Train model test*	The Video Augmentation enabler trains an ML model over the collected and annotated dataset	The Video Augmentation API must successfully connect to the backend trainer component, who should carry out the training properly.	Pass / Fail
2	*Inference model test*	The Video Augmentation enabler provides the ability to perform ML model inference over new not-trained local or streaming data (image/video).	The Video Augmentation API must successfully infer an already trained ML model over (i) new stored pictures/videos or (ii) from streaming real-time RTP messages. The validation is obtained via a 200-response message.	Pass / Fail

Table 57: Video Augmentation enabler's functional tests results

Enabler	Video Augmentation enabler (tests 1-2)
Description	Functional test for training an ML model in the Video Augmentation enabler
Approach	Fully automatic, with a Python script
Test tool/s	• Any CMD debugging console • Python unit-test library
Pre-test conditions	• Python v3.11 installed • Tensorflow 2 • NVIDIA GPU installed • Video Augmentation helm chart deployed • Annotated dataset stored in the corresponding data folder locally

Enabler	Video Augmentation enabler (tests 1-2)	
Test sequence	Step 1	Sends a POST request to train a pre-configured ML model.
	Step 2	Receive OK API response from the Video Augmentation enabler API, including a "Training model X" message.
	Step 3	Sends a POST request to perform an inference process over new data with the already trained ML model.
	Step 4	Receive OK API response from the Video Augmentation enabler API, including the predicted object over the new image.
Test verdict	Pending	
Additional logs/ Report (in case of manual)	Pending	

MR Enabler

The MR enabler will not be evaluated in automated way through the platform, as the under-development software (in *.appx file) will be deployed in specific hardware (Microsoft Hololens 2) and cannot be encapsulated (see deliverable 3.6 [16.], Chapter 5.2 Encapsulation exceptions). Nevertheless, the testing procedures will be followed in accordance with ASSIST-IoT methodology, which means that unit testing will be executed offline and integration tests will be performed with the rest of the required components, as follows:

Table 58: MR enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Receive alerts*	Receiving alert messages from real-time data streams and displaying them to the device.	MQTT messages will be send to MR enabler in order to visualise them.	Pass / Fail
2	*Send Data*	The MR enabler will send reports (data and image) to the LTSE.	Verify that the data is stored to the LTSE.	Pass / Fail
3	*Performance metrics*	Health metrics will be generated in the MR enabler and will be sent to the PUD enabler via APIs	Verify the PUD has received the health metrics through the provided API	Pass / Fail

Table 59: MR enabler's functional tests results

Enabler	MR enabler (tests 1-3)	
Description	Functional tests for MR Enabler	
Approach	Fully manual.	
Test tool/s	• Python mqtt library • LTSE instance • Web browser	
Pre-test conditions	• One laptop that can publish test alerts to a test topic, through a script • Run a LTSE instance on the laptop to hold the tables	
Additional information		
Test sequence	Step 1	Create a mqtt client connected to the EDBE and publish raw data to a test topic.
	Step 2	Connect the MRE client to the EDBE and subscribe to the test topic.
	Step 3	Check that the MRE receives the messages and visualises them to the user.
	Step 4	Fill a new report from the user interface of the MRE.
	Step 5	Send the report, with RESTAPI, to the LTSE database.
	Step 6	Check the LTSE database that the tables are filled correctly.

Enabler	MR enabler (tests 1-3)	
	Step 7	Make often API calls to the device to receive MRE important health metrics.
	Step 8	Adapt those metrics into one object and send them to PUD enabler via API call.
	Step 9	Check on PUD enabler that the health metrics of the MRE are being sent correctly and have valid values.
Test verdict	Pass	
Additional logs/ Report (in case of manual)		

4.1.2 Functional Testing of vertical enablers

4.1.2.1 Self-* enablers

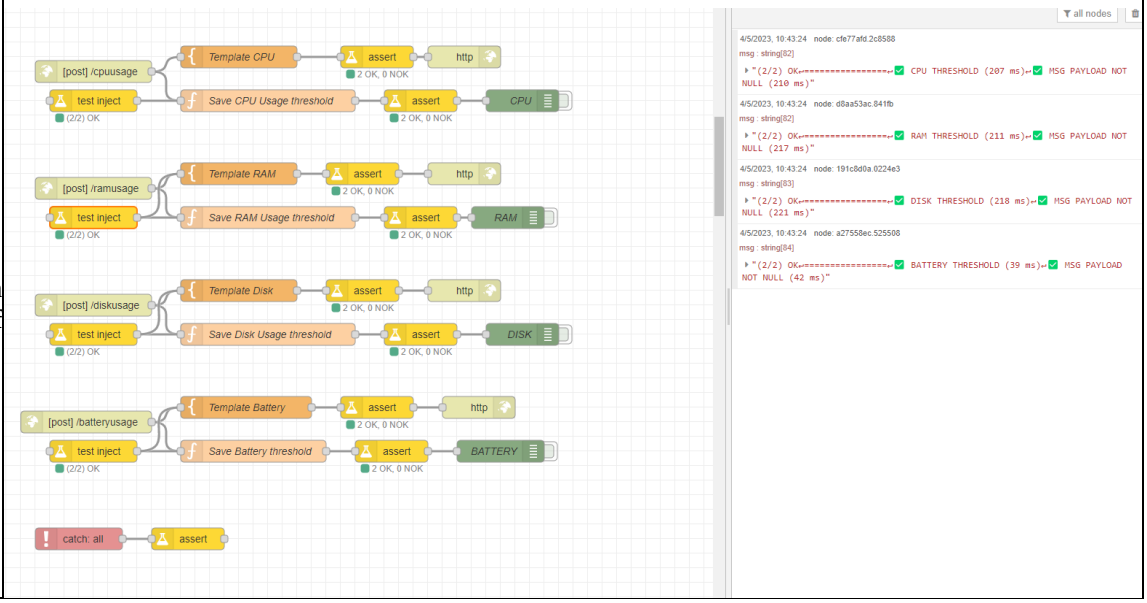
Self-healing enabler

Table 60: Self-healing enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Hardware testing (RAM/CPU)*	The self-healing enabler provides HW consumption metrics (RAM, CPU, disk percentage).	The self-healing UNIX commands implemented over its NodeRed flows must receive the percentage values of the device's resources.	Pass / Fail
2	*Hardware testing (network)*	The self-healing enabler provides current network connectivity status of the device.	The self-healing UNIX commands implemented over its NodeRed flows must receive either a numerical value associated with the latency for a successful ping request or a Boolean for a unsuccessful ping request.	Pass / Fail
3	*Hardware remediation (RAM/CPU/network)*	The self-healing enabler performs remediation actions over the HW resources (RAM/CPU/disk) when their consumption is beyond the user-defined threshold.	The self-healing UNIX commands implemented over its NodeRed flows receive an acknowledgment about the successful remediation action.	Pass / Fail
4	*Network remediation*	The self-healing enabler performs remediation actions over the network interface when there is no communication	The network interfaces are restarted, and a numerical value associated with the restarted status for a new ping request is received.	Pass / Fail

Table 61: Self-healing enabler's functional tests results

Enabler	Self-healing enabler (tests 1-4)	
Description	Functional tests for the Self-healing enabler	
Approach	Semi-automatic (the monitoring thresholds of the remediation triggers is customized by the user)	
Test tool/s	Node-red-contrib-flowtest library	
Pre-test conditions	• node.js, npm installed. • Node-red installed, and node-modules zip file executed. • In node-red (localhost:1880), import the self-healing flow JSON file	
Additional information	The scenario assumes that the tester has knowledge of the node-red interface provided by the self-healing enabler.	
Test sequence	Step 1	The test inject will inject a test message into the different flows

Enabler	Self-healing enabler (tests 1-4)	
	Step 2	The assert node will catch the injected test message and make assertions about it.
	Step 3	The reporting test is prompted in the node-red console
Test verdict	Pass	
Additional logs/ Report (in case of manual)		

Automated Configuration enabler

Table 62: Automated Configuration enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Add Requirements Model*	Test adding a new requirements model using the HTTP interface.	The test succeeds if the new requirements model is added and can be retrieved via the HTTP interface.	Pass / Fail
2	*Delete Requirements Model*	Test deleting an existing requirements model using the HTTP interface.	The test succeeds if the specified requirements model is deleted and can no longer be retrieved.	Pass / Fail
3	*Add Reaction Model*	Test adding a new reaction model using the HTTP interface.	The test succeeds if the new reaction model is added and can be retrieved via the HTTP interface.	Pass / Fail
4	*Delete Reaction Model*	Test deleting an existing reaction model using the HTTP interface.	The test succeeds if the specified reaction model is deleted and can no longer be retrieved.	Pass / Fail
5	*Register Resource*	Test registering a new resource using the Kafka interface.	The test succeeds if the new resource is registered and can be retrieved via the HTTP interface.	Pass / Fail
6	*Deregister Resource*	Test deregistering an existing resource using the Kafka interface.	The test succeeds if the specified resource is deregistered and can no longer be retrieved.	Pass / Fail
7	*CustomMessage Handling*	Test sending a custom message using the Kafka interface and triggering the appropriate reaction.	The test succeeds if the custom message is sent, the reaction is triggered, and the action is executed.	Pass / Fail
8	*ConditionalAction*	Test the ConditionalAction reaction to verify correct	The test succeeds if the appropriate action or fallback is executed based on the specified condition.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
		execution of action or fallback based on the condition.		
9	*FilterExpression*	Test the correct filtering of messages based on FilterExpression types.	The test succeeds if messages are filtered correctly, and reactions are triggered as specified.	Pass / Fail

Table 63: Automated Configuration enabler's functional tests results

Enabler	**Automated Configuration enabler (tests 1-9)**	
Description	Functional tests for the automated Configuration enabler	
Approach	Fully automatic (integrated in a pipeline)	
Test tool/s	• ScalaTest library • Akka HTTP TestKit • Java Virtual Machine • GitLab CI	
Pre-test conditions	• Deployment of all components of the enabler in a test environment.	
Additional information	The full functional test suite consists of multiple test cases and covers functionalities of the enabler. Only a simplified selection of the tests is presented in this deliverable. The full list of test cases can be examined in the enabler's source code and CI logs.	
Test sequence	Step 1	The components of the enabler are set up by GitLab CI in a containerised environment.
	Step 2	In each test (managed by ScalaTest), Akka HTTP TestKit simulates an HTTP request to the enabler.
	Step 3	Automated Configuration enabler performs the requested action.
	Step 4	ScalaTest checks if the enabler behaved as expected and reports the result.
Test verdict	Pass	
Additional logs/ Report (in case of manual)		

Table 64: Automated Configuration enabler's functional tests 1-4 results

Enabler	**Automated Configuration enabler (tests 1-4)**	
Description	This scenario tests the addition and deletion of requirements and reaction models.	
Approach	Manual	
Test tool/s	• HTTP client (e.g., Postman) • JSON formatter	
Pre-test conditions	An instance of the Automated Configuration Enabler should be running.	
Additional information	The scenario assumes that the tester has knowledge of the HTTP interface provided by the Automated Configuration Enabler.	
Test sequence	Step 1	Add a requirements model (Test 1) using the HTTP client.
	Step 2	Delete the requirements model (Test 2) using the HTTP client.
	Step 3	Add a reaction model (Test 3) using the HTTP client.
	Step 4	Delete the reaction model (Test 4) using the HTTP client.
Test verdict	Pass	
Additional logs/ Report (in case of	Manual testing is necessary to ensure that the HTTP interface works as expected and that the requirements and reaction models can be added and deleted correctly.	

Enabler	Automated Configuration enabler (tests 1-4)
manual)	

Table 65: Automated Configuration enabler's functional tests 5-9 results

Enabler	Automated Configuration enabler (tests 1, 3, 5-9)	
Description	This scenario tests the addition and deletion of resources and reaction models, custom message handling, conditional actions, filtering of messages, and executing various reaction actions.	
Approach	Manual	
Test tool/s	• HTTP client (e.g., Postman) • Kafka client • JSON formatter	
Pre-test conditions	An instance of the Automated Configuration Enabler should be running, and Kafka should be properly configured.	
Additional information	The scenario assumes that the tester has knowledge of the HTTP and Kafka interfaces provided by the Automated Configuration Enabler.	
Test sequence	Step 1	Add a requirements model (Test 1) using the HTTP client.
	Step 2	Add a reaction model (Test 3) using the HTTP client.
	Step 3	Register a new resource (Test 5) using the Kafka client.
	Step 4	Deregister the resource (Test 6) using the Kafka client.
	Step 5	Send a custom message (Test 7) using the Kafka client.
	Step 6	Test the ConditionalAction (Test 8) by sending appropriate messages using the Kafka client.
	Step 7	Test the FilterExpression (Test 9) by sending various messages and observing the reactions.
	Step 8	Test the execution of various ReactionAction types (Test 10) by sending appropriate messages.
	Step 9	Send another custom message (Test 7) using the Kafka client to verify the reaction.
Test verdict	Pass	
Additional logs/ Report (in case of manual)	Manual testing is necessary to verify that the system's behavior matches the expected behavior, as described in the documentation, under various conditions.	

Resource Provisioning enabler

Table 66: Resource Provisioning enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Obtain managed enablers*	Display enablers in cluster available for management and inference.	An appropriate output is obtained by displaying the enabler management configuration.	Pass / Fail
2	*Update managed enablers*	Update configuration on enablers or components to realise the inference	The correct string is displayed on the screen showing the success of the operation	Pass / Fail
3	*Get train values*	Display train values configuration to realise the inference based on the current criteria.	Formatted output with current criteria values and following the specific schema.	Pass / Fail
4	*Update train values*	Send same train values configuration so as not to no alter current criteria following the format received previously.	The correct string is displayed on the screen showing the success of operation	Pass / Fail
5	*Train*	Execute the train based on train values criteria for clusters and	Test output of the operation to see if any error has occurred or if the training has been	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
		components chosen previously.	completed.	
6	*Execute inference*	When the data is trained, infer module acts in each cluster/component predicting future resources requirements.	Test output of the operation to see if any error has occurred or if the inference has been completed.	Pass / Fail
7	*Get version*	Display enabler version	The correct version string is correctly display	Pass / Fail
8	*Get health*	Display health status of the environment	The current enabler environment is healthy or not.	Pass / Fail

Table 67: Resource Provisioning enabler's functional tests results

Enabler	**Resource Provisioning enabler (test 1-8)**	
Description	Formal description of functional tests. All tests follow the same steps based on the ability to self-manage without dependencies on other enablers.	
Approach	Fully automatic. OpenAPI Swagger file-based pipeline integration.	
Test tool/s	This enabler only requires any software capable of executing REST API calls such as POSTMAN, Gitlab pipelines or a script with curl or similar software.	
Pre-test conditions	Enabler deployed and prerequisites specified in documentation applied.	
Additional information	Self-managing enabler, suitable for automated testing and GitLab pipelines.	
Test sequence	Step 1	Send an HTTP Request (GET, POST, DEL) to each respective endpoint. In case of POST request, it is mandatory to include a custom body specified in the swagger file.
	Step 2a-1	Check if an HTTP 2xx or 3xx response code is returned.
	Step 2a-2	Check if response schema matches with the HTTP Request response.
	Step 2a-3	If last 2 steps are successful, it returns the success of the operation.
	Step 2b-1	Check if an HTTP 4xx or 5xx response code is returned.
	Step 2b-2	Response log is returned.
Test verdict	For each test, each answer is compared with the expected results and the final verdict will indicate the success or failure of the operation. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

NOTE: Several steps of each test may have different paths depending on the output obtained, it is good to keep this in mind.

Monitoring and Notifying enabler

Table 68: Monitoring and Notifying enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Receive data from IoT/Edge devices*	Monitor the status of devices by subscribing to topics created by the Edge Data Broker enabler and ensuring the data delivery.	The data arrives intact, and the user sees it on his consumer dashboard.	Pass / Fail
2	*Create notification*	Create a notification when a monitored device's threshold is breached.	The notification is successfully created.	Pass / Fail
3	*Push notification*	The notifications created in test #2 should be pushed to the responsible operator.	The notification (alongside with the related data) is successfully obtained by the correct operator, and can be seen on the dashboard.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
4	*Store notifications/critical events*	The notifications created in test #2 should be stored in the enabler's database for future consuming.	The notification (alongside with the related data) is successfully stored in the database in JSON format.	Pass / Fail
5	*Query notifications/critical events*	The notifications stored in the database, in test #4, should be able to be queried.	Successfully see the queried critical events from the database.	Pass / Fail

Table 69: Monitoring and Notifying enabler's functional tests results

Enabler	**Monitoring and Notifying enabler (test 1-5)**	
Description	The functional tests of the enabler, test an entire pipeline of connecting to the broker, creating a "normal" message, a "critical message", mapping them, sending the critical one to the enabler's internal storage, to DLT logging and auditing and finally querying it.	
Approach	Fully automatic (integrated in a pipeline)	
Test tool/s	Maven, Junit, GitLab pipelines	
Pre-test conditions	• Zookeeper up and running • Kafka broker up and running • MongoDB internal storage up and running	
Additional information	Maven's Junit has been used because the enabler is implemented in Java.	
Test sequence	Step 1	Check if a topic is created and subscribe to it
	Step 2	Send a message to the test topic
	Step 3	Check if the message is consumed
	Step 4	Receive a value above the predefined threshold
	Step 5	Check if the notification is produced
	Step 6	Check if the message with the notification is forwarded to the consumer
	Step 7	Check if the message is correctly mapped to a MongoDB document
	Step 8	Check if the Json with the notification is forwarded to the MongoDB collection
	Step 9	Check if the stored data can be queried
Test verdict	Passed. (If all the above steps are executed without any problem, then the test is considered as passed)	
Additional logs/ Report (in case of manual)	N/A	

Location Processing enabler

Table 70: Location Processing enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Create Query*	Test creating a new query using POST v1/queries	Test succeeds if the response status code is 201 and the created query is returned in the response body.	Pass / Fail
2	*Retrieve All Queries*	Test retrieving all queries using GET v1/queries	Test succeeds if the response status code is 200 and the list of queries is returned in the response body.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
3	*Retrieve Single Query*	Test retrieving a single query using GET v1/queries/{name}	Test succeeds if the response status code is 200 and the specified query is returned in the response body.	Pass / Fail
4	*Update Query*	Test updating a query using PUT v1/queries/{name}.	Test succeeds if the response status code is 200 and the updated query is returned in the response body.	Pass / Fail
5	*Delete Query*	Test deleting a query using DELETE v1/queries/{name}	Test succeeds if the response status code is 200 and the deleted queries count is returned in the response body.	Pass / Fail
6	*Trigger Query*	Test triggering a query using POST v1/queries/{name}/input	Test succeeds if the response status code is 200 and the output is returned in the response body.	Pass / Fail

Table 71: Location Processing enabler's functional tests 1-3 results

Enabler	**Location Processing enabler (tests 1-3)**	
Description	This test scenario will validate the successful creation of two queries, the retrieval of a single query, and the retrieval of all queries using the Location Processing enabler's HTTP interface.	
Approach	Manual	
Test tool/s	• Postman (or any other API testing tool)	
Pre-test conditions	The Location Processing enabler is up and running with the correct configuration. No query with the same name exists in the system.	
Additional information	Ensure access to the HTTP interface.	
Test sequence	Step 1	Create the first query by sending a POST request to v1/queries with the required query body.
	Step 2	Create the second query by sending a POST request to v1/queries with the required query body.
	Step 3	Semantic Repository enabler performs the requested action.
	Step 4	Verify the successful creation by checking the response with a status code 201.
	Step 5	Retrieve a single query by sending a GET request to v1/queries/{name}.
	Step 6	Verify the successful retrieval by checking the response with a status code 200 and the correct query information.
	Step 7	Retrieve all queries by sending a GET request to v1/queries.
	Step 8	Verify the successful retrieval by checking the response with a status code 200 and the correct list of queries.
Test verdict	Pass	
Additional logs/ Report (in case of manual)	Manual testing is necessary to validate the overall functionality and interaction between the HTTP interface and the Location Processing enabler, as well as the ability to retrieve and manage multiple queries.	

Table 72: Location Processing enabler's functional tests 4-6 results

Enabler	**Location Processing enabler (tests 4-6)**
Description	This test scenario will validate the successful creation, update, triggering, and deletion of a query using the Location Processing enabler's HTTP interface.
Approach	Manual
Test tool/s	• Postman (or any other API testing tool)
Pre-test conditions	The Location Processing enabler is up and running with the correct configuration. No query with the same name exists in the system.

Enabler	Location Processing enabler (tests 4-6)	
Additional information	Ensure access to the HTTP interface.	
Test sequence	Step 1	Create a query by sending a POST request to v1/queries with the required query body.
	Step 2	Verify the successful creation by checking the response with a status code 201.
	Step 3	Update the created query by sending a PUT request to v1/queries/{name} with the modified query body.
	Step 4	Verify the successful update by checking the response with a status code 200.
	Step 5	Trigger the updated query manually by sending a POST request to v1/queries/{name}/input with the required input data.
	Step 6	Verify the successful triggering by checking the response and output topic in the MQTT broker.
	Step 7	Delete the query by sending a DELETE request to v1/queries/{name}.
	Step 8	Verify the successful deletion by checking the response with a status code 200.
Test verdict	Pass	
Additional logs/ Report (in case of manual)	Manual testing is necessary to validate the overall functionality and interaction between the HTTP interface, the Location Processing enabler, and the MQTT brokers.	

4.1.2.2 Federated machine learning enablers

FL Training Collector enabler

Table 73: FL Training Collector enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Send training configuration*	FL Training Collector should be able to receive configuration for the training job to be run via API.	The API request is correctly handled and a message confirming a successful execution of a requested operation (accept configuration) is send in response.	Pass / Fail
2	*Request job status*	FL Training Collector should be able to provide status of a job which configuration it received.	The API request is correctly handled and in response all necessary information about job with a given it status are given	Pass / Fail

Table 74: FL Training Collector enabler's functional test 1 results

Enabler	FL Training Collector enabler (test 1)	
Description	Functional test 1 for the FL Training Collector enabler that tests the configurability of the training	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL). Additionally, logs of the FL Training Collector and FL Local Operations instances should be surveyed.	
Pre-test conditions	The FL Training Collector should, of course, be deployed. In order to properly test configurations involving custom strategies, FL Repository should be deployed. FL Repository should contain those objects. Additionally, a sufficient number of FL Local Operations should be deployed in order to test the full training.	
Additional information	N/A	
Test sequence	Step 1	Send a prepared configuration to FL Training Collector via a HTTP POST request to the `/job/config/<training_id>` endpoint.
	Step 2	Check the HTTP response. If the configuration is structurally valid, it should have the status of 200, with a 500 status otherwise.

Enabler	FL Training Collector enabler (test 1)	
	Step 3	Check FL Training Collector logs. If the enabler has properly connected to the FL Repository and FL Orchestrator, no error messages should appear.
	Step 4	Send the appropriate configuration to FL Local Operations instances. In the FL Training Collector and FL Local Operations appropriate logs should appear, detailing the beginning gRPC connection, the progression of the training process, as well as displaying the metrics. The information in the logs should be congruent with the configuration.
	Step 5	At the end of the training process check the collection of FL training results via the FL Repository by sending an HTTP GET request to the `/training-results` endpoint. It should contain the results of the conducted training. The final weights should be possible to download via an HTTP GET `/training-results/weights/<model_name>/<model_version>/<training_id>` request.
Test verdict	If the training process adheres to the detailed steps without the FL Training Collector generating any additional error messages, the test passes. Otherwise, it fails.	
Additional logs/ Report (in case of manual)	Logs from the FL Local Operations and FL Training Collector enablers	

Table 75: Training Collector enabler's functional test 2 results

Enabler	FL Training Collector enabler (test 2)	
Description	Functional test 2 for the FL Training Collector enabler that tests the ability to obtain the status of a given training process.	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL)	
Pre-test conditions	The FL Training Collector should be deployed along with the FL Repository and at least a single instance of the FL Local Operations.	
Additional information		
Test sequence	Step 1	Send an HTTP GET request to the `/job/status/{training_id}` endpoint on the FL Training Collector enabler with a random training_id.
	Step 2	Obtain a training response with the status INACTIVE.
	Step 3	Send sample training configurations to the FL Training Collector and FL Local Operations. Appropriate sample configurations should be found in the README files in those enablers. Wait for the training to start.
	Step 4	Send an HTTP GET request to the `/job/status/{training_id}` endpoint on the FL Training Collector enabler with the training id placed in the configuration. A response with the status TRAINING, along with the number of finished rounds, should be obtained.
	Step 5	Wait for the training to stop and send the same request once again. The status obtained should be FINISHED.
Test verdict	If all the results described in the steps were achieved properly, the test was passed.	
Additional logs/ Report (in case of manual)	N/A	

FL Orchestrator

Table 76: FL Orchestrator enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*FL training configuration setup test*	The FL Orchestrator needs to be in charge of defining the FL training configuration (including model to be trained, number of	The FL orchestrator must successfully retrieve from its own database the default values for FL training. In addition, the FL	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
		involved parties and training rounds, encryption mechanism, and evaluation criteria).	orchestrator API must successfully connect with the FL repository database.	
2	*FL training configuration delivery test*	The FL Orchestrator sends the defined FL training configuration to all the enablers involved in the training.	The FL orchestrator API must send in JSON documents about the FL configuration to the FL Training Collector, and FL Local Operations, which will acknowledge about its successful reception.	Pass / Fail
3	*FL training lifecycle monitoring test*	The FL Orchestrator needs to be aware of the current job status of the FL process.	The FL orchestrator API must periodically receive the status of the FL Training Collector and the FL Local Operations (either ON or OFF), as well as the number of finished epochs and training rounds.	Pass / Fail

Table 77: FL Orchestrator enabler's functional test 1 results

Enabler	**FL Orchestrator (test 1)**	
Description	Five different unit tests are conducted in order to assess the proper FL training configuration is set up via the enabler API calls.	
Approach	Semi-automatic relying on the enabler API endpoints responses.	
Test tool/s	• Any CMD debugging console • Unit test python library	
Pre-test conditions	Python installed. FL Orchestrator and FL Repository deployed	
Test sequence	Step 1	Generate a GET request for inserting the default values of all ML algorithms.
	Step 2	Generate a GET request for retrieving every ML algorithm configuration value from the FL repository.
	Step 3	Generate a POST request for collecting from a virtual GUI the default FL training configuration parameters (number of rounds, number of local operations, etc.).
	Step 4	Generate a POST request that modifies the default FL training configuration parameters.
	Step 5	Generate a POST request for visualizing the defined FL training configuration.
Test verdict	Pass	

Enabler	FL Orchestrator (test 1)
Additional logs/ Report (in case of manual)	Running tests (unittest): c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test01_FlOrchestrator Running tests: c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test01_FlOrchestrator::test_ShowConfiguration c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test01_FlOrchestrator::test_configurationsbyModel c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test01_FlOrchestrator::test_getConfigurations c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test01_FlOrchestrator::test_insertAditionalModelData c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test01_FlOrchestrator::test_recoverMergedModelData c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test01_FlOrchestrator::test_storeConfigurationModel ./test_florchestrator.py::Test01_FlOrchestrator::test_ShowConfiguration Passed ./test_florchestrator.py::Test01_FlOrchestrator::test_configurationsbyModel Passed ./test_florchestrator.py::Test01_FlOrchestrator::test_getConfigurations Passed ./test_florchestrator.py::Test01_FlOrchestrator::test_insertAditionalModelData Passed ./test_florchestrator.py::Test01_FlOrchestrator::test_recoverMergedModelData Passed ./test_florchestrator.py::Test01_FlOrchestrator::test_storeConfigurationModel Passed Total number of tests expected to run: 6 Total number of tests run: 6 Total number of tests passed: 6 Total number of tests failed: 0 Total number of tests failed with errors: 0 Total number of tests skipped: 0

Table 78: FL Orchestrator enabler's functional test 2 results

Enabler	FL Orchestrator (test 2)	
Description	Functional test conducted to assess the delivery of the FL training configuration to the FL Local Operations and FL Training Collector via the enabler API.	
Approach	Semi-automatic relying on the enabler API endpoints responses.	
Test tool/s	• Any CMD debugging console. • Unit test python library	
Pre-test conditions	Python installed. FL Orchestrator, FL Local Operations, and FL Training Collector deployed	
Test sequence	Step 1	Generate a POST request for delivering the FL training configuration to the FL Local Operations.
	Step 2	Generate a POST request for delivering the FL training configuration to the FL Training Collector
	Step 3	Receive an API response with the acknowledgement from FL Local Operations and FL Training Collector about the training request delivery.
Test verdict	Test ready, but whole environment not deployed yet	

Enabler	FL Orchestrator (test 2)
Additional logs/ Report (in case of manual)	Running tests (unittest): c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test02_FlOrchestrator Running tests: c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test02_FlOrchestrator::test_trainModel ./test_florchestrator.py::Test02_FlOrchestrator::test_trainModel Passed Total number of tests expected to run: 1 Total number of tests run: 1 Total number of tests passed: 1 Total number of tests failed: 0 Total number of tests failed with errors: 0 Total number of tests skipped: 0

Table 79: FL Orchestrator enabler's functional test 3 results

Enabler	FL Orchestrator (test 3)	
Description	Functional test conducted to assess the FL training lifecycle via the enabler API.	
Approach	Semi-automatic relying on the enabler API endpoints responses.	
Test tool/s	• Any CMD debugging console. • Unit test python library	
Pre-test conditions	Python installed. FL Orchestrator, FL Local Operations, and FL Training Collector deployed	
Test sequence	Step 1	Receive a GET request from the deployed FL Local Operations APIs informing about their ON status.
	Step 2	Receive a GET request from the deployed FL Training Collector API informing about a FL training round finished
Test verdict	Test ready, but whole environment not deployed yet	
Additional logs/ Report (in case of manual)	Running tests (unittest): c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test03_FlOrchestrator Running tests: c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test03_FlOrchestrator::test_FlTraininground c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test03_FlOrchestrator::test_RecoverStatusFromEnablers c:\prodevelop\ws\assist-iot\wp5\fl-orchestrator\tests\test_florchestrator.py::Test03_FlOrchestrator::test_RecoverTrainingEpochs ./test_florchestrator.py::Test03_FlOrchestrator::test_FlTraininground Passed ./test_florchestrator.py::Test03_FlOrchestrator::test_RecoverStatusFromEnablers Passed ./test_florchestrator.py::Test03_FlOrchestrator::test_RecoverTrainingEpochs Passed Total number of tests expected to run: 3 Total number of tests run: 3 Total number of tests passed: 3 Total number of tests failed: 0 Total number of tests failed with errors: 0 Total number of tests skipped: 0	

FL Repository enabler

Table 80: FL Repository enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Add new FL model metadata*	The enabler correctly adds the new metadata to the selected collection, in this case, models.	The model is listed by the enabler along with the other models. Its specific metadata can also be separately downloaded (test 1).	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
2	*Add new FL model files*	The enabler stores the model files and updates the metadata of the model by providing a new id under which the files are stored.	The model_id field in the metadata of the model is correctly updated. Additionally, the model files can be downloaded and reconstructed into the original model (test 2).	Pass / Fail
3	*List all available FL models*	The enabler returns all FL model metadata stored in the enabler.	The metadata of each of the listed FL models is the same as their separately obtained metadata (test 4).	Pass / Fail
4	*List only the FL models which were already trained.*	The enabler returns only the metadata of those FL models, which were already trained using the system and some of their training results have been saved.	The metadata of each of the listed FL models is the same as their separately obtained metadata. Additionally, each of the models listed has a form of training results stored in the repository (test 4) .	Pass / Fail
5	*Delete an FL model*	The enabler deletes the selected model from its repository, including metadata as well as any model files.	The metadata of the model does not show up in the listed files. Neither is it possible to separately download the metadata or the files (test 3).	Pass / Fail
6	*Get the metadata of selected FL model*	The enabler returns the full metadata of a selected model.	The metadata of a selected model obtained from the endpoint is the same as previously uploaded metadata (test 5).	Pass / Fail
7	*Get the files serializing a selected FL model*	The enabler returns the serialised data of a selected model.	The downloaded files have the same structure and contain the same information as previously uploaded files.	Pass / Fail
8	*Add new FL training results metadata*	The enabler correctly adds the new metadata to the selected collection, in this case, training-results.	The training results are listed by the enabler along with the other training results, both while using the functionality that lists all training results and all training results for a selected model. Their specific metadata can also be separately downloaded (test 1).	Pass / Fail
9	*Add new FL training results weights*	The enabler stores the final weights obtained as a result of the training.	The weights_id field in the metadata of the training results is correctly updated. Additionally, the resulting weights can be downloaded and applied to the training model (test 2).	Pass / Fail
10	*List all available FL training results*	The enabler returns all FL training results metadata stored in the enabler.	The metadata of each of the listed FL training results is the same as their separately obtained metadata (test 4).	Pass / Fail
11	*List only the FL training results which were obtained using a specific model.*	The enabler returns only the metadata of those FL training results, which were obtained in a training that used a specified model.	The metadata of each of the listed FL training results specify the selected model. Additionally, the model appears on the list of already trained models (test 4).	Pass / Fail
12	*Delete selected FL training results*	The enabler deletes the selected training results from its repository, including metadata as well as any training weights.	The training results do not show up in the listed results. Neither is it possible to separately download the metadata or the weights (test 3).	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
13	*Get the files serialising the model weights which were a part of the selected training results*	The enabler returns the serialised final weights obtained as a result of selected training.	The downloaded files have the same structure and contain the same information as previously uploaded files.	Pass / Fail
14	*Add new FL strategy metadata*	The enabler correctly adds the new metadata to the selected collection, in this case, strategies.	The strategies are listed by the enabler along with all the other aggregation strategies (test 1).	Pass / Fail
15	*Add new FL strategy object*	The enabler stores the pickled object which can be used by the Training Collector as an aggregation strategy.	The strategy_id field in the metadata of the model is correctly updated. Additionally, the resulting object can be downloaded and applied as an FL aggregation strategy (test 2).	Pass / Fail
16	*List all available FL strategies*	The enabler returns all FL strategy metadata stored in the enabler.	The metadata of each of the listed FL strategies is the same as their separately obtained metadata (test 4).	Pass / Fail
17	*Update the metadata of a selected FL strategy*	The enabler updates the description of the FL strategy located in the repository under a chosen name.	The listed FL strategy metadata is updated according to the request (test 2).	Pass / Fail
18	*Delete selected FL strategy*	The enabler deletes the selected aggregation strategy from its repository, including metadata as well as the pickled object.	The strategy does not show up in the listed strategies. Neither is it possible to separately download the strategy object (test 3).	Pass / Fail
19	*Get the pickled object implementing the functionality of an FL strategy*	The enabler returns the serialised strategy object.	The downloaded files have the same structure and contain the same information as previously uploaded files (test 6).	Pass / Fail
20	*Add new FL collector metadata*	The enabler correctly adds the new metadata to the selected collection, in this case, FL collectors.	The collector metadata is listed by the enabler along with all the other available FL collections (test 1).	Pass / Fail
21	*Add new FL collector object*	The enabler stores the pickled object which can be used by the FL Local Operations for dynamic data loading of a specific format	The collector_id field in the metadata of the model is correctly updated. Additionally, the resulting object can be downloaded and applied as a data loader (test 2).	Pass / Fail
22	*List all available FL collectors*	The enabler returns all FL collector metadata stored in the enabler.	The metadata of each of the listed FL collectors is the same as their separately obtained metadata (test 4).	Pass / Fail
23	*Update the metadata of a selected FL collector*	The enabler updates the description of the FL collector located in the repository under a chosen name.	The listed FL collector metadata is updated according to the request (test 2).	Pass / Fail
24	*Delete selected FL collector*	The enabler deletes the selected FL collector from its repository,	The collector does not show up in the listed collector. Neither is it possible to	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
		including metadata as well as the pickled object.	separately download the collector object (test 3).	
25	*Get the pickled object implementing the functionality of an FL collector*	The enabler returns the serialised FL collector object.	The downloaded files have the same structure and contain the same information as previously uploaded files (test 6).	Pass / Fail
26	*Add new FL transformation metadata*	The enabler correctly adds the new metadata to the selected collection, in this case, FL transformations.	The transformation metadata is listed by the enabler along with all the other available FL transformations (test 1).	Pass / Fail
27	*Add new FL transformation object*	The enabler stores the pickled object which can be used by the FL Local Operations for flexible data pre-processing	The transformation_id field in the metadata of the model is correctly updated. Additionally, the resulting object can be downloaded and applied as a data transformation (test 2).	Pass / Fail
28	*List all available FL data transformations*	The enabler returns all FL data transformation metadata stored in the enabler.	The metadata of each of the listed data transformations is the same as their separately obtained metadata (test 4).	Pass / Fail
29	*Update the metadata of a selected FL data transformation*	The enabler updates the description of the FL data transformation located in the repository under a chosen id.	The listed FL data transformation metadata is updated according to the request (test 2).	Pass / Fail
30	*Delete selected FL data transformation*	The enabler deletes the selected FL data transformation from its repository, including metadata as well as the pickled object.	The transformation does not show up in the listed data transformations. Neither is it possible to separately download the data transformation object (test 3).	Pass / Fail
31	*Get the pickled object implementing the functionality of an FL data transformation*	The enabler returns the serialised FL data transformation object.	The downloaded files have the same structure and contain the same information as previously uploaded files (test 6).	Pass / Fail

Table 81: FL Repository enabler's functional test 1 results

Enabler	**FL Repository enabler (test 1)**	
Description	Functional test 1 for the FL Repository enabler that tests the POST endpoints of the enabler	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL)	
Pre-test conditions	The FL Repository enabler needs to be deployed.	
Additional information		
Test sequence	Step 1	Send an HTTP POST request to a selected '/<collection-name>' endpoint.
	Step 2	If the response code is 400, access the HTTP GET "/<collection-name>" endpoint and see if an item with the same identifier (in the form of id or name and version) is

Enabler	FL Repository enabler (test 1)	
		already there.
	Step 3	If the response code is 201, access the HTTP GET "/<collection-name>" endpoint. The sent object should already be there.
Test verdict	If the results described in the tests are fulfilled, the test result is positive. Otherwise, it's negative.	
Additional logs/ Report (in case of manual)	N/A	

Table 82: FL Repository enabler's functional test 2 results

Enabler	FL Repository enabler (test 2)	
Description	Functional test 2 for the FL Repository enabler that tests the PUT endpoints of the enabler	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL)	
Pre-test conditions	The FL Repository enabler needs to be deployed.	
Additional information	N/A	
Test sequence	Step 1	In the case of endpoints allowing for metadata modification, send the new metadata to the `/<collection-name>/meta/<identifier>/<identifier 2, if exists>` endpoint. In the case of files or pickled objects, upload them to `/<collection-name>/<identifier>/<identifier 2, if exists>`.
	Step 2	If the response status is 204, the update has been successful. The HTTP GET "/<collection-name>" endpoint response should now showcase new, updated data (in the form of new storage id or new metadata).
	Step 3	If the response status is 404, the update has not been successful. The HTTP GET "/<collection-name>" endpoint response should have no items with the selected identifiers (they do not exist).
Test verdict	If the results described in the tests are fulfilled, the test result is positive. Otherwise, it's negative.	
Additional logs/ Report (in case of manual)	N/A	

Table 83: FL Repository enabler's functional test 3 results

Enabler	FL Repository enabler (test 3)	
Description	Functional test 3 for the FL Repository enabler that tests the DELETE endpoints of the enabler	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL)	
Pre-test conditions	The FL Repository enabler needs to be deployed.	
Additional information	N/A	
Test sequence	Step 1	First, lists the existing collection items using the `/<collection-name>` endpoint and check if you'll be trying to delete an item with identifiers already present in the collection.
	Step 2	If you're trying to delete an existing item, the response should have the 204 status. The item should no longer be present in the collection items listed via `/<collection-name>` endpoint.
	Step 3	If you're trying to delete a non-existing item, the response should have the 404 status. The item list accessed via the `/<collection-name>` endpoint should remain unchanged.

Enabler	FL Repository enabler (test 3)
Test verdict	If the results described in the tests are fulfilled, the test result is positive. Otherwise, it's negative.
Additional logs/ Report (in case of manual)	N/A

Table 84: FL Repository enabler's functional test 4 results

Enabler	FL Repository enabler (test 4)	
Description	Functional test 4 for the FL Repository enabler that tests the GET endpoints of the enabler that allow for listing the collection (partially and as a whole)	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL)	
Pre-test conditions	The FL Repository enabler needs to be deployed.	
Additional information	N/A	
Test sequence	Step 1	First, send an HTTP GET request to the `/<collection-name>` endpoint and check the response. In the case of training-results, a list of the training results obtained for a specific model and version is available by sending a GET request to the `/training-results/<model_name>/model_version` endpoint. In any case, the response should be a list of metadata (it may also be an empty list).
	Step 2	Send an HTTP POST request to add an element to the displayed collection. Send the previous HTTP GET request again to see if the list was updated.
	Step 3	Send an HTTP DELETE request to delete an element from the displayed collection. Similarly, repeat the first request and see if the response changed appropriately (the item in question is no longer displayed by the list).
Test verdict	If the results described in the tests are fulfilled, the test result is positive. Otherwise, it's negative.	
Additional logs/ Report (in case of manual)	N/A	

Table 85: FL Repository enabler's functional test 5 results

Enabler	FL Repository enabler (test 5)	
Description	Functional test 5 for the FL Repository enabler that tests the GET endpoints of the enabler that allow for downloading a specific set of metadata.	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL)	
Pre-test conditions	The FL Repository enabler needs to be deployed.	
Additional information	N/A	
Test sequence	Step 1	First, display the items in the collection by accessing the `/<collection-name>` endpoint via an HTTP GET request. Choose the identifiers of the metadata item to download.
	Step 2	Send an HTTP GET request to the `/<collection-name>/meta` endpoint. A valid metadata item should be obtained.
	Step 3	Then, send a similar HTTP GET request specifying identifiers that were not listed by the `/<collection-name>` endpoint. A response with 404 status should be obtained.
Test verdict	If the results described in the tests are fulfilled, the test result is positive. Otherwise, it is negative.	
Additional logs/ Report (in case of	N/A	

Enabler	FL Repository enabler (test 5)
manual)	

Table 86: FL Repository enabler's functional test 6 results

Enabler	FL Repository enabler (test 6)	
Description	Functional test 6 for the FL Repository enabler that tests the GET endpoints of the enabler that allow for downloading the specific objects stored in the enabler.	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL)	
Pre-test conditions	The FL Repository enabler needs to be deployed.	
Additional information	N/A	
Test sequence	Step 1	First, use the HTTP POST `/<collection-name>/<identifier1>/<identifier2>/…` to update the files of a specific, pre-existing item. The existing identifiers are visible in the collection by accessing the HTTP GET `/<collection-name>` endpoint.
	Step 2	Use an HTTP GET `/<collection-name>/<identifier1>/<identifier2>/…` to download and store an object.
	Step 3	Compare the downloaded object with the previously sent data.
Test verdict	If the results described in the tests are fulfilled, the test result is positive. Otherwise, it's negative.	
Additional logs/ Report (in case of manual)	N/A	

FL Local Operations enabler

Table 87: FL Local Operations enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Send configuration*	FL Local Operations should be able to receive configuration for the training job to be run via API.	The API request is correctly handled and a message confirming a successful execution of a requested operation (accept configuration) is send in response. The training process may appropriately begin (test 1).	Pass / Fail
2	*Send model metadata*	FL Local Operations should be able to accept the model with a given name, version and metadata.	The API request is correctly handled and a message confirming a successful execution of a requested operation (accept model) is send in response (test 2).	Pass / Fail
3	*Send model files*	FL Local Operations should correctly store the data necessary to reconstruct the model for later use.	The API request is correctly handled and a message confirming a successful execution of a requested operation (accept model) is send in response (test 3).	Pass / Fail
4	*Request status*	The FL Local Operations enabler should be able to provide its status.	The API request is correctly handled and in response status information is given (test 4).	Pass / Fail
5	*Request machine capabilities*	In order to determine which FL Local Operations instances will be able to perform the training of a selected model, the current capabilities of this instance must be obtained.	The API request is correctly handled and in response, the information detailing the current capabilities of the instance is given (test 5).	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
6	*Request current data format*	In order to determine which FL Local Operations instances will be able to perform the training of a selected model, or what data transformations would have to be applied for them to be, the current format of the data located on the instance must be obtained.	The API request is correctly handled and in response, the information detailing the current data format on the instance is given (test 6).	Pass / Fail
7	*Request predictions from the inference component*	The inference component of the FL Local Operations should be able to reconstruct an ML model and pre-process incoming data according to its local configuration. It should also be able to efficiently return these predictions.	The gRPC request is correctly handled with predictions returned in the proper format (test 7).	Pass / Fail
8	*Request the deployment of only the inference component of the charts (without the training or the database)*	It should be possible to deploy the inference component separately in order to provide inference without overloading the capabilities of edge environments.	The inference component is correctly deployed without any additional resources and is able to conduct inference (test 8).	Pass / Fail

Table 88: FL Local Operations enabler's functional test 1 results

Enabler	**FL Local Operations enabler a (test 1)**	
Description	Functional test 1 for the FL Local Operations enabler that tests the configurability of the training	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL). Additionally, logs of the FL Training Collector and FL Local Operations instances should be surveyed.	
Pre-test conditions	The FL Local Operations, as well as the FL Repository and the FL Training Collector should be deployed. Additionally, a training process that needs just one client should be started on the FL Training Collector using one of the sample configurations present in its README, started via an HTTP POST request sent to the its `/job/config/<training_id>` endpoint.	
Additional information	N/A	
Test sequence	Step 1	Send a training configuration to the FL Local Operations via an HTTP POST request to the `/job/config/<training_id>` endpoint. The configuration should be consistent with the previous FL Training Collector configuration, that is, correctly identify the FL TC IP address and aim to train the same model.
	Step 2	Obtain the response status. If the configuration was successful, the received response status should be 200.
	Step 3	The FL Local Operations logs is an indicator that the model is properly loaded into the instance.
	Step 4	The logs should later also display that client has been constructed properly, with the appropriate selected privacy mechanisms put in place (the mechanisms mentioned in logs should be the same as those in the configuration). A connection with the Flower server located on FL Training Collector should be stablished.
	Step 5	The training process should finish without any information about missed connections with the FL Training Collector or FL Repository.
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/	N/A	

Enabler	FL Local Operations enabler a (test 1)
Report (in case of manual)	

Table 89: FL Local Operations enabler's functional test 2 results

Enabler	FL Local Operations enabler a (test 2)	
Description	Functional test 2 for the FL Local Operations enabler that tests local model metadata storage	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL).	
Pre-test conditions	The FL Local Operations should be deployed.	
Additional information	N/A	
Test sequence	Step 1	Send an HTTP POST request to the `/model` endpoint on the FL Local Operations instance. This request should contain valid FL Model metadata.
	Step 2	Obtain the response status. If the configuration was successful, the received response status should be 200.
	Step 3	Try to upload model files via an HTTP PUT request to `/model/<model_name>/<model_version>` endpoint. The received response status should be 204.
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/ Report (in case of manual)	N/A	

Table 90: FL Local Operations enabler's functional test 3 results

Enabler	FL Local Operations enabler a (test 3)	
Description	Functional test 3 for the FL Local Operations enabler that tests local model file storage	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL).	
Pre-test conditions	The FL Local Operations should be deployed.	
Additional information		
Test sequence	Step 1	Send an HTTP POST request to the `/model` endpoint on the FL Local Operations instance. This request should contain valid FL Model metadata.
	Step 2	Try to upload model files via an HTTP PUT request to `/model/<model_name>/<model_version>` endpoint. The received response status should be 204.
	Step 3	Then, try to upload model files via an HTTP PUT request to `/model/<model_name>/<model_version>` specifying a non-existent model name and version. The received response status should be 404.
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/ Report (in case of manual)	N/A	

Table 91: FL Local Operations enabler's functional test 4 results

Enabler	FL Local Operations enabler a (test 4)
Description	Functional test 4 for the FL Local Operations enabler that tests job status retrieval.

Enabler	FL Local Operations enabler a (test 4)	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL).	
Pre-test conditions	The FL Local Operations should be deployed.	
Additional information	N/A	
Test sequence	Step 1	Send an HTTP GET to the `/job/status` endpoint on the FL Local Operations instance. This request should the number of training jobs the FL Local Operations is currently participating in, which for a newly created enabler should be 0.
	Step 2	Send a valid Local Operations training configuration via an HTTP POST request to the `/job/config/training_id` endpoint on the enabler.
	Step 3	Then send an HTTP GET request to the `/job/status` endpoint on the enabler again. The number of current jobs should be larger by 1.
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/ Report (in case of manual)	N/A	

Table 92: FL Local Operations enabler's functional test 5 results

Enabler	FL Local Operations enabler a (test 5)	
Description	Functional test 5 for the FL Local Operations enabler that tests machine capabilities retrieval.	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL), along with terminal commands like `htop`, `pip` and `glxinfo` or equivalent tools.	
Pre-test conditions	The FL Local Operations should be deployed.	
Additional information	N/A	
Test sequence	Step 1	Send an HTTP GET to the `/capabilities` endpoint on the FL Local Operations instance.
	Step 2	Check the response. It should contain information about the available RAM, storage, GPU capabilities, installed python packages and similar.
	Step 3	Use commands like `htop` (to check the RAM and storage), `pip` (to check installed dependencies) and `glxinfo` (to check GPU availability) to verify the obtained capabilities. They response should be consistent with the data from these commands.
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/ Report (in case of manual)	N/A	

Table 93: FL Local Operations enabler's functional test 6 results

Enabler	FL Local Operations enabler a (test 6)	
Description	Functional test 6 for the FL Local Operations enabler that tests data format retrieval.	
Approach	Semi-automatic	
Test tool/s	REST API client (one that is automatically set up for the enabler can be accessed via its API on the `/docs` URL).	
Pre-test conditions	The FL Local Operations should be deployed.	
Additional information	N/A	
Test sequence	Step 1	Send an HTTP GET to the `/format` endpoint on the FL Local Operations instance.

Enabler	FL Local Operations enabler a (test 6)	
	Step 2	The response should be a JSON file detailing the current training data format of the data located on the enabler.
	Step 3	Compare the JSON with the data format file, which should be located either in the enabler's folder marked in configuration as PREPROCESSED_FOLDER (for already pre-processed data) or its DATA_FOLDER (for its local data).
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/ Report (in case of manual)	N/A	

Table 94: FL Local Operations enabler's functional test 7 results

Enabler	FL Local Operations enabler a (test 7)	
Description	Functional test 7 for the FL Local Operations inference component to test its inference capabilities.	
Approach	Semi-automatic	
Test tool/s	gRPC testing client, for example tropicRPC available for Vscode.	
Pre-test conditions	The FL Local Operations should be deployed. An inference configuration compatible with the tests should be set up in the `inference_application/configurations` directory.	
Additional information	N/A	
Test sequence	Step 1	Send a gRPC data stream to the enabler adhering with the configuration established in `inference_application/code/proto/basic-inference.proto`. The requests in the data stream should have unique id fields. The format of the data should be compatible with the FL Local Operations data configuration.
	Step 2	Receive the predictions in the form of a data stream. The id of the response should be the same as the id of the request. The shape of the predictions should be compatible with the predictions of the model.
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/ Report (in case of manual)	N/A	

Table 95: FL Local Operations enabler's functional test 8 results

Enabler	FL Local Operations enabler (test 8)	
Description	Functional test 8 for the FL Local Operations inference component partial deployment.	
Approach	Fully manual	
Test tool/s	Existing Kubernetes and Helm installations	
Pre-test conditions	Internet connection to acquire the Docker image.	
Additional information	N/A	
Test sequence	Step 1	Run the command detailed in FL Local Operations README, ` helm install fllocaloperationslocal fllocaloperations --set inferenceapp.fullDeployment.enabled=false`.
	Step 2	Check nodes and services deployed in Kubernetes. Only the inference component should be deployed from the FL Local Operations Helm chart.
	Step 3	Uninstall the FL Local Operations chart using Helm commands.
	Step 4	Run ` kubectl apply -f fllocalops-config-map.yaml ` or `kubectl apply -f local-pv.yaml` if either the pvc-data-lo PersistentVolume or fllocalops-configmap ConfigMap were not deployed previously. Deploy the FL Local Operations chart using ` helm install fllocaloperationslocal fllocaloperations `.

Enabler	FL Local Operations enabler (test 8)	
	Step 5	Check if the training component, the database component and the inference component were all properly deployed.
Test verdict	If the tests proceed according to the described steps, the end result is a success.	
Additional logs/ Report (in case of manual)	N/A	

4.1.2.3 Cybersecurity enablers

Identity Manager enabler

Table 96: Identity Manager enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Ports exposed*	Identity Manager enabler needs to expose a set of external ports to check the service is up and running	A Dynamic Unit Test is deployed to verify that after the deployment of the enabler ports are responding accordingly to the definition on docker-compose. Ports 8080 and 2020 Verification can be done in CI/CD pipeline using https://github.com/gauntlt/gauntlt	Pass / Fail
2	*API REST exposed Keycloak*	Identity Manager enabler needs to expose REST API	http://<host>/auth/realms/	Pass / Fail
3	*API REST exposed*	Identity Manager enabler needs to expose REST API	http://:2020/health	Pass / Fail
4	*Key Cloak API response*	Enabler rest interfaces needs to process the response from the Keycloak API	Different static tests are deployed in order to process real and simulated connection attempts to the backend Described and documented in GitLab https://gitlab.assist-iot.eu/wp5/t53/identity-manager/-/blob/main/restenabler/test_keycloakapiconnector.py	Pass / Fail
5	*Rest Connector API response*	Enabler rest interfaces needs to process the response from the Rest Connector	Different static tests are deployed in order to process real and simulated connection attempts to the backend Described and documented in GitLab https://gitlab.assist-iot.eu/wp5/t53/identity-manager/-/blob/main/restenabler/test_restconnector.py	Pass / Fail

Table 97: Identity Manager enabler's functional tests results

Enabler	Identity Manager enabler (test 1-5)	
Description	Short description of the functional test that needs to be done to check that the enabler is properly deployed.	
Approach	Fully automatic (integrated in a pipeline)	
Test tool/s	This enabler only requires a tool that support REST API, such as POSTMAN, and a web browser.	
Pre-test conditions	The enabler must be deployed and configured with a test dataset.	
Additional information	N/A	
Test sequence	Step 1	Using the web browser go to the IP address and port that must be exposed and check that the webpage is shown.

Enabler	Identity Manager enabler (test 1-5)	
	Step 2	Send an HTTP Request (GET, POST) to each endpoint. In case of Post request, it must be included the body.
	Step 3	Check if the answer (HTTP Respond) received is according to the Request sent
Test verdict	For each test, each HTTP Respond received must be checked with the expected value and the final verdict will show the result of the test.	
Additional logs/ Report (in case of manual)	N/A	

Authorisation enabler

Table 98: Authorization enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Ports exposed*	Authorisation Server enabler needs to expose a set of external ports and API URL to check the service is up and running	A Dynamic Unit Test is deployed to verify that after the deployment of the enabler ports are responding accordingly to the definition on docker-compose. Port MySQL 3306 and 9000 Verification can be done in CI/CD pipeline using https://github.com/gauntlt/gauntlt	Pass / Fail

2	*API REST AuthServer*	Authorisation Server needs to expose REST API	Dynamic Unit Test to verify REST API
			• Test with role: http://<host>:9000/DcAuthzPap/rest/evaluate?resource=**domain**@**sourceOfIdentification**&action=**actionName**&code=**identificationCode**@**userRole**
			• Test Without role: http://<host>:9000/DcAuthzPap/rest/evaluate?resource=**domain**@**sourceOfIdentification**&action=**actionName**&code=**identificationCode**
			Parameter description:
			• domain: Required. Name of the domain in Authorisation Server.
			• sourceOfIdentification: Required. Name of the source of identification (resource) in Authorisation Server.
			• actionName: Required. Name of the action requested. Action names are defined in the rules of the policy in the Authorisation Server.
			• identificationCode: Required. Identification code of a user of the Authorisation Server.
			• userRole: Optional. If used in the rules, will be the user role used in the rules of the policy in the Authorisation Server.
			The response is a JSON with the following format:
			`Permit: {"retcode":"0","resource":"domain@sourceOfIdentification","action":"actionName","code":"identificationCode","response":"Permit","msg":""}`
			`Deny: {"retcode":"0","resource":"domain@sourceOfIden`
				Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
			`tification","action":"actionName","code":"iden tificationCode","response":"Deny","msg":""}`	
3	*API REST PAP*	Authorisation Server needs to expose REST API for PAP	http://<host>:9000/DcAuthzPap/	Pass / Fail

Table 99: Authorisation enabler's functional tests results

Enabler	**Authorisation enabler (test 1-3)**	
Description	Short description of the functional test that needs to be done to check that the enabler is properly deployed.	
Approach	Fully automatic (integrated in a pipeline)	
Test tool/s	This enabler only requires a tool that support REST API, such as POSTMAN, and a web browser.	
Pre-test conditions	The enabler must be deployed and configured with a test dataset.	
Additional information	N/A	
Test sequence	Step 1	Using the web browser go to the IP address and port that must be exposed and check that the webpage is shown.
	Step 2	Send an HTTP Request (GET, POST) to each endpoint. In case of Post request, it must be included the body.
	Step 3	Check if the answer (HTTP Respond) received is according to the Request sent
Test verdict	For each test, each HTTP Respond received must be checked with the expected value and the final verdict will show the result of the test.	
Additional logs/ Report (in case of manual)	N/A	

Cybersecurity Monitoring enabler

Table 100: Cybersecurity Monitoring enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Ports exposed*	Security monitoring enabler needs to expose a set of external ports to check the service is up and running	A Dynamic Unit Test is deployed to verify that after the deployment of the enabler ports are responding accordingly to the definition on docker-compose. Verification can be done in CI/CD pipeline using https://github.com/gauntlt/gauntlt	Pass / Fail

Table 101: Cybersecurity Monitoring enabler's functional tests results

Enabler	**Cybersecurity Monitoring enabler (test 1)**
Description	Short description of the functional test that needs to be done to check that the enabler is properly deployed.
Approach	Fully automatic (integrated in a pipeline)
Test tool/s	This enabler only requires a web browser.
Pre-test conditions	N/A
Additional	N/A

Enabler	Cybersecurity Monitoring enabler (test 1)	
information		
Test sequence	Step 1	Using the web browser go to the IP address and port that must be exponed and check that the webpage is shown.
Test verdict	The final verdict will be determined if the webpage of the enabler can be reached and shown	
Additional logs/ Report (in case of manual)	N/A	

Cybersecurity Monitoring Agent enabler

Table 102: Cybersecurity Monitoring Agent enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Ports exposed*	Security monitoring agent enabler needs to expose a set of external ports to check the service is up and running	A Dynamic Unit Test is deployed to verify that after the deployment of the enabler ports are responding accordingly to the definition on docker-compose. Verification can be done in CI/CD pipeline using https://github.com/gauntlt/gauntlt	Pass / Fail

Table 103: Cybersecurity Monitoring Agent enabler's functional tests results

Enabler	Cybersecurity Monitoring Agent enabler (test 1)	
Description	Short description of the functional test that needs to be done to check that the enabler is properly deployed.	
Approach	Fully automatic (integrated in a pipeline)	
Test tool/s	This enabler only requires a web browser to access to Cybersecurity monitoring enabler.	
Pre-test conditions	Cversecurity Monitoring enabler must be deployed and tested.	
Additional information	It is needed to access the Cybersecurity monitoring enabler to add the agent to the system.	
Test sequence	Step 1	Using the web browser, access the Cybersecurity monitoring enabler, and check if the agent is shown in the proper section
Test verdict	If the agent is shown in the proper section will determine the final verdict result	
Additional logs/ Report (in case of manual)	N/A	

4.1.2.4 DLT based enablers

Logging and Auditing

Table 104: Logging and Auditing enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Push logs*	The DLT has to have an operating API to receive messages.	Send data to verify that the API receives them and stores them in the ledger	Pass / Fail
2	*Store Logs*	Store logs with critical data to the DLT	Run a query to verify the data exists on the ledger	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
3	*Retrieve specific log*	Retrieve a log with critical data that is stored in the ledge	Provide a specific hash ID of the log to query the log	Pass / Fail

Table 105: Logging and Auditing enabler's functional tests results

Enabler	**Logging and Auditing enabler (test 1-3)**	
Description	The full functional test suite consists of test cases that covers all functionalities of the enabler.	
Approach	Semi-automatic. They are running all together but the main test function has to be called manually. The Postman part is manual.	
Test tool/s	• CCkit • gomega (library for testing in golang language) • ginkgo (library for testing in golang language) • Postman	
Pre-test conditions	Data ready in json format with one of the two sets of fields: • DeviceID, Value, Timestamp, Partition, Offset, Warning • Tag_ID, Lat, Lon	
Additional information	Pending to integrate them a pipeline.	
Test sequence	Step 1	Test every functionality of the enabler on chain (test the functionalities of the smart contract) depending on the input and the expected outcome.
	Step 2	Test the blockchain network through the api and if the process and the connection between them goes smoothly. In this step, Postman was used.
Test verdict	PASS	
Additional logs/ Report (in case of manual)	Ran 20 of 20 Specs in 0.003 seconds SUCCESS! -- 20 Passed | 0 Failed | 0 Pending | 0 Skipped PASS ok	

Integrity Verification

Table 106: Integrity Verification enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Push hashed data*	The DLT has to have an operating API to receive messages.	Send data to verify that the API receives them and stores them in the ledger	Pass / Fail
2	*Store hashed data*	The DLT stores the hashed data	Run a query to verify the data exists on the ledger	Pass / Fail
3	*Verification mechanism*	The DLT has to verify the integrity of the data.	Send hashed data (that already exists in the ledger) to verify that the verification mechanism works and matches the data with the already stored data	Pass / Fail

Table 107: Integrity Verification enabler's functional tests results

Enabler	**Integrity Verification enabler (test 1-3)**
Description	The full functional test suite consists of test cases that covers all functionalities of the enabler.
Approach	Semi-automatic. They are running all together but the main test function has to be called manually. The Postman part is manual.
Test tool/s	• CCkit

Enabler	Integrity Verification enabler (test 1-3)	
	• gomega (library for testing in golang language) • ginkgo (library for testing in golang language) • Postman	
Pre-test conditions	Data ready in json format with one of the two sets of fields: • Value	
Additional information	Pending to integrate them a pipeline.	
Test sequence	Step 1	Test every functionality of the enabler on chain (test the functionalities of the smart contract) depending on the input and the expected outcome.
	Step 2	Test the blockchain network through the api and if the process and the connection between them goes smoothly. In this step, Postman was used.
Test verdict	PASS	
Additional logs/ Report (in case of manual)	Ran 20 of 20 Specs in 0.003 seconds SUCCESS! -- 20 Passed | 0 Failed | 0 Pending | 0 Skipped PASS ok	

Broker Service

Table 108: Broker Service enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Push metadata*	The DLT has to have an operating API to receive messages.	Send data to verify that the API receives them and stores them in the ledger	Pass / Fail
2	*Store metadata*	Store metadata to the DLT	Run a query to verify the data exists on the ledger	Pass / Fail

Table 109: Broker Service enabler's functional tests results

Enabler	Broker Service enabler (test 1-2)	
Description	The full functional test suite consists of test cases that covers all functionalities of the enabler.	
Approach	Semi-automatic. They are running all together but the main test function has to be called manually. The Postman part is manual.	
Test tool/s	• CCkit • gomega (library for testing in golang language) • ginkgo (library for testing in golang language) • Postman	
Pre-test conditions	Data ready in json format with one of the two sets of fields: • ID, Endpoint, Type, Timestamp, Status	
Additional information	Pending to integrate them a pipeline.	
Test sequence	Step 1	Test every functionality of the enabler on chain (test the functionalities of the smart contract) depending on the input and the expected outcome.
	Step 2	Test the blockchain network through the api and if the process and the connection between them goes smoothly. In this step, Postman was used.
Test verdict	PASS	
Additional logs/ Report (in case of manual)	Ran 20 of 20 Specs in 0.003 seconds SUCCESS! -- 20 Passed | 0 Failed | 0 Pending | 0 Skipped PASS ok	

Federated Learning DLT

Table 110: FL DLT enabler's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Push model*	The DLT has to have an operating API to receive messages.	Send a model to verify that the API receives them and stores them in the ledger	Pass / Fail
2	*Store model*	Store a model to the DLT	Run a query to verify the model exists on the ledger	Pass / Fail

Table 111: FL DLT enabler's functional tests results

Enabler	**FL DLT enabler (test 1-2)**	
Description	The full functional test suite consists of test cases that covers all functionalities of the enabler.	
Approach	Semi-automatic. They are running all together but the main test function has to be called manually. The Postman part is manual.	
Test tool/s	• CCkit • gomega (library for testing in golang language) • ginkgo (library for testing in golang language) • Postman	
Pre-test conditions	Data ready in json format with one of the two sets of fields: • ModelName, ModelVersion, TrainingID, Round, Clients	
Additional information	Pending to integrate them in a pipeline	
Test sequence	Step 1	Test every functionality of the enabler on chain (test the functionalities of the smart contract) depending on the input and the expected outcome.
	Step 2	Test the blockchain network through the api and if the process and the connection between them goes smoothly. In this step, Postman was used.
Test verdict	PASS	
Additional logs/ Report (in case of manual)	Ran 25 of 25 Specs in 0.005 seconds SUCCESS! -- 25 Passed | 0 Failed | 0 Pending | 0 Skipped PASS ok	

4.1.2.5 Manageability enablers

Enablers' manager

Table 112: Enablers' manager functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Show enablers list*	The enabler provides the list of the deployed enablers.	The list of the deployed enablers is shown in a table.	Pass / Fail
2	*Deploy enabler*	Deploys a new enabler using the Smart Orchestrator under the hood.	The new enabler is shown in the table of the deployed enablers.	Pass / Fail
3	*Terminate an enabler*	Terminates a deployed enabler, interacting with the Smart Orchestrator under the hood.	The enabler is shown in the table of the deployed enablers and its operational status is "terminated". Now, the enabler can be deleted.	Pass / Fail

Nº	Test	Description	Evaluation criteria	Results
4	*Delete an enabler*	Deletes a terminated enabler using the Smart Orchestrator.	The enabler is not shown in the table of the deployed enablers.	Pass / Fail
5	*Show enabler logs*	Shows the logs of the enabler.	The list of logs of the selected enabler is shown.	Pass / Fail
6	*Show Helm chart repository list*	The enabler provides the list of the registered Helm chart repositories.	The list of the registered Helm chart repositories is shown in a table.	Pass / Fail
7	*Add a Helm chart repository*	Adds a new Helm chart repository using the Smart Orchestrator under the hood.	The new Helm chart repository is shown in the table of the registered Helm chart repositories.	Pass / Fail
8	*Delete a Helm chart repository*	Deletes a Helm chart repository using the Smart Orchestrator under the hood.	The deleted Helm chart repository is not shown in the table of the registered Helm chart repositories.	Pass / Fail

Table 113: Enablers' manager functional tests 1-5 results

Enabler	**Enablers manager (tests 1-5)**	
Description	Functional tests 1-5 for the Enablers manager, which are those related with the enablers management.	
Approach	Fully manual because the end user only interacts with the frontend component of the enabler, since it is a user-friendly dashboard deployed as a web page.	
Test tool/s	• Web browser	
Pre-test conditions	The enabler itself and the Smart Orchestrator must be previously deployed.	
Additional information		
Test sequence	Step 1	Navigate to the *Enabler list* page, which can be accessed through its entry located under the *Enablers management* section of the dashboard menu.
	Step 2	Wait until the table of the deployed enablers is loaded using the data obtained from the Smart Orchestrator.
	Step 3	Perform the needed action using the proper buttons of the page and wait for the final result. Some actions will need additional actions (e.g., fill in a form, apply to a confirmation dialog, …)
	Step 4	Check if the final displayed information is related with the performed action and the expected result.
Test verdict	The test passes if the user can perform the actions described in all the steps, even if the operational status of a deployed enabler is *failed* or an error message certainly related with the Smart Orchestrator enabler is shown. The test fails if the user cannot perform any action described in any step due to an unexpected error (not certainly related with the Smart Orchestrator), or if the dashboard backend loses its connection with the Smart Orchestrator. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 114: Enablers' manager functional tests 6-8 results

Enabler	**Enablers manager (tests 6-8)**
Description	Functional tests 6-8 for the Enablers manager, which are those related with the Helm chart repositories management.
Approach	Fully manual because the end user only interacts with the frontend component of the enabler, since it is a user-friendly dashboard deployed as a web page.
Test tool/s	• Web browser

Enabler	Enablers manager (tests 6-8)	
Pre-test conditions	The enabler itself and the Smart Orchestrator must be previously deployed.	
Additional information		
Test sequence	Step 1	Navigate to the *Helm chart repositories* page, which can be accessed through its entry located under the *Enablers management* section of the dashboard menu.
	Step 2	Wait until the table of registered Helm chart repositories is loaded using the data obtained from the Smart Orchestrator.
	Step 3	Perform the needed action using the proper buttons of the page and wait for the final result. Some actions will need additional actions (e.g., fill in a form, apply to a confirmation dialog, …)
	Step 4	Check if the final displayed information is related with the performed action and the expected result.
Test verdict	The test passes if the user can perform the actions described in all the steps, even if an error message certainly related with the Smart Orchestrator enabler is shown. The test fails if the user cannot perform any action described in any step due to an unexpected error (not certainly related with the Smart Orchestrator), or if the dashboard backend loses its connection with the Smart Orchestrator. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Composite Services manager

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

Table 115: Composite Services manager's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Show deployed pipelines*	The enabler provides the deployed pipelines.	The enabler shows the deployed pipelines in a user-friendly way.	Pass / Fail
2	*Deploy a new pipeline*	Deploys a new pipeline using the Smart Orchestrator under the hood.	The new pipeline is shown in a user-friendly way and the agents of the new pipeline are deployed in the proper Ks cluster.	Pass / Fail
3	*Delete an existing pipeline*	Deletes an existing pipeline using the Smart Orchestrator under the hood.	The deleted pipeline is not shown, and the agents of the deleted pipeline are deleted in the K8s cluster.	Pass / Fail
4	*Update an existing pipeline*	Updates an existing pipeline using the Smart Orchestrator under the hood.	The updated pipeline is shown in a user-friendly way and the agents of the updated pipeline are deployed or deleted (if needed) in the proper K8s cluster.	Pass / Fail

Table 116: Composite Services manager's functional tests results

Enabler	Composite services manager tests
Description	Functional tests for the Composite services manager.
Approach	Fully manual because the end user only interacts with the frontend component of the enabler, since it is a user-friendly dashboard deployed as a web page.
Test tool/s	• Web browser
Pre-test conditions	The enabler itself, the Smart Orchestrator and the LTSE must be previously deployed.
Additional information	The agents deployed to accomplish the pipelines are not shown in the table of deployed enablers since they are not considered enablers.

Enabler	Composite services manager tests	
Test sequence	Step 1	Navigate to the *Manageability flow* page through its entry in the menu of the dashboard.
	Step 2	Modify the existing pipelines: add, delete or update them.
	Step 3	Click on the *Deploy* button to deploy the displayed pipelines.
	Step 4	Check in the debug panel, which is located in the right part of the view, appears a message informing that an HTTP 200 code is returned
Test verdict	The test only passes if an HTTP 200 code is returned. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Clusters and Topology manager

Table 117: Clusters and Topology manager's functional tests

Nº	Test	Description	Evaluation criteria	Results
1	*Show clusters list*	The enabler provides the list of the registered K8s clusters.	The list of the deployed enablers is shown in a table.	Pass / Fail
2	*Register cluster*	Registers a new K8s cluster using the Smart Orchestrator.	The new cluster is shown in the table of the registered clusters and its status is "ENABLED".	Pass / Fail
3	*Delete cluster*	Deletes a K8s cluster using the Smart Orchestrator.	The K8s cluster is not shown in the table of the registered K8s clusters.	Pass / Fail
4	*Show clusters topology graph*	The enabler shows the topology (K8s nodes and its role: master or worker) of the registered K8s clusters.	A user-friendly graph is shown using the information about the topology of the registered K8s clusters.	Pass / Fail
5	*Show a list of the enablers deployed on a cluster*	The enabler shows a list of the enablers deployed on the selected K8s cluster.	The list of the deployed enablers in the selected K8s cluster is shown in a dialog.	Pass / Fail
6	*Deploy an enabler on a cluster*	Deploys a new enabler on the selected K8s node using the Smart Orchestrator under the hood.	The new enabler is shown in the table of the deployed enablers of the *Enabler list* page (Enabler manager).	Pass / Fail

Table 118: Clusters and Topology manager's functional tests 1-3 results

Enabler	Clusters and Topology manager (tests 1-3)
Description	Functional tests 1-3 for the Clusters and topology manager, which are those related with the K8s cluster management.
Approach	Fully manual because the end user only interacts with the frontend component of the enabler, since it is a user-friendly dashboard deployed as a web page.
Test tool/s	• Web browser
Pre-test conditions	The enabler itself and the Smart Orchestrator must be previously deployed.
Additional information	N/A

Enabler	Clusters and Topology manager (tests 1-3)	
Test sequence	Step 1	Navigate to the *K8s clusters* page, which can be accessed through its entry located under the *K8s clusters & devices* section of the dashboard menu.
	Step 2	Wait until the table of registered K8s clusters is loaded using the data obtained from the Smart Orchestrator.
	Step 3	Perform the needed action using the proper buttons of the page and wait for the final result. Some actions will need additional actions (e.g., fill in a form, apply to a confirmation dialog, …)
	Step 4	Check if the final displayed information is related with the performed action and the expected result.
Test verdict	The test passes if the user can perform the actions described in all the steps, even if the status of a registered cluster is *DEGRADED* or *FAILED*, or an error message certainly related with the Smart Orchestrator enabler is shown. The test fails if the user cannot perform any action described in any step due to an unexpected error (not certainly related with the Smart Orchestrator), or if the dashboard backend loses its connection with the Smart Orchestrator. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 119: Clusters and Topology manager's functional tests 4-5 results

Enabler	Clusters and Topology manager (tests 4-5)	
Description	Functional tests 1-3 for the Clusters and topology manager, which are those related with the K8s cluster topology management.	
Approach	Fully manual because the end user only interacts with the frontend component of the enabler, since it is a user-friendly dashboard deployed as a web page.	
Test tool/s	• Web browser	
Pre-test conditions	The enabler itself and the Smart Orchestrator must be previously deployed.	
Additional information	N/A	
Test sequence	Step 1	Navigate to the *K8s clusters topology* page, which can be accessed through its entry located under the *K8s clusters & devices* section of the dashboard menu.
	Step 2	Wait until the user-friendly graph is created using the information about the topology of the registered K8s clusters obtained from the Smart Orchestrator.
	Step 3	Click on a K8s cluster.
	Step 4	Check if the list of deployed enablers on the selected cluster is shown.
Test verdict	The test passes if the user can perform the actions described in all the steps, even if an error message certainly related with the Smart Orchestrator enabler is shown. The test fails if the user cannot perform any action described in any step due to an unexpected error (not certainly related with the Smart Orchestrator), or if the dashboard backend loses its connection with the Smart Orchestrator. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

Table 120: Clusters and Topology manager functional test 6 results

Enabler	Clusters and topology manager (test 6)
Description	Functional test 6 for the Clusters and topology manager, which is related with the K8s cluster topology management.
Approach	Fully manual because the end user only interacts with the frontend component of the enabler, since it is a user-friendly dashboard deployed as a web page.
Test tool/s	• Web browser
Pre-test conditions	The enabler itself and the Smart Orchestrator must be previously deployed.
Additional information	N/A

Enabler	Clusters and topology manager (test 6)	
Test sequence	Step 1	Navigate to the *K8s clusters topology* page, which can be accessed through its entry located under the *K8s clusters & devices* section of the dashboard menu.
	Step 2	Wait until the user-friendly graph is created using the information about the topology of the registered K8s clusters obtained from the Smart Orchestrator.
	Step 3	Click on a K8s node and fill in the form to deploy a new enabler on the selected node.
	Step 4	Navigate to the *Enabler list* page, which can be accessed through its entry located under the *Enablers management* section of the dashboard menu.
	Step 5	Check if the deployed enabler is displayed in the table of the deployed enablers and if in the *K8s cluster* column appears the same K8s cluster to which the previously selected K8s node belongs.
Test verdict	The test passes if the user can perform the actions described in all the steps, even if the operational status of a deployed enabler is *failed* or an error message certainly related with the Smart Orchestrator enabler is shown. The test fails if the user cannot perform any action described in any step due to an unexpected error (not certainly related with the Smart Orchestrator), or if the dashboard backend loses its connection with the Smart Orchestrator. -> Passed	
Additional logs/ Report (in case of manual)	N/A	

4.2 Integration testing

The integration testing methodology and approach for ASSIST-IoT is outlined in D6.2 [15.], where it is described as a sequential process following functional testing. However, given that integration testing involves checking the interaction between internal components of an enabler, it is often confounded with functional testing, as the communication between components is itself a functionality. Due to this fact, it was decided to approach integration testing in a different way, addressing specifically the integrations between enablers.

The aim is to test the connection of enablers without considering the specific internal processes, which have already been tested in unit and functional testing. Hence, this conforms the first phase of integrating components, to be later enhanced towards entire pilot trials (end-to-end testing phase). The table below reports the status of integrations that have been already implemented in the project. The partners that implemented the integrations are marked with blue color.

Another important note is that after the unified testing environment has been running, many of the integrations have progressed immediately. The current state of the project requires the quick integration of enablers, so the deployments in pilots can be eventually launched.

Table 121: Integration progress of ASSIST-IoT enablers

#	Enablers Involved	Integration Description	Partner Responsible & Involved	Pilots Involved	Current status
1	**EDBE – LTSE**	The integration involves the information (camera position, licence plate number etc.) of the pictures that arrive from the scanners, which have to be transferred through EDBE. The information is sliced into separate attributes and then stored to the LTSE.	CERTH - TT	P3b	Completed
2	**Monitoring & Notifying – ALL DLT enablers**	Integration of Kafka and DLT enablers. Firstly, Logging and Auditing in order to store critical event notifications from IoT devices. Secondly with Integrity Verification to store hashes of information in order to ensure that it remains	CERTH	P3b	Completed

#	Enablers Involved	Integration Description	Partner Responsible & Involved	Pilots Involved	Current status
		intact. Third with Broker Service to monitor the status of edge devices and gateways.			
3	**OpenAPI - IdM**	Configuring Kong API Gateway to use Open ID Connect (OIDC) plugin to integrate with Keycloak IdM in order to secure exposed endpoints. Also using Keycloak OIDC to connect to the openAPI Portal.	CERTH - S21SEC	ALL	Completed
4	**Semantic Annotation enabler – Semantic Repository enabler**	Downloading RML files from the Semantic Repository. The Semantic Annotation enabler needs these files to annotate incoming data streams. The integration will be used in Pilot 2 to integrated data from various sources.	SRIPAS	P2	Completed
5	**Semantic Annotation enabler – EDBE**	Integration of the MQTT protocol with the Semantic Annotation enabler. The Semantic Annotation enabler is able to produce and consume data using the MQTT protocol. The integration is needed for Pilot 2 to ensure interoperability with other enablers and components.	SRIPAS - ICCS	P2	Pending
6	**Semantic Translation enabler – EDBE**	In addition to the existing Apache Kafka support, Semantic Translation enabler will offer integration of the MQTT protocol	SRIPAS – ICCS	P2	Pending
7	**Location Processing enabler – EDBE**	Integration of the MQTT protocol with the Location Processing enabler. The Location Processing enabler is able to produce and consume data using the MQTT protocol. The integration is needed for Pilot 2 to ensure interoperability with other enablers and components.	SRIPAS – ICCS	P2	Pending
8	**Location Processing enabler – Location Tracking enabler**	Configuration in the Location Tracking enabler was prepared for ingesting location data from the location tags. A common schema for data exchanged between the enablers was established. The integration is needed for location-tracking related use cases in Pilot 2.	SRIPAS – NEWAYS	P2	Completed
9	**MR enabler – Location Processing enabler**	Obtaining worker locations from the Location Processing enabler through mqtt protocols, using the EDBE and displaying those new locations inside the MR enabler. The MR enabler, as soon as it receives the mqtt message, it compares the tags in the message with the workers' database and displays the new location of the workers that are closer to the MR enabler.	SRIPAS – ICCS	P2	Completed
10	**MR enabler – Semantic Repository enabler**	Downloading 3D models and other media files from the Semantic Repository using REST API and displaying them through the MR enabler.	ICCS – SRIPAS	P2	Completed

#	Enablers Involved	Integration Description	Partner Responsible & Involved	Pilots Involved	Current status
11	**Smart orchestrator - PUD**	To let the orchestrator decide the optimal place to automatically deploy enablers in the available clusters, different metrics (resource and latency-related) are needed. This required implementing a federated version of the PUD enabler to access such information from a central location, without needing to involve the LTSE nor custom synchronization mechanisms.	UPV - ICCS	P2	Completed
12	**Smart orchestrator - Manageability enablers**	The Smart orchestrator API is extensive and unfriendly to be directly utilized for administrating a given deployment (i.e., involved clusters, enablers and repositories). Manageability enablers provide interfaces and forms to abstract it, which required integrating its backend with the right orchestrator endpoints, including some aggregation of calls and filtering of the results.	UPV	ALL	Completed
13	**Smart orchestrator - EDB**	ETSI MANO architecture was not thought for massive deployments, in which several clusters (managed under the umbrella of "groups") will deploy the same set of enablers. To cope with this kind of cases, an extended MQTT-based architecture was implemented, in which the orchestrator incorporates a dedicated MQTT client and data model to manage the entire flow. MQTT bridges between the main EDBE and edge instances have been also provisioned. Buffering and timeout strategy under refinement.	UPV	ALL	Completed
14	**SD-WAN & WAN acceleration enablers**	These enablers are naturally working together to implement IPSec tunnels among connected network sites. Dedicated code was needed to enable the configuration of the WAN acceleration enabler instances from the SD-WAN controller, which acts as the orchestrator of the entire SD-WAN solution.	UPV	ALL	Completed
15	**VPN enabler - LTSE**	Information about clients provisioned was initially stored in an internal database of the VPN enabler. This information has been moved to the LTSE, and to that end the API of the VPN enabler has been modified to implement the required LTSE endpoints. In this way, less storage systems are needed in a given deployment,	UPV	ALL	Completed
16	**Manageability enablers - Tactile dashboard**	Manageability enablers are essential to manage a deployment, and therefore it was natural to integrate them with the main user framework of the project. To that end, the development guides of the tactile dashboard enabler were leveraged to generate such interfaces (including backends) and then integrating them in the same solution (VUI + Spring /PUI9 framework), avoiding the need of deploying different web applications independently and optimizing in this way the available resources.	UPV – PRO	ALL	Pending

#	Enablers Involved	Integration Description	Partner Responsible & Involved	Pilots Involved	Current status
17	**Traffic classification enabler - Semantic repository enabler**	The traffic classification enabler can generate and needs models to work. At this moment, involved models are stored and consumed in a storage volume of the host containing an instance of the enabler. The idea is to modify its API to interact with the Semantic repository enabler to manage the trained models (store and get them), being the latter a dedicated solution for that purpose.	UPV – SRIPAS	-	Completed
18	**FL Training Collector enabler**	The FL Training Collector has been fully integrated with the FL Repository and the FL Local Operations, allowing for download and storage of models and custom components in the case of FL Repository and for configurable FL training and evaluation in the case of FL Local Operations. Proper connections can be established with the FL Orchestrator, but in order to fully integrate the system some internal changes in FL Orchestrator are needed. These integrations will be then necessary for the proper functioning of Pilot 3b.	UPV – SRIPAS	P3b	Completed
19	**FL Repository enabler**	The FL Repository has been fully integrated with the FL Orchestrator, FL Training Collector and the FL Local Operations. The integration allows for flexible ML model, training results and custom modules download and storage. These integrations will be then necessary for the proper functioning of Pilot 3b.	SRIPAS – PRO	P2, P3b	Completed
20	**FL Local Operations enabler**	The FL Local Operations has been fully integrated with the FL Training Collector and the FL Repository, allowing for the configurable start and monitoring of the FL training process and the flexible ML model and custom component loading respectively. The integration with the FL Orchestrator still necessitates some internal updates in the code. These integrations will then be used in pilots 3b and 2.	SRIPAS – PRO	P3b, P2	Completed
21	**MR enabler - EDBE**	The MR enabler receives messages in real time, through topics that it has subscribed, every time that the EDBE publishes a message to the specific topic (mqtt protocols). The message could contain either information for a real time alert, or an update on the location of the construction's workers.	ICCS	P3b, P2	Completed
22	**MR enabler - Tactile dashboard**	Integration of the interface holding the updatable fields for the MR enabler inside the PUI9 framework. Those fields will be configurable through the PUI9 framework and will be received by the MR enabler through a REST API call, when the MR enabler is executed.	ICCS – PRO	P2	Pending
23	**MR enabler - PUD**	The MR enabler receives data, containing health metrics, from the Hololens device that is executed on, through constant REST API calls, and stores them. Then, the PUD	ICCS	P2	Completed

#	Enablers Involved	Integration Description	Partner Responsible & Involved	Pilots Involved	Current status
		enabler make constant REST API calls to receive the latest values and display them.			
24	**MR enabler - LTSE**	Retrieve workers' data from the LTSE database and display them on MR through REST API calls. Also prepare reports on the MR enabler's UI and send them to the LTSE database through REST API requests.	ICCS	P2	Completed
25	**SDN controller - ACN enabler**	Integration of SDN controller and ACN enablers is for collecting information from SDN controller and use controller to configure (rerouting paths) the network, to execute the results of optimisation from AI module into the SDN network. ACN modules are using ONOS applications: Path Generator and Maintainer in ACN communicates directly with ONOS and independently with two ONOS build-in applications, i.e., Intent Forwarding (IFWD) and Intent Monitoring and Rerouting (IMR).	OPL	P2	Completed
26	**Tactile dashboard - IdM**	Integration of the tactile dashboard and the IdM in order to authenticate users by means of the stored tokens in the IdM.	PRO – S21Sec	ALL	Completed
27	**Tactile dashboard – Authorisation enabler**	Integration of the tactile dashboard and the Authz in order to authorized specific roles to specific users in the dashboard.	PRO – S21Sec	ALL	Completed
28	**Multi-link – LTSE**	The Multi-link enabler makes use of VPN tunnels and dedicated information to perform. Similarly to the VPN-LTSE integration, the multi-link enabler data will be stored in the LTSE, thus requiring some modifications of its API to be implemented.	UPV	ALL	Pending
29	**LTSE – BKPI**	Visualization of historical data from Malta Freeport stored in the NoSQL part of the LTSE through the BKPI enabler	PRO	P1	Partially completed
30	**PUD – BKPI**	PUD's underlying technology uses Grafana as main representation framework. Project envisions BKPI as the main one, and therefore some dedicated dashboards and plugins (as data are to be stored in LTSE) are needed	ICCS	ALL	Partially completed
31	**FL Local operations & DLT-based FL**	A custom FL training strategy will store aggregated weights along with singular client weights in the FL-DLT enabler in order to later retrieve from the FL-DLT computed client reputation scores. More precisely, the FL Local Operations should send a series of requests, first sending the files containing the aggregated weights, then client weights, and then finally a JSON file with metadata like the round index and number of clients the FL-DLT should wait for. Then the FL Local Operations should, on sending an HTTP request	SRIPAS - CERTH	P3b	Pending

#	Enablers Involved	Integration Description	Partner Responsible & Involved	Pilots Involved	Current status
		with a previously specified format, receive a JSON response with the reputation score for a given client. This integration will not be used in any pilots.			
32	**Cybersecurity agents and server**	Cybersecurity Server has 2 parts, one the Incident Response must be installed in the cloud and the Incident detection can be implement in the cloud or the edge (Depends on each Pilot). Finally, the agent needs a Linux or Windows compatible OS.	S21Sec	ALL	Completed
33	**BKPI – Tactile Dashboard**	BKPI enabler is essential to visualized time-series KPI pilot data, and therefore it was natural to integrate them with the main user framework of the project. To that end, the development guides of the tactile dashboard enabler were leveraged to generate such interfaces, integrating them in the same solution, avoiding the need of deploying different web applications independently.	PRO	P1	Completed

4.3 End-to-end testing

As already mentioned in the previous deliverable, end-to-end testing is the last phase of software functional testing. After unit, functional and integration testing of enablers, the environment now becomes the entire application and the interconnection between enablers. The best testbed for conducting end-to-end testing is no other than the actual pilot trials and sub-trials. The recommended way to approach end-to-end testing, since there are enablers involved from different organizations, is to create an integration team composed of all the technical partners in such trial, which will be responsible for developing, executing and reporting the tests. It is important to note that the integration team is not officially recognized as a standalone unit within the project. However, it is important for all organizations involved in the project to collaborate closely on testing and integration, to ensure that all components work seamlessly together.

Table 122: End-to-end testing report final table

Pilot #						
Trial #						
Enabler's connection under test	**Description**	**Input**	**Outputs**	**Already tested / Which phase**	**Test result**	**Automated or manual / Comments in case of manual**
Which enablers are under test	*Brief description of the connection being tested*	*Which is the input data of the interacting enablers*	*Which is the output which will be input to the next connection*	*Yes / no* *If yes in which phase?*	*Pass / Fail*	*Auto / Manual* *Any comment*

The amendment of the project (entry into force in M30) allows to present a general approach for test development in each pilot, and in the last deliverable of this deliverable series, the results of the tests will be reported with precision and thoroughness. The following section presents the general idea of what will tested in be end-to-end phase and provide an overview of the expected outcomes. An important note is that in cases that

there are already implemented automated tests between enablers in the previous phases, these tests are modified and transferred into each trial.

4.3.1 Pilot 1: Port Automation

4.3.1.1 Trial #1: Tracking assets in terminal yard

This pilot trial involves the tracking of assets in the terminal yard. The architectural diagram delivered in D7.2 [17.] includes a straightforward connection of the enabler.

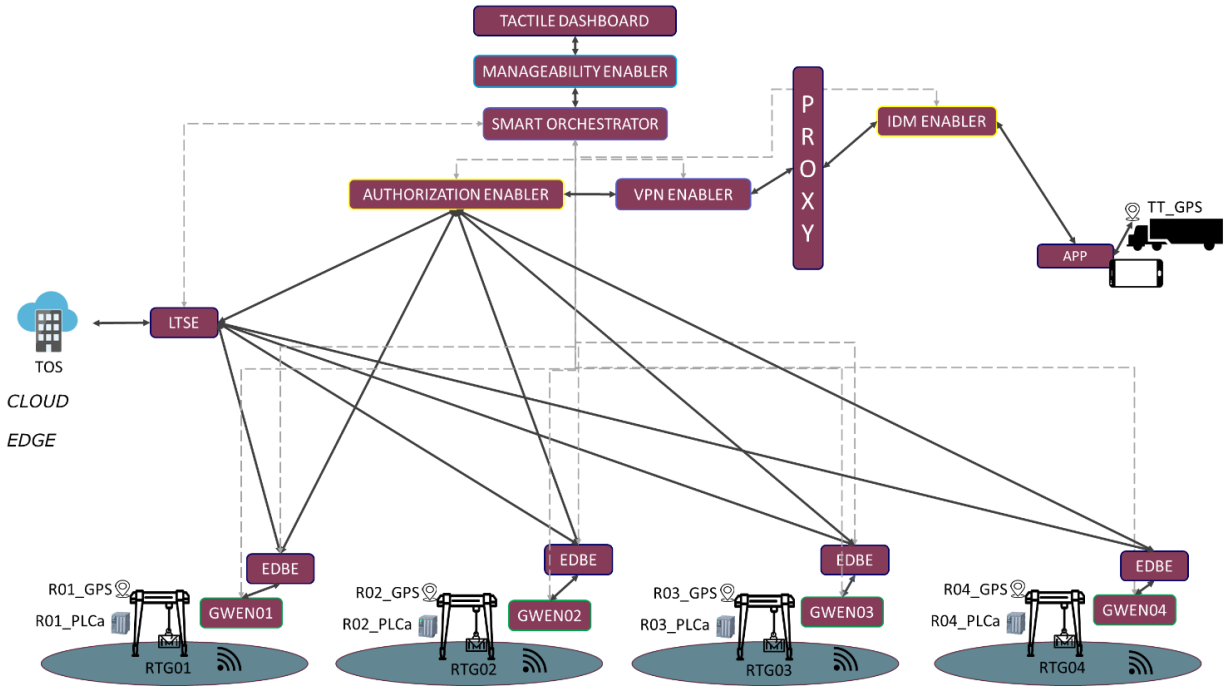


Figure 25. Architectural block diagram of Pilot 1 – Trial #1

In principle, to have a complete tested pilot trial, every line connecting the enablers / components has to be tested. In detail, the end-to-end testing of this trial includes the following steps:

- Test the connection between GWEN with the EDBE deployments
- Test the connection between EDBE and LTSE
- Test the connection of Authorisation Enabler with EDBE, LTSE, VPN
- Test the connection of tactile dashboard with the manageability enablers and with smart orchestrator

In fact, all of the above have been already tested multiple times, and the tests are automated and integrated in the pipeline. The point of this phase is to monitor that all of the application's functionalities are present, stable and reliable, in order to be delivered to the end user.

4.3.1.2 Trial #2: Automated CHE cooperation

Similar to the first pilot trial, the testing activities involve:

- The connection of GWEN with EDBE
- The connection of Authorisation enabler and IdM enabler with the application's dashboard

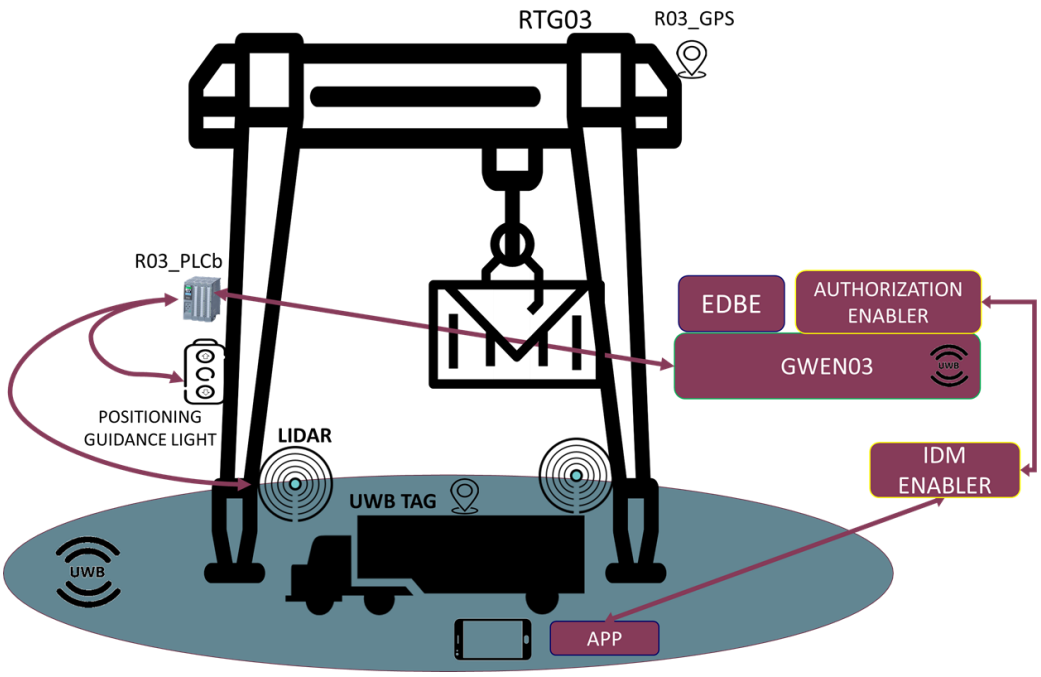


Figure 26. Architectural block diagram of Pilot 1 – Trial #2

As mentioned before, some of the tests have been already conducted in the previous phases of testing, making the monitoring of the application's stability and reliability the main purpose of this phase.

4.3.1.3 Trial #3: RTG remote control with AR support

The third trial of pilot one involves the following testing activities:

- GWEN, EDBE, Multilink enabler and video augmentation enabler working together
- The connection of EDBE with LTSE
- The connection Video augmentation enabler with FL repository
- The integration of IdM enabler and Authorisation enabler with the platform
- The connection of VPN enabler, Tactile dashboard, Manageability and Smart orchestrator, which will actually work as a unity

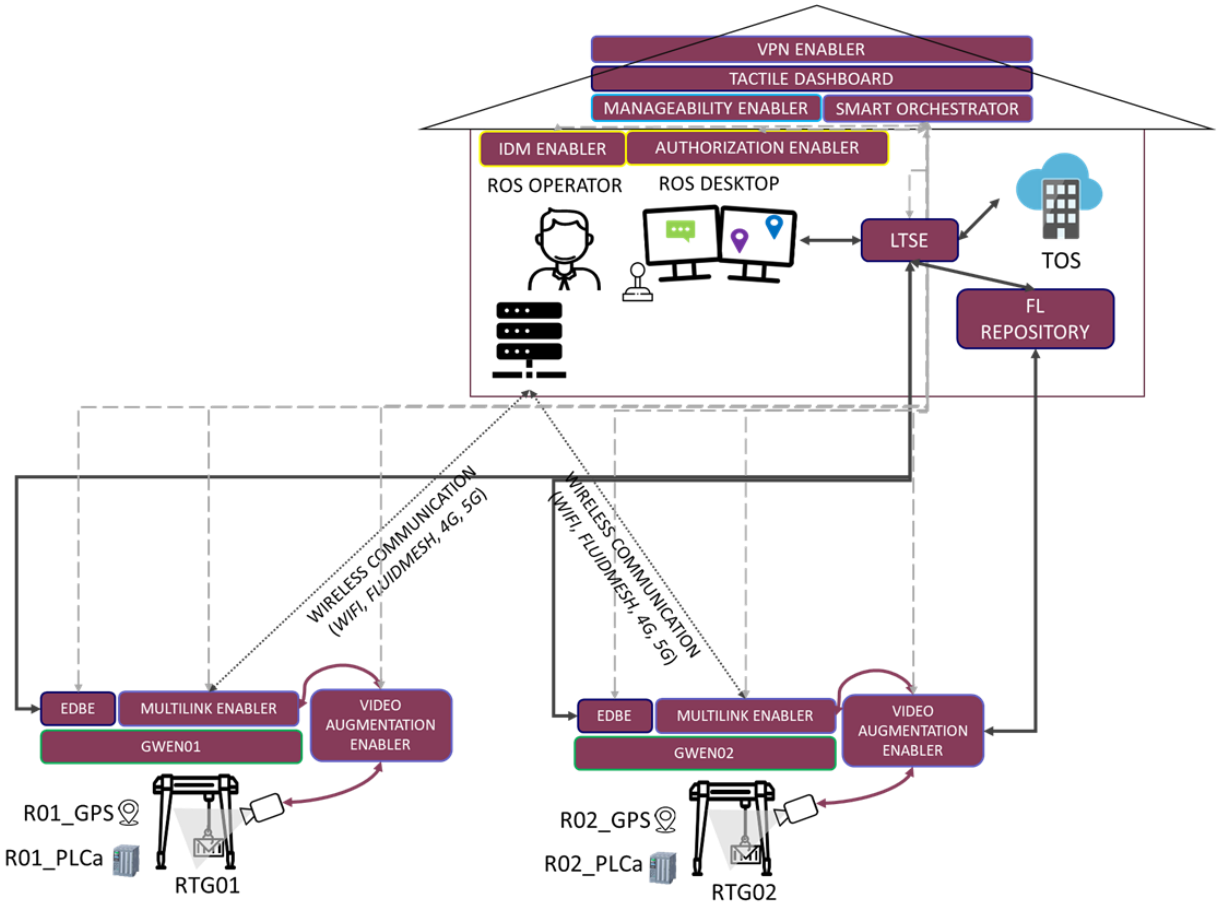


Figure 27. Architectural block diagram of Pilot 1 – Trial #3

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

4.3.2 Pilot 2: Smart safety of workers

4.3.2.1 Trial #1: Occupational safety and health monitoring

SUB TRIAL – 1: Workers' health and safety assurance sub-trial

The first sub-trial of pilot two involves the following testing activities:

- Connection between Authorisation and IdM enablers with Tactile dashboard
- Connection of Tactile dashboard and Construction site controller with Workplace safety controller
- Connection of EDBE with Workplace safety controller, PCS interface, Weather data collector, Wristband interface and semantic annotator
- Connection of Workplace safety controller with Semantic annotator and Semantic repository

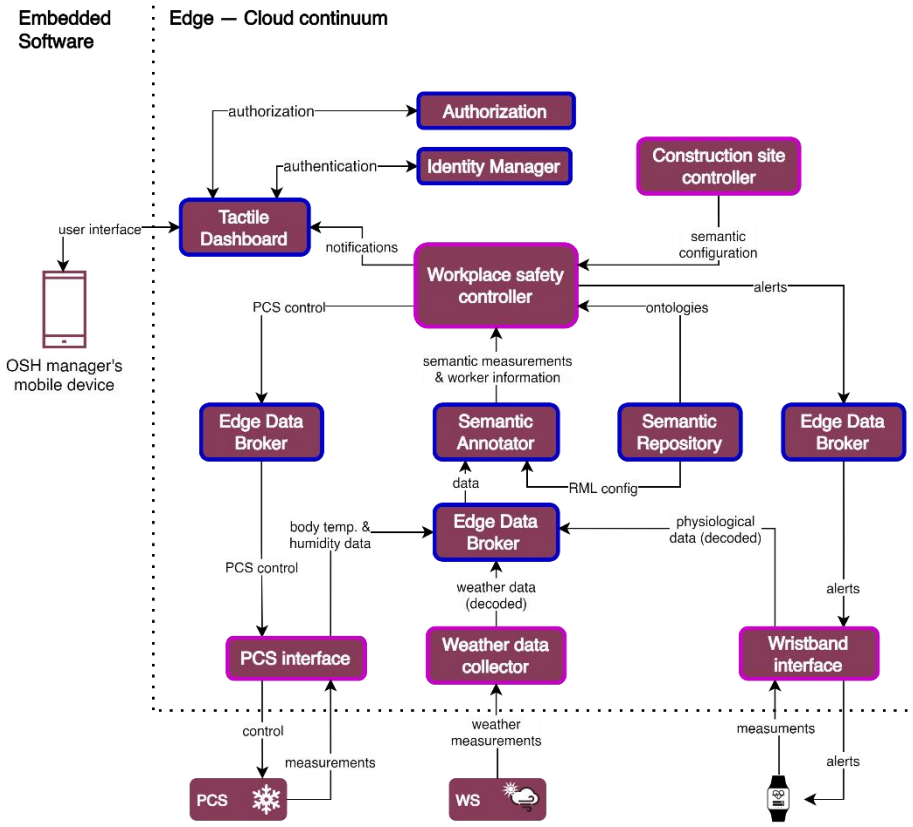


Figure 28. Architectural diagram for Workers' health and safety assurance sub-trial

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

SUBTRIAL – 2: Geofencing boundaries enforcement sub-trial

The second sub-trial of pilot two involves the following testing activities:

- Connection of location tracking enabler with EDBE
- Connection of EDBE with LTSE, Tactile Dashboard and Workplace safety controller
- Connection of Workplace safety controller with Semantic repository enabler, Constructions site controller and Integrity Verification enabler
- Connection of Tactile dashboard with Authorisation and IdM enablers
- Connection of Semantic repository enabler with BIM processor and Location processing enabler
- Connection of BKPI reporting enabler with LTSE and Tactile dashboard

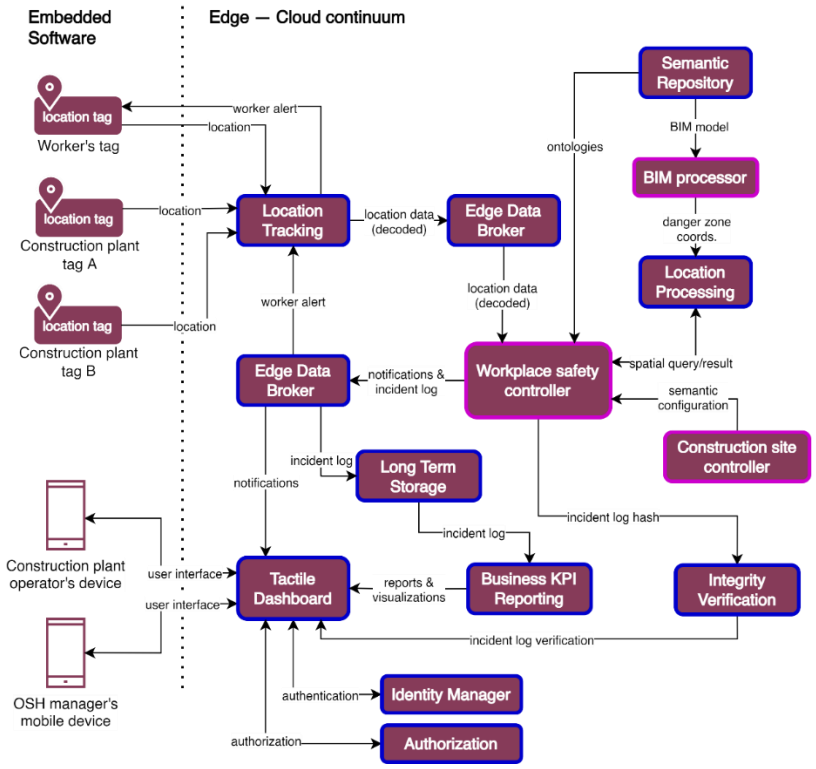


Figure 29. Architectural diagram for Geofencing boundaries enforcement sub-trial

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

SUB TRIAL – 3: Construction site access control sub-trial

The third sub-trial of pilot two involves the following testing activities:

- Connection of EDBE with Location tracking enabler, LTSE and Workspace safety controller
- Connection of Workspace safety controller with Amazon rekognition, Construction site controller, Semantic repository enabler and Integrity verification enabler
- Connection of Tactile dashboard with Authorisation and IdM enablers, BKPI reporting enabler, LTSE and Integrity verification enablers

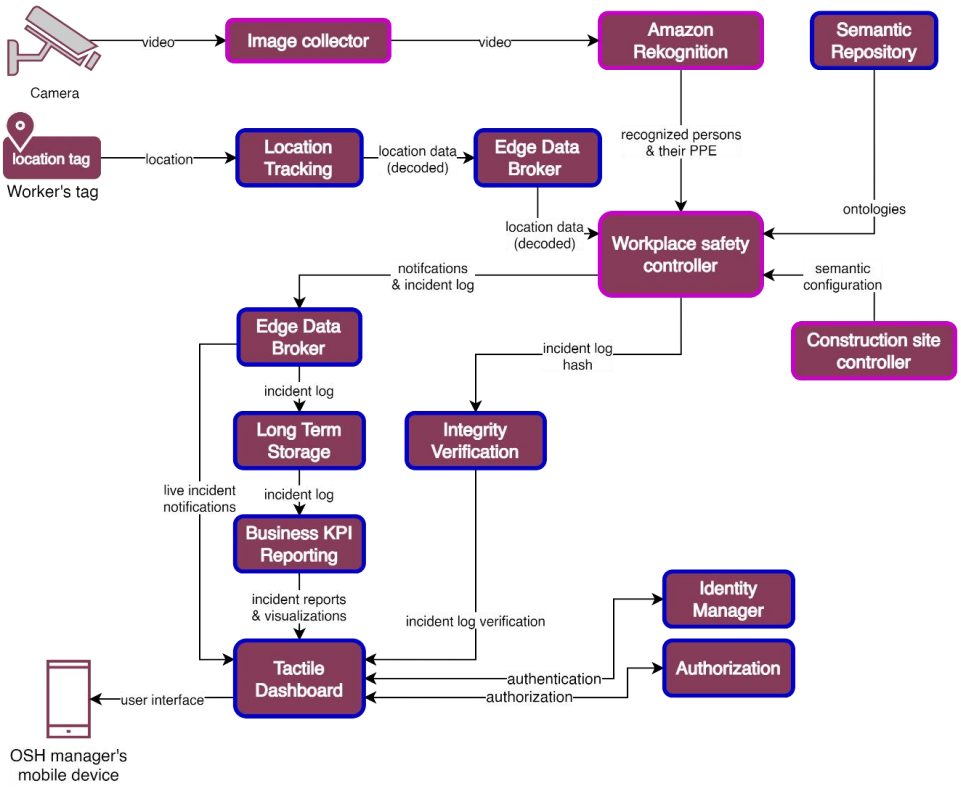


Figure 30. Architectural diagram for Construction site access control sub-trial

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

4.3.2.2 Trial #2: Fall-related incident identification

The second trial of pilot two involves the following testing activities:

- Connection of EDBE with Location tracking enabler, LTSE and Workspace safety controller
- Connection of Workspace safety controller with Amazon rekognition, Construction site controller, Semantic repository enabler, Integrity verification enabler and FL local operations
- Connection of Tactile dashboard with Authorisation and IdM enablers, BKPI reporting enabler, LTSE and Integrity verification enablers
- Connection of FL local operations with all the remaining FL enablers

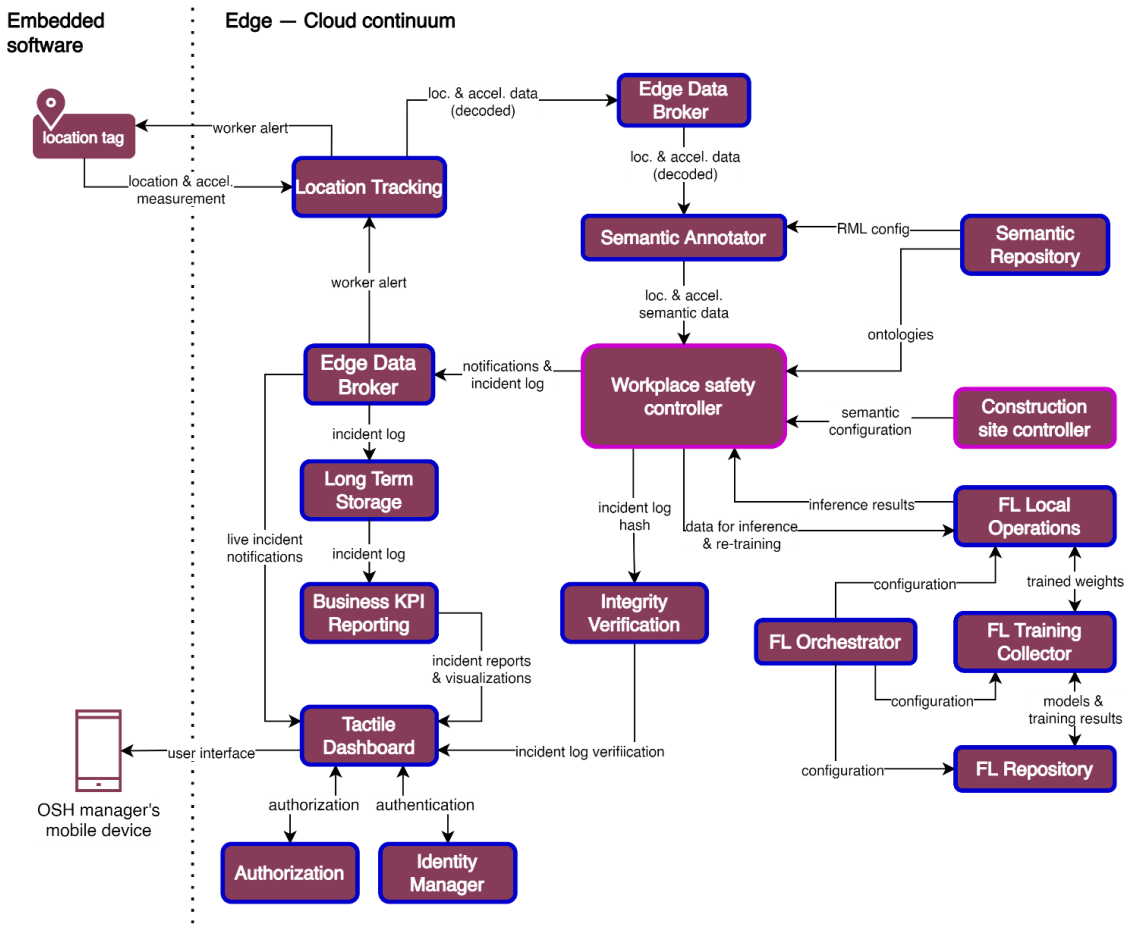


Figure 31. Architectural diagram for Fall-related incident identification trial

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

4.3.2.3 Trial #3: Health and safety inspection support

The third trial of pilot two involves the following testing activities:

- Connection of MR glasses with PUD, Tactile dashboard, Semantic repository enabler and EDBE
- Connection of Tactile dashboard with Authorisation and IdM enablers
- Connection of Workplace safety controller with EDBE, Location processing enabler and Construction site controller

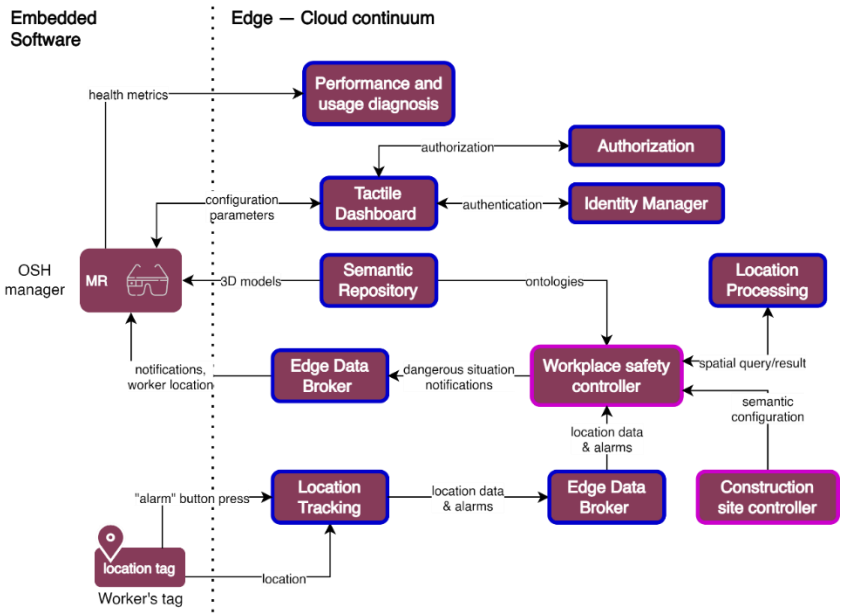


Figure 32. Architectural diagram for Safe navigation instructions sub-trial

- Connection of MR glasses with LTSE
- Connection of Location tags with Location tracking enabler
- Connection of EDBE with Location tracking enabler

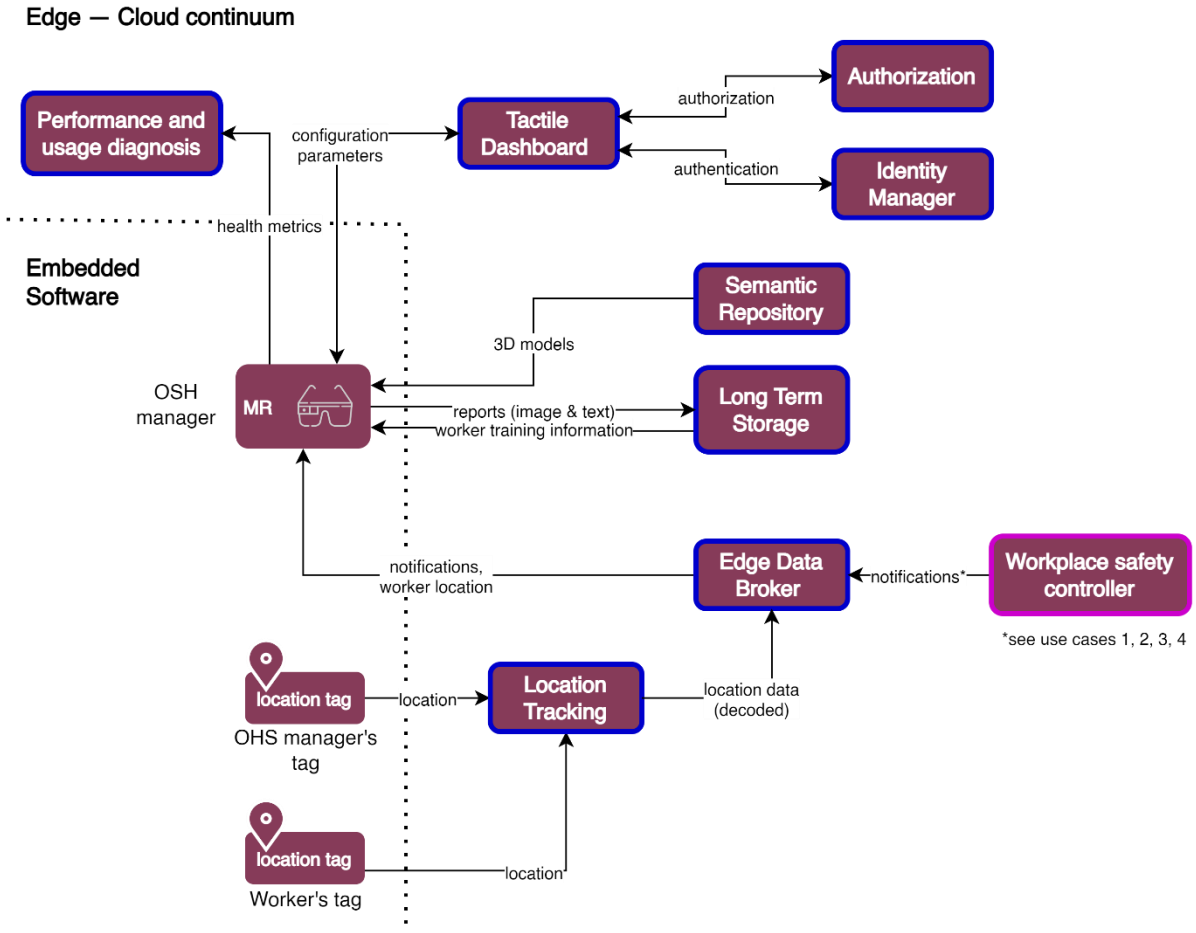


Figure 33. Architectural diagram for Health and safety inspection support sub-trial

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

4.3.3 Pilot 3A: Vehicle in-service emission diagnostics

4.3.3.1 Trial #1: Fleet in-service emission verification

The first trial of pilot three-a involves the following testing activities:

- Integration of GWEN, EDBE, Multilink enabler, Authorisation enabler and IdM as one technology block
- Connection of LTSE with the above block of enablers, Authorisation and IdM enablers
- Connection of Manageability enablers with Smart orchestrator

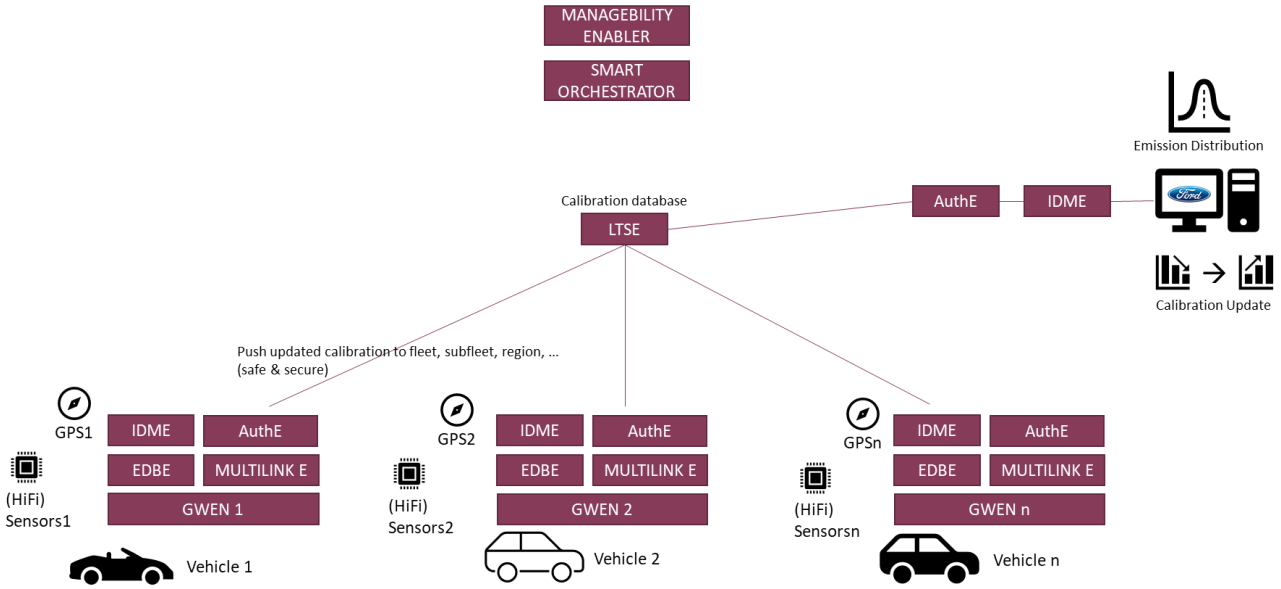


Figure 34. BS-P3A-1: Fleet in-service emission verification

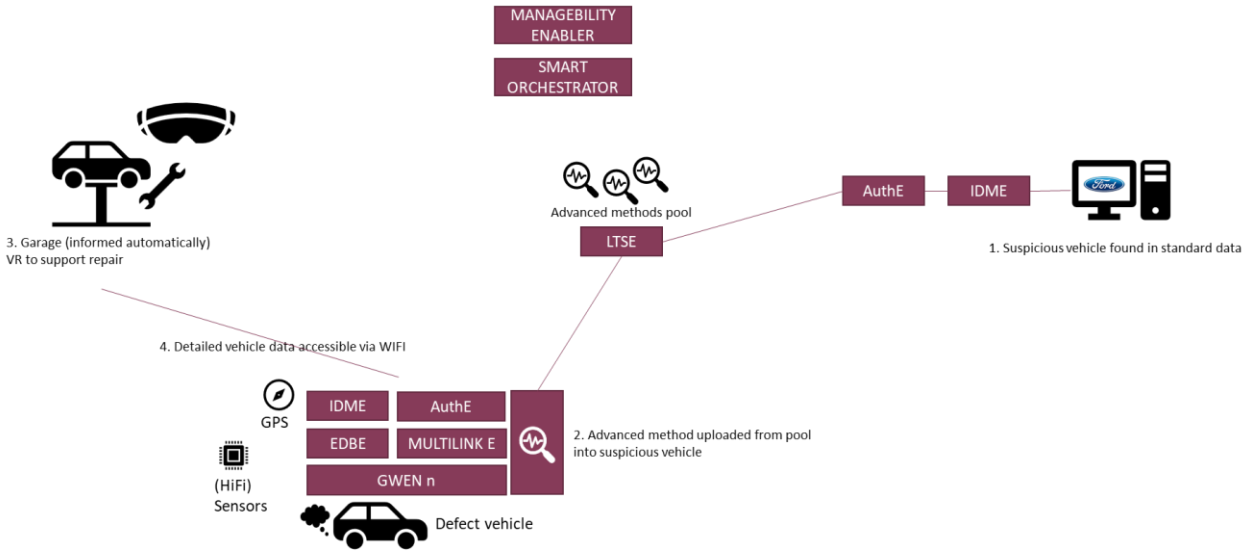


Figure 35. BS-P3A-2: Vehicle diagnostics

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

4.3.4 Pilot 3B: Vehicle exterior condition inspection and documentation

4.3.4.1 Trial #1: Vehicle exterior condition inspection and documentation

The first trial of pilot three-b involves the following testing activities:

- Connection of the scanner with EDBE
- Connection of Smart orchestrator with Tactile dashboard, LTSE and BKPI reporting enabler
- Connection of EDBE with LTSE
- Connection of the local Filesystem with FL enablers
- Connection of Tactile dashboard with Authorisation and IdM enablers

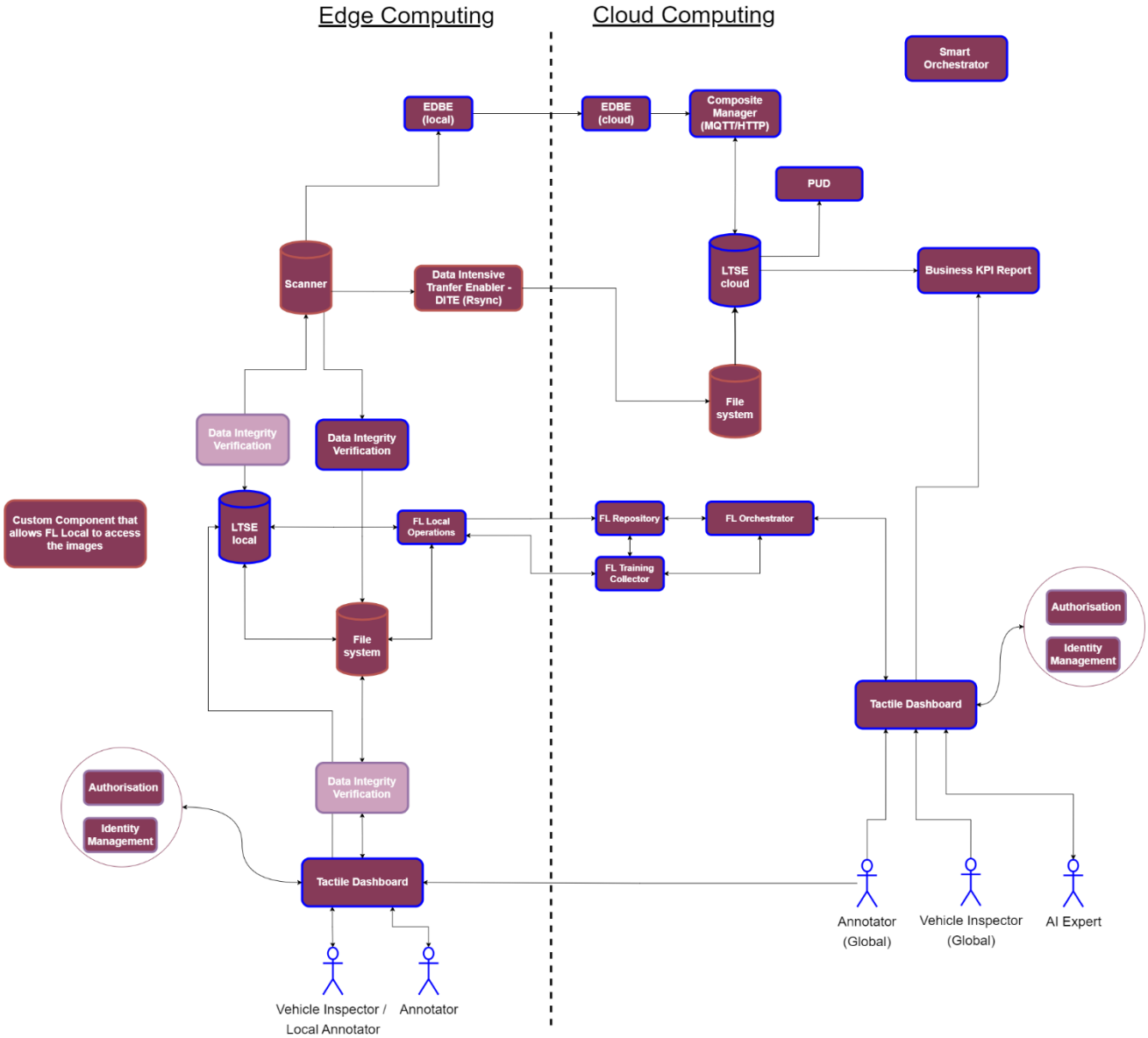


Figure 36. Architectural diagram for Vehicle exterior condition inspection and documentation

If there are tests already implemented, they will be reported in the corresponding table in the last deliverable.

4.4 Acceptance testing

Due to the amendment to the Grant Agreement of the project, that has extended the duration of WP6 till WP36 (originally, ending now at M30), the focus of this deliverable has shifted to reporting the tests that have been implemented in the first three testing phases agreed upon by the consortium. The aim is to have a practical approach for acceptance testing after the conclusion of the former, given that the project is now in a phase where pilots have to be validated. The definition of acceptance testing was documented in the previous deliverable D6.2, along with various technologies and a preliminary approach. The priority now is to ensure that the approach can be accurately implemented and executed in the following months.

In order to properly understand the purpose of acceptance testing in ASSIST-IoT, it is important to establish a link between the requirements defined in WP3, and specifically in D3.3 [4.]. This involves mapping the requirements onto the corresponding business needs and defining the solutions that will fulfill those needs. The primary goal of acceptance testing is to verify compliance with the gathered requirements and assess whether the solution is ready for delivery to the pilots. This is done in a laboratory environment subsequently to the end-to-end testing process in which the pilot trials have been tested.

From functional to non-functional and with their priorities set, the requirements will be analyzed and classified, with the next step being to verify the acceptance criteria formulated in WP3 to ensure that the solution meets the requirements and is suitable for deployment in the pilots.

4.5 Performance testing

Performance testing is the final testing process for software deployment that focuses on speed, response time, stability, reliability, and scalability [20.]. This test relates closely to business requirements as the system's operations will be evaluated against business indicators [21.]. The result of this testing phase is the diagnostic information leading to eliminating bottlenecks and improving the poor performance of components. The literature divides the performance testing into smaller units [22.], which are: i) load, ii) stress, iii) endurance, iv) spike, and v) configuration testing. The aforementioned tests demand various conditions for assessing the system's performance.

In terms of ASSIST-IoT, performance testing is to take place in the deployment of pilot sites. The work in other packages is essential for providing the basic indicators for running performance tests. The indicators range from technical deliverables relevant to architecture and development to business driven by the pilot sites. These indicators will provide an overview of the system's scalability, reliability, stability, and efficiency in different IoT scenarios.

The requirements that were appointed for acceptance testing can be segregated, and the ones that are more relevant to performance testing should be applied to this phase. In a sense, the aforementioned requirements along with some KPIs will form the criteria for performance testing. Practically, the same pilot pipelines tested in the acceptance phase should be used for this testing phase by stressing and loading them.

Since each component has its own performance metrics, it is important to verify the quantitative values under different conditions. Therefore, the following units should be the core of performance testing:

- **Throughput**: The number of requests or transactions that the system can handle per unit of time, which applies to all pilots that interact with LTSE for example.
- **Response time**: The time it takes for the system to respond to a request.
- **Latency**: The time it takes for a specific operation to complete, for example, the time it takes for a notification to arrive to a worker for pilot 2.
- **Concurrent users**: The number of users that can use the system simultaneously without degrading performance. (KPI 4.7.1)
- **Resource allocation**: The amount of system resources, such as CPU, memory, and disk space that the system uses during different types of operations. It is essential to have the capacity to dynamically expand or contract resources without sacrificing performance or availability.

5 Conclusion / Future Work

The objective of this deliverable is to provide a detailed report on the tests that have been conducted during the integration process, following the DevSecOps methodology. The report covers the procedures, tools, and tests utilized to ensure the security of the deployed architecture. The tools employed in the testing and integration process include GitLab, GitLab CI/CD, GitLab Runner, Container registry, Helm registry, and Kubernetes.

The testing strategy for each enabler and the level of integration achieved so far have been thoroughly documented, while the upcoming testing phases have been analysed in both theoretical and practical manners to facilitate their reporting in the final deliverable. The time plan for the testing and integration phase has been updated to reflect the amendment of the project.

This deliverable will be updated in conjunction with the other two WP6 deliverables, which will document the final testing and integration results, packaging and releasing, technical support, and documentation.

References

[1.] ASSIST-IoT (2022). D6.7: Release and Distribution Plan. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[2.] ASSIST-IoT (2022). D6.6: Technical and Support Documentation. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[3.] ASSIST-IoT (2021). D6.1: Devsecops Methodology and tools. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[4.] ASSIST-IoT (2021). D3.3: Use Cases Manual & Requirements and Business Analysis. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[5.] GitLab website. https://about.gitlab.com/

[6.] GitLab. (2021, 1 June). GitLab is setting the standard for DevSecOps. https://about.gitlab.com/blog/2021/06/01/gitlab-is-setting-standard-for-devsecops/

[7.] GitLab. GitLab CI/CD. https://docs.gitlab.com/ee/ci/

[8.] GitLab. GitLab Runner. https://docs.gitlab.com/runner/

[9.] GitLab. GitLab Registry. https://docs.docker.com/registry/

[10.] Helm. Helm Registry. https://helm.sh/docs/helm/helm_registry/

[11.] Helm. The package manager for Kubernetes. https://helm.sh/

[12.] Helm. Charts. https://helm.sh/docs/topics/charts/

[13.] Kubernetes. https://kubernetes.io/

[14.] Óscar López, Jordi Blasi, Mikel Uriarte, Ignacio Lacalle, Gonzalo Galiana, Carlos E. Palau, Eduardo Garro, Maria Ganzha, Marcin Paprzycki, Piotr Lewandowski, Katarzyna Wasielewska, Konstantinos Votis, Georgios Stavropoulos, Iordanis Papoutsoglou, DevSecOps Methodology for NG-IoT Ecosystem Development Lifecycle – ASSIST-IoT perspective, Journal of Computer Science and Cybernetics, 37(3):321-33, Sept 2021.

[15.] ASSIST-IoT (2022). D6.2: Testing and integration plan - Initial. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[16.] ASSIST-IoT (2022). D3.6: ASSIST-IoT Architecture Definition. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[17.] ASSIST-IoT (2022). D7.2: Pilot Scenario Implementation. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[18.] Guru99. (2022, 16 April). Integration Testing: What is, Types, Top Down & Bottom Up Example. Source. Accessed on 3rd of May 2022. https://www.guru99.com/integration-testing.html

[19.] Software testing fundamentals. (2020, 13 September). Integration testing. Source. Accessed on 3rd of May 2022. https://softwaretestingfundamentals.com/integration-testing/

[20.] Guru99. (2023, 21 January). Performance Testing Tutorial – Types. Source. https://www.guru99.com/performance-testing.html

[21.] Microsoft Learn. (2022, 12 January). Performance testing. Source. https://learn.microsoft.com/en-us/azure/architecture/framework/scalability/performance-test

[22.] Mustafa, K. M., Al-Qutaish, R. E., & Muhairat, M. I. (2009, December). Classification of software testing tools based on the software testing methods. In 2009 Second International Conference on Computer and Electrical Engineering (Vol. 1, pp. 229-233). IEEE.