*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

# D6.2 -Testing and integration plan – Initial

| | | | |
|---|---|---|---|
| **Deliverable No.** | D6.2 | **Due Date** | 30/04/2022 (delivered June 1st, 2022) |
| **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 |
| Ignacio Lacalle | P01 UPV | iglaub@upv.es |
| Eduardo Garro | P02 PRO | egarro@prodevelop.es |
| Paweł Szmeja | P03 IBSPAN | pawel.szmeja@ibspan.waw.pl |
| Katarzyna Wasielewska-Michniewska | P03 IBSPAN | katarzyna.wasielewska@ibspan.waw.pl |
| Georgios Stavropoulos | P04 CERTH | stavrop@iti.gr |
| Evripidis Tzionas | P04 CERTH | tzionasev@iti.gr |
| Iordanis Papoutsoglou | P04 CERTH | ipapoutsoglou@iti.gr |
| Aggeliki Papaioannou | P06 INF | apapaioannou@infolysis.gr |
| Theoni Dounia | P06 INF | tdounia@infolysis.gr |
| Óscar López | P13 S21Sec | olopez@s21sec.com |
| Jordi Blasi | P13 S21Sec | jblasi@s21sec.com |


# History

| Date | Version | Change |
|---|---|---|
| 17-Dec-2021 | 0.1 | ToC and task assignments |
| 28-Feb-2022 | 0.2 | ToC updates |
| 03-Mar-2022 | 0.3 | First round of contribution |
| 01-Apr-2022 | 0.4 | Update to ToC |
| 27-Apr-2022 | 0.5 | Second round of contribution |
| 14-May-2022 | 0.8 | IR Review |
| 30-May-2022 | 1.0 | Final vesion after PIC review |

# Key Data

| | |
|---|---|
| **Keywords** | IoT, equipment, SDN, NFV, smart control, architecture… |
| **Lead Editor** | P04 CERTH – Georgios Stavropoulos |

# 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 first one of two deliverables for testing and integration within the ASSIST-IoT project. This first deliverable focuses on tools, testing phases, and test plan for the developed components of the project following the DevSecOps methodology.

Essentially, DevSecOps methodology drives the adoption of tools and sets the testing phases for the Continuous Integration (CI). The document presents the current tools in place supporting the developing team of each partner to coordinate their work. The main tools are currently Gitlab and Kubernetes along with Gitlab Runner and Docker Registry. The tools will help in packaging the work and facilitate the future integration. The test strategy presents the testing phases to follow during the project in order to deliver functional components abiding by the requirements set in WP3 and technical WPs with special attention to security. Finally, the testing and integration plan includes details on the testing phases in the project, presents the environments to be involved during the tests, and presents the time plan and the status and provisional test results.

.

# Table of contents

1 About this document 8
1.1 Deliverable context 8
1.2 The rationale behind the structure 8
2 Integration infrastructure and tools 9
2.1 Gitlab 9
2.2 Gitlab Runner 9
2.3 Docker registry 9
2.4 Kubernetes 10
3 Test Strategy 11
3.1 Unit testing 11
3.2 Functional testing 12
3.3 Integration testing 12
3.4 End-to-end testing 13
3.5 Acceptance testing 15
3.6 Performance testing 16
4 Acceptance and integration test plan 18
4.1 Development of the Testing Methodology 18
4.2 Testing process in ASSIST-IoT 21
4.2.1. Functional Testing of horizontal enablers 23
4.2.2. Functional Testing of vertical enablers 32
4.3 Test environment 39
4.4 Time plan 39
5 Conclusion / Future Work 41
Appendix 1: Enabler's status 42

# List of tables

Table 1: Template for Integration Tests ....13
Table 2: Software Test & Integration plan ....20
Table 3: Smart Orchestrator's functional tests ....23
Table 4:Traffic Classification's functional tests ....24
Table 5: Multi-link's functional tests ....24
Table 6: SD-WAN's functional tests ....24
Table 7: WAN Acceleration's functional tests ....25
Table 8: VPN's functional tests ....26
Table 9: Semantic Repository's functional tests ....26
Table 10: Semantic Translation's functional tests ....27
Table 11: Semantic Annotation's functional tests ....28

Table 12: Edge Data Broker's functional tests....29
Table 13: Long-Term Storage's functional tests....29
Table 14: Tactile dashboard's functional tests....30
Table 15: Business KPI Reporting's functional tests....30
Table 16: PUD's functional tests....30
Table 17: OpenAPI Management's functional tests....31
Table 18: Video Augmentation's functional tests....31
Table 19: MR's functional tests....31
Table 20: Self-healing's functional tests....32
Table 21: Automated Configuration's functional tests....32
Table 22:Resource Provisioning's functional tests....32
Table 23: Monitoring and Notifying's functional tests....33
Table 24: Location Processing's functional tests....33
Table 25: FL Training Collector's functional tests....34
Table 26: FL Orchestrator's functional tests....34
Table 27: FL Repository's functional tests....35
Table 28: FL Local Operations' functional tests....35
Table 29: Identity Manager's functional tests....35
Table 30: Authorisation's functional tests....36
Table 31: Cybersecurity Monitoring's functional tests....36
Table 32: Cybersecurity Monitoring Agent's functional tests....37
Table 33: Logging and auditing's functional tests....37
Table 34: Integrity Verification's functional tests....37
Table 35: Broker service's functional tests....37
Table 36: FL DLT's functional tests....38
Table 37: Registration and status of enablers' functional tests....38
Table 38: Device Management's functional tests....38

# List of figures

Figure 1. Unit Testing Life Cycle....12
Figure 2. Functional Testing high level diagram....12
Figure 3. Architectural diagram for Construction site access control sub-trial....14
Figure 4. Acceptance Testing high level diagram....16
Figure 5. Performance Testing Life Cycle....17
Figure 6. Waterfall model of ASSIST-IoT Development Life Cycle....18
Figure 7. V-model of ASSIST-IoT Development Life Cycle....19
Figure 8: DevSecOps embedded security control....19
Figure 9. Test environment to simulate pilot site premises....22
Figure 10. ASSIST-IoT testing and integration time plan....39

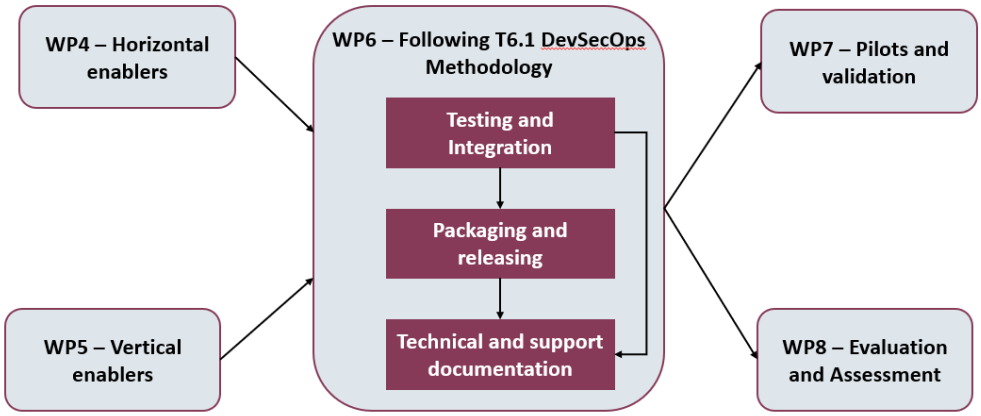
# List of acronyms

| Acronym | Explanation |
|---|---|
| **AC** | Automated Configuration |
| **AD** | Active Directory |
| **API** | Application Programming Interface |
| **CI** | Continuous Integration |
| **CI/CD** | Continuous Integration/continuous delivery |
| **CNCF** | Cloud Native Computing Foundation |
| **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 |

# 1 About this document

The key objective of this document is to provide the initial plan of testing and integration for the project. The testing and integration follow different phases for enforcing the DevSecOps methodology, and each phase can use tools and metrics to evaluate the performance. The final version and outcomes will be part of the second iteration of the deliverable scheduled for M30.

## 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.<br>**O2 to O5:** Each of the implementations is a subject to testing.<br>**O6:** The testing environment will be used for testing and integrating the developed solutions before validating them in the pilots. |
| **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. |
| **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 (D6.4 [1.]) and documentation (D6.5 [2.]). The next iteration is expected in M30. It partially draws from the D6.1 [3.] – DevSecOps methodology delivered in M6. |

## 1.2 The rationale behind the structure

This deliverable follows a straightforward approach: Section 2, presents the infrastructure and tools in place; then, Section 3, the test strategy is elaborated; Section 4 presents the acceptance and integration test plan for the project. Finally, conclusions are drawn in Section 5, introducing the outcomes expected in the next iteration of the current deliverable.

# 2 Integration infrastructure and tools

## 2.1 Gitlab

GitLab [4.] is a web-based Git repository that offers open and private repositories for free, as well as issue tracking and wikis. It is a suitable platform to apply DevSecOps methodology, as it allows developers to handle all aspects of a project, from project planning to source code management, monitoring and security. Furthermore, GitLab enables teams to cooperate and create better software. Teams can shorten product lifecycles and boost productivity resulting in greater value for customers. Users are not required to handle authorisations for each tool in the program. Roles and permissions are part of the project to evaluate access to individual components depending on the access rights.

Security is a fundamental part of the methodology in ASSIST-IoT, as it becomes everyone's responsibility. GitLab is a platform that facilitates and automates DevSecOps procedures with an updated approach depicted in a forum post [5.]. In detail, security testing is applied within the CI pipeline raising developers' awareness of vulnerabilities introduced in their code and making it actionable to be corrected. Moreover, the CI pipeline is not costly, and a single license manages it.

Apart from security being a focal point, GitLab is a platform that enterprises use in daily operations for different reasons. One reason is to support the collaboration between teams since this is vital for a successful project deployment. Moreover, GitLab available features like repositories, issues tracking and roles are helpful in project management by assisting the stages of software development.

As GitLab is already organized for testing and integration purposes, additional functionalities and details have also been set up. In detail, the established tools are GitLab runner and Docker Registry, and new tools can be added in the future for providing additional functionalities. A security policy with measurements is in place to safeguard the work developed in the project. Indicative rules applied on GitLab are the obligatory two-factor authentication and password update in timely intervals. The project has divided the GitLab platform into work packages and tasks for the most appropriate management. The partners can define their personnel, apply access rights, and integrate their work by uploading code and binaries on the platform.

## 2.2 GitLab Runner

GitLab Runner [6.] is introduced as the application for running CI/CD jobs in a pipeline, developed by GitLab. Generally, runners are agents that run jobs and communicate with the Gitlab instance.

GitLab Runner was selected based on the matching features to the project's specifications. A feature by GitLab Runner allows running Docker containers harmonised with the project's idea for the containerisation of the designed enablers. Moreover, the application is able to run on a plethora of systems providing flexibility to the development. Finally, Prometheus analytics' availability was deemed valuable for the project.

## 2.3 Docker registry

The Registry [7.] is a server-side application that stores and distributes Docker images. It is stateless, as the application does not read or store information about its state, and extremely scalable. The Registry is open-source and licensed under the Apache permissive licence.

Docker registry offers improved latency, better availability, greater integration with enterprise-grade AD/LDAP and SSO, better image access control, better security (protected from external threats), ability to run containers in private subnets and reduced cost. Authenticated users are able to push and pull Docker images.

Docker Registry adds the functionality to store the project's Docker images. Moreover, teams can manage the image versioning to streamline their work. Finally, the project's partners are experienced in working with Docker registries.

## 2.4 Kubernetes

Kubernetes [8.], also referred to as K8s, is a system for orchestrating containerised workloads and has been chosen for the instantiation of the project's enablers (or rather, their components). The system was originally deployed by Google, but the Cloud Native Computing Foundation (CNCF) and has become the de facto standard for container orchestration in production environments.

Kubernetes extends the functionalities provided by Docker (understanding it as the engine for deploying containers). To begin with, Docker run workloads in a single node, which is a clear point of failure: in case this node fails, or an update is needed, the services/applications from the hosted containers are halted. K8s avoids this kind of issues by working with clusters with several nodes as well as with dedicated updating strategies, making it much more suitable for production environments.

Enablers must be prepared and packaged to work in this environment, and therefore it should be part of the integration environment.

# 3 Test Strategy

This chapter lays the foundation for coordinating the tests to be performed over the technical components of the project. Definitions are provided to get familiar with the different testing cycles that should be followed to adhere to the DevSecOps methodology.

## 3.1 Unit testing

Unit testing is a term defined as a test executed by developers in a laboratory environment to demonstrate a program's ability to meet the requirements set in the design specification [9.]. The units in this kind of testing can either be individual components or a bundle of them [10.]. The focus is set on the unit, while the general system is not part of the consideration during this testing. Runeson's survey indicates that the development team should conduct unit tests on the component.

It is vital to determine the unit in terms of ASSIST-IoT project. The work in the architectural design defines a microservice architecture for the project where each component is responsible for the execution of a single function. In other words, the architectural design has divided the functionality into **enablers**. Enablers act as a bundle of internal components, such as databases, APIs, registries and services in general. Unit testing will be applied on the aforementioned individual components' classes, methods and internal interfaces, as each enabler is a complex structure of individual components.

The methodology that will be followed in ASSIST-IoT will not deviate from the general principles of unit testing. The idea is that a "unit" will be defined as the smallest testable part of each enabler's component. Unit tests should be automated and written using a unit test framework, corresponding to the programming language that each developing team is making use of. For example JUnit for Java or unittest for Python etc.

Unit testing should take place before (or in parallel with) uploading code into repositories, in order to ensure each unit's correct behaviour, and only components that have passed all unit tests should be committed. Specifically for each major method – function – class (units) – internal API of each enabler's component a single test should be implemented. Methods should have corresponding test-methods that with known inputs, produce expected outputs. Furthermore, classes should have abstract test classes implemented, to be initialized before the main classes are called. When calling the main program, the test methods – functions – classes will be called first and if unexpected outputs occur, there will be an error assertion that the test has failed.

Some simple guidelines are:

- Unit test means that each test tests exactly one thing.
- Each test method is one test.
- Each test method should have one assert (Pass/Fail).
- Data inputs and outputs for the test should be deterministic.
- Failing tests should have clear and unambiguous error messages.

To further enhance the software quality, it is recommended to create a test for each bug occurrence with a concrete bug statement number to automate the bug tracking and fixing. There will be no criteria for test coverage percentage at this point of development.

The unit testing considers the individual components and ignores the overall system. As enablers are natural to have dependencies on each other, these dependencies should be substituted by mock objects for running tests. There are different scenarios to cover during testing, like positive, negative, and limit tests.

All in all, a unit test covers a specific functionality of the unit. Units with multiple functionalities should consider matching tests to these functionalities. Components meeting the acceptable behaviour should be the ones to be included in the system.

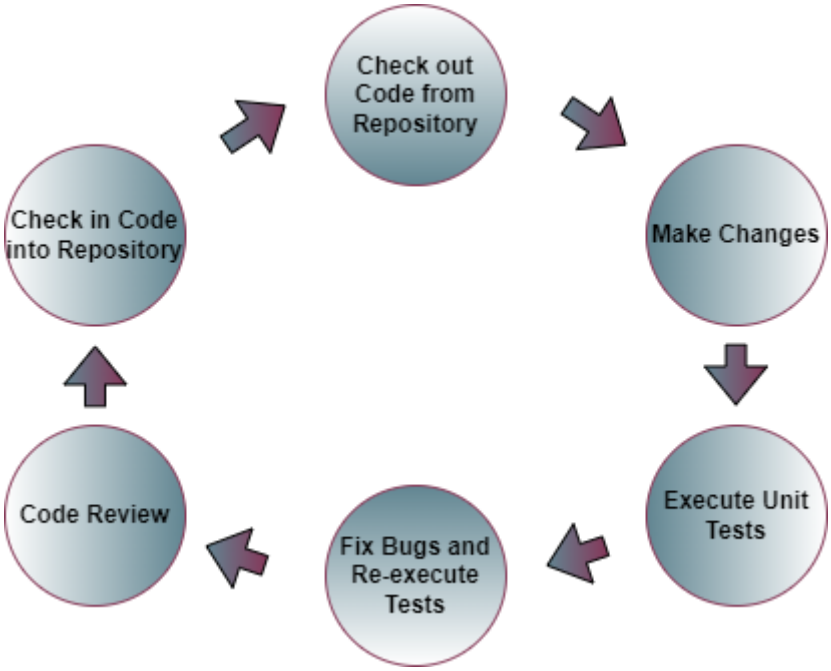



*Figure 1. Unit Testing Life Cycle*

## 3.2 Functional testing

Functional testing is another testing method that ASSIST-IoT will incorporate into its pipeline to guarantee the system's functionality. In detail, the requirements are drafted in WP4 and WP5 and are to be validated with functional testing on the delivered platform. In this testing phase, the system will be given an appropriate input to examine the adherence to the requirements.

Generally, functional testing is a black box test method that does not focus on the platform's source code. An initial step for this test method is to define clearly the functions expected to be performed by the platform.

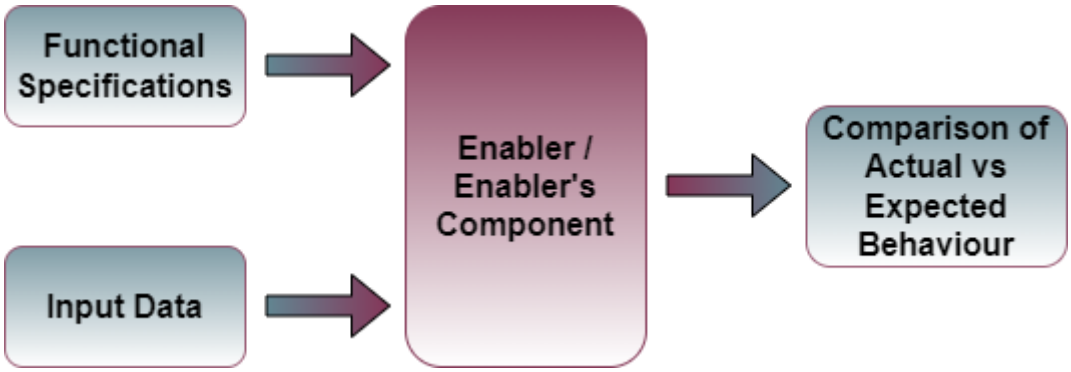



*Figure 2. Functional Testing high level diagram*

## 3.3 Integration testing

The next phase of testing in the project is the integration testing. While the individual components of the system are tested in the unit testing, the integration focuses on the components' interaction to provide higher-level functionality. Hence, integration testing considers runtime factors such as compatibility with interfaces, services, and dependency resolution. The collaboration of multiple units is under the integration testing concept.

In ASSIST-IoT, the enablers are mainly part of the two technical work packages (WP4, WP5). The integration tests act as grey box tests meaning that a complete workflow and its results are to be tested. The workflow structure is set by the architectural structure that will be realised in the pilot sites.

Since the testing strategy should follow a sequential manner, integration testing's focus is built around the next phase of ensuring the correct behaviour of interacting internal components of an enabler. Essentially, we have to make sure that internal enablers' APIs are working properly, by creating methods that are exchanging expected data, meaning that with the correct input we are expecting a desirable output between the internal components. There are tools and frameworks for integration testing that will be specific to programming and development environments, since the functionalities of each enabler differs in a variety of ways. The general idea is to develop tests for a specific use case or tool under test and deliver a corresponding template:

*Table 1: Template for Integration Tests*

| **ASSIST-IoT enabler #** | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Tool / Use Case under Test** | | | | | | | |
| **Test ID** | **Test** | **Description** | **Input** | **Outputs** | **Associated tool / Use case** | **Test result** | **Comment** |
| # | Integration | Describe the function, interaction, module, tool to test. | Inputs for integration test | Expected outputs | Tool: Jenkins / Maven / Git<br><br>Use case: Enabler #, Use case # | Pass/Fail | If failed, explain why |
| # | | | | | | | |
| # | | | | | | | |

The majority of the enablers follow the encapsulation logic for delivering their functionality in an independent manner. Messages and endpoints are vital parts in the execution of a pipeline between the different enablers that will support the successful integration testing. Hence, as a second phase of integration testing the approach should be to connect the enablers as a whole in a use case scenario, by developing the tests to establish the connection and determine the workflow of the use case. Use cases will be derived from D4.2, D5.2 and D5.3 deliverables, in which every enabler has documented several cases of its internal components interacting with each other, and to be distinguished from end-to-end testing which will draw pilot trials from D7.2. Gitlab is the main tool for implementing this phase of testing, though many of the partners, who have already started developing integration tests are using their own frameworks, such as Citrus, FitNesse or Rational Integration Tester. Of course Docker and building test containers will be the major and vital part of this phase of integration testing, to follow the enablers' encapsulation logic.

## 3.4 End-to-end testing

End to end testing is a methodology used in the software development lifecycle to test the functionality and performance of a given application from start to end. Today's applications are rarely stand-alone products, and their effective operation requires a wide web of interconnected systems (e.g., networks, external databases etc.). In extension, the application requires multiple underlying subsystems to operate smoothly, and if one subsystem fails, the entire application fails as well. Hence, teams nowadays must not only test the high quality and performance of the application as a whole but also the interconnected subsystems and how they communicate. End-to-end testing is the most effective approach to the above.

ASSIST-IoT is a foreground for conducting end-to-end testing in a vast environment of interconnected subsystems. The goal is to validate the system under test by validating the components of the project application (UI & API) and making sure the system behaves as expected. In a nutshell, the ultimate goal is to simulate a pilot trial and validate that the actual results are equal to expected.

In general, the benefit of end-to-end testing in ASSIST-IoT is to confirm the applications' health, expand the test coverage, detect bugs and, lastly, reduce the testing resources. This testing phase considers the application as a unit to ensure the optimal operation of the workflow as a whole.

For example we can refer to D7.2, pilot 2, trial #1 – Construction site access control sub-trial:

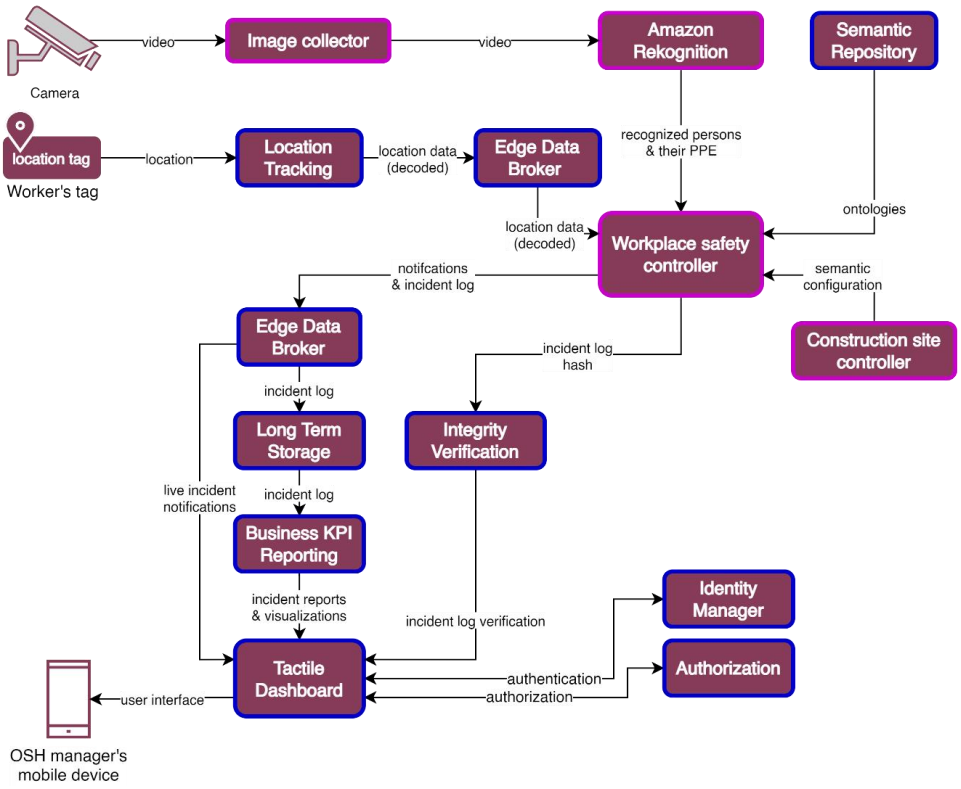



*Figure 3. Architectural diagram for Construction site access control sub-trial*

The above diagram's flow is explained in D7.2 section 3.2.1. Here we are going to try to give the outlines of the end-to-end testing of this trial.

- Develop a test for ensuring the connection of the camera to the image collector, a test for ensuring that the images arrive intact to Amazon Rekognition and a test for ensuring that the recognized persons & their PPE arrive at the Workplace safety controller.
- Develop the tests that known location tags are parsed to the location tracking enabler, then collected by EDBE and finally arrive at the workplace safety controller, decoded and with the expected output.
- Develop tests for the semantic repository and construction site controller to ensure that basic ontologies and configurations are loaded correctly.
- Develop tests that ensure the flow of data from workplace safety controller -> EDBE -> LTSE -> Business KPI reporting -> Tactile Dashboard.
- Develop Tests for the incident log has it parsed to the integrity verification enabler and from there to the tactile dashboard.
- Develop tests that ensure that the identity manager and authorization enabler correctly identify and give access to the related authorized personnel.

All the above actions consist an approach of developing test that give a holistic point of view of the construction site access control sub-trial, under the condition that the unit and integration testing have already taken place and passed. Thus far, the test will be conducted by developers and integration teams without end users involved.

While the above is just an example, the exercise of drilling down the tests for every "end-to-end" trial is being performed for all cases. In ASSIST-IoT, pilots are composed of trials (3 in pilot 1, 3 in pilot 2 and 2 in pilot 3 – divided in 1 for the pilot 3A and 1 for the pilot 3B), thus this will be the structure followed in T6.3 for the testing. This activity has been initiated across the board, but details on sub-tasks have not been able to be completed by the time of submitting this deliverable. Technical teams have been focused on delivering MVPs of the enablers, as well as on producing enough documentation for M18 (D6.5, D4.2 and D5.3). On the other hand, the diagrams that are being used as the baseline for devising testing plans (see Figure 3) have been just generated to be included in D7.2. It is, thus, planned, to elaborate those detailed end-to-end testing scenarios during the next weeks and to apply them as soon as the software artefacts will be ready.

These plans will be detailed (both in terms of planning and the evidences of its performance) in deliverable D6.3, by the time of finishing the work package (M30 – April 2023).

## 3.5 Acceptance testing

Validation in ASSIST-IoT is accomplished through Factory Acceptance Testing (FAT) and Site Acceptance Testing (SAT). Before deploying the system to the actual production environment, a vendor usually conducts factory acceptance testing. The goal of FAT is to ensure that the created systems meet the stated or contractually promised specifications and regulatory obligations prior to delivery and final installation. The system can be then installed in a real-world production environment, and site acceptance testing can be performed.

User tests in FAT, which are typically conducted by business experts or end-users, are neither focused on minor issues or errors nor on major software/services crashes, which are supposed to be acknowledged and fixed in earlier unit testing, integration testing and system testing phases. The FAT simulates real-world and real-time settings and operates as a final verification of the system's required business functionality and appropriate operation. If the software performs as expected and without major issues during routine operation, the same level of stability can be anticipated in production.

After FAT is conducted, the Site Acceptance Testing will occur on the pilot site. As a part of a quality management system, Operational Acceptance Testing (OAT) is used to measure the operational readiness testing of the solution in general.

SAT, a common type of non-functional software testing, concentrates on the system's operational readiness to be supported and/or integrated into the production environment. Prior to handing over the system to the final users on the pilot site, SAT serves as a final verification of the system's required functionality and proper performance. It tackles the problems of interfering with the already installed system by covering key quality attributes of functional stability, portability, and reliability, as well as procedures for disaster recovery, end-user training, maintenance and security.

The strategy followed by ASSIST-IoT platform is built in a sense of beta testing. The already tested pilot trials should be now tested by end users and vendors to assess if the system can support day-to-day business and ensure that the system is sufficient and correct for business usage. After building the test scenarios, which in our case will be structured around the pilot trials, the approach is to provide guidelines to the users on how to operate the platform and get feedback if their requirements are met.

The flow can be described in the following steps:

- Set up the operational environment for the tests and connect the interfaces.
- Perform the actions needed with desired inputs.
- After ensuring the delivery of expected outputs in real time, deploy the tests in the pilot site environment.
- After ensuring the user/vendor criteria is met proceed to disaster recovery, training of end users, maintenance and security guidelines.

There are several tools providing complete solutions for UAT. In ASSIST-IoT, as GitLab is the already used platform for developing and integrating software, it will be the main choice for testing as well. It is one of the most common software to design and execute tests, which allows to track issues, bugs and other work items through a predefined workflow that users can modify to fit their requirements. Other solutions could include Jira, Zephyr for Jira tool, which can be integrated with a variety of automated testing frameworks, such as Jenkins CU, Selenium, Appium and SmartBear products. Finally, Rally Software and qTest platform are popular solutions in the field of testing, by providing KPI tracking and reporting, high scalability and issue tracking. In a Consortium including as many partners as ASSIST-IoT (used/forced/set up to use specific tools already in place), it is not straightforward to oblige on a unified framework for executing the required test methodology, because it is not easily applicable for all the technologies involved. Using more than one tool is probably required, though it is not mandatory.

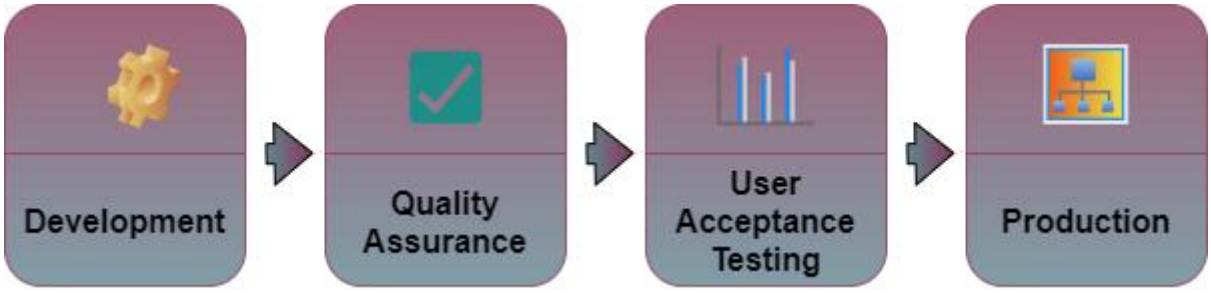



*Figure 4. Acceptance Testing high level diagram*

## 3.6 Performance testing

The goal of performance testing mainly revolves around responsiveness and stability under predetermined conditions. Furthermore, it is a suitable testing phase to produce notions for the scalability and reliability of the system. There are different test methods for gauging the system's performance. Apart from the methods, measurements and goals should be clearly defined for the complete testing. An early definition of performance testing points to the evaluation from a user's perspective [11.]. All in all, the goal of performance testing goal lies in alleviating performance bottlenecks rather than concerning bugs.

The methods for performance testing covered in the literature [12.] are numerous. The methods are briefly described as follows:

- Load testing: focuses on the application's ability to perform under the expected loads by a user. The goal is to catch bottlenecks before the application's production phase.
- Stress testing: gauges the limits of a system as extreme workloads are applied on the software.
- Endurance testing: estimates the behaviour of the system over a long time period as systems have to sustain the continuous load.
- Spike testing: tests the application for unexpected changes in user loads. Spikes are causes for system failures.
- Configuration testing: is to test the developed system with a combination of software and hardware to gauge the functional requirements.

Performance testing comes last in the testing pipeline, which means that it will take place after acceptance testing. Essentially, if we consider that the final product of ASSIST-IoT is ready, operational and deployed on the pilot sites, then it is the time for conducing performance tests. This process will include the integration team in collaboration with developers and the stakeholders, who better understand the scenarios that performance is critical and vital for providing optimal solutions.

Load, stress, endurance ,and spike testing are not mandatory for all enablers, due to the nature that they will not be used uniformly throughout the project. The suggestion is to apply the above tests in the essential enablers realized in WP4 and WP5, that provide the vital and most used functionalities of the project. Nevertheless the developing and integration teams may apply the tests in their enablers even if they are not essential ones, to optimize their product. Configuration testing should apply at all enabler in order to find the best balance between performance and scalability, taking into account both software and hardware infrastructure.

GitLab uses k6, a free and open source tool, for measuring the system performance of applications under load. It can be exploited as the main tool for performance testing along with some suggested tools such as LoadNinja, Apache Jmeter for Java apps, Gatling, Webload, SmartMeter, LoadRunner etc. They all provide virtual users and/or machines to manually load the system and inspect its behaviour in a real-world and real-time "heavy load" scenario.


*Figure 5. Performance Testing Life Cycle*

# 4 Acceptance and integration test plan

This section will present for each ASSIST-IoT component, the acceptance and integration test cases that will be executed in order to have a successful ASSIST-IoT platform. Apart from the description of the test cases, the time schedule of their activities is also presented.

## 4.1 Development of the Testing Methodology

Different development methodologies determine the software development lifecycle and the testing methodologies. Such well-known cases for methodologies are the waterfall model and the V-model. The waterfall method defines fixed steps in a downwards structure, as one can observe in **¡Error! No se encuentra el origen de la referencia.**. The general steps for the waterfall method are the definition of user requirements, the architecture and technical specifications, unit implementation and testing, system integration, and maintenance. The V-shape method extends the waterfall method by clearly associating each step with a corresponding testing phase to evaluate the results of the development, as it can be seen in **¡Error! No se encuentra el origen de la referencia.**. The aforementioned methods possess weaknesses for developing a novel NGIoT architecture for humans. The waterfall method postpones the testing and security for the final stages in a separate step, while the V-shape is a rigid method and hard to apply for the novelty of architecture. Another method of bringing security in earlier stages and allowing a flexible adaptation to foster innovation is required by the project. For that reason, DevSecOps is the selected methodology to apply to the project and harmonise the testing around the software cycle of the methodology.

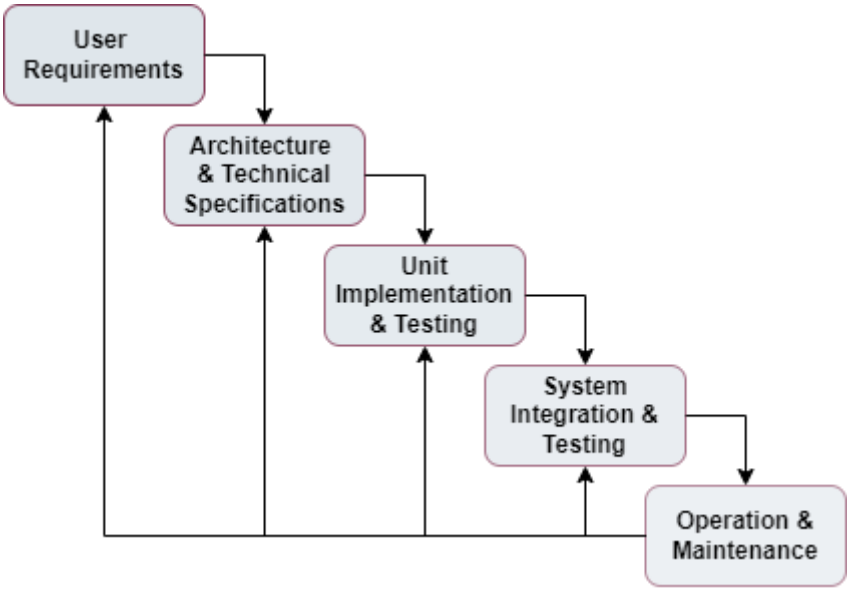



*Figure 6. Waterfall model of ASSIST-IoT Development Life Cycle*

The previous development methods reserved a place for security and testing at the end as an individual and separate procedure. The growing security threats for IoT devices require an active approach to solving security threats prior to the deployment of the software in the production environment. The shift from DevOps to DevSecOps is detailed in the project's publication [13.] as security moves to the left and is part of every phase of the development.

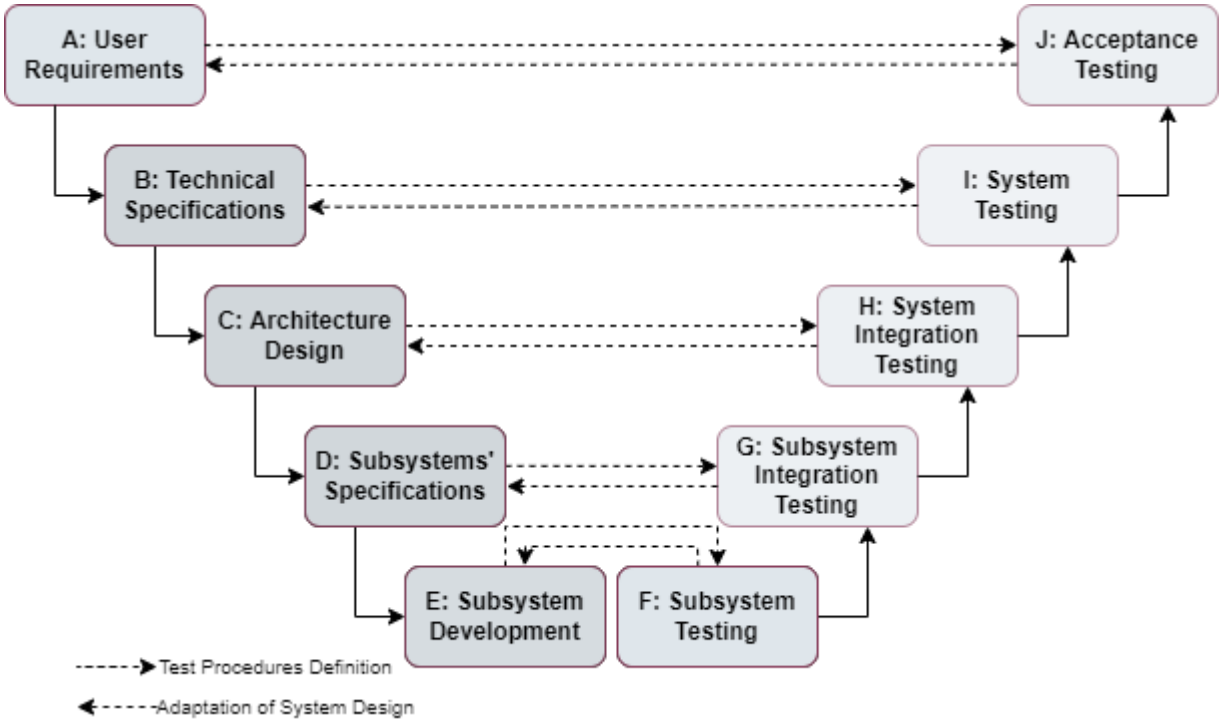



*Figure 7. V-model of ASSIST-IoT Development Life Cycle*

ASSIST-IoT adopts the following phases for executing the DevSecOps methodology: Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor. The plan phase begins before the code development for considering potential threats and is a dynamic process changing within the project's lifecycle. The code phase sets guidelines to be followed and enforced through plug-ins for the Integrated Development Environment (IDE). During the build phase, developers are to solve dependencies issues. The test phase harmonises the hardware and software into one functioning system where unit and system integration security tests are conducted. If the software successfully clears the tests, the release phase has the software packaged in an artifact repository like Gitlab. In the deployment and configuration phases, the application of different environments like staging, preproduction, and production are to be developed to permit the execution of validation and acceptance testing.

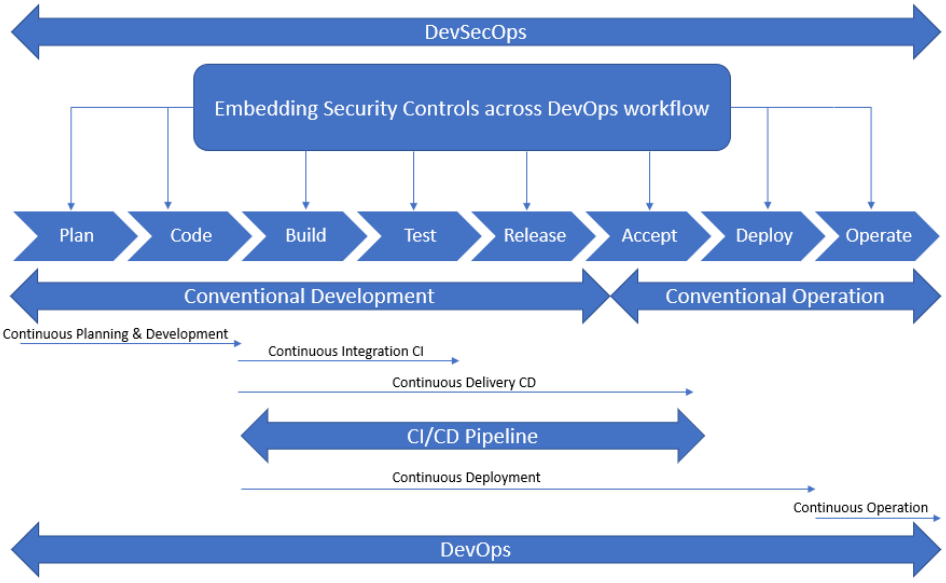



*Figure 8: DevSecOps embedded security control*

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

*Table 2: Software Test & Integration plan*

| Level of Testing | Activities | Test environments | Frequency of testing | Responsible | | |
|---|---|---|---|---|---|---|
| | | | | Writing test cases | Providing test data | Running tests |
| **Unit** | Select test cases<br>Write automated tests cases | Developer environment<br>Continuous Integration Infrastructure | Create test before / while developing<br>Automated tests run continuously when component is built on the CI Infrastructure | Developer | Developer | Component/Unit provider |
| **Functional** | Select test cases according to requirements<br>Prepare demos with test data<br>Run demos | Developer environment<br>CI Infrastructure | Create tests whenever a new functionality is introduced<br>Run tests continuously when adding the functionality to the enabler | Developer<br>Integration team | Developer<br>Integration team | Developer<br>Integration team |
| **Integration** | Select test cases<br>Manage unit dependencies<br>Write automated tests<br>Prepare non automated test cases | CI Infrastructure | Automated tests run continuously when binding enabler's components together<br>Manual testing each time a new component is introduced to the enabler | Developer | Developer<br>Integration team | Developer<br>Integration team |
| **End-to-end** | Design and prepare tests around pilot trials<br>Automate tests and run them<br>Design, prepare, and run manual tests | CI Infrastructure | Automated tests run continuously when all the enablers of a pilot trial are ready<br>Manual tests run whenever a new version of an enabler / component is introduced in the trial | Integration team | Integration team | Integration team |
| **Factory / Site Acceptance** | Define test cases according to the pilot trials | Factory / Pilot Site environment | Tests run on the integrated / production platform | Integration team<br>Pilot site stakeholders | Integration team<br>Pilot site stakeholders | End users / vendors / pilot site stakeholders |

| Level of Testing | Activities | Test environments | Frequency of testing | Responsible | | |
|---|---|---|---|---|---|---|
| | | | | Writing test cases | Providing test data | Running tests |
| | Prepare test data for real time case scenario<br>Run the integration tests of the system<br>Identify the observations and track the issues<br>Acceptance test review | | which will be used, whenever a trial is validated | | | |
| **Performance** | Design test cases for scalability, stress, load, endurance and extreme unlikely scenarios<br>Run the tests along with integration team and pilot site stakeholders<br>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 Pilot site stakeholders |

## 4.2 Testing process in ASSIST-IoT

Unit testing is about automated tests for testing each component's functionality in the enablers' functionality. The developing teams of each partner are to specify the individual tests to run as the responsible developers. Unit tests can be run in each version, and updates of the enablers to realise the DevSecOps methodology.

Unit tests should be implemented on each class and method uniquely. The architectural structure of the project defined by the WP3 actions defines the highest grade of unit testing, as enablers are to be the highest component in granularity for testing. Since unit testing is about testing the smallest testable units, each method, class and enabler's internal API should have its own test case to achieve as quickly as possible bug fixing. Only components that pass all unit tests should be allowed to be committed to the source code repository since they assure the correct behaviour of the unit. An important note is that unit testing will take place before any containerisation or deployment of the code to ensure the system's functionality.

It is also recommended to develop a new unit test for each detected bug, to further improve software quality. After fixing a bug, the commitment should state a bug number in order to automate the tracking of bug fixes.

For functional testing, it is important to draft into a table each test to run to prove the functionality of each enabler. The definition for its functional test will take place after the definition of deliverables D4.2 and D5.3 after consulting each responsible partner. The basic steps for functional testing will be to understand the

functional requirements for the aforementioned deliverables, clearly define the test input and data based on requirements, run the tests on the input to compute the expected outcomes, and finally compare actual and computed expected results. In the Annex, the table will be filled with the tests, criteria, and the verdict for each enabler.

A first step to enabling integration testing is the availability of a testbed to run the developed system. Essentially, machines will be available to replicate the conditions that the pilot sites will have. The review of the work from WP3 and WP7 led the project to the initial specifications for 3 machines of i5 processor, 16GB RAM, and 1TB SSD. The work in WP3 (D3.5 [15.], D3.6 [16.]) provides the functional requirements per pilot site to consider for this testing. The functionalities will be achieved with the combination of multiple enablers running in a system. Essentially, it is the lab test prior to the pilot release where issues on the harmonisation of the different components will come to the surface.

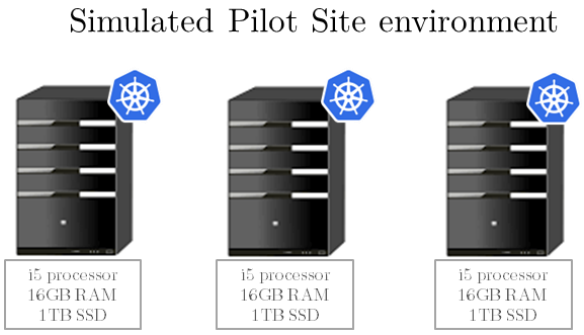



*Figure 9. Test environment to simulate pilot site premises*

End-to-end testing is about testing the functionality and performance of a given trial of ASSIST-IoT. The scenarios for this kind of testing are closely related to WP7 work on the pilots and the project's validation efforts. The proposed architecture and developed components are to be tested in the pilot premises. In D7.2 [17.], each pilot sets several trials to cover the needs and make use of ASSIST-IoT's enablers. The trials are detailed in the aforementioned deliverable, and they are the following:

- Port automation:
  - Tracking assets in the terminal yard
  - Automated CHE cooperation
  - RTG remote control with AR support
- Smart safety of workers:
  - Occupational safety and health monitoring
  - Fall-related incident identification
  - Health and safety inspection support
- Vehicle in-service:
  - Fleet in-service emission verification
- Vehicle exterior condition:
  - Vehicle exterior condition inspection and documentation

Finally, acceptance and performance testing will commence at the same time. The acceptance testing will use as a basis for the WP3 and WP8 work, where KPIs and other measurements are provided. The KPIs and measurements will guide the acceptance testing, as the tests will be formulated in the most appropriate way to

measure them. The performance testing will provide the details of the system performance while operating with the enablers. It is essential for testing the load on the system as the built system refers to IoT devices.

### 4.2.1. Functional Testing of horizontal enablers

This chapter aims to present the functional testing for each enabler. The functional testing is closely related to the deliverables detailing the enablers operations (D4.1 [18.], D4.2 [19.], D5.1 [20.], D5.2 [21.]), where the basic functionalities are described.

**Smart Network and Control Plane**

**Smart Orchestrator Enabler**

*Table 3: Smart Orchestrator's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *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. A Helm repository must have been added previously to complete the test. | Pass/Fail |
| 2 | *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 |
| 3 | *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 |
| 4 | *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 |
| 5 | *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 |
| 6 | *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 |
| 7 | *Add enabler manually* | 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 |
| 8 | *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 |
| 9 | *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 |
| 10 | *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 enabler. | Pass/Fail |
| 11 | *Automatic enabler placement* | An enabler is placed automatically in one of the added K8s clusters depending on the resources available and the placement policy | The enabler is deployed in the correct cluster depending on the placement policy chosen. Three cases will be assessed (after setting up a realistic environment to evaluate it):<br>• Most traffic: the cluster with more data traffic.<br>• Best fit: the cluster with less resources available. | Pass/Fail |

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| | | | • Worst fit: the cluster with more resources available. | |
| 12 | *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 |

## Traffic Classification Enabler

*Table 4:Traffic Classification's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *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 train a model with test samples and will substitute the former model. Error handling (e.g., database not present, bad labelling) will be evaluated. | Pass/Fail |
| 2 | *Classify packet* | The classifier will classify packets according to different classes. | A set of API calls with extracted packet features will be made to test the capability to classify packets. | Pass/Fail |

## Multi-link enabler

*Table 5: Multi-link's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Add interface* | Adds a tunnel interface to be managed by the enabler. | An API call will be made to train a model with test samples and will substitute the former model. Error handling (e.g., database not present, bad labelling) will be evaluated. | Pass/Fail |
| 2 | *Delete interface* | Delete a tunnel interface that was previously added. | The successful deletion of an inserted channel by a public key. | Pass/Fail |
| 3 | *Primary channel transmission* | Data should transmit normally using the primary wireless channel. | Two wireless networks will be considered for the test (WiFi vs 3G/4G/5G), and data packets will travel through the primary one. | Pass/Fail |
| 4 | *Backup channel transmission* | In case the primary channel is down, a backup one should take over. | The primary WiFi channel will be manually deactivated, traffic should swap over the second one in less than 50 ms. | Pass/Fail |
| 5 | *Primary channel recovery* | In case that the primary channel is recovered while in backup transmission, the former should take over | In case that the primary WiFi channel is recovered, traffic should swap back from the backup one to the primary one. | Pass/Fail |

## SD-WAN enabler

*Table 6: SD-WAN's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Add hub cluster* | A hub cluster is provisioned so traffic between edge clusters and from/to Internet travels through it. | The hub cluster is added correctly to the overlay managed by the enabler. | Pass/Fail |
| 2 | *Add edge cluster* | A dedicated IPSec tunnel is established to connect an edge cluster to the SD-WAN. | Two edge clusters will be added to the overlay, so two IPSec tunnels will be established automatically with the hub cluster so that traffic between them travels through them. A traffic sniffer will be used to check this performance (e.g., Wireshark). | Pass/Fail |

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 3 | *Get overlay* | Lists all the resources managed or defined in the overlay (edge/hub clusters belonging, IPSec data, IP ranges, etc.) | An API call will be made to obtain these data, which should include the clusters added in the previous tests. Although not included as tests, IP ranges and IPSec proposals must have been previously provisioned. | Pass/Fail |
| 4 | *Delete edge cluster* | An edge cluster is removed from the overlay of the SD-WAN, and its IPSec tunnel eliminated. | The edge cluster is detached from the hub, and the IPSec tunnel removed. Traffic might travel between edge clusters, but insecure (without tunnel). | Pass/Fail |
| 5 | *Delete hub* | The hub cluster is reconfigured and detached from the SD-WAN. | The hub cluster is removed from the overlay. It can only be done if edge clusters are not connected to it with dedicated IPSec tunnels. | Pass/Fail |

## WAN Acceleration enabler

The following tests are defined to be performed in consecutive order, as the execution of some of them require that previous steps are performed before their execution.

*Table 7: WAN Acceleration's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Get services* | Lists all the operation allowed by the enabler. | Returns the result of an API query, with the set of services allowed (should be firewall, mwan3, IPSec-related). | Pass/Fail |
| 2 | *Get interfaces* | Lists all the available interfaces and their specific information | Returns the result of an API query, with the set of interfaces managed by the enabler. | Pass/Fail |
| 3 | *Enable interfaces* | The interfaces of the system can be enabled (and disabled), so WAN rules can be applied over them or not. | An interface is enabled, and the next tests will affect this and the rest of enabled interfaces. It is considered successful if the following tests apply to the newly enabled interface. | Pass/Fail |
| 4 | *Add WAN policy* | Define how traffic will be routed through the WAN interfaces. | A test WAN policy is introduced via custom resource with the K8s API. This is not working until attached to a “rule”. | Pass/Fail |
| 5 | *Get WAN policies* | Lists the WAN policies added to the enabler. | Returns the result of an API query, with the test WAN policy recently added. | Pass/Fail |
| 6 | *Add WAN rule* | A policy is attached to a rule (i.e., to act when traffic matches a specific port and/or IP source and/or destiny, IP traffic type, or protocol. | A test WAN rule is added via custom resource with the K8s API. The effect of the test policy is evaluated with traffic tools (to define). | Pass/Fail |
| 7 | *Get WAN rules* | Lists the WAN rules added to the enabler. | Returns the result of an API query, with the WAN rule recently added. | Pass/Fail |
| 8 | *Delete WAN rule* | Deletes a test WAN rule from the enabler. | Test #7 is executed to check that this WAN rule is no longer part of the system. | Pass/Fail |
| 9 | *Delete WAN policy* | Deletes a test WAN policy from the enabler. | Test #5 is executed to check that this WAN policy is no longer defined in the system. | Pass/Fail |
| 10 | *Add firewall zone* | Groups one or many interfaces to be source or destination for forwardings, rules and redirects. | A set of interfaces are added to a firewall zone via custom resource with the K8s API. It is considered successful if the following tests apply to the defined zone. | Pass/Fail |
| 11 | *Get firewall zones* | Lists the interfaces belonging to a test zone. | Returns the result of an API query, with the set of interfaces belonging to the test zone. | Pass/Fail |
| 12 | *Add firewall rule* | Adds a firewall rule to a zone (i.e., accept, drop and reject rules for specific ports or hosts). | A test rule is introduced via custom resource with the K8s API. The effect of the test rule is evaluated with traffic tools (to define). | Pass/Fail |
| 13 | *Get firewall rules* | Lists all the firewall rules introduced in the enabler. | Returns the result of an API query, with a JSON containing the previously-added test firewall rule (if added correctly). | Pass/Fail |

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 14 | *Delete firewall rule* | Deletes a test firewall rule from the enabler. | Test #13 is executed to check that this firewall rule is no longer defined in the system. | Pass/Fail |
| 15 | *Delete firewall zone* | Deletes a test firewall zone from the enabler. | Test #11 is executed to check that this firewall zone is no longer defined in the system. Rules cannot be applied to it. | Pass/Fail |

### 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 8: VPN's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *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 connection can be stablished using the generated client (test #7). | Pass/Fail |
| 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 |

## Data management Plane

### Semantic Repository enabler

*Table 9: Semantic Repository's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Add model group* | An empty model group is created. | A model group is created. This request should be rejected if the declared model group namespace already exists. | Pass/Fail |

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 2 | *Get model groups* | Retrieve the list of existing model groups. | An API call is performed, returning a JSON with all existing model groups. | Pass/Fail |
| 3 | *Add model (default)* | A model is added with default options, to a model group. | A model is added to an empty existing model group, 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 model group. | A model is added to a model group, 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 model group* | Retrieve all models with versions under a given model group | An API call is performed, returning a JSON with the list of all models and their metadata, under the given model group. | Pass/Fail |
| 6 | *Get model* | A model is retrieved | An API call is performed, returning a model file, provided, that a model with given group 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 group, 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 model group* | A model group is removed | Remove a model group by name. Removing an existing group should be possible only, if the 'allow removal' parameter is set, and the model group does not contain any models. Otherwise, the request should be rejected. | Pass/Fail |

## Semantic Translation enabler

*Table 10: Semantic Translation'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 |
| 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 | Pass/Fail |

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| | | | request comes. After or before the timeout, the channel should no longer exist. | |
| 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 |

## Semantic Annotation Enabler

*Table 11: Semantic Annotation'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 |
| 10 | *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 |

## Edge Data Broker enabler

*Table 12: Edge Data Broker'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 preconfigured rule* | 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 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. | The consumer receives the alert. | Pass / Fail |

## Long-Term Storage Enabler

*Table 13: Long-Term Storage's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Authorized access* | The LTSE API needs to communicated with the IdM and Authorization enablers in order to confirm the read-and-write permissions of an enabler | The IdM and Authorization enablers provides the access rights of the connected enabler | Pass/Fail |
| 2 | *Manage SQL server* | Create the SQL tables for storing SQL information from authorized enablers via LTSE API | The success of the operation can be checked by exploring the existence of the SQL tables. | Pass/Fail |
| 3 | *Manage noSQL cluster* | Create the noSQL indices for storing corresponding noSQL information from authorized enablers via LTSE API | The success of the operation can be checked by exploring the existence of the noSQL indices. | Pass/Fail |
| 4 | *Ingest Raw SQL Data* | The authenticated enabler sends raw data to the corresponding SQL table of the LTSE via the LTSE API. | The SQL raw data is collected into the corresponding LTSE SQL table. If the enabler is not authorized a denied permission is provided. | Pass/Fail |
| 5 | *Ingest Raw noSQL Data* | The authenticated enabler sends raw data to the corresponding noSQL index of the LTSE via the LTSE API. | The noSQL raw data is collected into the corresponding LTSE noSQL index. If the enabler is not authorized a denied permission is provided. | Pass/Fail |
| 6 | *Retrieves filtered SQL data* | An authenticated enabler requests some filtered SQL data to the LTSE through the LTSE API. | The range of requested data is successfully obtained in XML format. If the enabler is not authorized a denied permission is provided. | Pass/Fail |
| 7 | *Retrieves filtered noSQL data* | An authenticated enabler requests some filtered noSQL data to the LTSE through the LTSE API. | The range of requested data is successfully obtained in JSON format. If the enabler is not authorized a denied permission is provided. | Pass/Fail |

## Application and Services Plane

### Tactile dashboard

*Table 14: Tactile dashboard's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Integration with other ASSIST-IoT enablers* | The tactile dashboards of ASSIST-IoT will be the main User Interface presented to the stakeholders to configure the values of their deployed enablers, and/or to visualize their pilots results or potential notifications/alerts. | The backend component of the different implemented dashboards in each pilot will communicate through rest API messages with the supported APIs of the deployed enablers. | Pass/Fail |

### Business KPI Reporting enabler

*Table 15: Business KPI Reporting's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Integration with the tactile dashboard* | The Business KPI enabler is conceived as a web iFrame that will be embedded within the tactile dashboard. | The Business KPI enabler helm chart must be properly integrated within the tactile dashboard. | Pass/Fail |
| 2 | *Integration with pilots databases for historical data* | The business KPI enabler visualizes in the form of graphs/charts different filtered data extracted from the historical time-series data stored in the pilots repositories | The Business KPI enabler API must be properly connected with the LTSE NoSQL cluster | Pass/Fail |
| 3 | *Integration with pilots gateways for real-time data* | The business KPI enabler visualizes in the form of graphs/charts different filtered data extracted from the real time data collected in the pilots gateways storage | The Business KPI enabler API must be properly connected with the GWEN storage | Pass/Fail |

### Performance and usage diagnosis (PUD) enabler

*Table 16: PUD'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 Prometheus time series database and be accessible through its UI. | Pass / Fail |
| 2 | *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 |

### OpenAPI Management Enabler

*Table 17: OpenAPI Management'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 |
| 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 | *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 |
| 6 | *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 |

## Video Augmentation enabler

*Table 18: Video Augmentation's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *FL repository integration* | The Video Augmentation enabler may use an already stored ML model or obtain new ones from the FL repository | The Video Augmentation API must successfully obtain the requested configuration of an ML model stored in the FL repository | Pass/Fail |
| 2 | *Inference model* | The Video Augmentation enabler provides the ability to perform ML model inference over new cameras deployed in ASSIST-IoT pilots | The already trained ML model is properly inferenced through Video Augmentation API over stored pictures/videos or from streaming real-time RTP messages | Pass/Fail |

## 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 19: MR'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 |

### 4.2.2. Functional Testing of vertical enablers

**Self-enablers**

**Self-healing enabler**

*Table 20: Self-healing's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Hardware testing (RAM/CPU)* | The self-healing enabler provides different methods for monitoring HW resources. | The self-healing UNIX commands implemented over its NodeRed flows must receive the numerical values of the resources' status. | Pass/Fail |
| 2 | *Hardware testing (network)* | The self-healing enabler provides different methods for monitoring device's network interfaces. | The self-healing UNIX commands implemented over its NodeRed flows must receive the status of the network device manager. | Pass/Fail |
| 3 | *Hardware remediation (RAM/CPU/network)* | The self-healing enabler performs remediation actions over the HW which it is deployed and being monitored. | The self-healing UNIX commands implemented over its NodeRed flows must perform the remediation actions over the device's resources. | Pass/Fail |

**Automated Configuration enabler**

*Table 21: Automated Configuration's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Add resource* | The Automated Configuration (AC) enabler should allow adding a new (configurable) resource | A new resource with a provided initial configuration is created. | Pass / Fail |
| 2 | *Modify resource configuration* | The AC enabler should allow updating an existing resource configuration. | An update action to an existing resource configuration is performed and verified via a set of predefined tests. | Pass / Fail |
| 3 | *Delete resource configuration* | The AC enabler should allow removing an existing resource configuration. | A resource delete request is issued to the enabler. The request should be rejected if the resource specified does not exist. | Pass / Fail |
| 4 | *(Re)configure sample test resource* | The AC enabler has to perform non-trivial configuration with fallbacks of the test resource. | The test configuration has to execute a set of predefined integration tests. | Pass / Fail |

**Resource Provisioning enabler**

*Table 22: Resource Provisioning's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Get artifacts* | It returns a list with the enablers and components correctly configured and ready to be optimised. | Correctly configured enablers and components are successfully obtained in JSON format. | Pass / Fail |
| 2 | *Train model* | The training of the model is performed for each component with historical data. | It will be possible to check the training result in the future database with the new trained data and without duplicates. | Pass / Fail |
| 3 | *Get range* | The enabler returns the training range based on both historical and predicted data. | The range of training data is successfully obtained in JSON format. | Pass / Fail |

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 4 | *Update range* | The data range values of historical and predicted data are updated. | The evaluation criterion will be linked to the verification of the change of variables by performing test #3. | Pass / Fail |
| 5 | *Inference* | Data inference is performed on each component of each enabler based on the latest training. | An update is performed on the HorizontalPodAutoscaler object of each managed component. | Pass / Fail |
| 6 | *Select enablers to manage* | Add or remove the enablers or components that you want to apply the inference. | The success of the operation can be checked by performing the inference based on test #5. | Pass / Fail |

### Monitoring and Notifying enabler

*Table 23: Monitoring and Notifying'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 |
| 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 |

### Location Processing enabler

*Table 24: Location Processing's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Add region* | The Location processing (LP) enabler should allow adding a new region. | A new region with a provided initial geometry is created. | Pass / Fail |
| 2 | *Modify region* | The LP enabler should allow updating an existing region geometry. | An update action to an existing region geometry is performed. The request should be rejected if the region specified does not exist. | Pass / Fail |
| 3 | *Delete region* | The LP enabler should allow removing an existing region. | A region delete request is issued to the enabler. The request should be rejected if the region specified does not exist. | Pass / Fail |
| 4 | *Query position(s) wrt sample test region* | The LP enabler has to allow checking if a given location lays within a specified region. | The test configuration has to execute a set of predefined, location specific, integration tests. | Pass / Fail |

## Federated machine learning enablers

### FL Training Collector enabler

*Table 25: FL Training Collector'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 |

## FL Orchestrator

*Table 26: FL Orchestrator's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Web GUI app integration* | The Web GUI application of the ASSIST-IoT FL system will provide a human friendly interface. It will provide to the end user the ability of configuring the FL training graphically. To do so, it should be properly integrated with the FL Orchestrator. | The different action buttons, forms, or edit fields must connect with the FL orchestrator API, which in turn is connected with the FL workflow manager that is in charge of properly configuring the JSON documents distributed with the rest of FL enablers | Pass/Fail |
| 2 | *Integration with FL Repository for retrieving ML models* | The FL Orchestrator needs to be integrated with the FL repository in order to have access to the different ML models supported, as well as to obtain their corresponding model_id field | The FL orchestrator API must successfully connect with the FL repository databases | Pass/Fail |
| 3 | *Integration with FL Training Collector for sending customized training configuration* | The FL Orchestrator needs to be integrated with the FL Training Collector to inform about the new Federated Learning process to be performed | The FL orchestrator API must send in JSON documents, the FL configuration to the FL Training Collector, which will acknowledge about its successful reception. | Pass/Fail |
| 4 | *Integration with FL Local Operations for sending customized training configuration* | The FL Orchestrator needs to be integrated with the FL Local Operations to inform about the new Federated Learning process to be performed | The FL orchestrator API must send in JSON documents, the FL configuration to the FL Local Operations, which will acknowledge about its successful reception. | Pass/Fail |
| 5 | *Get status of FL Training Collector / FL Local Operations* | The FL Orchestrator needs to be aware of the current job status of the FL process by being integrated with the FL Training Collector and the FL Local Operations | The FL orchestrator API must periodically receive in raw text the status of the FL Training Collector and the FL Local Operations (either ON or OFF). | Pass/Fail |
| 6 | *Get finished FL training round* | The FL Orchestrator needs to be aware of the last training round performed by the FL training collector | The FL orchestrator API must periodically receive in raw text (or via a JSON message) an ending notification of a new round from the FL Training Collector. | Pass/Fail |
| 7 | *Get finished FL training process* | The FL Orchestrator needs to be aware of the ending of the FL training from the FL training collector | The FL orchestrator API must receive in raw text (or via a JSON message) an ending notification of FL training from the FL Training Collector. | Pass/Fail |

## FL Repository enabler

*Table 27: FL Repository's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *ML model storage* | ML model is send to FL Repository to be stored. Using the model's id and version the same model is retrieved from the repository. Finally the model is deleted from the repository. | Each API request is correctly handled and a message confirming a successful execution of a requested operation is send in response. | Pass / Fail |
| 2 | *ML algorithm storage* | ML algorithm is send to FL Repository to be stored. Using the algorithm's id and version the same algorithm is retrieved from the repository. Finally the algorithm is deleted from the repository. | Each API request is correctly handled and a message confirming a successful execution of a requested operation is send in responses. | Pass / Fail |
| 3 | *ML training collector algorithm storage* | ML training collector algorithm is send to FL Repository to be stored. Using the training collector algorithm's id and version the same algorithm is retrieved from the repository. Finally the algorithm is deleted from the repository. | Each API request is correctly handled and a message confirming a successful execution of a requested operation is send in responses. | Pass / Fail |

## FL Local Operations enabler

*Table 28: FL Local Operations' 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. | Pass / Fail |
| 2 | *Send model* | FL Local Operations should be able to accept a model identified with a given id and version. | The API request is correctly handled and a message confirming a successful execution of a requested operation (accept model) is send in response. | Pass / Fail |
| 3 | *Request status* | The Local Operations enabler should be able to provide its status. | The API request is correctly handled and in response status information is given. | Pass / Fail |

# Cybersecurity enablers

## Identity Manager enabler

*Table 29: Identity Manager'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<br>Verification can be done in CI/CD pipeline using https://github.com/gauntlt/gauntlt | Pass / Fail |

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 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<br><br>Described and documented in GitLab<br><br>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<br><br>Described and documented in GitLab<br>https://gitlab.assist-iot.eu/wp5/t53/identity-manager/-/blob/main/restenabler/test_restconnector.py | Pass / Fail |

## Authorisation enabler

*Table 30: Authorisation's functional tests*

| Nº | Test | Description | Evaluation criteria | Results |
|---|---|---|---|---|
| 1 | *Ports exposed* | Authorization 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.<br>Port MySQL 3306 and 9000<br>Verification can be done in CI/CD pipeline using https://github.com/gauntlt/gauntlt | Pass / Fail |
| 2 | *API REST AuthServer* | Authorization Server needs to expose REST API | Dynamic Unit Test to verify REST API<br><br>http://<host>:9000/DcAuthzPap/rest/evaluate?resource=a&action=ib&code=c<br><br>The response is a JSON | Pass / Fail |
| 3 | *API REST PAP* | Authorization Server needs to expose REST API for PAP | http://<host>:9000/DcAuthzPap/ | Pass / Fail |

## Cybersecurity Monitoring enabler

*Table 31: Cybersecurity Monitoring'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.<br>Verification can be done in CI/CD pipeline using https://github.com/gauntlt/gauntlt | Pass / Fail |

## Cybersecurity Monitoring Agent enabler

*Table 32: Cybersecurity Monitoring Agent'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 |

## DLT based enablers

### Logging and auditing

*Table 33: Logging and auditing'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 |
| 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 |

### Integrity Verification

*Table 34: Integrity Verification'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 |

### Broker service

*Table 35: Broker service'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 |

### Federated Learning DLT

*Table 36: FL DLT'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 |

## Manageability enablers

### Enabler for registration and status of enablers

*Table 37: Registration and status of enablers' 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 and its operational status is “running”. | 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 |
| 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 |

### Enabler for management of services and enablers’ workflow

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.

### Device Management enablers

*Table 38: Device Management'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 |

## 4.3 Test environment

Generally, the different test phases will be executed in different environments. For example, the environment for unit testing can be the local system where the unit is developed as unit testing is defined by the development team. The below environments are to be considered for the testing phases building towards the validation process of the pilot cases:

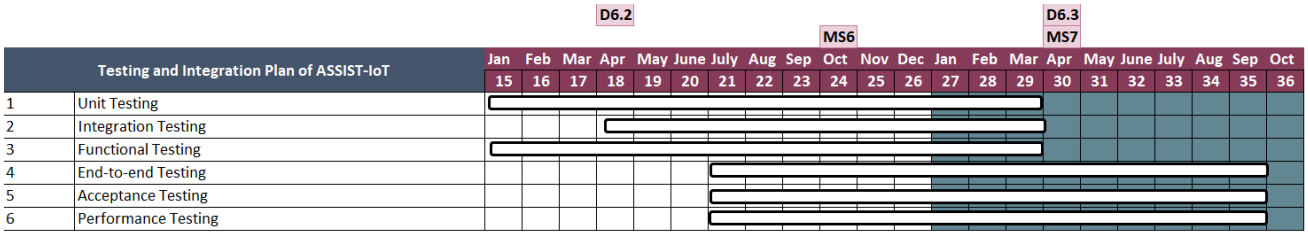
- Local system: Each developers' local machine environment. (unit testing)
- Development premises: Developers' local machines along with Gitlab testing environment. (integration testing)
- Staging: As previously mentioned, three machines provided by CERTH will be available for integration testing. The aim is to provide a staging environment for developers to test their components in a laboratory environment. This environment will present bottlenecks between the developed components prior to the final testing environment, the pilot sites. (integration testing, some end-to-end testing and some performance testing)
- Pilots: In D7.2, there are diagrams for the pilots' implementations that will used for the validation of the developed architecture. (final end-to-end testing, acceptance testing and final performance testing)

## 4.4 Time plan

ASSIST-IoT aims to introduce a reference architecture for the NG-IoT and use it in a human-centric manner. Three different pilot sites are in place to implement and test the developed architecture of the project. As the integration and testing are demanding, a time plan is advised to be followed.

The time plan aims to facilitate the implementation of the DevSecOps methodology. While the contemporary literature [22.], [23.] on the testing phases proposes incremental, step-by-step testing, the DevSecOps philosophy aims to operate within a cycle and include the testing within this cycle. For that reason, the testing phases will coexist for the duration of the project, and the maturity of the development will have to initiate a different testing phase during the progression of the project.

The initial time plan foresees the continuous testing of the developed tools. The testing methods beginning earlier than the rest of the methods are the unit and functional testing, which does not require the deployment of the whole system. As the development of the enablers matures, the integration testing is to follow.




| | 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 10. ASSIST-IoT testing and integration time plan*

Acceptance and performance testing are tests that will be realised in a later stage of the project, while the maturity of the development and integration will be higher. To elaborate more on the time plan, we should mention that unit testing begins with the first code commitment on the Gitlab repositories. It is part of CI/CD and therefore new bugs and issues can come up unexpectedly. All these should be assessed in parallel with functional testing, as the internal components of enablers are responsible for providing the complete

functionality of use cases. Following the testing workflow, integration testing comes next which begins as soon as the units of an enabler are already tested and need to be integrated in an enabler to be seen as a complete application. The main issue here is that integration testing is also part of CI/CD, hence until the enablers are put in a real pilot trial scenario, bugs and issues are continuously arising and have to be dealt with. The procedure continues with end-to-end testing, when enablers responsible for a pilot trial application scenario are ready to be integrated and tested. After ensuring that the requirements are met, we proceed to the UAT/FAT in which we deploy the platform to the pilot premises and apply beta testing by end users and stakeholders. Finally, performance testing as the last part of the testing lifecycle, begins when acceptance testing is finished, the product requirements are met, and we are ready to enter to the phase of maintaining the product, tackle rare scenarios, configure and optimize the product.

# 5 Conclusion / Future Work

The current deliverable aims at establishing the testing and integration strategy for the project. The strategy follows the DevSecOps methodology defines the tools, the procedures, and tests for the project to incorporate security in the deployed architecture. The current tools that contribute to the continuous integration and namely, are GitLab, GitLab Runner, Docker registry, and Kubernetes. The introduced tools are open-source tools enhancing the auditability of the project. More tools can be introduced to grow the capabilities of the pipeline depending on the needs for testing and the demands of the pilot cases.

The test strategy familiarises the readers with the testing methods that are to be in place for the project implementation. Tests are devised to include units ranging in scale, from the smallest ones like functions up to the whole system. The tests are to pinpoint bottlenecks in the code and application prior to the deployment and alleviate the burden on the development team. The deliverable provides details to be followed in each test phase. While a time plan is included, the development and testing follow a circular application.

The update of the current deliverable will be with the conclusion of the project on M30 presenting the final results. As the elapsed time between this version and the final is considerable, updates on the methodology and strategy are to be included within deliverables of other work packages such as WP7 and WP8.

# Appendix 1: Enabler's status

This chapter aims to provide an overview of the enablers defined in published deliverables architectural deliverables (D3.5) and detailed in core enablers (D4.1) and transversal enablers (D5.1). Updated versions of the deliverables are to be available at the same time as this deliverable, so changes and updates are included in D3.6, D4.2, and D5.2.

The following subchapters present the work on each separate enabler in a concise table. The readers can familiarise themselves with the enabler's functionality. The connection with other enablers can help the next step of testing and integration for the project as it can provide the initial points to focus on. Due to the project's phase, the majority of enablers are in the development process with the next stages to be the testing and integration.

**Smart Network and Control**

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| T42E1 | Smart Orchestration enabler | This enabler facilitates the interaction of user interfaces and other enablers with the main components of the MANO framework, namely the Network Function Virtualisation Orchestrator (NFVO) and the Kubernetes clusters, exposing only the required inherent functionalities. In particular, this enabler will control the whole lifecycle of Virtualised Network Functions (VNFs), from their instantiation to their termination, allowing their deployment in any k8s cluster available. | 50% |
| T42E2 | SDN Controller | The SDN Controller is the key element of an SDN-enabled network, being the software that takes over the responsibilities of the control plane from the hardware elements (switches mostly), including monitoring and management of packet flows. | 70% |
| T42E3 | Auto-configurable network enabler | This enabler provides optimised network routing configuration capabilities to the SDN Controller of an ASSIST-IoT ecosystem. This enabler will consist of an application that consumes the northbound APIs of the SDN Controllers to generate a policy that improves the performance of one/many KPIs of the network (e.g., latency). | 50% |
| T42E4 | Traffic classification enabler | The aim of this enabler is to classify network traffic into a number of application classes (video streaming, VoIP, Network control, best effort, OAM, etc.), making use of an AI/ML framework and dedicated algorithms. The traffic classification enabler can be seen as a service of the application layer of the general SDN architecture. | 20% |
| T42E5 | Multi-link enabler | Multi-link wireless network capabilities provide the possibility of sending video streams of data over different Radio Access Networks and different channels in each of them (for instance, regarding cellular, using more than 1 connection). Besides, it should provide reliability mechanisms: in case one channel is down, the signal cannot be lost or at least it should be recovered almost in real-time. | 15% |

| | | | |
|---|---|---|---|
| T42E6 | SD-WAN enabler | The objective of this enabler is to provide access between nodes from different sites based on SD-WAN technology. | 0% |
| T42E7 | WAN acceleration enabler | This enabler aims at increasing the efficiency of data transfer in Wide Area Network. This enabler will contain a set of independent, standalone VNFs with that purpose. These functions can be either chained (so data that requires of different techniques travels through the different functions) or selected for specific purposes. | 0% |
| T42E8 | VPN enabler | This enabler will facilitate the access to a node or device from a different network to the site's private network using a public network (e.g., the Internet) or a non-trusted private network. | 50% |

**Data Management**

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| T44E1 | Semantic repository enabler | This enabler offers a "Nexus" for data models and ontologies, that can be uploaded in different file formats, and served to users with relevant documentation. This enabler is aimed to support files that describe data models or support data transformations, such as ontologies, schema files, semantic alignment files etc. | 70% |
| T44E2 | Semantic translation enabler | Semantic Translation enabler offers a configurable service to change the contents of semantically annotated data in accordance with translation rules – so-called "alignments", or alignment files. | 50% |
| T44E3 | Semantic annotation enabler | This enabler offers a syntactic transformation service, that annotates data in various formats and lifts it into RDF. The full list of formats is yet to be decided and the first version will support JSON. | 70% |
| T44E7 | Edge data broker enabler | This enabler enables the efficient management of data demand and data supply from/to the Edge Nodes. It optimally distributes data where it is needed for application, services, and further analysis. | 50% |
| T44E8 | Long-term data storage enabler | The role of this enabler is to serve as a secure and resilient storage, offering different storage sizes and individual storage space for other enablers (which could request back when they are being initialising in Kubernetes pods). It also guarantees that the data will be kept safe, in face of various kinds of unauthorised access requests, or hardware failures, by only allowing access to the data once the Identity Manager and the Authorisation enablers have confirmed their access rights. | 65% |

**Application and Services**

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| T44E1 | Tactile Dashboard enabler | The Tactile Dashboard enabler has the capability of representing data stored in the ASSIST-IoT pilots, through meaningful combined visualisations in real-time. It also provides (aggregates and homogenises) all the User Interfaces for the configuration of the different ASSIST-IoT enablers, and associated components. | 75% |
| T44E2 | Business KPI reporting enabler | This enabler will illustrate valuable KPIs within Graphical User Interfaces embedded into the tactile dashboard. It will facilitate the visualisation and combination of charts, tables, maps, and other visualisation graphs in order to search for hidden insights. | 80% |
| T44E3 | Performance and usage diagnosis (PUD) enabler | PUD enabler aim at collecting performance metrics from monitored targets by scraping metrics HTTP endpoints on them and highlighting potential problems in the ASSIST-IoT platform so that it could autonomously act in accordance or notify to the platform administrator to fine-tuning machine resources. | 50% |
| T44E4 | OpenAPI management enabler | The OpenAPI management enabler will be an API Manager that allows enablers that publish their APIs, to monitor the interfaces lifecycles and also make sure that the needs of external third parties (including granted open callers), as well as applications that are using the APIs, are being met. | 35% |
| T44E5 | Video Augmentation enabler | This enabler receives data (mainly images or video streams) captured either from ASSIST-IoT Edge nodes, or from ASSIST-IoT databases, and by means of Machine Learning Computer Vision functionalities, it provides object detection/recognition of particular end-user assets (e.g., cargo containers, cars' damages). | 40% |
| T44E6 | MR enabler | The MR enabler receives data and transforms it into a format suitable for visualisation through head-mounted MR devices. Data, which may come from long-term storage or real-time data streams, are requested according to their relevance to the user. | 70% (1st release is ready, but the integration with the other enablers is not developed yet) |

**Self-***

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| SELF11 | Self-healing device enabler | This enabler aims at providing to IoT devices with the capabilities of actively attempting to recover themselves from abnormal states, based on a pre-established routines schedule. Hence, it should not require high computation capabilities in order to be deployed on any customizable device. | 80% |
| SELF12 | Resource provisioning enabler | This enabler will be able to horizontally scale (up or down) the resources devoted to a specific enabler (inside a node) in a dynamic fashion. | 50% |
| SELF13 | Monitoring and notifying enabler | This is an enabler responsible for monitoring the uninterrupted functionality of devices and notifying in case of malfunction incidents. Specifically, it has to ensure the departure of data, the arrival, the validity and its own self-monitoring functionality. | 50% |
| SELF14 | Geo(Localisation) enabler | To solve challenges of pilots we need to localize physical objects (containers in ports, workers on construction sites), some devices should be aware of their position in relation to each other (aligning cranes and tractors). This enabler will provide that functionality and will be directly used in pilot usages varying from coordinating heavy machinery in port to locating workers on construction site. This enabler will be a basic building block providing Self-awareness (about the location of devices) to the ASSIST-IoT deployment | 10% |
| SELF15 | Automated configuration enabler | This enabler aims to keep heterogeneous devices and services synchronised with their configurations. Users can update configuration and define its fallback versions in case of errors. Self-* component will detect if fallback configuration should be used. Self-* component will be responsible for reacting to changes in the environment and updating configuration as necessary/required. | 20% |

**Cross-context Federated Machine Learning**

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| T52E1 | FL Orchestrator | The FL orchestrator is responsible of specifying details of FL workflow(s)/pipeline(s). This includes FL job scheduling, managing the FL life cycle, selecting, and delivering the initial version(s) of the shared algorithm, as well as modules used in various stages of the process, such as training stopping criteria. Finally, it can specify ways of handling different "error conditions" that may occur during the FL process. | 50% |
| T52E2 | FL Training Collector | The FL training collector will consist of two components: (i) the combiner responsible of providing updates with respect to the shared averaged model, and (ii) the I/O component which will | 75% |

| | | | |
|---|---|---|---|
| | | carry out the input and output communications of the enabler. | |
| T52E3 | FL Repository | This repository will store (and deliver upon request/need) the ML algorithms or ML models. The FL repository will consist of four main components: ML Algorithms libraries (that gathers ML algorithms without involving any modelling associated with a particular training data set), ML models libraries (which have been already trained with a particular data set), Collectors (averaging algorithms to be used on the FL training process), and Auxiliary component (for any needed additional module, such as privacy mechanisms or stopping criteria). | 70% |
| T52E4 | FL Local Operations | The FL Local Operation enabler will consist of four components: Local data transformer component (that will be in charge of guaranteeing that data is appropriately formatted for the FL model in use), Local Model training component, Local Model inference component, and Communication component (to enable in and out communications between involved local parties and FL orchestrator and FL collector). | 50% |

**Cybersecurity components**

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| T53E1 | Authorization Enabler | The authorisation server offers a decision-making service based on XACML policies. It has different modules that interact and can be deployed independently such as, PEP (Policy Enforcement Point), PAP (Policy Administration Point), PIP (Policy Information Point) and PDP (Policy Decision Point). | 50% (Authorization Server mvp development completed. docker compose completed) |
| T53E2 | Identity Manager Enabler | This enabler will facilitate the use of OAuth2 protocol and will offer a federated identification service where service requester and provider will be able to establish a trusted relation without previously knowing each other. This way a secure identification process is completed without the service provider having received the requester credentials. | 50% (API development completed. Pending interface requirements. IdM basic deployment completed. Pending definition of usecase needs and architecture to complete integration.) |
| T53E3 | Cybersecurity monitoring enabler | This enabler provides security awareness and visibility and infrastructure monitoring. Having raw data as input, the enabler will set a series of processing steps that will enable the discovery of cybersecurity threats, going through a sequence step: (i) collecting, parsing, and normalizing input events, (ii) enriching normalized events, (iii) correlating events for detecting cybersecurity threats. | 70% (docker-compose orchestration tested on a virtual environment, two use cases on Windows and Linux operating systems, monitoring all the agents, the status and security of the hosts.) |

| | | | |
|---|---|---|---|
| T53E4 | Cybersecurity monitoring agent enabler | This enabler performs functions of an endpoint detection and response system, monitoring and collecting activity from end points that could indicate a threat. Security agent runs at a host-level, combining anomaly and signature-based technologies to detect intrusions or software misuse. | 70% (docker-compose orchestration, tested on a virtual environment two use cases on Windows and Linux operating systems, agents located on hosts sending status and security to the wazuh-server) |

**DLT decentralized infrastructure for privacy, trust, and interoperability**

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| T54E1 | Logging and auditing | This enabler will log critical actions that happen during the data exchange between ASSIST-IoT stakeholders to allow for transparency, auditing, non-repudiation, and accountability of actions during the data exchange. | 75% |
| T54E2 | Data integrity verification | This enabler will provide DLT-based data integrity verification mechanisms that allow data consumers to verify the integrity of the exchanged data. | 50% |
| T54E3 | Distributed broker service | This enabler will provide secured data sharing mechanisms as regards the data exchange between different heterogeneous IoT devices belonging to various edge domains and/or between different enablers of the architecture. | 50% |
| T54E4 | DLT-based Federated Learning | This enabler will foster the use of DLT-related components to exchange the local, on-device models (or model gradients) in a decentralised way avoiding single points of failure acting as a component to manage AI contextual information in an immutable form and avoiding alteration as well to the data. | 50% |

**Manageability and control**

| Enabler ID | Enabler Name | Enabler Description | Development Status |
|---|---|---|---|
| T54E1 | Management of enablers existence in a deployment | This enabler will log critical actions that happen during the data exchange between ASSIST-IoT stakeholders to allow for transparency, auditing, non-repudiation, and accountability of actions during the data exchange. | 50% |
| T54E2 | Management of service and enablers | This enabler will provide DLT-based data integrity verification mechanisms that allow data consumers to verify the integrity of the exchanged data. | 50% |
| T54E3 | Management of devices in an | This enabler will provide secured data sharing mechanisms as regards the data exchange between | 65% |

| | | | |
|---|---|---|---|
| | ASSIST-IoT deployment | different heterogeneous IoT devices belonging to various edge domains and/or between different enablers of the architecture. | |

# References


[1.] ASSIST-IoT (2022). D6.4: Release and Distribution Plan. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[2.] ASSIST-IoT (2022). D6.5: 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.] Gitlab website. https://about.gitlab.com/

[5.] 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/

[6.] Gitlab. Gitlab Runner. https://docs.gitlab.com/runner/

[7.] Gitlab. Gitlab Registry. https://docs.docker.com/registry/

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

[9.] Koomen, T., & Pol, M. (1999). Test Process Improvement: A practical step-by-step guide to structured testing. Addison-Wesley Longman Publishing Co., Inc..

[10.] Runeson, P. (2006). A survey of unit testing practices. IEEE software, 23(4), 22-29.

[11.] Vokolos, F. I., & Weyuker, E. J. (1998, October). Performance testing of software systems. In Proceedings of the 1st International Workshop on Software and Performance (pp. 80-87).

[12.] 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.

[13.] 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.

[14.] Kumar, R., & Goyal, R. (2020). Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC). Computers & Security, 97, 101967.

[15.] ASSIST-IoT (2021). D3.5: ASSIST-IoT Architecture Definition. 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.] ASSIST-IoT (2021). D4.1: Initial Core Enablers Specification. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[19.] ASSIST-IoT (2022). D4.2: Core Enablers Specification and Implementation. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[20.] ASSIST-IoT (2021). D5.1: Software Structure and Preliminary Design. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[21.] ASSIST-IoT (2022). D5.2: Traversal Enablers Development Preliminary Version. Deliverable of the Horizon-2020 ASSIST-IoT project, Grant Agreement No. 957258.

[22.] 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

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