Deliverable D5.4

MEDINA integrated solution-v2

Editor(s):	Debora Benedetto, Claudio Caimi, Ahmed Ibrahim, Claudia Zago
Responsible Partner:	Hewlett Packard Italiana, SRL (HPE)
Status-Version:	Final – v1.0
Date:	31.01.2023
Distribution level (CO, PU):	PU

Project Number:	952633
Project Title:	MEDINA

Title of Deliverable:	MEDINA integrated solution-v2
Due Date of Delivery to the EC	31.01.2023

Workpackage responsible for the Deliverable:	WP5 - MEDINA Framework Integration
Editor(s):	Debora Benedetto, Claudio Caimi, Ahmed Ibrahim, Claudia Zago (HPE)
Contributor(s):	TECNALIA, Bosch, CNR, Fabasoft, FhG, XLAB
Reviewer(s):	Juncal Alonso (TECNALIA) Cristina Martinez (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3, WP4, WP6

Abstract:	This deliverable will integrate all the components developed by the other technical WPs in the MEDINA Framework. Different versions of the solution will be provided following an incremental approach. The first version will be an initial prototype with the core functionalities implemented (at M15); the second version (at M27) will augment these functionalities taking into consideration the feedback coming for the use cases and the final version (M33) will include corrections and feedback coming from the implementation of the use cases. The software will be accompanied by a Technical Specification Report. This set of deliverables is the result of Task 5.3.
Keyword List:	Architecture, Workflows, Components Integration, CI/CD, Integrated UI
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	10.11.2022	Initial TOC	HPE, HPE CDS
v0.2	20.11.2022	Added Test Bed environment	HPE, HPE CDS
v0.3	28.11.2022	Added Evidence Collection	FhG
v0.4	29.11.2022	Added Hardware Infrastructure	TECNALIA
v0.5	29.11.2022	Added Catalogue of Controls and Metrics	TECNALIA
v0.6	30.11.2022	Added RAOF section	CNR
v0.7	30.11.2022	Added NL2CNL Translator and DSL Mapper	CNR
v0.8	30.11.2022	Added details for organizational evidence gathering	Fabasoft
v0.9	30.11.2022	Added details for NL2CNL Translator	Michela Fazzolari CNR
v0.91	03.12.2022	Added Life Cycle Manager and Orchestrator and Databases	FhG
v0.92	06.12.2022	Added Wazuh, VAT, CCE	XLAB
v0.93	06.12.2022	Added SSI and Trustworthiness System	TECNALIA
v0.94	07.12.2022	Added Workflows and Workflows Appendix	TECNALIA, BOSCH
v0.95	16.12.2022	Added Integrated UI, CNL Editor and CI/CD solution	HPE, HPE CDS
v0.96	10.01.2023	Merged all the contribution, fixed references and figures	HPE, HPE CDS
v0.97	12.01.2023	Increase RAOF and block#4	CNR, FhG
v0.98	25.01.2023	Internal Review	TECNALIA
v0.99	30.01.2023	Addressed all comments from internal review	HPE, HPE CDS
v1.0	31.01.2023	Ready for submission	Cristina Martínez (TECNALIA)

Table of contents

List of tables

List of figures

Terms and Abbreviations

AMOE	Assessment and management of organizational evidence
API	Application Programming Interface
CAB	Conformity Assessment Body
CCE	Continuous Certification Evaluation
CCD	Company Compliance Dashboard
CCE	Continuous Certification Evaluation
CI/CD	Continuous Integration / Continuous Deployment
CISO	Chief Information Security Officer
CNL	Controlled Natural Language
CSA or EU CSA	Coordination and Support Action
CSP	Cloud Service Provider
DLT	Distributed Ledger Technologies
DoA	Description of Action
DSL	Domain Specific Language
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
gRPC	Google Remote Procedure Call
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure As A Service
IT	Information Technologies
KPI	Key Performance Indicator
KR	Key Result
LCM	Life Cycle Manager
NCCA	National Cybersecurity Certification Authority
NL	Natural Language
NL2CNL	Natural Language To Controlled Natural Language
NLP	Natural Language Processing
OWASP	Open Web Application Security Project
PaaS	Platform As A Service
RAM	Random Access Memory
RAOF	Risk Assessment and Optimisation Framework
RBAC	Role Based Access Control
REO	Requirements and Obligations
REST	Representational State Transfer
SATRA	Self-Assessment Tool for Risk Analysis
SCA	Software Composition Analysis
SPDX	Software Package Data Exchange
SSH	Secure Shell
SSI	Self-Sovereign Identity
SSL	Secure Sockets Layer
SSO	Single Sign-On
SW	Software
ToE	Target of Evaluation
ToC	Target of Certification

TOM	Technical and Organizational Measure
TRL	Technology Readiness Level
UC	Use Case
UI	User Interface
VAT	Vulnerability Assessment Tools
WF	WorkFlow
VM	Virtual Machine

Executive Summary

This document is the second version of the D5.3 [1], that points out the result of the task 5.3 in M27 (January 2023). The goal of this second version is to have a more stable environment and a more automated solution for the MEDINA Framework using the CI/CD approach.

In this deliverable we present a second version of the MEDINA integrated solution with increased functionalities compared to the initial prototype in M15, and also taking into consideration the feedback coming from the evaluation in the two MEDINA use cases. The document shows how some of the main objectives of the work package 5 are achieved in relation to the maintenance of the SecDevOps infrastructure for MEDINA and the support of the continuous integration with dedicated session, workshops and webinars.

The document reports the same structure of D5.3, highlighting updates or changes in each section, and placing the unchanged parts in the Appendix. First, it recapitulates the current state of the Test Bed environment with hardware and operating details, and the methodology adopted throughout the integration phase of the components in the MEDINA Framework exploiting webinars and demos. An overview of the entire integrated environment including the Kubernetes cluster and the CI/CD infrastructure is provided. The document then goes deep into the description of MEDINA CI/CD implemented solution, how it supports the automation of the processes with the pipelines and their stages with a focus on security aspects. Compared to the previous version, the seven workflows are presented in a new view based on the roles of the users in MEDINA that has been agreed by the consortium for every component involved in each workflow. For each of the eight building blocks that compose the MEDINA architecture their current status and their published APIs are reported. The last part of the document is dedicated to the MEDINA Integrated User Interface, with updates on its technical implementation and usage.

The next version of this deliverable published in M33 will provide the final MEDINA integrated solution including corrections and feedback from the implementation of the use cases.

1 Introduction

This section includes an overview of the context of the deliverable, how it is structured and the updates respect to the previous deliverable D5.3 [1].

1.1 About this deliverable

As stated in the "Introduction" section of D5.2 [2], WP5 "MEDINA framework Integration" has five deliverables that can be divided in two parallel series:

- Those that define the MEDINA framework in detail (D5.1 [3] and D5.2 [2])
- Those that describe the developed solution (D5.3 [1], D5.4 and D5.5).

This deliverable is the second version of the three deliverables of WP5 dedicated to the *developed* "MEDINA integrated solution". It reports about the current status and the advancements achieved on the integration of the MEDINA components and is the result of task T5.3 "System Continuous Integration and Optimization".

Since this is a self-contained document, you can find here the description of the integration strategy and implementation adopted during the whole period, although the improvements introduced in the last year, from M15 to M27, have been highlighted. Further details about the updates with respect to D5.3 can be found in Section 1.3.

The document starts by describing the details of the hardware infrastructure provided to set up the Test Bed environment and how this environment is implemented and used. The Test Bed environment hosts the MEDINA components, further details about its installation and configuration can be found in the *APPENDIX A: Operating Environment*. Once the Test Bed environment has been set up, partners can release their components and the following sections describe the methodology adopted to achieve this integration. Finally, current status of the MEDINA framework release and the integration of its component is explained.

Secondly, the document describes the overall design of the CI/CD solution that has been put in place to support the development and integration activities of the MEDINA Framework. This solution foresees three pipelines of build, deploy and security to perform the automation of the integration component.

Thirdly, the document presents the workflows used by the Use Cases to test the correct behaviour of the MEDINA framework. The workflows have minor updates and are described in detail in the *APPENDIX D: Generic Architectural Workflows*. In this period, partners have focused on the introduction of the user's roles point of view, implementing the authorization and filtering strategies in the components.

Fourthly, the document presents an overview of the implementation status of each component, explaining the interaction with the other MEDINA modules and providing brief details on the component interface (if any). One important goal achieved in M27 is the introduction of new connections between MEDINA components.

Finally, the document includes the description of the MEDINA Integrated User Interface component, which is the entry point for the user to access to the MEDINA framework.

A third release of this deliverable is foreseen in M33 (July 2023). It will describe the final infrastructure and components integration, and will leverage the information received from the implementation of the Use Cases as feedback to revise the solution.

1.2 Document structure

The rest of the document is structured as follows:

Section 2 presents the Test Bed Environment, describing its configuration and the hardware infrastructure provided, the description of the methodology adopted for the component integration through the "Keycloak", "Authorization and Filtering" and "CI/CD" webinars, and the current status of the integration of components. It then describes the implementation and strategy adopted for the CI/CD solution.

Section 3 describes the generic workflows based on seven example scenarios with related architectural components. These workflows are described from the authorization and filtering point of view and are presented from the user's role and permissions perspective.

Section 4 presents the MEDINA Framework components. There is a sub section for each block describing all components that belong to it. Each component is presented with an overview of its scope in MEDINA, its implementation status and its integration with the other MEDINA components. If available, its graphical interface is also described.

Section 5 is dedicated to the MEDINA Integrated User Interface component, which is the component implemented in Work Package 5.

Finally, Section 6 reports the conclusions.

The Appendices sections are dedicated to topics that have not changed much from the previous deliverable or are too extensive to be included in the main sections of the document. They are structured as follows:

- *APPENDIX A: Operating Environment,* describes the installation and configuration of the Kubernetes cluster into the Test Bed environment and the final results achieved.
- *APPENDIX B: Docker and Kubernetes Webinar with Sample Component Integration example,* describes the webinar organized for the explanation of the main aspects and operations of Docker and Kubernetes and the demonstration through a demo example on how manually release the components into the Test Bed environment.
- *APPENDIX C: First integration workshop,* describes the workshop held to complete the first release of the MEDINA framework in the "dev" environment and the status of component integration achieved.
- *APPENDIX D: Generic Architectural Workflows,* describes the workflows in detail, going step-by-step through the iterations between architectural components and the generic role(s) being involved.
- *APPENDIX E: Published APIs,* describes the REST API exposed by the components, divided into a section dedicated to each of them.

1.3 Updates from D5.3

This deliverable evolves from D5.3 [1], so much of its content is common to that included in the previous document, with the ultimate goal of providing a self-contained deliverable that facilitates the reader´s understanding. To simplify the tracking of progress and updates with respect to the previous version of the deliverable (D5.3), Table 1 shows a brief summary of the changes and additions made in each of the sections.

Table 1. Overview of deliverable updates with respect to D5.3

Section	Change
2	The Hardware Infrastructure provided has been updated to better suit the components integration requirements. More steps of the integration methodology have been completed and new webinars have been released. The current integration status and the point-to-point connections have been updated. Finally, the CI/CD pipelines are now implemented by all the partners and the final status is shown.
3	This section is evolved by introducing the MEDINA user roles and presenting the workflows from a role point of view.
4	This section contains an update of the description of the MEDINA components and includes the description of two new implemented components SSI and AMOE.
5	Main changes here are in the "components description" and "user interface structure" sections, since more components have been integrated and it has implemented the possibility to login with the Bosch Use Case external identity provider.
Appendix A	No changes here.
Appendix B	No changes here.
Appendix C	No changes here.
Appendix D	This section contains small changes to the workflow steps, which are largely the same as in the previous deliverable.
Appendix E	This section contains the REST APIs of the *Self-Sovereign Identity* and AMOE components. The REST APIs of the other components have updates.

2 MEDINA Test Bed and Secure DevOps infrastructure

This section presents the configuration of the Test Bed environment and the hardware infrastructure used for its installation The organization of the Test Bed environment has not changed, while the hardware infrastructure has undergone some updates.

We also describe here the methodology followed to achieve the second release of the MEDINA framework, giving details about the new webinars held to help partners during this process and the situation at M27 of the current state of the integration of components and the point-to-point connections.

2.1 Test Bed environment

The Test Bed environment is the environment where the MEDINA Framework is delivery on to test and verify all the functionalities.

The main changes introduced during this period are the hardware infrastructure which has been reorganized to better adapt to component requirements and the "Production" environment hosted by the Use Cases, which has been replaced by a "Validation" environment.

In order to have a self-contained document, we are going to resume here the configuration of the Test Bed Environment.

As described in *APPENDIX A: Operating Environment*, the Test Bed environment was installed and configured from scratch and it consists of a three nodes Kubernetes [4] cluster with two different, independent and isolated virtual environments:

- **Development**: is used by developers for testing their modules without fear of bugs or errors. This environment does not affect the end users and is used to improve the code of the MEDINA micro-services before deploying them to the Test environment.
- **Test**: the main purpose here is to ensure that all the updates made on the different modules work as expected. This environment, that is more stable than the development environment, is used by developers for integration testing and by Use Case owners for the validation and quality assessment of MEDINA components.

All the micro-services in the Test Bed environment are containerized and communicate with each other via a RESTful API over a secure HTTPS protocol.

Since the Use Cases, Bosch and Fabasoft are validating the components released in the "Test" environment, they are hosting a "Validation" environment that will replace the former "Production" environment, as described in deliverable D5.2 [2].

2.1.1 Hardware Infrastructure

This section describes the list of the hardware equipment used to setup the Development and Test environments. These environments run on Virtual Machines (VM) hosted by TECNALIA and based on Ubuntu OS 20.04. The domain for all the machines is *medina.esilab.org*. The access to the virtual machines is provided via SSH (Secure Shell) protocol, using digital certificates.

The Development and Test Environments are implemented on a 3-node Kubernetes cluster that virtualizes both environments, making them independent and isolated (see Figure 1). These environments run the MEDINA micro-services in containers.

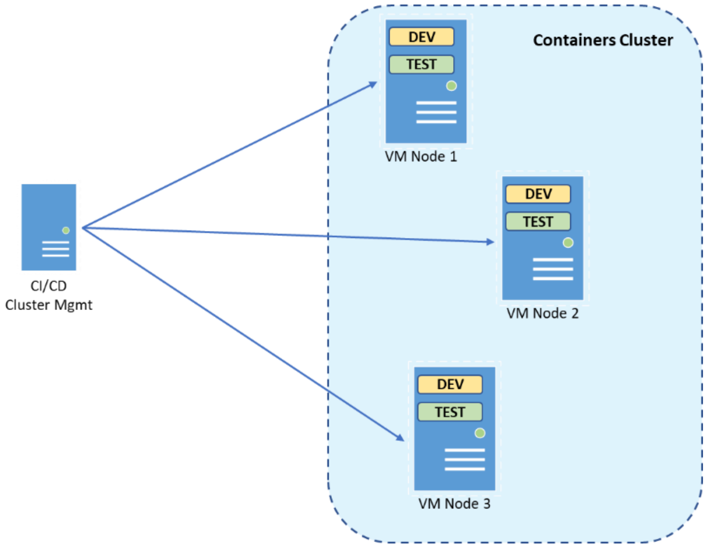


Figure 1. Kubernetes cluster on MEDINA infrastructure

A dedicated VM hosts the CI/CD orchestration engine, the tools that support the CI/CD processes, and the Kubernetes cluster management. Its current resource status is as follows:

- RAM: 16 GB
- Cores: 4
- Hard Disk: 400 GB

The CI/CD is reachable at: ***cicd.medina.esilab.org***.

The three nodes for the Kubernetes cluster (**k8s00, k8s01, k8s02.medina.esilab.org**) share the same specifications:

- RAM: 16 GB
- Cores: 8
- Hard Disk: 200 GB + 200 GB

The 200 GB of storage of each node are organized as a distributed filesystem for data persistent layer. The Kubernetes cluster offers 200 GB of storage, and the data is duplicated among the three nodes.

An additional VM is provided for the Wazuh and VAT tools, in order to produce fake data for the MEDINA Framework. The specifications are:

- RAM: 8 GB
- Cores: 4
- Hard Disk: 60 GB
- OS: Ubuntu 20.04

2.1.2 Components Integration Methodology

Once the Test Bed environment has been properly configured and all the necessary installations have been performed, the next step is to deploy all the component in the cluster.

This section describes the methodology adopted to perform component integration, which has not changed with respect to D5.3 [1], and reports on the progress achieved during this period.

We describe here the new delivery of webinars to enhance the integration and the current status of component connection.

In order to better organize the work of the integration we adopted the following methodology which presents the actions to be taken until the complete release of the MEDINA framework:

1. Each component must be available on the internal private GitLab repository
2. Each component must be containerized into a docker image, the docker image must be available on the internal private docker registry Artifactory
3. Deployment of each component into the development environment in the MEDINA Kubernetes cluster, named "dev"
4. Standalone tests to check each component have been correctly deployed in the development environment
5. Point to point tests for the communication in pairs of the components in the development environment
6. Test end to end in the development environment verifying that the workflows described in Section 3 have been correctly implemented
7. Deployment of the stable version of each component in the test environment in the MEDINA Kubernetes cluster, named "test"
8. Standalone tests to check each component has been correctly deployed in the test environment
9. Point to point tests for the communication in pairs of the components in the test environment
10. Test end to end in the test environment verifying that the workflows described in Section 3 have been correctly implemented.

This methodology is implemented through two instruments: workshops and webinars. The overall integration consists of three rounds: M15, M27 and M33. The webinars are recorded and shared with all partners in the Fabasoft cloud, in a specific folder named "TECHNICAL WEBINARS" (see Figure 2). This allows partners to view them again whenever they need to.

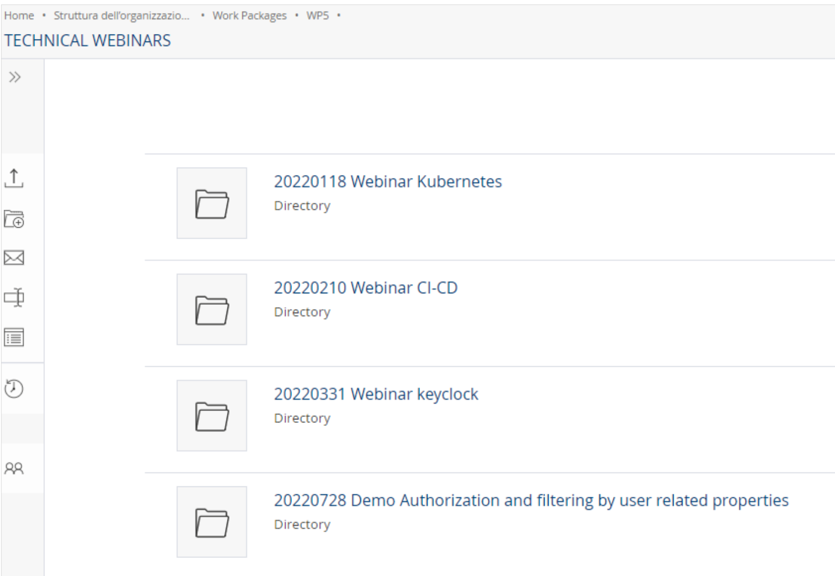


Figure 2. Technical Webinars

During the first round (M15) HPE coordinated the integration of components, which was done manually by each partner. To support this, we delivered a webinar and a workshop. During the webinar we illustrated the main concepts and functionalities of Docker and Kubernetes, as reported in *APPENDIX B: Docker and Kubernetes Webinar with Sample Component Integration example*. During the workshop (see *APPENDIX C: First integration workshop*) we supported the partners for the implementation of the first five actions of the methodology: integration in GitLab, build and push of the docker images into Artifactory, and deployment and tests in the development environment of the MEDINA Kubernetes cluster.

During this second round (M27), we have delivered three webinars and we collaborated with the Use Cases for the validation of the workflows described in Section 3. The integration of the components has been automated and we have released the first stable version of MEDINA framework in the “test” environment. In addition, partners can now automatically release their components and we are adopting a continuous integration strategy by checking during the biweekly Work Package 5 meetings the status of the connection between components.

The integration status of each component and the advancements of the methodology actions are tracked using a spreadsheet available in the Fabasoft shared repository, which is reviewed and updated during the WP5 regular meetings.

The following sections present the description of the webinars conducted in the second round.

2.1.2.1 Keycloak Webinar

Keycloak [5] is an open-source identity and access management tool. It supports multiple standards, the one used in MEDINA is OpenID. Its role in MEDINA is to act as source of truth for identity and to provide login UI. The Keycloak server is reachable, for example, for the DEV environment at this URL: https://catalogue-keycloak-dev.k8s.medina.esilab.org/auth. Every microservice client uses a Keycloak adapter in order to communicate with the Keycloak server.

The Keycloak webinar aims to help partners with their microfrontend configuration. It is divided in two parts. The first one describes theoretically how Keycloak works and the flow it covers when a user initiates a request: the result is the token containing the user’s information for authentication and authorization. The second part shows a demo with a SpringBoot application for the configuration of a Keycloak adapter and the configuration on Keycloak server.

2.1.2.1 Authorization and Filtering Webinar

This webinar consists of a demonstration about the topics of Authorization and Filtering in Keycloak for MEDINA. The first topic deals with the configuration in Keycloak of the Composite Roles used by each component to give access permissions to endpoints within a component. In Keycloak it is possible to manage users and roles. For example, a user without any role assigned cannot see anything in the UI, to grant permission it is necessary to define roles. These roles are defined within the Client (microfrontend) and are only available to this Client.

The second topic is addressed by using the user-related properties obtained from the token used for authentication. These properties correspond to the token fields “cloudserviceproviderid” and “cloudserviceid” which are used to restrict the visibility of the provider (Fabasoft or Bosch) and the resources the user is interested in.

2.1.2.2 CI/CD Webinar

This webinar is focused on Continuous Integration and Continuous Delivery. It can be considered a second part of the previous webinar dedicated to the integration with Kubernetes cluster. The webinar is structured by presenting first the CI/CD environment already setup for MEDINA, then the ad-hoc pipelines developed and finally a live demo with a sample project called “springswagger-template” has been shown. This example provides guidelines for partners to create their own pipelines. This webinar it also mentioned throughout Section 2.2.2 with reference to the demo to explain how to set up the Jenkins Seed Job for pipeline creation and what the pipelines do.

2.1.2.3 Second Round - Continuous integration

During the first round we dedicated a workshop session to release the first version of the MEDINA Framework in the "dev" environment of the Kubernetes cluster. The partners manually released their components and we coordinated and helped them during this phase. The integration was successfully completed and the point2point connections planned were implemented. Further details about this workshop are in *APPENDIX C: First integration workshop.*

During this second round, we didn't have a dedicated workshop session but the partners continuous integrated and updated their components. Thus, all component owners implemented the CI/CD pipelines, which allowed the partners to automatically release them in the Kubernetes cluster. Section 2.2 describes the strategy and implementation in the CI/CD pipelines.

One of the goals we reached during this second round is the integration of the latest MEDINA components into the Kubernetes cluster. In fact, the *Organizational Evidences Gathering and Processing* (AMOE) component was not integrated during the first workshop session and the *Automated Self-Sovereign Identity-based certificates management* (SSI) component was introduced in recent months. AMOE and SSI are now deployed in the "dev" and "test" environments.

Figure 3 lists all the components of the MEDINA Framework: the green ones are released in the Development and Test environments and the blue ones will not be released in the Kubernetes cluster. The Codyze component will be integrated in the MEDINA Security pipeline and Wazuh and VAT run in a dedicated standalone VM provided by TECNALIA. Interested readers can see the progress of the integration of the MEDINA components by comparing Figure 3 with the previous status of integration in M15, that is shown in Figure 48 in the *APPENDIX C: First integration workshop*.

INTEGRATION COMPONENTS STATUS

				Integration Steps						
Component Name	**Owner (Partner**	**Work Packag**	**Task**	**TECNALIA Private GitLab**	**Containerizatio**	**K8s file**	**OpenAPI specs**	**Push to Docker Registry**	**Deploy Dev**	**Deploy Test**
CNL Editor	HPE	WP2	T2.4	yes	yes	yes	yes	yes	yes	yes
Metrics and measures catalogue	TECNALIA	WP2	T2.2	yes	yes	yes	yes	yes	yes	yes
NL2CNL Translator	CNR/Fabasoft	WP2	T2.3	yes	yes	yes	yes	yes	yes	yes
DSL Mapper	CNR/Fabasoft	WP2	T2.5	yes	yes	yes	yes	yes	yes	yes
Cloud Evidence Collector (Clouditor)	FhG	WP3	T3.2	yes	yes	yes	yes	yes	yes	yes
Security Assessment (Clouditor)	FhG	WP3	T3.2	yes	yes	yes	yes	yes	yes	yes
Orchestrator (Clouditor)	FhG	WP3	T3.1	yes	yes	yes	yes	yes	yes	yes
Codyze	FhG	WP3	T3.3	yes (partly)	yes	\	\	yes	no (integrated Jenkins)	no (integrated Jenkir
Blockchain Monitoring Tool	TECNALIA	WP3	T3.5	no (proprietary component)	yes	yes	yes (partially)	yes	yes	yes
Static Risk Assessment and Optimisation Framework	CNR	WP2	T2.4	yes	yes	yes	yes	yes	yes	yes
Dynamic Risk Assessment and Optimisation Framework	CNR	WP4	T4.4	yes	\	\	\	no	\	\
Wazuh + VAT evidence collector (interface to sec.ass.)	XLAB	WP3	T3.2	yes	yes	no	no (uses clouditor's API)	yes	no	no
Wazuh & VAT proprietary	XLAB	WP3	T3.2	no (proprietary component)	no	\	no (uses clouditor's API)	yes	no (dedicated VM)	no (dedicated VM)
Continuous Certification Evaluation	XLAB	WP4	T4.1	yes	yes	yes	yes	yes	yes	yes
Life Cycle Manager	FhG	WP4	T4.3	yes	yes	yes	yes	yes	yes	yes
Assessment and management of organisational evidences (AMOE)	Fabasoft	WP3	T3.4	yes	yes	yes	partially	yes	yes	yes
Integration UI	HPE	WP5	T5.3	yes	yes	yes	no apis	yes	yes	yes
Self-Sovereign Identity (SSI)	TECNALIA	WP4	T4.3	yes	yes	no	yes	yes	yes	yes

Figure 3. Status of integration of MEDIAN components

The last four actions foreseen by the defined methodology were successfully completed by all partners: first of all, each project has been released in the Kubernetes "test" environment and the standalone and point2point tests have been performed, finally the Use Cases tested the end to end scenarios verifying that the workflows described in Section 3 were working properly in their own "Validation" environment. Further details on the validation of the workflows can be found in D5.2 [2] and in D6.3 [6].

During the regular bi-weekly WP5 meetings we checked the status of the components and the updates of the point-to-point connections. Table 2 shows the current status of these connections as follows:

- Light green: the connection was implemented during the first round
- Dark green: the connection has been successfully implemented during this second round
- Orange: the connection is in progress
- Grey: the connection is no longer needed

Comparing the contents of Table 2 with the previous status shown in Table 19 in *APPENDIX C: First integration workshop,* we can see that most of the point-to-point connections are completed: 20 connections have been implemented in addition to the previous 6, 3 connections have been discarded and 3 connections are still in progress.

Table 2. Status of Point-to-point connections

Component Name A	Component Name B	Status
Orchestrator	Continuous Certification Evaluation	CONNECTED
Orchestrator	Trustworthiness System	CONNECTED
Orchestrator	Security Assessment	CONNECTED
Orchestrator	Catalogue of Controls & Metrics	CONNECTED
Orchestrator	NL2CNL Translator	CONNECTED
Codyze	Orchestrator	CONNECTED
Cloud Evidence Collector	Security Assessment	CONNECTED
Security Assessment	Evidence Collection from VAT	IN PROGRESS
Security Assessment	Evidence Collection from WAZUH	CONNECTED
DSL Mapper	Orchestrator	CONNECTED
DSL Mapper	Catalogue of Controls & Metrics	DISCARDED
NL2CNL Translator	Catalogue of Controls & Metrics	CONNECTED
NL2CNL Translator	CNL Editor	CONNECTED
CNL Editor	DSL Mapper	CONNECTED
CNL Editor	Catalogue of Controls & Metrics	DISCARDED
Assessment and Management of Organizational Evidence	Catalogue of Controls & Metrics	CONNECTED
Assessment and Management of Organizational Evidence	Orchestrator	CONNECTED
Catalogue of Controls & Metrics	Static Risk Assessment and Optimisational Framework	IN PROGRESS
Countinuous Certification Evaluation	Catalogue of Controls & Metrics	CONNECTED
Countinuous Certification Evaluation	Dynamic Risk Assessment and Optimisation Framework	CONNECTED
Countinuous Certification Evaluation	Life Cycle Manager	CONNECTED

Component Name A	Component Name B	Status
Dynamic Risk Assessment and Optimisation Framework	Life Cycle Manager	CONNECTED
Assessment and Management of Organizational Evidence	Orchestrator	CONNECTED
Self-Sovereign Identity	Life Cycle Manager	CONNECTED
Integrated UI	Catalogue of Controls & Metrics	CONNECTED
Integrated UI	NL2CNL Translator	DISCARDED
Integrated UI	Orchestrator	CONNECTED
Integrated UI	CNL Editor	CONNECTED
Integrated UI	Self-Sovereign Identity	IN PROGRESS
Integrated UI	Static Risk Assessment and Optimization Framework	CONNECTED
Integrated UI	Continuous Certification Evaluation	CONNECTED
Integrated UI	Assessment and Management of Organizational Evidence	CONNECTED

2.2 Implementation of the CI/CD solution

This section provides updates in M27 on the status of the implementation of the CI/CD strategy supported by CI/CD tools. It first gives an overview of the operating environment that involves all CI/CD components and the Kubernetes cluster and how they work together in our automated solution designed for MEDINA. During this period, full automation of software release has been achieved through the use of pipelines. Secondly, more details are provided on the three standardized pipelines and their stages, and how they are setup through the Jenkins Seed Job. In addition, a new pipeline for cleaning dangling Docker images has been created. Also mentioned is the webinar that was held during this period to support the integration activities by giving an example to partners.

2.2.1 Operating Environment

This section describes the overview of the MEDINA Operating Environment proposed to support the CI/CD implementation.

The MEDINA framework is made up by the collaboration of multiple components developed by the partners and published over the Internet. Each component corresponds to one or more microservices and the code is stored in the TECNALIA GitLab version control, which provides repositories both for private[1] and open-source[2] projects.

All open-source projects are published in TECNALIA's public GitLab, organized with a folder per component where every microservice reports its license, as shown in Figure 4.

[1] https://git.code.tecnalia.com/medina - [authentication required]
[2] https://git.code.tecnalia.com/medina/public

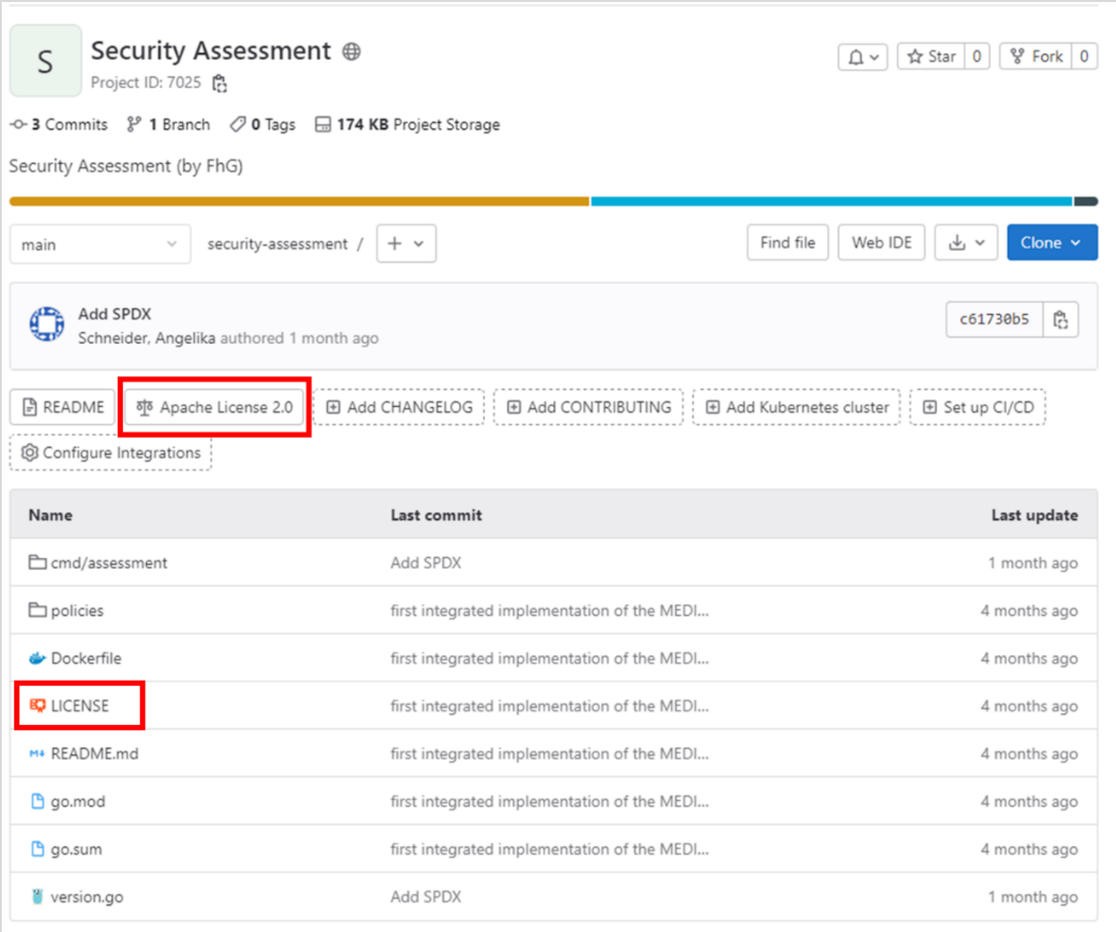


Figure 4. Public GitLab - license

In addition, the license is also provided using the SPDX [7] standard. Thus, in each source code file of the open-source projects there is a header indicating the licence details, which is for all components the Apache-2.0.

On the other hand, we organized the TECNALIA's private GitLab repository in folders that support work packages and tasks, so that each partner can use a dedicated path for its components. For example, the *CNL Editor* component belongs to the work package 2, Task 2.4 and that is the folder where it is stored, as shown in Figure 5.

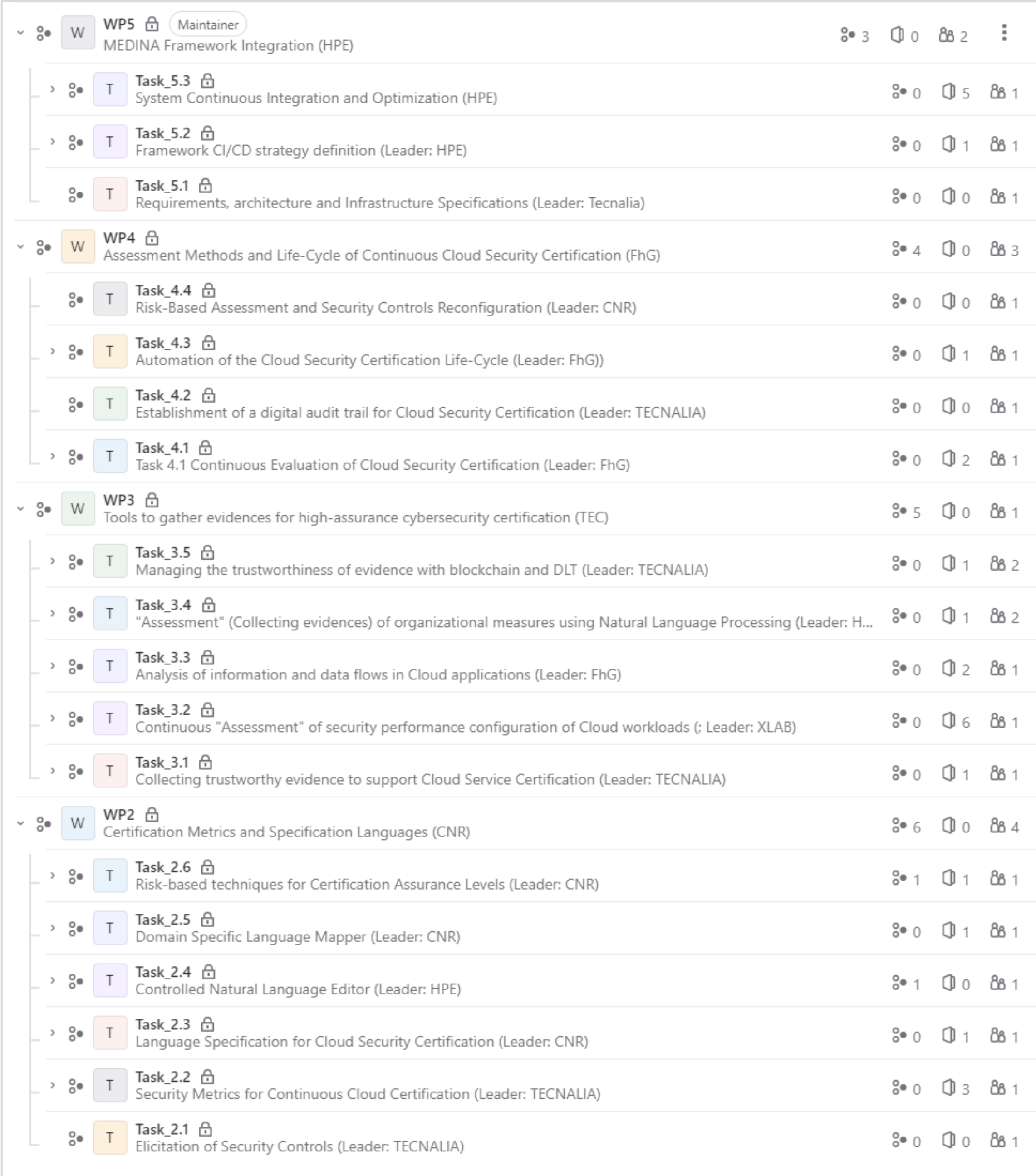


Figure 5. Private GitLab repository

During our regular WP5 meeting, we coordinated and checked that all the components followed the conventions explained above.

The microservice has to be containerized into a Docker image in order to be deployed. For this reason, we provided a private Docker registry hosted by TECNALIA, which is the JFrog Artifactory[3] [8], to store the docker images.

Finally, the docker images are deployed to the Kubernetes cluster and exposed over the Internet. The Jenkins automation server hands the delivery of each microservices: it fetches the code from

[3] https://artifact.tecnalia.com/ui - [authentication required]

GitLab, builds and stores the Docker image and finally releases it into the Kubernetes cluster (see Figure 6).

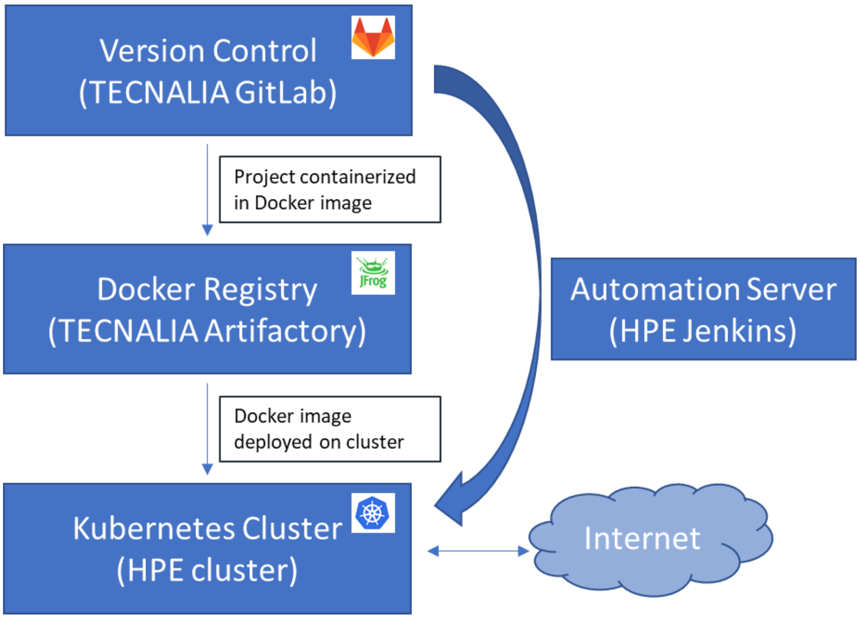


Figure 6. CI/CD tools

More details about the Jenkins pipelines are explained in the following section.

2.2.2 Pipelines

This section describes the implementation of the CI/CD solution that is put in place for supporting the MEDINA Framework through the pipelines schema that has been described in D5.3 [1]. The implemented pipelines are three, named **Build** pipeline, **Deploy** pipeline and **Security** pipeline.

These pipelines are called following a hierarchy: the Build pipeline is triggered automatically at every push of a project in the MEDINA public GitLab and automatizes the build of the project, the creation of the Docker image and its push on the TECNALIA Artifactory. Then, if the previous pipeline succeeds, without any errors, the second Deploy pipeline is triggered that will automatically deploy the component to the “development" environment by default. Finally, the Security pipeline starts automatically if the Build and the Deploy pipelines succeed.

As described in D5.2 [2], to automate the deployment process we make use of the Jenkins Seed Job that will automatically create the pipelines for each component of the MEDINA Framework. This is a plugin that consists in filling a form by entering parameters such the software repository URL where to retrieve the source code, the container file descriptor (in Docker format), the generated container image for publishing to an internal private registry and a list of one or more Kubernetes deployment manifest files.

This procedure is quite the same for all components because all the CI/CD tools involved are organized to simplify the deployment with a convention agreed by the consortium. The GitLab repository is divided into groups that are folders which contain the projects. The structure reflects the Work Package and Tasks division of the MEDINA project. Also, Jenkins and Artifactory are organized following this convention.

All these concepts and steps were described during the CI/CD Webinar (see Section 2.1.2.2) using a Demo with the sample project "springswagger-template”. First of all, a new project named “springswagger-template” was created in GitLab. The Jenkins Seed Job could then be run by filling it with the parameters customized for the project.

Following there is the description of these parameters and an example how to compile the form to create the specific pipelines for the project “springswagger-template”. Figure 7 shows these parameters:

- **Work Packages/Task folder,** where the Jenkins Jobs will be created. We can choose the correct path from the picklist, that are previous created in Jenkins. Select wp5/task_5.2.
- **Job basename**, i.e., the component name: for example, springswagger-template.
- **GitLab URL**, retrieved from the TECNALIA GitLab web interface, is the source code repository for the project.
- **GitLab branch**, is the default “master”.
- **Build template**, chosen from a preconfigured template, can be empty or customized with a build automation tool like Maven. Select Maven.
- **Docker file**, the name of the dockerfile that contains the instructions to build the container image. In this case the folder in which is the file is “docker” and the name of the file is “Dockerfile”.
- **Image**, the name of the container image pushed to the private registry, that is the Artifactory owned by TECNALIA. The image will have the absolute path, for example: wp5/t52/springswagger-template.
- **Kubernetes manifests**, the yaml files used for the deployment in the Kubernetes cluster, that are contained in GitLab folder “kubernetes”.

Once these details are provided, the Seed Job automatically creates the three pipelines for build, deploy and security for the “springswagger-template” in the selected folder (see Figure 8).

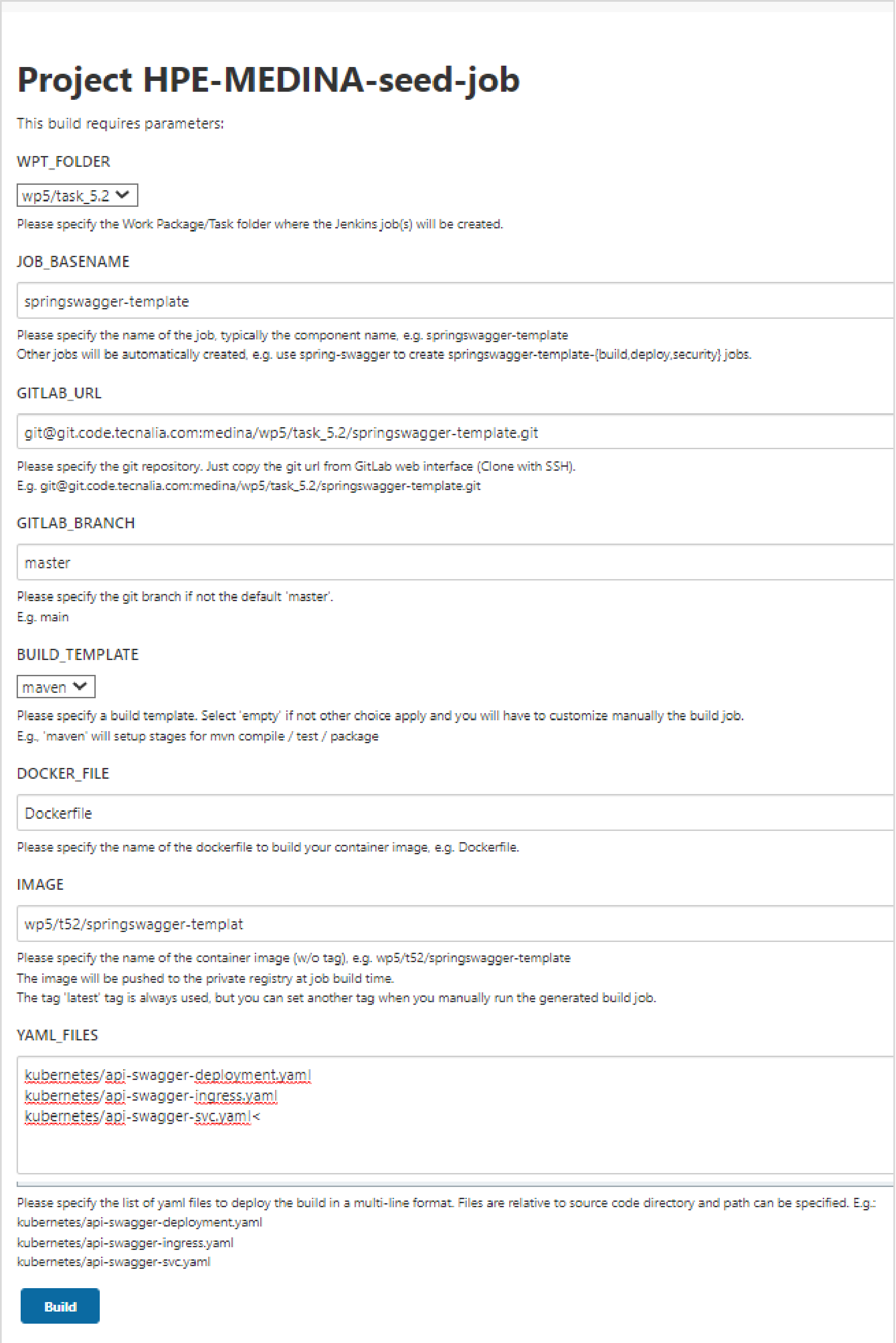


Figure 7. Jenkins Seed Job

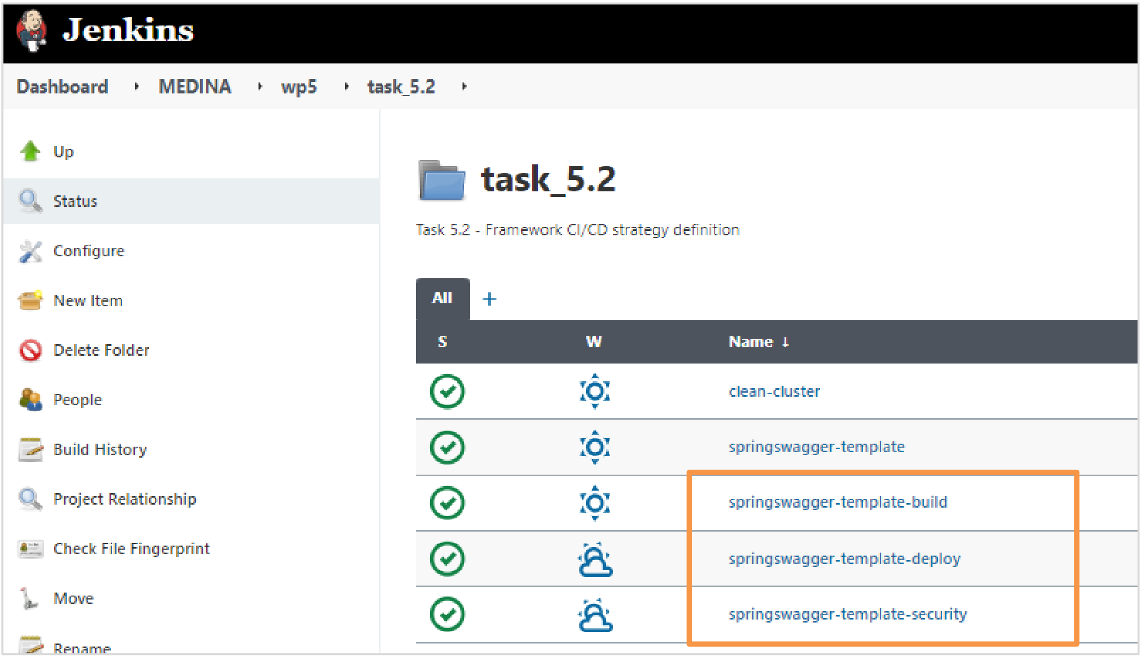


Figure 8. Pipelines

During the CI/CD Webinar demo it has been shown how the creation of the pipelines flows through their stages after configuring and building the Jenkins Seed Job. Every pipeline has several stages, with a name describing what they have done.

As described theoretically in the CI/CD strategy in D5.2 [2], the Build pipeline foresees stages where the code is checked out from GitLab and the docker container is setup to execute the other build stages. These stages are the compile, testing and package stages that can be different depending on the Build template field selected in the Seed Job before running it. In this case we have selected Maven, so *"mvn"* commands are executed. The next three stages are referred to the Docker image building and pushing to the Artifactory repository. By default, the image is pushed with the "latest" tag but there is an optional phase to tag it differently. At last, if no errors occur the Deploy Job is automatically called.

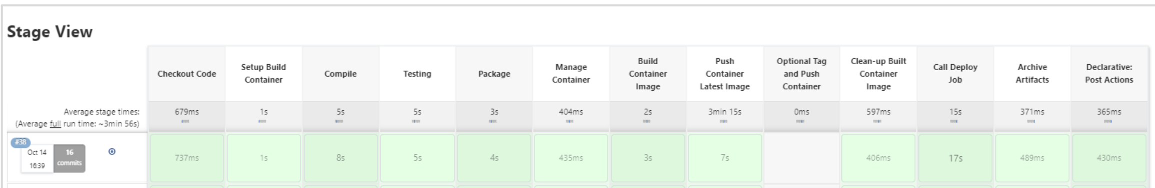


Figure 9. Build pipeline

The Deploy pipeline deals with the release of the components in the Kubernetes cluster. As described in Section 2.1, the Kubernetes cluster is divided in two isolated and virtual environments, "dev" and "test". The stages of this pipeline (see Figure 10) include first the step where Jenkins accesses to the Kubernetes cluster with exchanged credentials, and then the step in which the Kubernetes manifests files are applied to release the configuration to the environment. By default, the Deploy pipeline releases the component on the "dev" environment.

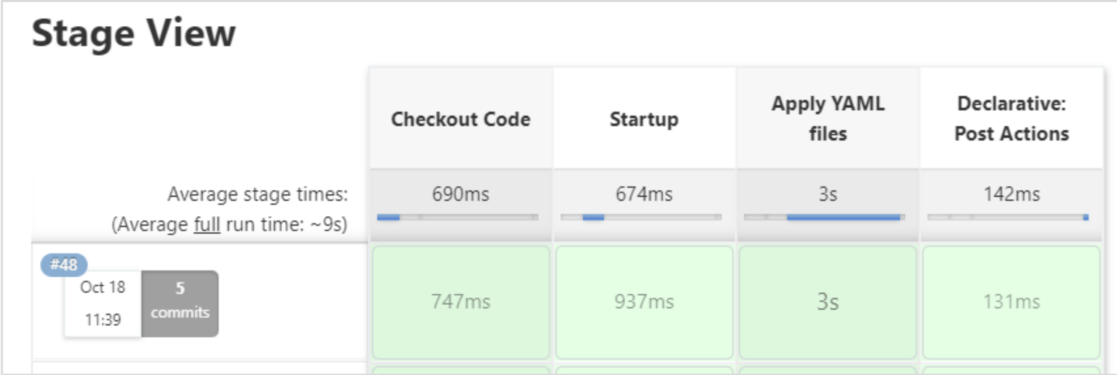


Figure 10. Deploy pipeline

Partners can also use this pipeline to manually release the component on the "test" environment changing it with one click from the Deploy pipeline, rebuilding the pipeline and choosing among the available environments, as shown in Figure 11.

Figure 11. Deploy pipeline with available ENV

The Security pipeline is automatically triggered upon a successful Build and Deploy.

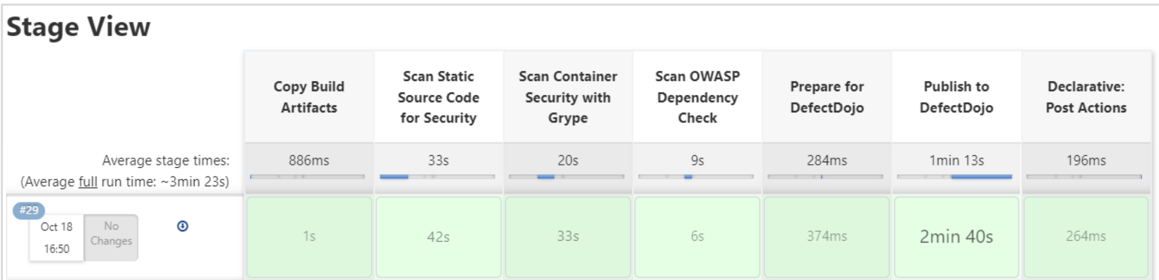


Figure 12. Security pipeline

This pipeline includes various steps (shown in Figure 12) representing the different types of security analysis performed: Static Code analysis for checking the source code, Container security for scanning vulnerabilities into the container packages, and Software Composition Analysis (SCA) for spotting security issues in third party libraries.

The two first security controls are performed respectively by Semgrep and Anchore. These tools are running into containers called in the security pipeline. Once the scanning is done, these containers, in which the tools are installed, are destroyed but the output file of the analysis persists. The advantage of this choice to use the container lives in the fact that it is possible to fast and easily update the tool to the latest version, forcing the download of the latest tag of the container images.

Regarding the third security control, SCA, the tool that performs this analysis is OWASP Dependency Check, installed via command line. In the latest stages of this Security pipeline a report is prepared, that collects all the analysis outputs of the previous stages, and finally is published to DefectDojo, the vulnerability report aggregator tool adopted to make possible to see all the analysis results in a unique view. The report is visible directly inside Jenkins, but DefectDojo provides a graphical interface with several metrics and dashboards to analyse the results using different parameters, such as the time or the severity of the vulnerabilities.

Figure 13 shows a view from the DefectDojo Dashboard of the findings “springswagger-template” that are found running its Security pipeline. It represents an easy way to control, for example, the Severity of the vulnerabilities and help to do mitigation actions with suggestions.

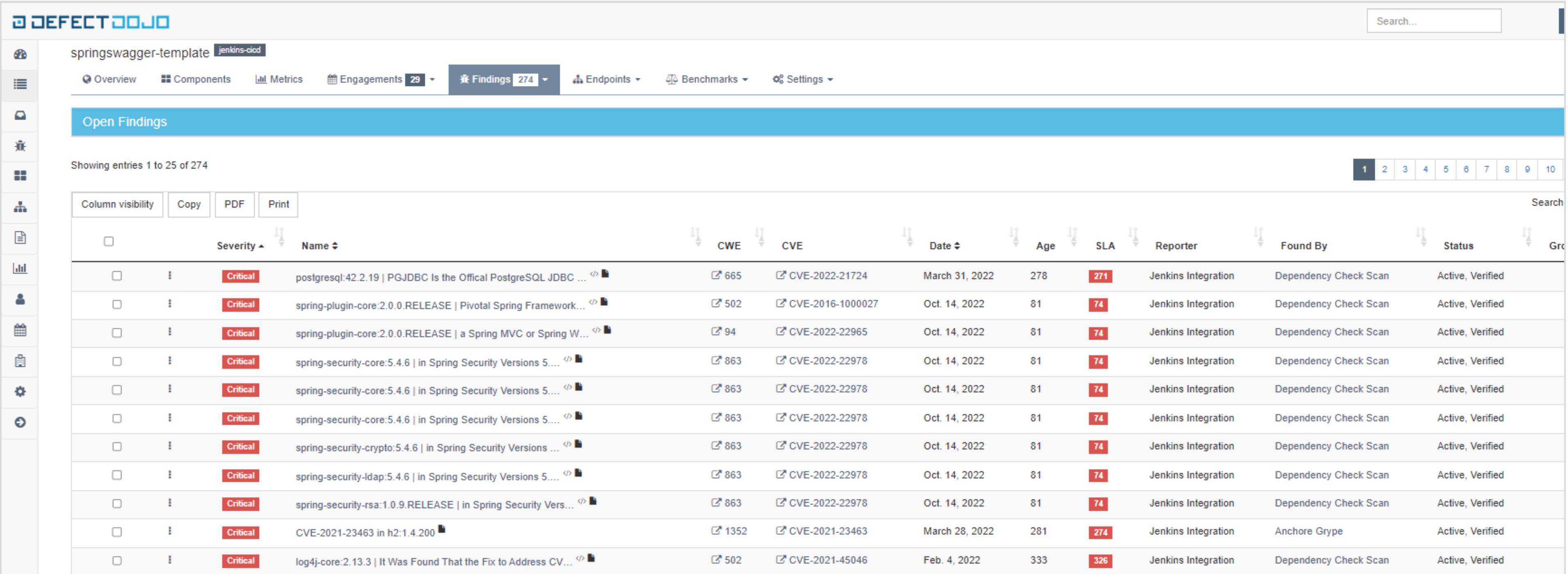


Figure 13. DefectDojo Dashboard: springswagger template

As a next step in the Security pipeline, the MEDINA component Codyze [9] will be added. Codyze is a static code analysis tool developed by FhG partner (see Section 4.7.1).

In addition to the three pipelines available in M15, we needed to add a new pipeline called “clean-cluster” to deal with dangling docker images that caused no disk space to be left. Figure 14 shows the stages that compose this pipeline. Basically, the dangling docker images are listed, then removed, and finally the disk space is shown.

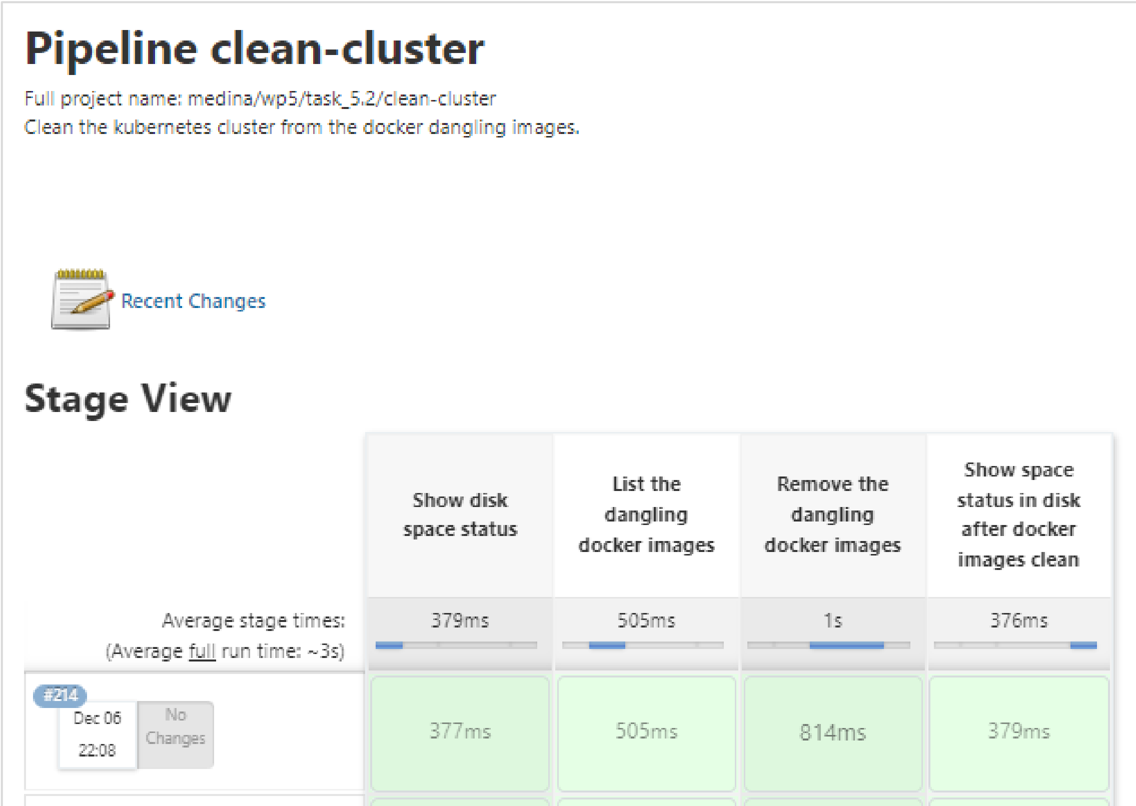


Figure 14. Clean-cluster pipeline

All these steps provide an example of how to use the CI/CD tools to adopt the SecDevOps approach in MEDINA. The aim is to give guidelines to partners to enable a conventional way of using the overall infrastructure that is setup.

The result is that all the components that build the MEDINA Framework were deployed in M27 using these Jenkins pipelines and were released in the two Kubernetes environment “dev” and “test”.

3 Generic Architectural Workflows

This section updates the generic MEDINA workflows (WFs), which were first introduced in D5.3 [1] and further detailed in D6.3 [6], with the initial version of an "Authorization and Filtering Concept" by defining the use-case 1 (UC1) roles and access-levels assigned to MEDINA users on the different components of the developed framework.

At this point it is worth to notice that MEDINA's authorization and filtering concept only applies to those components where user interaction (UI) has been considered, whereas components/API methods not exposing any UI element are out of scope[4].

Next, to keep this report self-contained, we review the basics related to the generic MEDINA workflows, as presented in D5.3. For interested readers, an updated version of the MEDINA workflows' details can be found in *APPENDIX D: Generic Architectural Workflows.*

3.1 Generic MEDINA Workflows

This section provides as background the generic workflows which comprise the MEDINA framework and consist of the seven different scenarios/interactions shown in the Table 3 below.

Table 3. Generic MEDINA workflows

Workflow	Comment	Other/Dependency
WF1 - Preparation of Target of Certification (ToC)	Setup, configure and deploy the cloud service to certify (ToC) on top of the chosen hyperscaler(s). This process includes configuring the underlying PaaS/IaaS.	Mandatory workflow CSP Responsibility Dependencies: None
WF2 - Preparation of MEDINA components	Setup, configure and deploy the MEDINA components. Only related to those components under the responsibility of the CSP.	Mandatory workflow CSP Responsibility Dependencies: WF1
WF3 - EUCS deployment on ToC	Setup, configure and deploy the corresponding EUCS framework (for the chosen assurance level basic/substantial/high) on the ToC.	Mandatory workflow CSP Responsibility Dependencies: WF1, WF2
WF4 - EUCS Preparedness – ToC Self-Assessment	Self-assess preparedness for EUCS certification based on the chosen assurance level. This is a risk-based approach.	Optional workflow CSP Responsibility Dependencies: WF1, WF2, WF3
WF5 - EUCS – compliance assessment	Performs a point-in-time (discrete) EUCS compliance assessment for the ToC. When such discrete assessment is periodically executed, then we achieve the MEDINA notion of "continuous".	Mandatory workflow CAB Responsibility Dependencies: WF1, WF2, WF3
WF6 - EUCS – maintenance of ToC certificate	Start certificate maintenance life-cycle for the ToC. Based on current EUCS, the maintenance process comprises the following stages: (issuance[5]), renewal, continuation, update, re-issuance (new certificate), withdrawal, and	Mandatory workflow CAB Responsibility CSP Responsibility Dependencies: WF1, WF2, WF3, WF5

[4] It must be noticed that API-level authorization shall be needed in productive MEDINA environments (TRL7). The authorization/filtering model introduced in this section provides the basis for implementing more complex/productive scenarios.

[5] Despite initial certificate's issuance is not mentioned in the maintenance process defined by the core EUCS document, for the purposes of MEDINA this discussion is part of the life-cycle manager (WP4).

Workflow	Comment	Other/Dependency
	suspension.	
WF7 - EUCS – report on ToC certificate	Reports on EUCS certificate status for a ToC. The report can be obtained by the CAB and the CSP, in which case the level of provided details might vary.	Optional workflow CAB Responsibility CSP Responsibility Dependencies: WF1, WF2, WF3, WF5

Based on these generic workflows, the rest of this chapter focuses on presenting the initial roles and authorization concept for the framework.

3.2 Roles and Levels of Visibility

To present the initial authorization/filtering concept, first we proceed to re-introduce the basic roles in MEDINA (cf. D6.3 [6]) along with their "visibility level", which is defined in terms of the CSP information available for EUCS certification. This is shown in Table 4.

Table 4. MEDINA Roles and Levels of Visibility

Roles	Explanation (cf. D6.3 [6])	Level of Visibility
IT Security Governance	Its main objective is the protection of Bosch business models, products, services, and data.	Cloud Service Provider[6]
Security Analyst	Responsible for ensuring that the Bosch Group's digital assets and sensitive information are protected as well as evaluating and reporting on the efficiency of the security policies in place.	Cloud Service Provider
Domain Governance	Acts as the core competence holder and responsible topic owner for product security.	One or more Cloud Services
Product and Service Owner	The Product & Service Owner is the central point of contact for all questions concerning a specific Bosch IT product or service.	Cloud Service[7]
Product (Security) Engineer	Oversees the build, deploy, and run of a product and its system components.	Cloud Service
Chief Information Security Office (CISO)	The Chief Information Security Officer (CISO) is who the Compliance Manager has to report to.	Cloud Service Provider
Customer	The customer[8] is either a company consuming cloud products or services (B2B, business-to-business context), or an individual (B2C, business-to-customer context).	Cloud Service
Auditor[9]	The Conformity Assessment Body (CAB) is a body that performs conformity assessment services with the goal of demonstrating that specified requirements are fulfilled.	One or more Cloud Services

[6] Including all underlying certifiable Cloud Services.
[7] For the purposes of MEDINA, we consider visibility to at most one Cloud Service.
[8] For the purposes of MEDINA, the Customer is the only non-authenticated role in the framework.
[9] This role also refers to internal Auditors and NCCAs (National Cybersecurity Certification Authority).

Next, for each defined role we introduce the actual set of allowed actions based on both the relevant WF and the involved MEDINA framework components. This is presented in the following section.

3.3 Authorization Model for MEDINA Workflows

MEDINA leverages the Role Based Access Control model (RBAC[10]) to enforce specific permissions on the Integrated UI for certain components. This section presents the initial version of MEDINA's RBAC concept based on the generic workflows, whereas details associated to its technical implementation are presented later on this document.

3.3.1 WF1 - Preparation of Target of Certification (ToC)

This initial workflow, despite not invoking any of the MEDINA components, is an evident pre-requisite for the CSP to fulfil before the certification process starts. Its main goal is for the CSP to prepare the Target of Certification (ToC), both from a technical (e.g., deploying the actual cloud service in the hyperscaler) and organizational (e.g., gather the operational manuals in electronic format) perspectives.

Table 5. Workflow 1

Short Explanation	Associated MEDINA Components
Setup, configure and deploy the cloud service to certify (ToC) on top of the chosen hyperscaler(s). This process includes configuring the underlying PaaS/IaaS.	CSP testbed

For this initial workflow, the only role allowed to operate on the platform is the so-called Product (Security) Engineer, as shown in Table 6.

Table 6. RBAC Model for Workflow 1

Roles	Component	Allowed Actions
IT Security Governance	Testbed	None
Security Analyst	Testbed	None
Domain Governance	Testbed	None
Product and Service Owner	Testbed	None
Product (Security) Engineer	Testbed	Setup, configure, deploy
Chief Information Security Office (CISO)	Testbed	None
Customer	Testbed	None
Auditor	Testbed	None

3.3.2 WF2 - Preparation of MEDINA Components

The second generic workflow of the architecture (WF2) refers to the actual configuration and deployment of those MEDINA components which are needed for certifying the Cloud Service. This WF2 does not perform any actual assessment, but it executes a required set of deploying actions before the certification process is triggered by WF3.

[10] Please refer to https://en.wikipedia.org/wiki/Role-based_access_control

Table 7. Workflow 2

Short Explanation	Associated MEDINA Components
Setup, configure and deploy the MEDINA components. Only related to those components under the responsibility of the CSP.	Evidence Collectors, Integrated UI

The evidence collectors (e.g., Clouditor and Wazuh), along with the Integrated UI are deployed and configured by the Product (Security) Engineer exclusively.

Table 8. RBAC Model for Workflow 2

Roles	Component	Allowed Actions
IT Security Governance	Testbed	None
Security Analyst	Testbed	None
Domain Governance	Testbed	None
Product and Service Owner	Testbed	None
Product (Security) Engineer	Testbed	Setup, configure, deploy (Catalogue, SSO, DLT, Clouditor, Wazuh, Codyze, VAT)
Chief Information Security Office (CISO)	Testbed	None
Customer	Testbed	None
Auditor	Testbed	None

3.3.3 WF3 - EUCS deployment on ToC

After the ToC has been deployed on the hyperscaler (WF1) and the corresponding MEDINA components were configured/deployed by the CSP (WF2), then it is possible to use the later for certifying the Cloud Service. That is the goal of this WF3.

Table 9. Workflow 3

Short Explanation	Associated MEDINA Components
Setup, configure and deploy the corresponding EUCS framework (for the chosen assurance level basic/substantial/high) on the ToC.	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper

The third workflow in MEDINA involves interaction between different components of the architecture to orchestrate complex processes (e.g., NLP for recommending EUCS metrics). In this case we identify roles without permission to modify certification information, and once again the Product (Security) Engineer as the only role capable of changing information about the framework. It must be noted that the Catalogue cannot be modified by any of the MEDINA roles, and it is pre-filled by the "MEDINA framework provider" with the required standards.

Table 10. RBAC Model for Workflow 3

Roles	Component	Allowed Actions
IT Security Governance	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper	Read[11]
Security Analyst	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper	Read
Domain Governance	Catalogue, NL2CNL Translator, CNL Editor,	Read

[11] Descriptions can be updated for Entities on the Catalogue, in order to match specific organizational needs.

Roles	Component	Allowed Actions
	DSL Mapper	
Product and Service Owner	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper	Read
Product (Security) Engineer	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper	Read/Write[12]
Chief Information Security Office (CISO)	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper	Read
Customer	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper	None
Auditor	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper	Read

For WF3 and in the specific case of the CNL Editor UI, the "Write" actions means that only the role "Product (Security) Engineer" is allowed to operate on the Rego objects by applying *Complete* and *Map*.

3.3.4 WF4 - EUCS Preparedness – ToC Self-Assessment

This workflow relates to the components in charge of performing the static risk management (SATRA) and the self- assessment Questionnaire (*Catalogue of controls and metrics*) as documented by D2.7 [10] and D2.2 [11] respectively. Although SATRA implements a "stand alone functionality", which does not need to be technically deployed in the Cloud Service (cf. WF3), it is integrated into the whole MEDINA framework thanks to the unified UI.

Table 11. Workflow 4

Short Explanation	Associated MEDINA Components
Self-assess preparedness for EUCS certification based on the chosen assurance level. This includes a risk-based approach.	SATRA, Catalogue

Both components in WF4 only allow the Product and Service Owner to perform all available actions. The rest of roles are assigned read-only/reporting actions according to the least privilege principle.

Table 12. RBAC Model for Workflow 4

Roles	Component	Allowed Actions	Component	Allowed Actions
IT Security Governance	SATRA	Risk Computation (Reporting)	Catalogue	Load questionnaire, Generate report
Security Analyst	SATRA	Risk Computation (Reporting)	Catalogue	Load questionnaire, Generate report
Domain Governance	SATRA	Risk Computation (Reporting)	Catalogue	Load questionnaire, Generate report
Product and Service Owner	SATRA	Create SoC, SoC Info, Questionnaire, Asset Information, Risk Computation (Reporting)	Catalogue	Start/Edit questionnaire, Save questionnaire, Generate report
Product (Security) Engineer	SATRA	Risk Computation (Reporting)	Catalogue	Load questionnaire, Generate report
Chief Information	SATRA	Risk Computation	Catalogue	Load questionnaire,

[12] Only for AMOE it is allowed to Delete uploaded PDF security policies.

Roles	Component	Allowed Actions	Component	Allowed Actions
Security Office (CISO)		(Reporting)		Generate report
Customer	SATRA	None	Catalogue	None
Auditor	SATRA	Risk Computation (Reporting)	Catalogue	Load questionnaire, Generate report

3.3.5 WF5 - EUCS Compliance Assessment

This WF5 describes **discrete compliance assessments**, which should then be periodically executed for the MEDINA framework to start the certification lifecycle (cf. WF6).

Table 13. Workflow 5

Short Explanation	Associated MEDINA Components
Performs a point-in-time (discrete) EUCS compliance assessment for the ToC. When such discrete assessment is periodically executed, then we achieve the MEDINA notion of "continuous".	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors

WF5 contains the interactions for performing discrete assessments, where only the role (internal) Auditor is allowed to change AMOE recommended assessments and submit them for evaluation to the Orchestrator.

Table 14. RBAC Model for Workflow 5

Roles	Component	Allowed Actions
IT Security Governance	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	Read[13]
Security Analyst	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	Read
Domain Governance	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	Read
Product and Service Owner	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	Read
Product (Security) Engineer	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	Read
Chief Information Security Office (CISO)	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	Read
Customer	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	None
Auditor	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors	Read/Write

3.3.6 WF6 - EUCS – Maintenance of ToC certificate

This WF6 departs from the current definition of certificate maintenance in the EUCS core document, and for the purposes of MEDINA, it also adds an initial stage of "certificate issuance".

Despite WF6 plays an important role in MEDINA (i.e., continuous execution and analysis of discrete assessments), there is no user interaction envisioned within the Integrated UI. **For this reason, WF6 is not associated to any RBAC model.**

[13] Filtering assessment results in the Orchestrator is consider a "Read" action.

3.3.7 WF7 - EUCS –Report on ToC Certificate

The goal of this WF7 is to report about the status of an EUCS certificate corresponding to the ToC and at different levels of detail, depending on the targeted audience (CAB, CSP, etc.). The final WP7 takes care of reporting the status of the certificate (and related evidence) to authorized stakeholders.

Table 15. Workflow 7

Short Explanation	Associated MEDINA Components
Report on EUCS certificate status for a ToC. The report can be obtained by the CAB or by the CSP, in which case the level of provided details might vary.	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI

In this case, the proposed RBAC model considers read-only actions for all roles except the Auditor. The latter must have read and write access in order to operate at the level of evidence and assessments in the corresponding components' UIs.

Table 16. RBAC Model for Workflow 7

Roles	Component	Allowed Actions
IT Security Governance	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read
Security Analyst	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read
Domain Governance	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read
Product and Service Owner	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read
Product (Security) Engineer	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read
Chief Information Security Office (CISO)	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read
Customer	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read[14]
Auditor	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI	Read/Write

[14] For non-authenticated users (i.e., Customer role), the usage of SSI technology provides selective disclosure of attributes related to the EUCS certificate. This measure avoids leaking confidential attributes used during the certification process.

4 MEDINA Framework components and integration

This section describes the status of the integration activities of the MEDINA components.

The figure below represents the evolution of the architecture presented in D5.3 [1] and identifies eight building blocks, each one corresponding to a different functionality. Further information about the architecture can be found in deliverable D5.2 [2].

For each block there is a dedicated section presenting the components which are part of it. The block#8 represents the Integrated UI, which is a WP5 component. For this reason, it is described in the dedicated Section 5.

For each component, we present a brief description of its role in the MEDINA framework and a reference to the deliverable containing more details about it. This is followed by information on the integration of the component with the other MEDINA components, the improvements achieved during this second round and the APIs exposed. Finally, if available, a brief description of the implemented Graphical User Interface (GUI) is included.

All component REST APIs are listed in *APPENDIX E: Published APIs*.

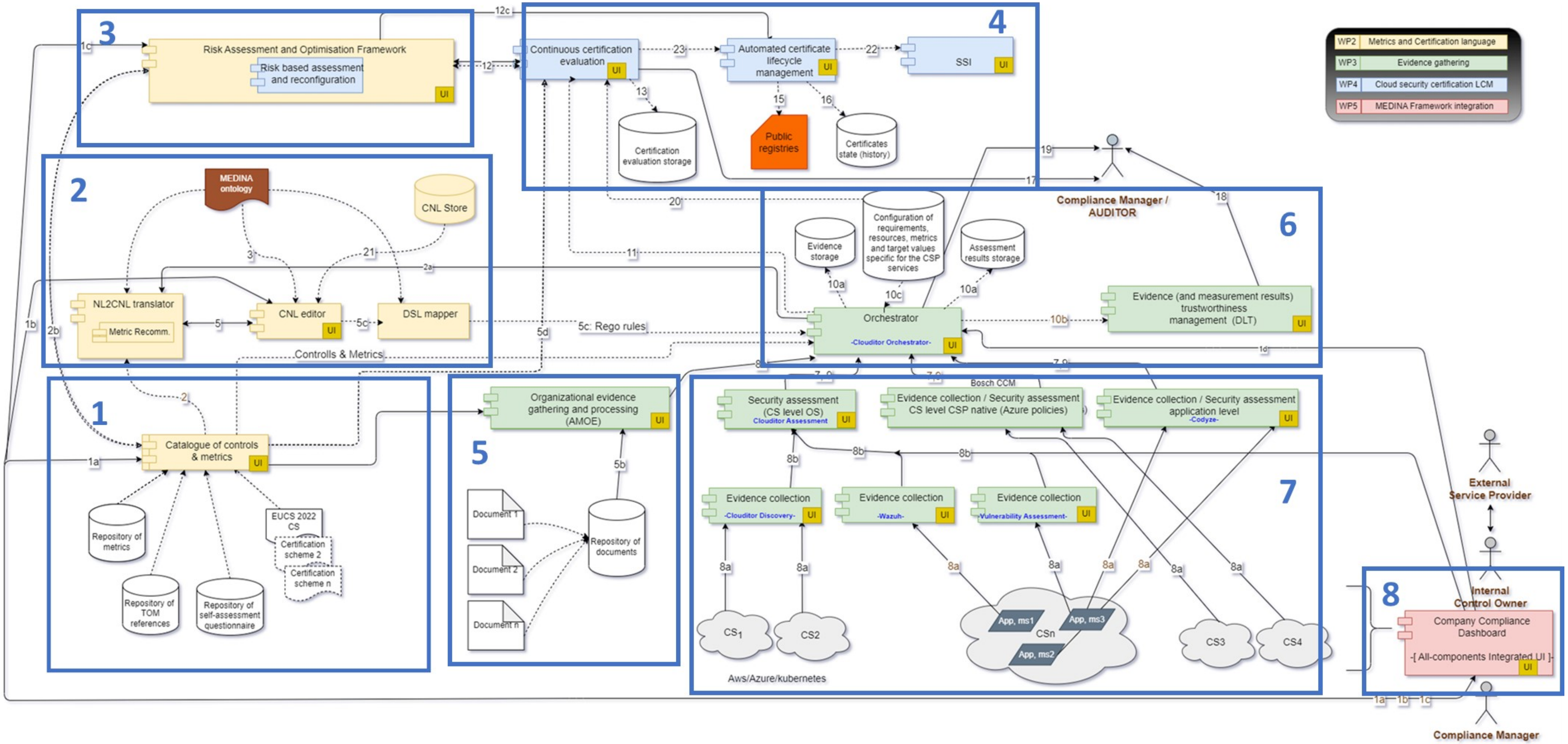


Figure 15. MEDINA Architecture and data flow

4.1 Catalogue (block #1)

The Catalogue is the component that implements most of the KR1 (Repository of metrics and measures). The main goal is to have an automated tool where a CSP compliance manager or an auditor can obtain all the information and guidance related to a security scheme (in MEDINA, we are focused in the EUCS scheme). Namely the controls, the security requirements, its assurance levels, etc. That is, everything that can be considered "static" information that appears in the standard.

As a result of the research performed in MEDINA, the Catalogue has been extended with extra information/functionalities such as reference TOMs, metrics, mapping to controls that are similar in other schemes, and self-assessment questionnaires.

For more information about the Catalogue, see the deliverable D2.2 [11].

4.1.1 Catalogue of Controls and Metrics

The *Catalogue of Controls and Metrics* component provides the following functionalities:

- Endorsement of Security Control Frameworks and related attributes: security requirements, categories, controls, reference TOMs, metrics, and assurance levels.
- Provision of guidance for the (self-)assessment of the requirements.
- Shows and filters the information based on some values for the attributes:
    - Shows the controls by category, navigation through categories, controls, requirements, and metrics
    - Selection of requirements of a certain assurance level
    - Selection of metrics related to TOM.
    - Provides filters in each screen based in field names
- Mapping of certification schemes, providing information about related controls from different frameworks, with respect to controls in EUCS.
- Provides a self-assessment questionnaire (of about 500 questions) to check the degree of compliance of the EUCS 2022 security framework [12].

More information about all the Catalogue can be found in D2.2 [11].

4.1.1.1 Implementation and Integration Status

The main updates of the second release of the *Catalogue of Controls and Metrics* (M27) with respect to the previous version (M12) are related to:

- Updates to EUCS draft version August 2022 [12]
- Development of the questionnaire functionality
- Refurbishment of the GUI for easier navigation through the application data
- Updates to the mapping of controls, including the new version of ISO 27002 and the Cisco Cloud Controls Framework[15]
- Inclusion of the Reference TOMs
- Configuration of the deployment of the component for Kubernetes into the development and test environments
- Updates on the data model used, specifically in the database, as required by the Use Cases.

[15] Cisco CCF: https://www.cisco.com/c/en/us/about/trust-center/compliance/ccf.html

The second version of the Catalogue in M27 implements all mandatory requirements as defined in deliverable D5.2 [2], at least partially. Concretely, 7 out of 9 (77%) of the requirements are fully implemented, and the rest is very advanced. A docker-compose file for deployment has been provided that can be deployed locally for development using vagrant and docker-compose. The deployment in the Development and Test environments has been done through the Kubernetes cluster provided by WP5.

The Catalogue frontend is now integrated with the MEDINA Integrated UI. It is also integrated with the user management tool (Keycloak) and is able to control the logged user and its properties. The Catalogue provides a GUI for end users, as well as a RESTful API to interact with it. Both are described in Section 4.1.1.3.

The Catalogue provides data to the following MEDINA components: *Orchestrator* (controls and metrics), *NL2CNL Translator* (metrics), *SATRA-Risk Assessment and Optimization framework* (answers to the questionnaire), *AMOE-Assessment and Management of Organizational Evidence* (control and metrics), and *CCE-Continuous Certification Evaluation* (relations between metrics, requirements, controls, categories).

The Catalogue is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[16].

4.1.1.2 Published APIs

The Catalogue has implemented all the internal functionality to access and modify the database elements as a REST API, so the number of interfaces and endpoints is quite large. The list of the available APIs is gathered in Annex E, *Component: Catalogue of Controls* . All of them are available for the components that want to interact with the Catalogue.

The complete API is also available online at the repository[17] as an OpenAPI definition.

4.1.1.3 Graphical Interface

The Catalogue offers a GUI to access and manipulate the different entities that are stored in the database. A CRUD screen (Create/Retrieve/Update/Delete) has been developed for each of the main entities, although not all these actions have been allowed in all cases.

The GUI allows the user to navigate through the EUCS framework elements, using the visual elements on the different screens -like buttons, links, and filters-. For example, the user can select information like the requirements of a certain assurance level, controls of a category, metrics related to a requirement, reference TOMs, etc.

In the following, some screenshots are presented as a sample of the GUI, in particular to show:

- Controls (see Figure 16)
- Requirements (see Figure 17)
- Filters (see Figure 18 and Figure 19)
- Metrics (see Figure 20) and details of a metric (see Figure 21)
- Questionnaires (see Figure 22)

The interested readers can find a more detailed user manual in D2.2 [11].

[16] https://git.code.tecnalia.com/medina/public/catalogue-of-controls
[17] https://git.code.tecnalia.com/medina/public/catalogue-of-controls/-/blob/main/openapi.json

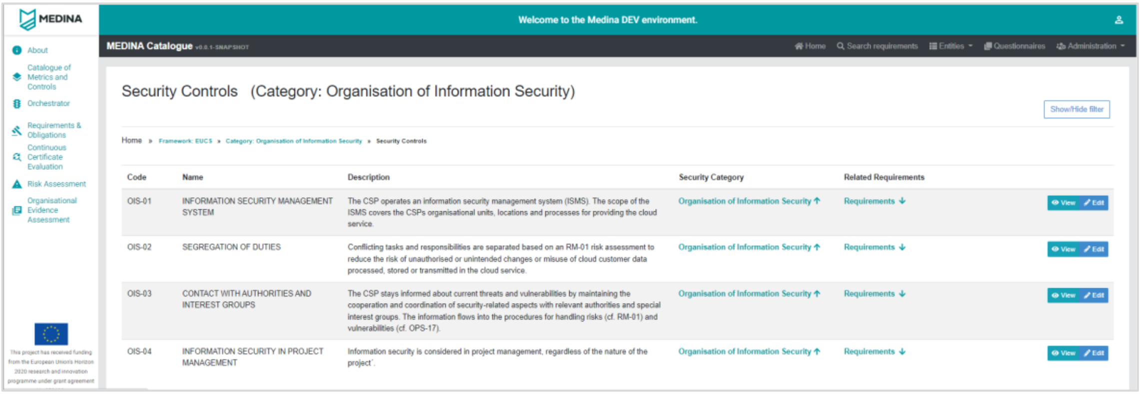


Figure 16. Controls of the "Organisation of Information Security" category

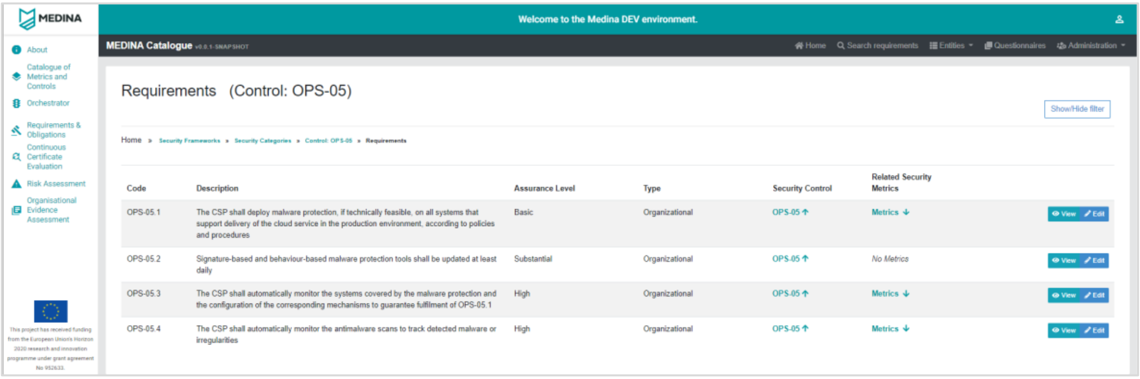


Figure 17. Requirements of the "OPS-05" control

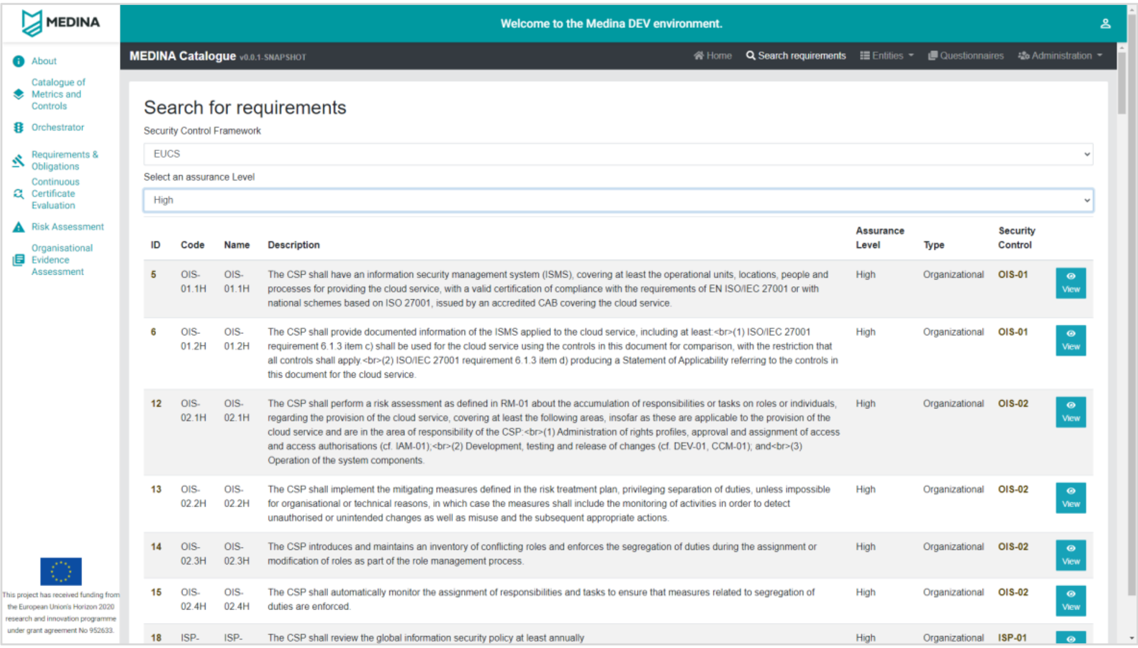


Figure 18. Filter to search for "EUCS & High" requirements

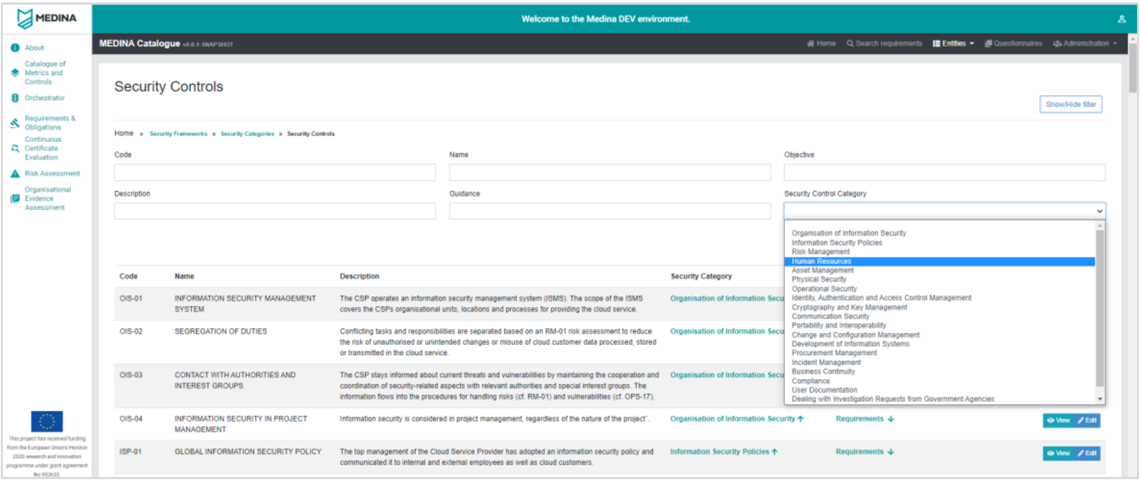


Figure 19. Filter to search for controls

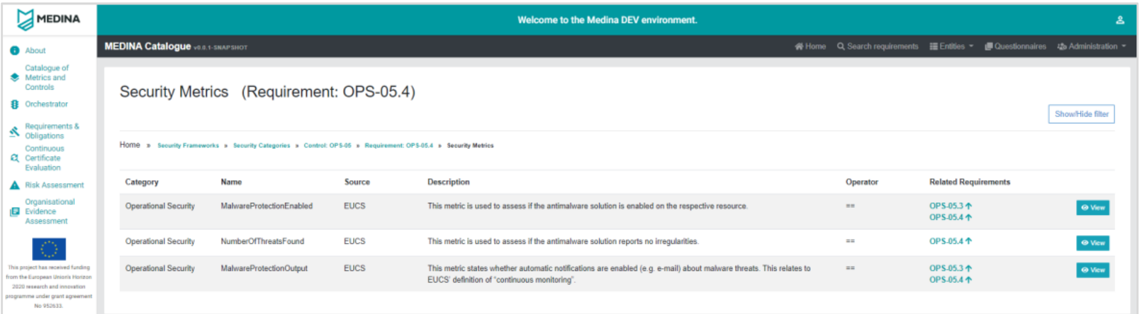


Figure 20. Metrics implemented for the "OPS-05.4" requirement

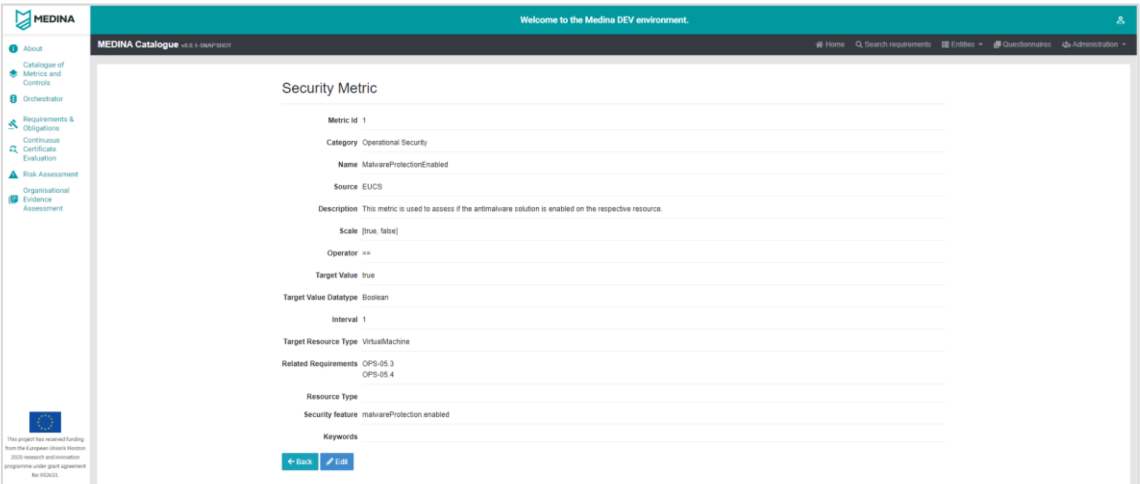


Figure 21. Details of a metric

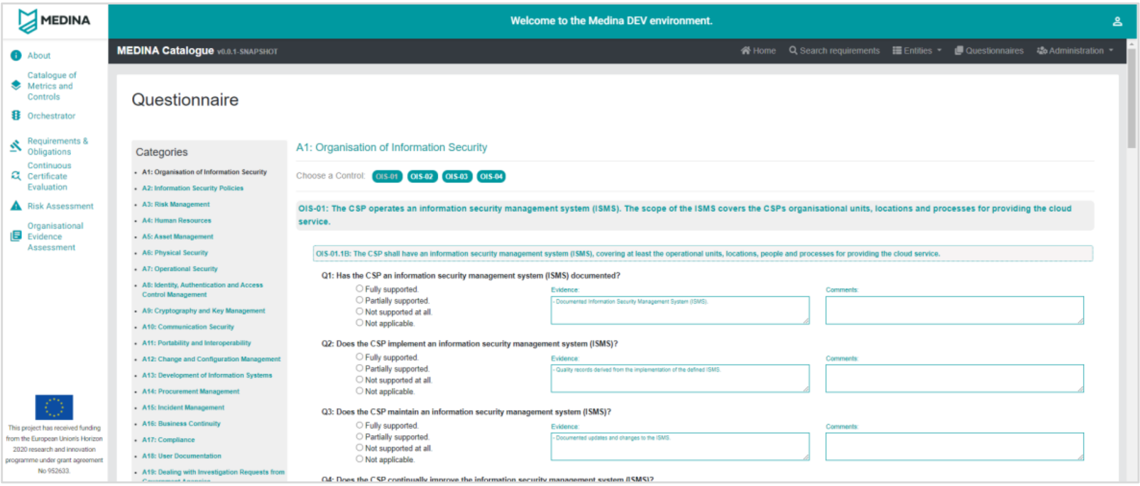


Figure 22. Questionnaire. Questions for the OIS-01 Security control

4.2 Certification Metrics and Language (block #2)

The components belonging to the "Certification Metrics and Languages" block are mainly related with KR3 (Certification Language), whose objective is to provide a language specification which expresses the most relevant aspects of a security certification scheme in machine-readable format using a Domain Specific Language (DSL).

More information about all the components described in this section can be found in the deliverable D2.4 [13].

4.2.1 NL2CNL Translator

The NL2CNL Translator is the MEDINA component used to map EUCS NL (Natural Language) requirements into their MEDINA CNL (Controlled Natural Language) translation. This translation is performed in two steps: the first one selects a set of metrics that could be useful to evaluate a certain security requirement, also called TOM (Technical and Organizational Measure). After associating a set of metrics with a requirement, the second step translates those metrics into policies. Specifically, requirements and metrics are expressed in NL, while the translated policies are expressed in CNL.

The NL2CNL Translator interacts with the *Catalogue of Controls and Metrics*, with the *CNL Store* through the CNL Editor APIs, and with the *Orchestrator*.

4.2.1.1 Implementation and Integration Status

Compared with M15, the NL2CNL Translator prototype has undergone some changes, and its development and integration status are at an advanced stage. Specifically, the component now consists of three modules, each with a specific function. The Translation module and the Metric Recommender module were available in M15. The first one implemented a set of RESTful APIs and the translation of metrics into obligations, from NL to CNL. The second one implemented the association of a TOM with a set of metrics. The functionality of the Metric Recommender is unchanged, whereas there is a new module, called API Server, which is responsible of coordinating all the modules and of implementing the API interface towards the outside.

The whole prototype with all modules is correctly deployed to the MEDINA Kubernetes cluster and is connected to the other components of the Certification Language block.

Specifically, the NL2CNL Translator connects to the *Catalogue of Controls and Metrics* through its API, to retrieve the TOMs and metrics descriptions and metadata. Several tests have been carried out to verify proper operation, and currently no errors have been found in retrieving the necessary information from the Catalogue.

After the translation, the result is stored in the CNL Store, a database managed by the CNL Editor, accessible though the CNL Editor APIs. At M15 this connection was still in a testing phase, while currently the two components are fully connected through the CNL Editor APIs. Also in this case, the tests performed did not generate errors in the creation of REOs, which are the objects managed by the CNL Store. In addition, compared with M15, the CNL Editor ontology has been updated and the NL2CNL Translator has been modified accordingly.

Lastly, the NL2CNL Translator interacts with the Orchestrator, which triggers the translation through its User Interface. To this aim, the NL2CNL Translator has been modified with respect to M15 to provide an endpoint that is invoked by the Orchestrator.

An additional feature currently available that had not yet been developed in M15 is the implementation of authentication via Keycloak server. This allows the NL2CNL Translator to verify that the user invoking its API is actually logged into the MEDINA framework.

The NL2CNL Translator is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[18].

4.2.1.1 Published APIs

The NL2CNL Translator provides a REST API that can be used by the other components interacting with it. The list of the available APIs is provided in Annex E, *Component: NL2CNL Translator and DSL Mapper*.

4.2.2 CNL Editor

CNL Editor is the component that allows a user of a Cloud Service provider to manage, with a Graphical Interface, the REO objects that are the association, in CNL format, between Requirements and policies as compiled from NL2CNL Translator. CNL Editor takes as input REO created from NL2CNL Translator and produces in output updated REO for DSL Mapper.

With the Editor Frontend, user can visualize REO, change Target Value specified for the Metrics and delete Obligations not considered suitable for CSP. Finally, the user can send the REO to the DSL Mapper with "map" operations which convert CNL obligations into Rego Code.

4.2.2.1 Implementation and Integration Status

CNL Editor is composed by these different modules:

- CNL Editor Interface: the web GUI to access CNL Editor.
- Vocabulary: a file in RDF format with extension .owl where the Ontology structures and terms needed for the Editor the control of user changes on Obligations are defined.
- CNL Editor REST API: APIs used by the Editor and eventually by other Certification Languages tools like CNL Translator and DSL Mapper for basic operations.
- CNL Store: database with Req&Obl xml files.
- Back Store Interface: REST APIs for access to the CNL Store used by CNL Editor.

CNL Editor was partially containerized on a VM standalone in M15. At the time of writing CNL Editor is implemented in a mature version and has been fully deployed in the MEDINA Kubernetes cluster. It provides both a GUI for end users and a set of RESTful APIs to interact with

[18] https://git.code.tecnalia.com/medina/public/nl2cnl-translator

it. The vocabulary used by Editor was updated to the requirements available the Catalogue in M26 (EUCS 2022 [12]).

From M15 onwards we revised the xml structure of REO to reflect the needs of MEDINA based on partners requests, and the API was also renamed and adapted to better fit the MEDINA context.

CNL Editor can be invoked from the MEDINA UI by selecting the option "Requirements & Obligations" and access is allowed by Keycloak.

CNL Editor interacts with the other tools, *NL2CNL Translator* and *DSL Mapper*, by REST APIs.

4.2.2.2 Published APIs

CNL Editor makes available APIs that can be used from other tools (e.g., create by NL2CNL Translator) to manage REO and that are listed in Annex E, *Component: CNL Editor*.

4.2.2.3 Graphical interface

CNL Editor has a Web Interface that allows a user visualizing and managing some changes to the REOs. Operations allowed for a REO are: delete obligations or change the Target Values of Obligations.

When the user invokes CNL Editor a list of REOs is shown (see Figure 23).

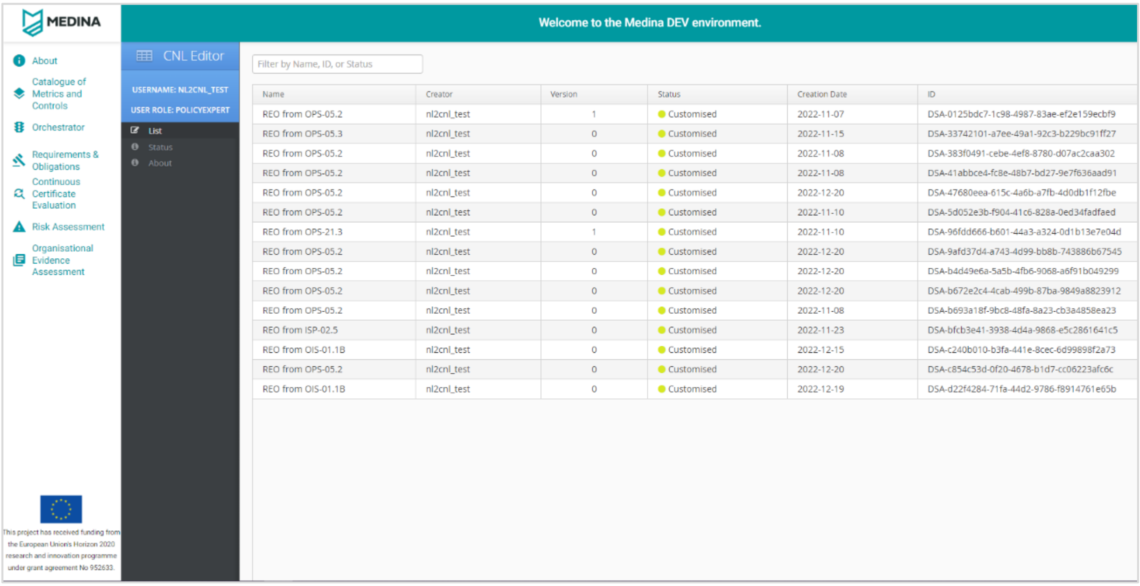


Figure 23. CNL Editor – REOs visualization

When the user selects a specific REO the window shown in Figure 24 is displayed.

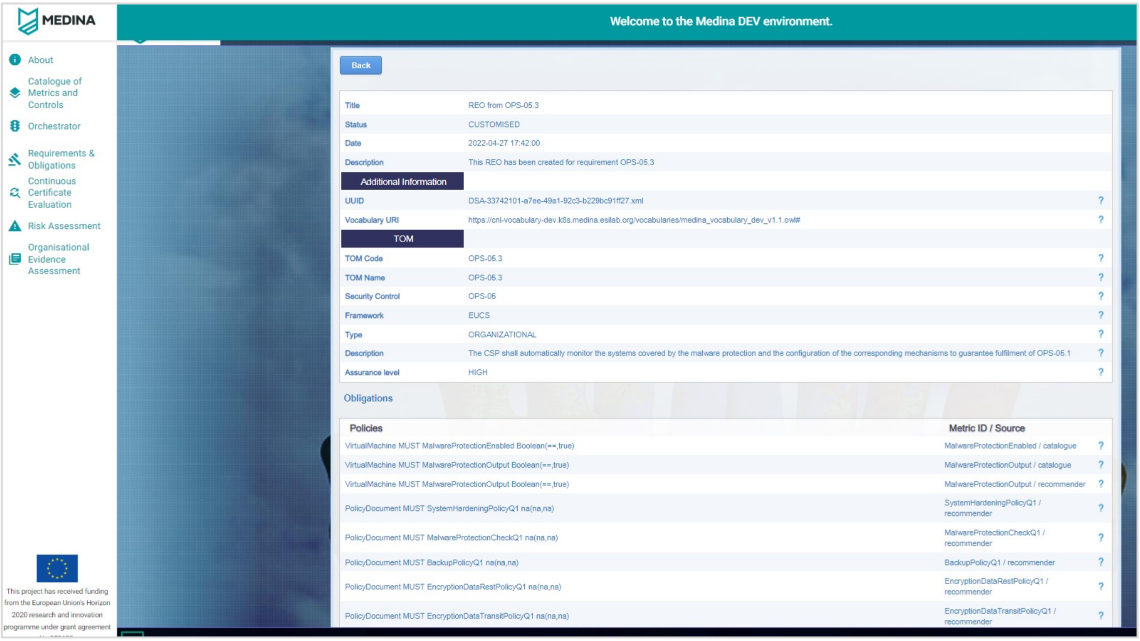


Figure 24. CNL Editor – Showing a specific REO

4.2.3 DSL Mapper

The DSL Mapper is a component of the MEDINA framework that has the aim of mapping the obligations expressed in CNL into executable policies expressed in DSL. In particular, the obligations resulting from the previous steps are embedded in an XML object, called REO, and read from the CNL Store, while the output generated by the DSL Mapper is expected to be compliant with the DSL chosen in MEDINA, i.e., the Rego language. The Rego language allows the creation of policies that can be used to automatically assess evidence, collected by the evidence collector components. The output of the DSL Mapper is sent to the *Orchestrator*, which performs the assessment of the policies.

4.2.3.1 Implementation and Integration Status

Compared with M15, several steps forward have been made regarding this component. The most important change is in the implementation of a new stand-alone prototype, whereas previously the DSL Mapper had been implemented directly in the NL2CNL Translator due to the immaturity of the prototype.

There are two main subcomponents in the DSL Mapper: the first is called API Server and is responsible for the API interface to other MEDINA components. Moreover, it coordinates all the DSL Mapper operations. The second is the Mapping component, which implements the generation of the Rego rules.

The prototype is currently deployed in the MEDINA Kubernetes cluster and is connected to the other needed components of the MEDINA framework. Specifically, the mapping functionality is triggered by the CNL Editor, through its UI. Then, the DSL Mapper uses the information stored in the CNL Store as source of data. Currently, the connection between the DSL Mapper and the CNL Editor is fully working.

After performing the mapping of obligations into Rego rules, the output is sent to the Orchestrator in order to be further processed. The latter feature is not yet fully supported and is in a beta stage, as errors sometimes occur that prevent the DSL Mapper from sending mapping results to the Orchestrator.

Similar to the other components of this block, the DSL Mapper also implements the authentication via Keycloak server.

The DSL Mapper is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[19].

4.2.3.1 Published APIs

The DSL Mapper provides a REST API that can be used by the other components interacting with it. The list of the available APIs is shown in Annex E, *Component: NL2CNL Translator and DSL Mapper*.

4.3 Risk Assessment and Optimisation Framework (block #3)

4.3.1 Risk Assessment and Optimisation Framework (RAOF)

RAOF is a service for supporting the non-conformity assessment process with a risk-based decision-making capability. This component evaluates the current risk of the CSP, by estimation of the CSP's needs and protection against possible threats. The computed risk value is used to evaluate how far is the CSP from full compliance with the selected certification scheme (and assurance level). Not only does this analysis help to identify which security requirements are missing, but also how risky it is for this CSP if these requirements are not fulfilled. By implementing these functionalities, RAOF contributes to two Key Results: KR2 (by providing the risk-aware support to a compliance manager before applying for certification) and KR6 (supporting the MEDINA's auditor, i.e., *Certification Life-Cycle Manager*, with a risk-based evaluation of detected non-conformities).

Additional details about this component are available in Deliverables D2.7 [10] and D4.4 [14].

4.3.1.1 Implementation and Integration Status

The RAOF component is used in two parts of the MEDINA process. First, the component provides the support during the bootstrapping, when a compliance manager evaluates if the cloud service could be certified (i.e., fulfill the requirements for certification). In this case, the compliance manager interacts with the RAOF directly through the GUI.

RAOF is also used during the dynamic evaluation of compliance. CCE component notifies RAOF about the requirements which have been evaluated by assessment tools and the result of these assessments. If non-conformities are detected, RAOF re-computes the risk using initially provided input and the assessment results and analyses the non-conformity gap. The result of this analysis (i.e., whether the non-conformity is to be counted as major or minor) is provided to the *Life-Cycle Manager* (LCM) for further evaluation of the status of the certificate.

In the current version all the main features are implemented. The engine for the non-conformity gap analysis is set up to compute and compare risk values for different assurance levels and different cloud market types. The computation is based on the cloud resources expected values of which should be initially provided by the CSP and the fulfilled requirements of the certification scheme. Moreover, the recently added functionality helps the compliance manager to optimise its investment in covering certification scheme's requirements to achieve at most minor non-compliance. The dynamic part implements the communication between *Continuous Certification Evaluation* (CCE) and *Life-Cycle Manager* (LCM) components and is set up to perform the risk-based non-conformity gap assessment automatically.

[19] https://git.code.tecnalia.com/medina/public/dsl-mapper

The component is integrated into the MEDINA's platform and implements the common functionalities for it. In particular, it uses the Keycloak mechanisms to authenticate users and authorise access to the risk analysis functionalities only for associated Targets of Evaluations. During the final period of the project, a more detailed use of this mechanism is envisaged, for more accurate separation of duty management.

Another functionality implemented by the RAOF is importing results of the questionnaire provided by the *Catalogue of Controls and Metrics*. This option aims to ensure that a user can report which EUCS requirements the considered service satisfies only once, but benefit from both analyses provided as by the Catalogue (compliance score) as well as RAOF (risk-based analysis of non-conformities and optimised planning for implementation of additional requirements).

Yet some work is expected to improve the component and integrate it better with other components of MEDINA. Integration with other components should undergo deeper testing (e.g., various options should be considered). Especially, integration with the dashboard of the compliance manager may require additional endpoints to be implemented to simplify the work of the compliance manager. Some modifications in the logic of the dynamic risk computation could be implemented to enhance its computation of risks per resource. The values used to set up the components and its GUI could see changes after getting the feedback from the use case providers.

The *Risk Assessment and Optimisation Framework* is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[20].

4.3.1.2 Published APIs

RAOF provides a REST API with several endpoints. This API is to be used by the compliance manager dashboard during the bootstrapping phase. All Targets of Evaluation (ToEs) managed by the RAOF are created and could be modified by *Clouditor* using this API. As well, as the CCE is supposed to invoke RAOF using a dedicated endpoint of this API.

The list of the available APIs is provided in Annex E, *Component: Risk Assessment and Optimisation Framework.*

4.3.1.3 Graphical interface

RAOF provides a GUI for the direct interaction with a compliance manager, which can be used to set up all settings for a Target of Evaluation (see Figure 25), add the list of resources and their sensitivity (see Figure 26), and report fulfilled requirements (see Figure 27).

[20] https://git.code.tecnalia.com/medina/public/static-risk-assessment-and-optimization-framework

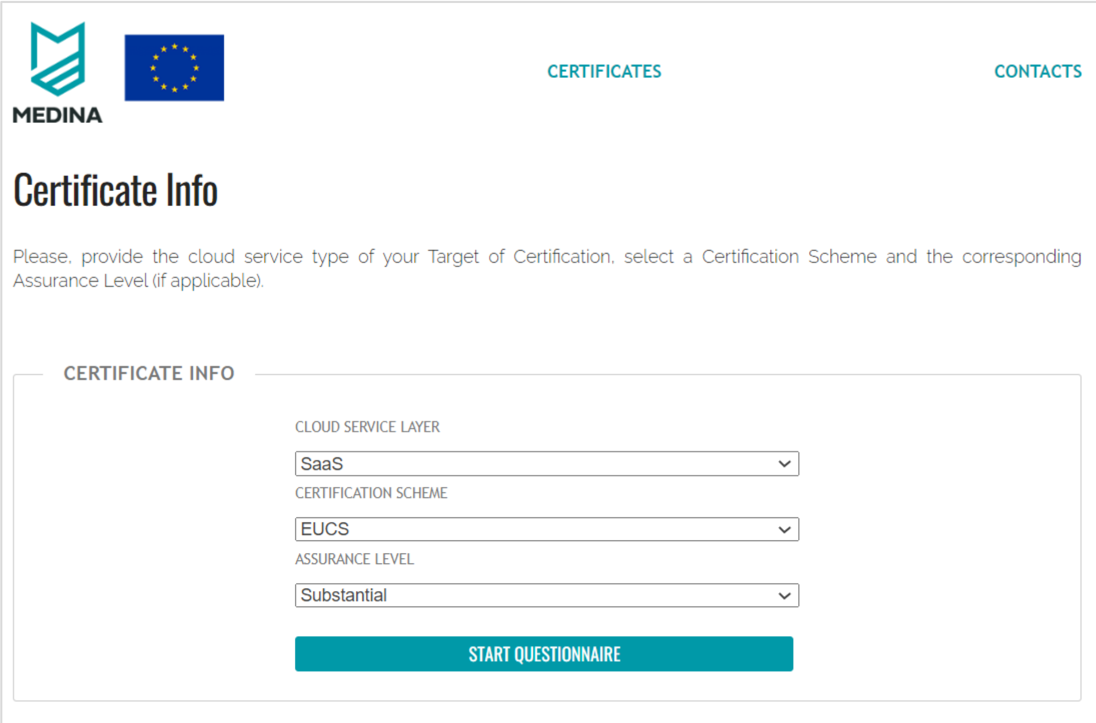


Figure 25. RAOF - Setup of Targets of Evaluation

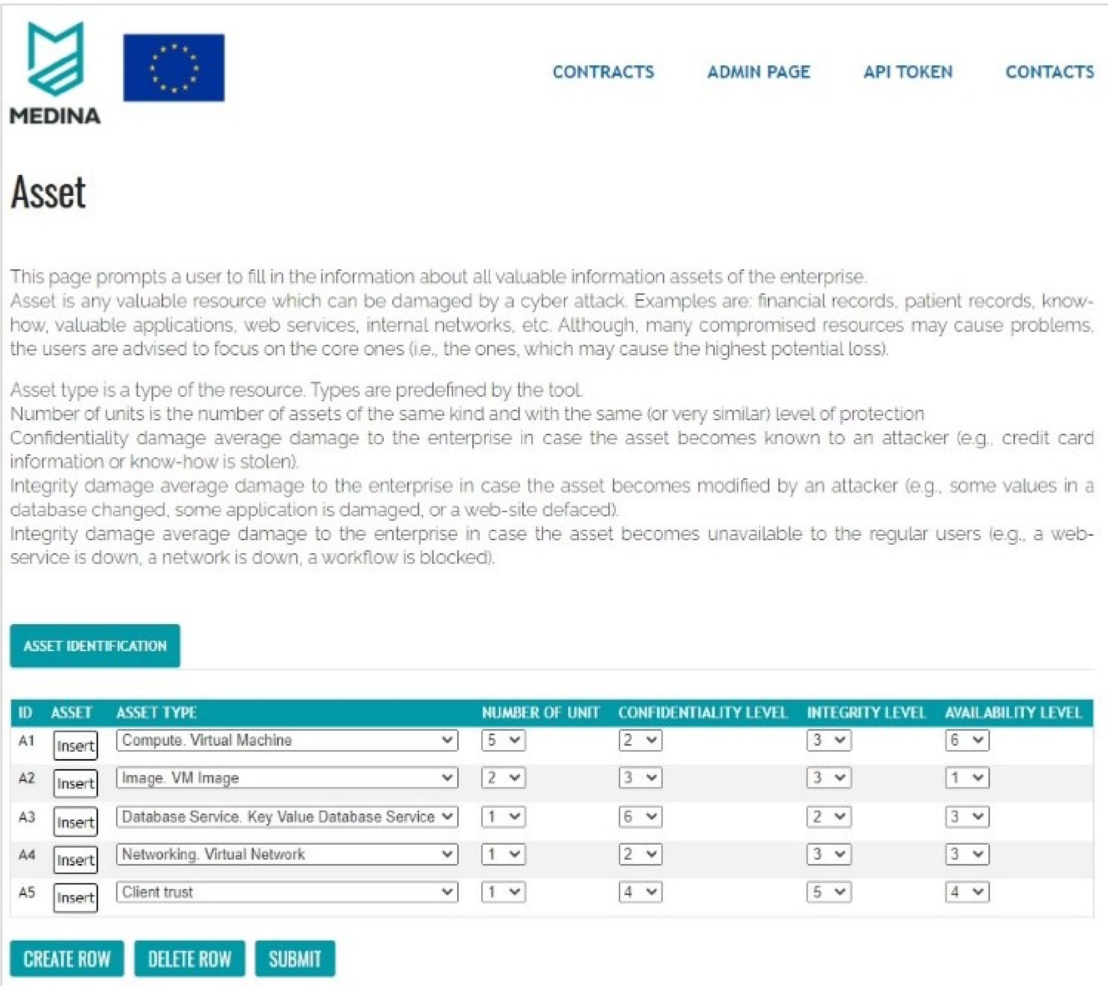


Figure 26. RAOF - List of resources

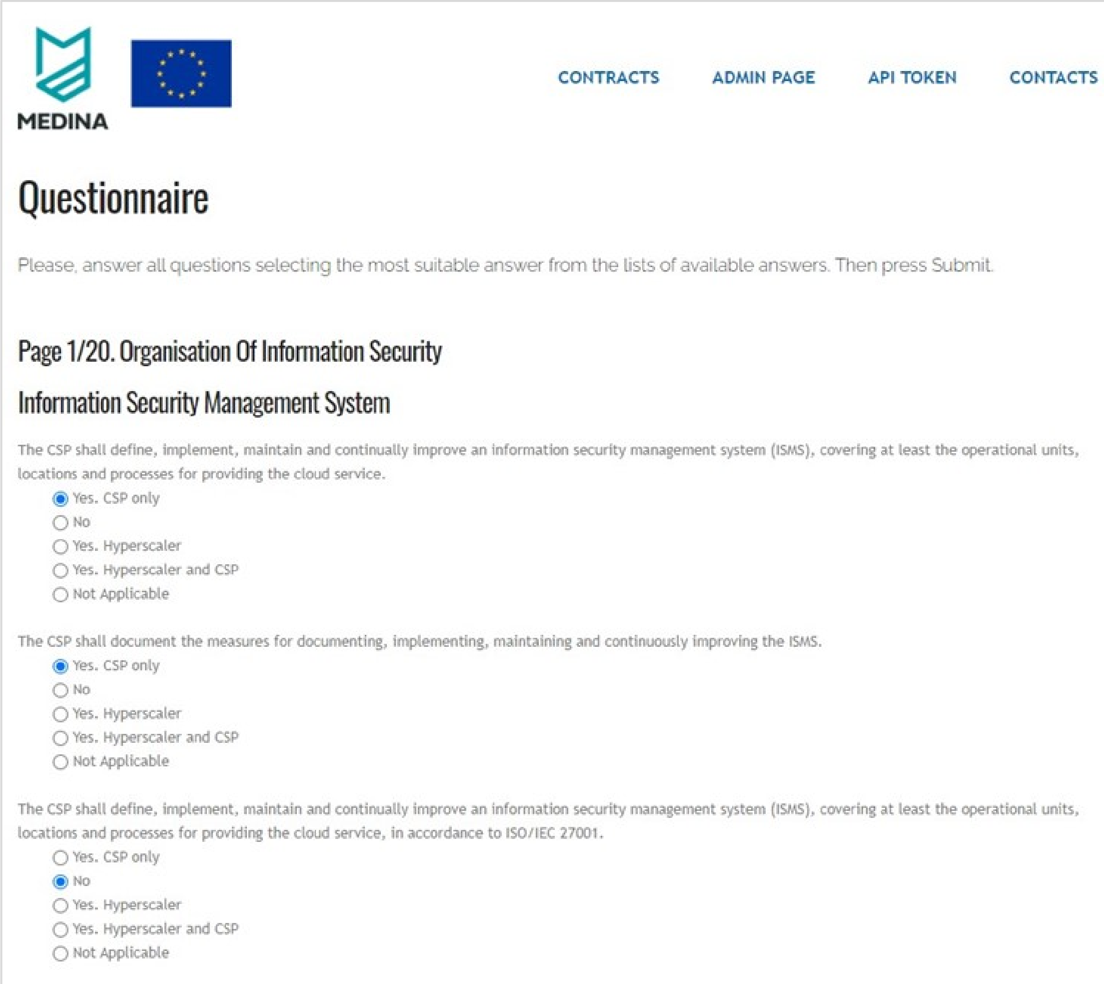


Figure 27. RAOF - Requirements to be fulfilled

Also, the GUI displays the results of the analysis and the computed risk values, as shown in Figure 28.

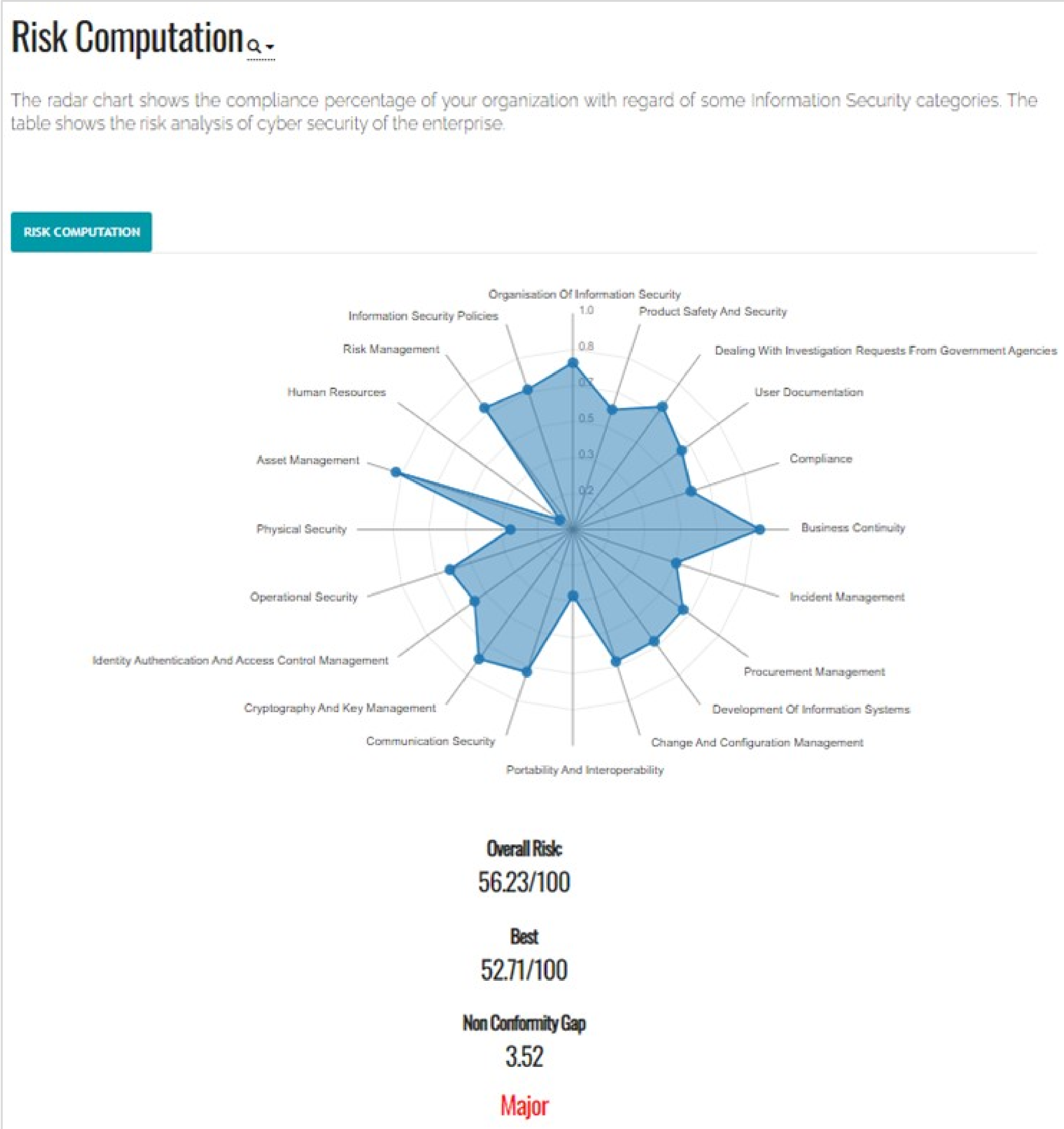


Figure 28. RAOF - Results of the analysis

4.4 Continuous Evaluation and Certification Life-Cycle (block #4)

4.4.1 Continuous Certification Evaluation

In the previous version of this report (D5.3 [1]), the CCE component only implemented the basic evaluation aggregation and supported a hard-coded sample standardisation schema. The updates of the current version with regards to D5.3 include full integrations with the MEDINA *Catalogue of Controls and Metrics*, the *Orchestrator*, and RAOF. A number of other features were also implemented, such as storing the history of past evaluation states and support for handling multiple Targets of Evaluation with a single CCE instance and calculation of operational effectiveness values. A front-end web UI component was added as well.

The components belonging to the "Continuous Certification Evaluation" are mainly related with KR5 (Continuous Cloud Certificate Evaluator, CCE), whose objective is to collect assessment results gathered by Security Assessment components and continuously build an evaluation tree representing aggregation of assessment results to determine compliance with the different controls.

Additional details about the component's architecture and methodology used is available in Deliverable D4.2 [15].

4.4.1.1 Implementation and Integration Status

All the elicited functional requirements are now implemented in the CCE. They are implemented using three microservices: CCE core (back-end), CCE UI (front-end) and MongoDB database.

The *Continuous Certification Evaluation* component (CCE) receives assessment results gathered by Security Assessment components through the *Orchestrator* and continuously builds an evaluation tree representing the aggregation of assessment results to determine compliance with the different certification elements.

Beside the assessment results, CCE also receives data about the Cloud Services and related Targets of Evaluation from the *Orchestrator*. Another required input is the structure of the evaluation scheme used (relations between metrics, requirements, controls, categories) that is obtained from the MEDINA *Catalogue of Controls and Metrics*.

Outputs of the CCE are consumed by the *Risk Assessment and Optimisation Framework* (RAOF) and the *Life-Cycle Manager* (LCM). CCE periodically sends the changed values of the evaluation tree to RAOF for the risk-based evaluation of the severity of incompliances. The LCM queries the CCM's API to obtain operational effectiveness values which help determine the overall certification state.

The evaluation aggregation is implemented for multiple Targets of Evaluation (multi-tenancy support), history of evaluation tree states is being stored in a database and is exposed through an API, the operational effectiveness values are being calculated and integration with all components needed for the complete functionality is complete. A web user interface of the component is also implemented with minor updates still pending.

The authentication and authorization of users (using Keycloak) are not yet implemented.

The source code of both the CCE core[21] (back-end) and the UI[22] (front-end) is available on the public GitLab repository. Dockerfiles are available for simple deployment and integrated with the project's development and testing environments on Kubernetes.

4.4.1.2 Published APIs

CCE exposes two APIs:

- HTTP REST-like API, mainly used for communication with the web front-end (UI)
- gRPC API, for communication with the *Orchestrator* and the *Life-Cycle Manager*

Details of both these APIs are described in Annex E, *Component: Continuous Certification Evaluation*

[21] https://git.code.tecnalia.com/medina/public/continuous-certification-evaluation
[22] https://git.code.tecnalia.com/medina/public/cce-frontend

4.4.1.3 *Graphical interface*

The CCE frontend provides a tree visualization of the assessment results (as shown in Figure 29). Additional web UI updates are still pending.

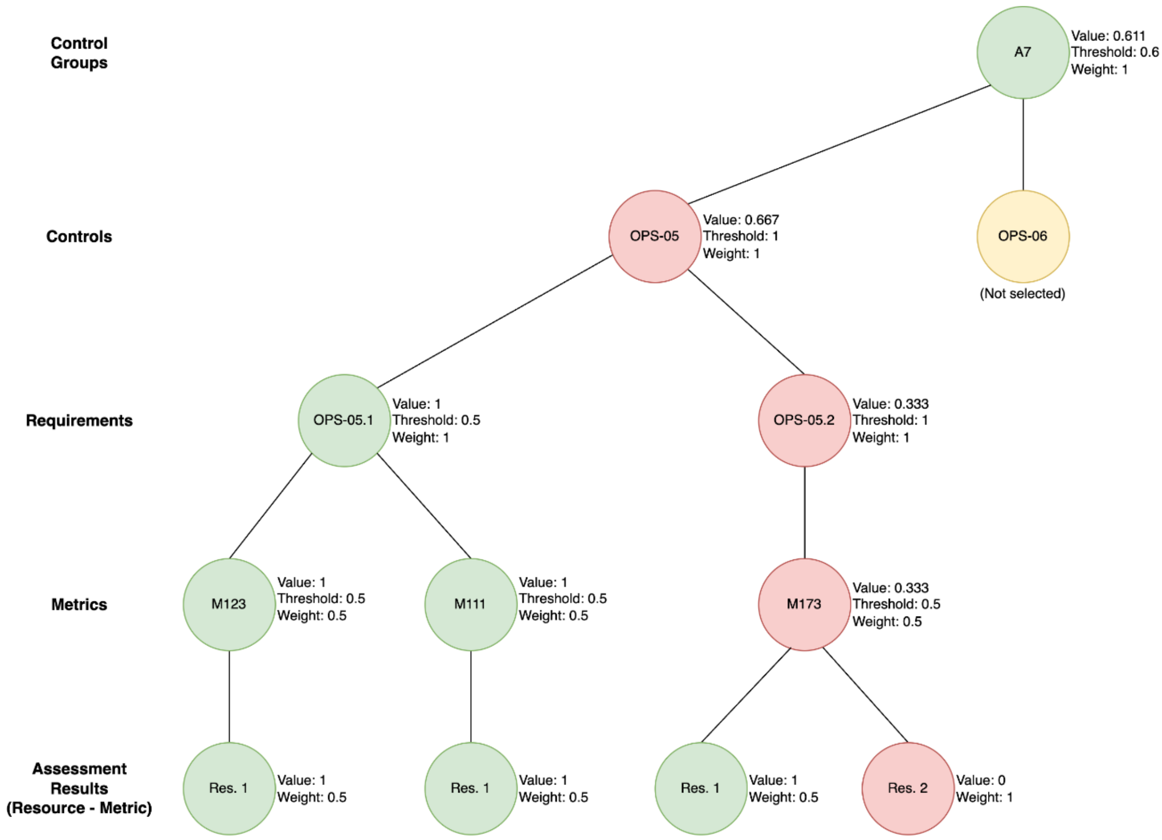


Figure 29. Sample assessment results tree

4.4.2 Automated Certificate Life Cycle Manager

The *Automated Certificate Life Cycle Manager* (LCM) integrates different sources of information to decide on the certificate status. As such, it presents the final step of the MEDINA framework, consolidating all assessments/evaluations into one result. It addresses KR6 (Risk-Based Auditor Tool).

4.4.2.1 *Implementation and Integration Status*

The LCM is integrated with the components mentioned above: the integration with the CCE is implemented via a periodical HTTP request that retrieves the operational effectiveness data. Risk-based evaluations are reported to the LCM by the RAOF after every evaluation, e.g., every five minutes. This connection is also implemented via an HTTP API. A further HTTP request is sent to the *SSI Framework* component. Currently, this request is only sent if the certificate is suspended or withdrawn, since only in this case should the CAB review the available evaluation results and possibly issue a new certificate with a changed status. The connection to the Orchestrator is implemented using gRPC and serves the purpose of storing certificate data including their state history in a permanent storage.

As inputs, it obtains information from the *Continuous Certification Evaluation* component which calculates data on operational effectiveness, i.e., compliance data over a certain time frame. Additionally, it obtains information from the *Risk Assessment and Optimisation Framework*, which provides a risk-based evaluation of deviations present in the cloud service.

As outputs, the LCM provides the certificate status to the *Orchestrator* to be saved in a database and presented in the Orchestrator UI. Furthermore, it reports to the *SSI Framework*, since it is possible that a new certificate credential needs to be issued. For more details, see deliverable D4.2 [15].

The Automated Certificate *Life Cycle Management* is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[23].

4.4.2.2 Published APIs

The Automated Certificate Life Cycle Management APIs are listed in Annex E, *Component: Life Cycle Manager.*

4.4.2.3 Graphical interface

The status of existing certificates and their histories are presented in the Orchestrator UI. Figure 30 shows an example of this.

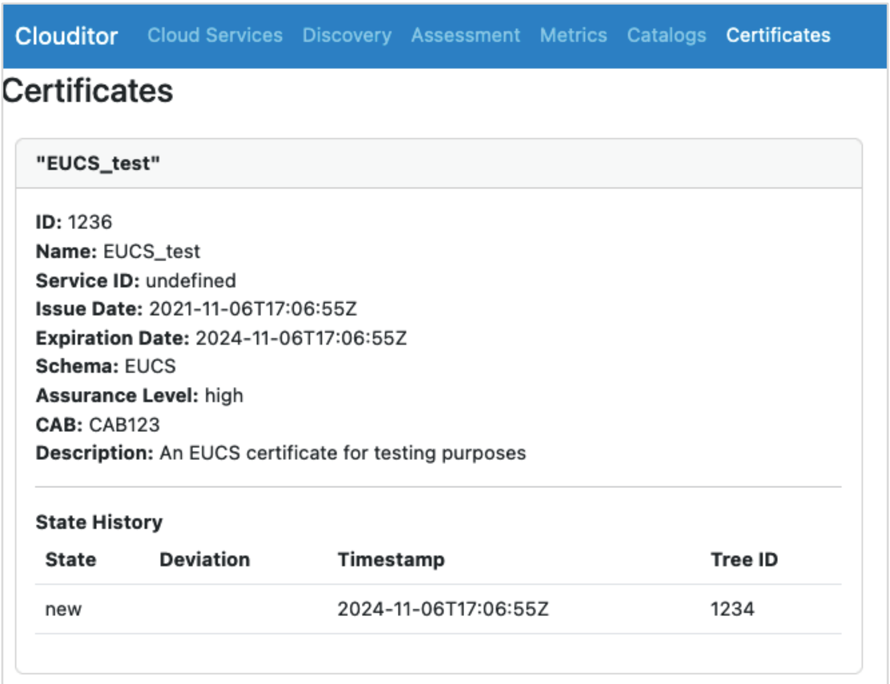


Figure 30. Screenshot of the certificates overview presented in the Orchestrator UI

4.4.3 Automated Self-Sovereign Identity-based certificates management (SSI)

The *Self-Sovereign Identity (SSI) Framework* provides the CSPs with the capability to manage their own security certificates as part of their identity through verifiable credentials. “To manage their own identity” ultimately means that they store their identity on their own “user space” without intervention of a third-party.

The *SSI Framework* is not only composed of the CSP component to store and control the credentials. It is also composed of the issuer component which provides the CAB a way to issue verifiable credentials about the security certificates related to the CSP; and the client´s component which provides a way to ask and verify proofs of different security certificate

[23] https://git.code.tecnalia.com/medina/public/life-cycle-manager

features. In this sense, privacy is an important requirement within MEDINA, as several security certificate features are considered sensitive and must be treated carefully. The *SSI Framework* is capable of sharing sensitive information in a confidential way by keeping user´s identity out of third parties, which act as identity silos, reducing the risk of identity theft; but also, by using Zero-Knowledge Proofs (ZKPs). ZKPs preserve user´s privacy using cryptography to proof that a CSP has some attributes without disclosing these attributes.

The *SSI Framework* is part of KR5 (Cloud Certificate Evaluator). Details about this component are available in deliverable D4.2 [15].

4.4.3.1 Implementation and Integration Status

A first prototype of the *Self-Sovereign Identity (SSI) Framework* has been implemented since M15 completely covering its functionality. It is composed by one SSI-network, three SSI-agents (issuer, holder and verifier, for the complete SSI flow) and two SSI-webapps (one for the holder and another one for the issuer and verifier).

The SSI-network, two of the SSI-agents (issuer and verifier), one of the SSI-webapps (the one for the issuer and verifier) and the SSI-API are provided as a service by TECNALIA emulating the CAB and a potential CSP customer. All these components are correctly deployed and integrated with each other. Additionally, the SSI-API is also correctly integrated with the LCM for receiving the certificate state after the MEDINA framework execution.

Additionally, one SSI-agent (holder) and one SSI-webapp (the one for the holder) are correctly deployed on the MEDINA environment and are correctly integrated with Keycloak. These components are also integrated with the rest of the SSI components. No integration with additional components is needed in this case. The associated functional requirements are fully covered.

4.4.3.2 Published APIs

The SSI-API component of the SSI Framework exposes an API described in detail in Section 6.3.2.3 in D4.2 [15]. The list of these APIs is also available in Annex E, *Component: Automated Self-Sovereign Identity-based certificates management (SSI)*.

4.4.3.3 Graphical interface

The *Self-Sovereign Identity (SSI) Framework* is controlled by means of a web-app application. Figure 31 shows an example of the graphical interface.

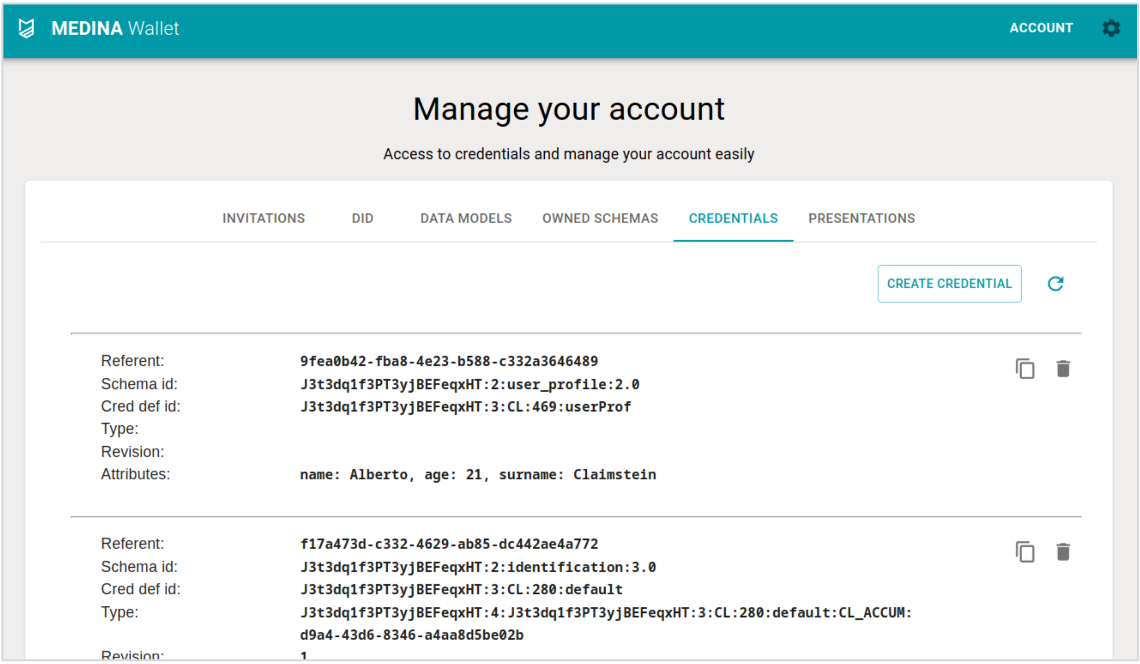


Figure 31. MEDINA Self-Sovereign Identity (SSI) Framework graphical interface

4.5 Organizational Evidence Gathering and Processing (block #5)

4.5.1 Organizational Evidence Gathering and Processing

The *Assessment and Management of Organizational Evidence* (AMOE) component extracts and collects evidence from policy documents. The component is addressing the NLP and organizational measure aspects of KR4 (Continuous Evidence Management Tools). It can compute pre-assessments (hints) that can be used to speed up the audit process. After uploading a document, the component extracts the evidence for a set of organizational metrics with the help of the built-in Natural Language Processing (NLP) pipeline.

The processed data can be analyzed in the UI and assessment results can be set/confirmed. Once complete, the assessment results can be forwarded to the *Orchestrator* on demand.

Additional details about this component are available in deliverables D3.2 [16] and D3.5 [17].

4.5.1.1 Implementation and Integration Status

The main implementation of AMOE started in M15. Current status of the implementation consists of one component made up by the webservice for the user interface (UI) and REST API (e.g., for the CCD).

For the evidence gathering functionality the following subprocesses have been implemented. Pre-processing for PDF to transform unstructured policy documents into semi-structured content usable for faster and more accurate extraction. The evidence extraction pipeline itself, which consists of one main method (keyword-based approach) that is used by default. For research purposes three other similar evidence extraction pipelines have been built, however, tests have shown they would need additional work. All evidence extraction approaches make use of standard NLP techniques and utilize the pre-trained question answering system roberta-base-squad2[24].

[24] https://huggingface.co/deepset/roberta-base-squad2

The integration of the component into the MEDINA framework uses the API of the *Catalogue of Controls and Metrics* and has a hardcoded fall back to a static metric file if the connection would fail. Furthermore, the connection to the *Orchestrator* for metric implementation details and sending assessment results and evidence has been implemented.

To store the metadata, logging and extracted evidence internally, a connector to internal data base (MongoDB) has been added. The user action information (on edit/upload/delete/submit) is logged into the data base.

User authentication is done via the MEDINA Keycloak service and respective component client. Role based access (Keycloak roles) as well as filtering of information based on cloud service information in the authentication token has been implemented. A dockerfile and kubernetes configuration for deployment of webservice, db and redis cache have been created.

A quality check pipeline for manual checks on the status and aid of research tasks for evidence extraction has been implemented. It enables comparison of annotated information in the tool Inception[25] to the evidence extraction approaches.

The AMOE component is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[26].

4.5.1.2 Published APIs

The AMOE APIs are listed in Annex E, *Component: Assessment and Management of Organizational Evidence – AMOE*.

4.5.1.3 Graphical interface

AMOE provides a GUI for users to interact (see Figure 32). The following access types are defined in M27, configurable through the Keycloak authentication token roles:

- no access
- read only access
- upload/delete files or stop running processes
- edit/submit assessment results
- admin (full read/write access)

[25] https://inception-project.github.io/
[26] https://git.code.tecnalia.com/medina/public/amoe

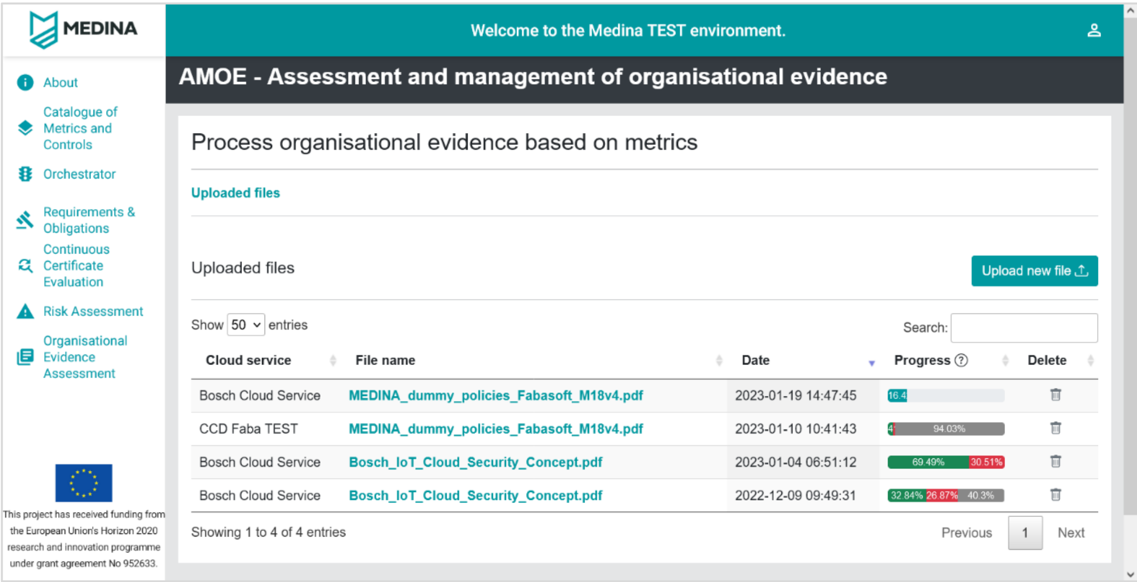


Figure 32. AMOE landing page

4.6 Orchestrator and Databases (block #6)

4.6.1 Orchestrator and Databases

The *Orchestrator* is the central interface in MEDINA and manages evidence and assessment results. It provides access to databases, forwards data between components, and provides the main user interface. As such, it is an essential component in the integrated MEDINA framework. It addresses KR4 (Continuous Evidence Management Tools).

For more details, please refer to deliverables D3.2 [16] and D3.5 [17].

4.6.1.1 Implementation and Integration Status

The *Orchestrator* provides a graphical user interface which, among others, allows to view assessment results, register cloud services, and manage the evaluation of cloud services. It is integrated with the evidence collection tools (including AMOE), the security assessment tools, the *Continuous Certification Evaluation*, the *Catalogue of Controls and Metrics*, the Keycloak component, and the *DSL mapper*. It also provides a persistent (and in-memory) database.

The *Orchestrator* component is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository [27].

4.6.1.2 Published APIs

The Orchestrator implements numerous APIs, since it is integrated with many components. Please see Annex E, *Component: Orchestrator* for an overview[28].

4.6.2 Trustworthiness System

The *MEDINA Evidence Trustworthiness Management System* provides a secure storage for evidence and assessment results hashes. It is implemented through Smart Contracts backboned

[27] https://git.code.tecnalia.com/medina/public/orchestrator

[28] An up-to-date API specification can also be found on GitHub: https://github.com/clouditor/clouditor/blob/main/openapi/orchestrator/openapi.yaml

by a common Blockchain network for all the MEDINA framework instances, providing the following functionalities:

- Includes the logic for all Orchestrator instances in MEDINA to provide the required information to be audited (about evidence and assessment results). For this purpose, an API is exposed by the Blockchain client.
- Provides secure long-term information recording, thanks to the inherent advantages of Blockchain (integrity, decentralization, authenticity…).
- Includes the logic for external users to access MEDINA's audited information (about evidence and assessment results) in a graphical and user-friendly way through a kibana-based dashboard.

The *MEDINA Evidence Trustworthiness Management System* is part of the KR4 (Continuous Evidence Management tools). Details about this component are available in D3.2 [16].

4.6.2.1 Implementation and Integration Status

The *MEDINA Evidence Trustworthiness Management System* is almost completely implemented and integrated in M27. It is composed by three main components: Blockchain ledger and deployed Smart Contracts, Blockchain monitor provided as a service from TECNALIA, and Blockchain client needed by the *Orchestrator* to interact with the Blockchain. The integration between the Blockchain client and the Orchestrator component by means of an API has been improved since M15 fixing some bugs and including error management.

Keycloak is not needed by the *MEDINA Evidence Trustworthiness Management System* because it includes its own user management functionality as it is provided as a common service to different use-cases and users. For this reason, users and roles are not limited to those defined in MEDINA Keycloak.

The associated functional requirements are almost covered except for the trustworthiness guaranteeing capabilities by extracting checksums from DLT and comparing with current checksums to detect modifications. Manual ways have been defined but automatic ways are under consideration. Usability and security have been highly improved since M15.

4.6.2.2 Published APIs

The Blockchain client exposes an API described in detail in Appendix C of D3.2 [16]. The list is also available in Appendix E, *Component: Trustworthiness System.*

4.6.2.3 Graphical interface

The MEDINA Evidence Trustworthiness Management System exposes a Kibana-based graphical interface available at: https://medina.bclab.dev/ [authentication required]. For more details, refer to Section 4.2.2.5 in D3.2 [16].

Figure 33 shows an example of dashboard of the graphical interface.

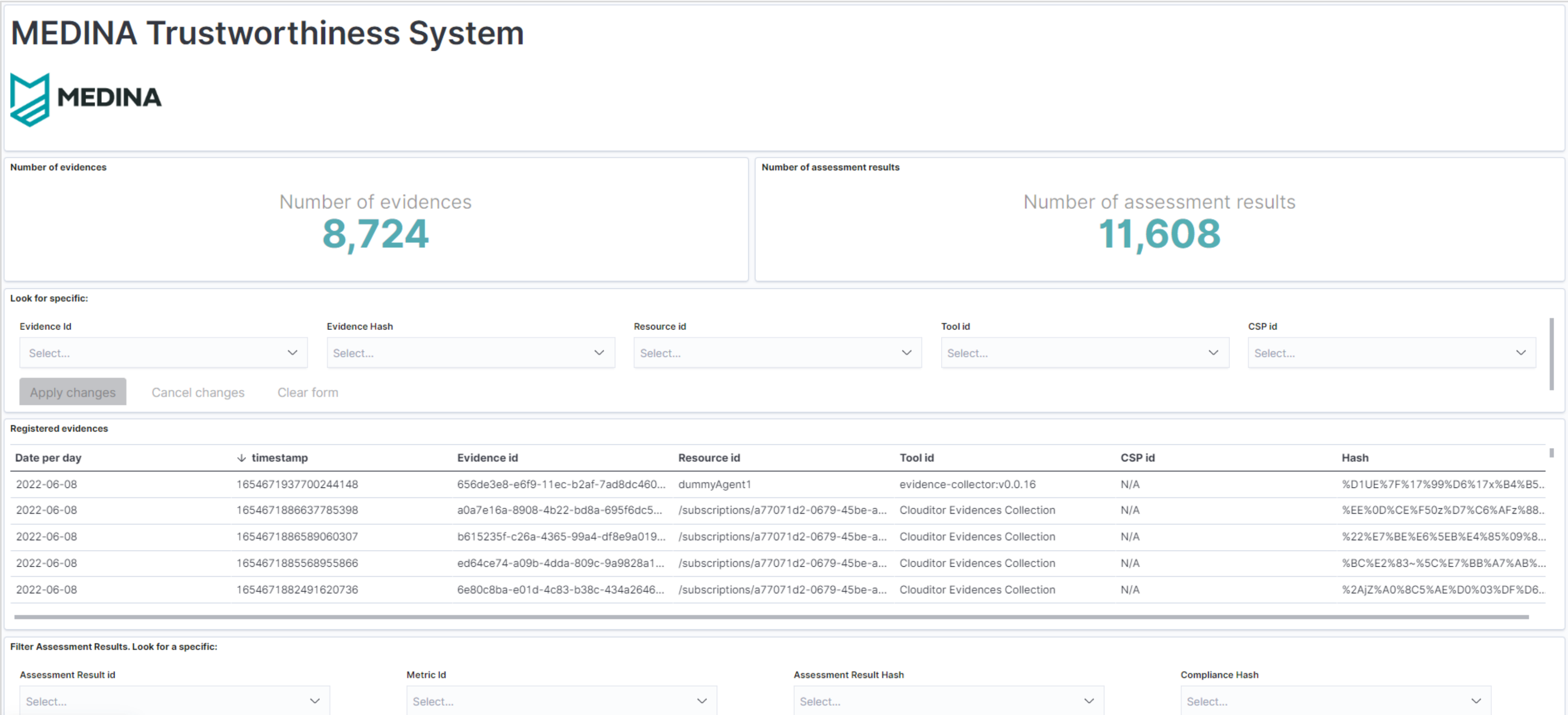


Figure 33. MEDINA Evidence Trustworthiness Management System graphical interface

4.7 Evidence Collection and Security Assessment (block #7)

4.7.1 Evidence Collection

4.7.1.1 Evidence Collection (Clouditor Discovery)

The evidence collectors form the first automated step in the MEDINA evidence pipeline. They scan a certain resource and compile information about it to be assessed by the *Security Assessment*. The *Cloud Evidence Collection* provided by *Clouditor* discovers existing cloud resources, e.g., from Microsoft Azure systems, and retrieves information about them. It then creates a MEDINA evidence and sends it to the Security Assessment. This component addresses KR4 (Continuous Evidence Management Tools).

For more details, please refer to deliverables D3.2 [16] and D3.5 [17].

4.7.1.1.1 Implementation and Integration Status

The *Clouditor Evidence Collection* implements all requirements defined in D5.2 [2]. Still, we will extend its functionality regarding the cloud resource types it can discover. It is furthermore integrated with the *Security Assessment* to which it sends the evidence, as well as with the *Orchestrator* which receives the raw evidence to be stored in a database.

The Evidence Collection components are now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[29].

4.7.1.1.2 Published APIs

The Evidence Collection offers two APIs: One for starting the discovery, and one for retrieving the evidence collected in the last iteration. See also Annex E, *Component: Evidence Collection (Cloud Discovery)*.

4.7.1.2 Evidence Collection (Wazuh)

Wazuh [18] is a host-based intrusion detection system that features several modules for threat detection, integrity monitoring, incident response, and basic compliance monitoring. It is deployed on individual machines in the CSP's infrastructure and gathers data about security-related events on these machines. An additional component, the *Wazuh & VAT Evidence Collector* is used to connect Wazuh with the rest of the MEDINA framework by querying Wazuh and producing evidence based on its state and reported events. While Wazuh is a standalone component, *Wazuh &VAT Evidence Collector* functions as a microservice within the MEDINA framework.

Wazuh addresses the KR 4 (Continuous Evidence Management Tools).

Additional details about this component are available in deliverables D3.2 [16] and D3.5 [17].

4.7.1.2.1 Implementation and Integration Status

All requirements defined [3] for evidence collection with Wazuh are implemented. The evidence is currently produced for a limited number of metrics, which is planned to be extended. It is integrated (through the *Wazuh & VAT Evidence Collector*) to the *Security Assessment* component to which it sends the produced evidence.

[29] https://git.code.tecnalia.com/medina/public/cloud-evidence-collector

The *Evidence Collection* component is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[30].

4.7.1.2.2 Published APIs

There are no APIs exposed externally (to other MEDINA components). Internally, Wazuh publishes an API for querying its state which is used by the *Wazuh & VAT Evidence Collector*.

4.7.1.3 Evidence Collection (Vulnerability Assessment Tools)

Vulnerability Assessment Tools (VAT) act as a vulnerability scanning and detection framework. The component incorporates multiple web application scanning tools that can be configured to periodically scan the CSP's services in testing or in production environments and report about detected vulnerabilities. It also provides capabilities to run user-provided vulnerability detection scripts which can be used with VAT to produce MEDINA-compliant evidence.

VAT address the KR4 (Continuous Evidence Management Tools).

Additional details about this component are available in deliverables D3.2 [16] and D3.5 [17].

4.7.1.3.1 Implementation and Integration Status

Similar to Wazuh, VAT is also connected to MEDINA by means of the *Wazuh & VAT Evidence Collector* component. Currently, evidence can be produced for custom user-provided vulnerability detection scripts with a configurable metric identifier.

The *Evidence Collection* component is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[31].

4.7.1.3.2 Published APIs

No APIs are externally exposed by VAT. Internally, VAT exposes an API to provide information about the configuration and results of all scheduled and completed tasks. This API is used by the *Wazuh & VAT Evidence Collector* to produce evidence based on the state of VAT. The evidence is forwarded to the Security Assessment component (*Clouditor*).

4.7.1.4 Security Assessment (Clouditor)

Once the evidence has been collected, it must be assessed regarding the requirements specified in the respective certification catalogue. The *Security Assessment* first obtains pre-defined metrics data and policies from the *Orchestrator*. It then uses this data to assess incoming evidence regarding their compliance with the metric data. Assessment Results are the output of this component and include the compliance state, resource ID, and other information that enable auditors to trace a non-compliance to its exact source. It addresses KR4 (Continuous Evidence Management Tools) and KR5 (Cloud Certificate Evaluator).

For more details, please refer to deliverables D3.2 [16] and D3.5 [17].

[30] https://git.code.tecnalia.com/medina/public/wazuh-vat-evidence-collector
https://git.code.tecnalia.com/medina/public/wazuh-deploy
[31] https://git.code.tecnalia.com/medina/public/wazuh-vat-evidence-collector
https://git.code.tecnalia.com/medina/public/vat-deploy
https://git.code.tecnalia.com/medina/public/vat-genscan

4.7.1.5 Implementation and Integration Status

The *Security Assessment* component currently implements all mandatory requirements as defined in deliverable D5.2 [2]. It is integrated with the *Evidence Collection* and the *Orchestrator*, and therefore implements all necessary integrations. These also include the integration with the Keycloak component.

The *Security Assessment* component is now Open Source with license Apache 2.0 and the source code is available on the public GitLab repository[32].

4.7.1.6 Published APIs

The *Security Assessment* offers two APIs: one for providing evidence to be assessed, and one for querying assessment results. See also Annex E, *Component: Security Assessment (Clouditor)*.

[32] https://git.code.tecnalia.com/medina/public/security-assessment

5 MEDINA Integrated User Interface (block #8)

This section provides an in-depth description of the MEDINA Integrated User Interface (IUI).

5.1 Implementation

5.1.1 Functional description

The goal of the tool is to provide a main access point for the MEDINA Framework: it integrates with existing authentication and guides users based on their authorization level to the user interfaces of specific components.

5.1.1.1 Fitting into overall MEDINA Architecture

The MEDINA Framework is developed with a microservices architecture. Thus, each component implements its own Graphical User Interface (GUI). For this reason, the MEDINA Framework GUIs are separated, and the final users need a leading thread that makes it easier to navigate through content. The MEDINA Integrated UI integrates all these GUIs into a single and organized entry point.

5.1.2 Technical description

In order to facilitate independent team frontend development of functionalities, the architecture chosen for this implementation is “micro-frontends” [19]. This kind of architecture allows to embed in a main frontend component (Integrated UI) any other UI in the framework regardless of the underlying technology.

5.1.2.1 Prototype architecture

The following diagram describes a simplified architecture from the Integrated UI perspective. The client lands on the Integrated UI and then the user can navigate to the GUI provided by other components. These components need to implement a Keycloak adapter in order to enforce authentication.

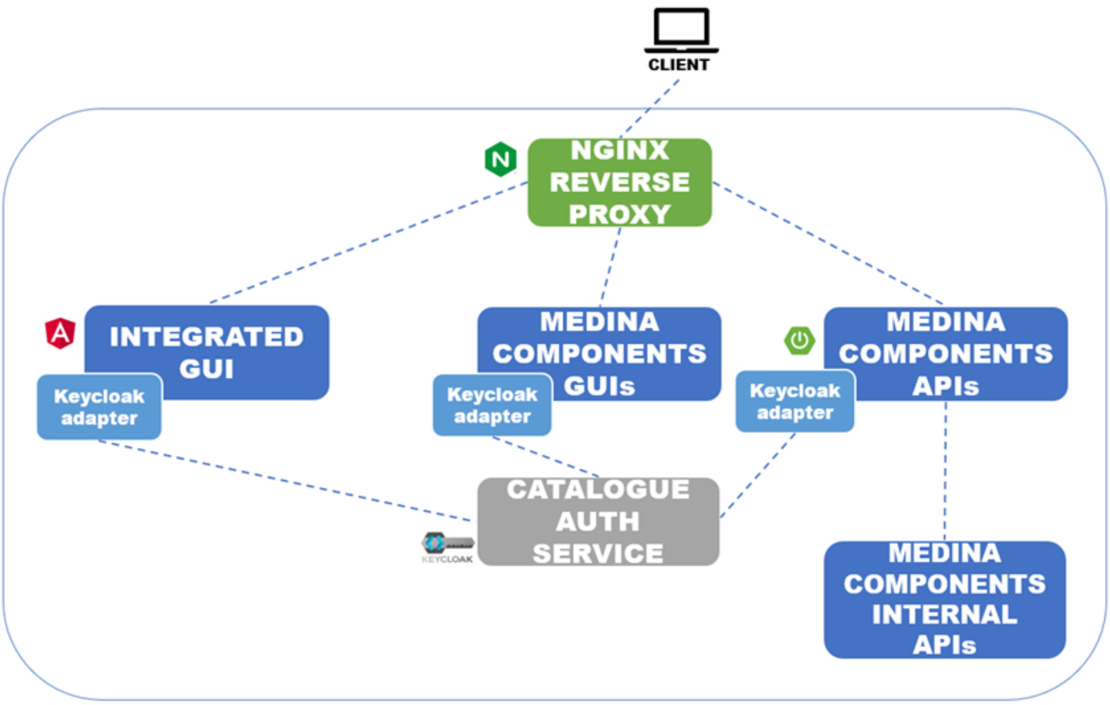


Figure 34. MEDINA UI Architecture

5.1.2.2 Description of components

5.1.2.2.1 Authentication and authorization

Authentication is managed by Keycloak[33], which is a standalone component based on an open-source solution. It provides a UI and, with due initial configuration, advanced authentication and authorization capabilities, including SSO, Identity Brokerage and role mapping. Every component implements a "Keycloak adapter" which acts as an HTTP interceptor and checks on resources requests whether:

- The client requesting user authentication is a registered client
- The user is authenticated, if not it redirects to login page
- The user is authorized for the requested resource based on its role on Keycloak configuration, if not it redirects to an appropriate error page

Once a user is authenticated, a JWT is provided which contains user information and roles. It allows us to implement in a safe way features like conditional formatting and routing based on user's role. For example, a CSP wouldn't see what concerns an Auditor accessing the same panel.

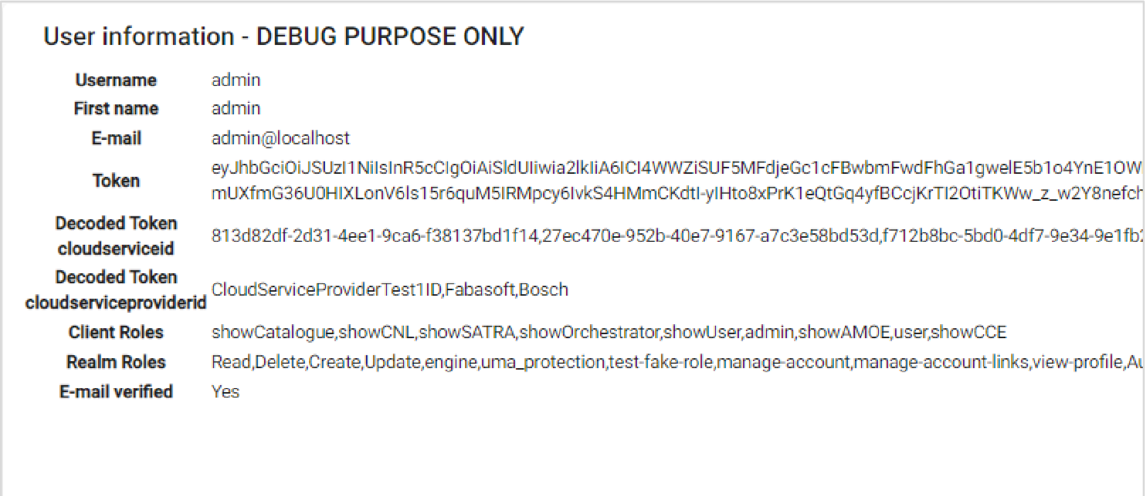


Figure 35. Bearer Token Fields

As shown in Figure 35, the token provides user related information available to the components that are being accessed. In particular during this second round we took advantage of it in order to provide the cloud services and cloud service providers references that are linked to the current user.

[33] https://www.keycloak.org/

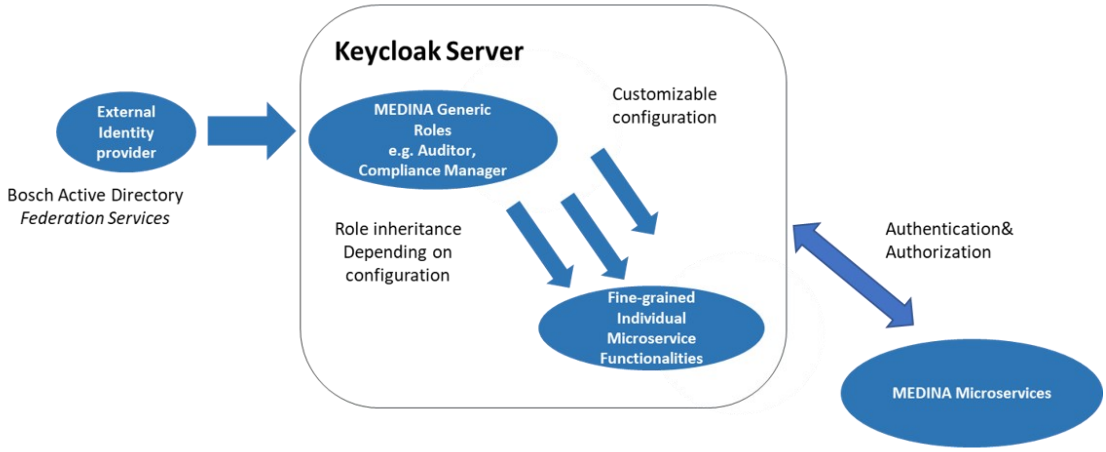


Figure 36. Keycloak server

In MEDINA, the Keycloak identity management server acts as:

- A source of truth for authentication and authorization of users and microservices communication
- An identity broker for existing enterprise identity providers
- Based on OpenID Connect standard

Microservices take advantage of Keycloak adapters to communicate with Keycloak server. Each microservice has its corresponding configuration on Keycloak server.

As a result, we have successfully integrated the Enterprise Identity Provider authentication provided by the UC1 (Bosch Active Directory) in M27.

5.1.2.2.2 Integration of components

Table 17 shows the list of components integrated in M27 in the MEDINA integrated UI and the chosen integration strategy. Respect to M15, during this period we integrated four new components. These new integrated components are highlighted in light green in Table 17.

Table 17. Integration strategy for the different MEDINA components

Component name	Integration strategy
Catalogue of Metrics and Controls	Iframe
Orchestrator* *This integration has been set up, but will be finalized in the next phase	Iframe
CNL Editor	Iframe
Continuous Certificate Evaluation	Iframe
Risk Assessment and Optimisation Framework	Iframe
Keycloak	Rest API
Organizational Evidence Gathering and Processing	Iframe

5.1.2.3 Technical specifications

The prototype is developed using Angular 12 [20], a modern typescript framework that allows us to build high-performance, scalable, component-based single page web applications The framework is enriched with Angular Material 2 library [20], a set of high quality animated responsive components that follow Material Design UI specifications. The application runs on a Nginx web server [21].

Integration of micro-frontends is obtained through Iframes and REST API. In particular, since the micro-frontends are deployed in the Kubernetes cluster, we are able to integrate them by providing the URL of the component and update automatically the referred services in the application, with great benefits to productivity.

Web application source code is packaged as ES flattened module and added to a Nginx:alpine image, in order to containerize it.

5.1.2.4 User Interface structure

In this section we present the authentication provided by the MEDINA Integrated UI.

5.1.2.4.1 Login, authentication and Iframe embedding

Unauthenticated users that try to access the integrated-UI are redirected to Keycloak's login page (see Figure 37). We collaborated with UC1 in order to integrate the possibility to authenticate with the external identity provider provided by Bosch in the MEDINA Login page. Thanks to this approach a user registered into the Bosch Active Directory can be also recognized in the MEDINA Framework.

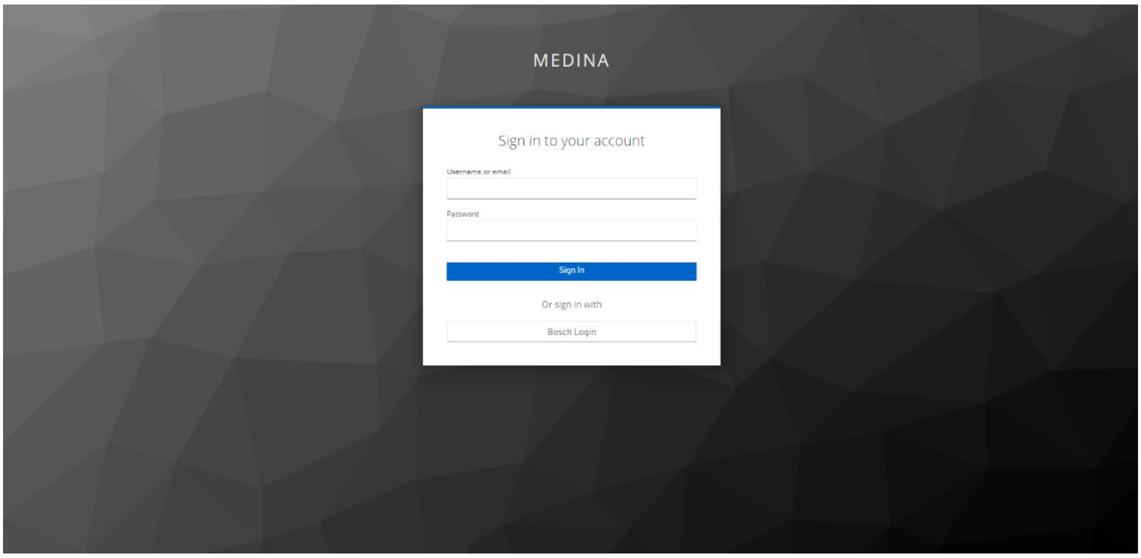


Figure 37. Keycloak Login Page

After inserting correct credentials, users are redirect to the page that the request was originated from (see Figure 38).

The UI is composed of a fixed top navigation bar and a dynamic lateral navigation bar, so that the latter can be hidden or shown depending on screen size. Main content is rendered inside the container by Angular Routing Component, depending on the requested endpoint. For example, path /frame renders an Iframe component which embeds a different application.

In the following example, the Integrated-UI embeds *Catalogue* dashboard. As explained before, the authentication is received correctly by the embedded component, without the need to log-in again.

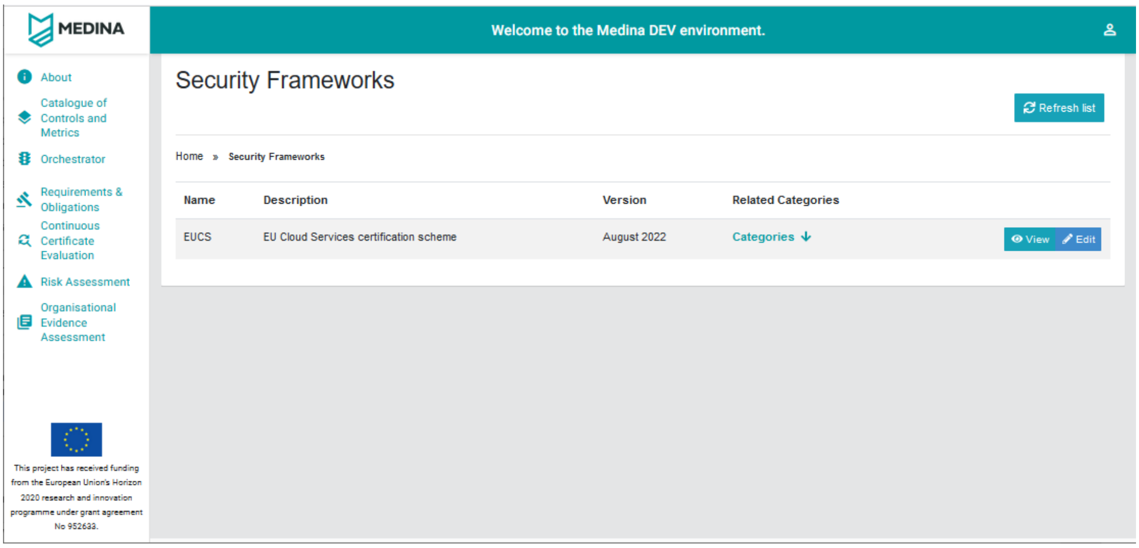


Figure 38. FullScreen IFrame Embedding - Catalogue and Integrated UI

5.1.3 Delivery and usage

5.1.3.1 Package information

The package has the following structure:

Table 18. Package Structure

Path	Description
/conf	Contains specifications that are used by docker when generating an image to configure Nginx web server
/dist	Contains the result of the build
/keycloak-dev-docker-compose	The docker compose for local development described in the readme file
/kubernetes	Contains kubernetes configuration files for deployment
/Kubernetes-test	Contains kubernetes test configuration files for deployment
/node_modules	Contains installed npm modules
/realm-config	This file contains a backup of the keycloak realm configurations
/src/Dockerfile	This file contains specifications that are used in order to build a docker image
/src/assets/config/config.json	Contains application configuration which can be modified at runtime
/src/environments/	Contains static configurations based on environment (dev or test)
/src/app/services	Contains services that are generated via OpenAPI specs in order to integrate with other applications in MEDINA Framework
/src/app/	Contains the main components of the application

5.1.3.2 Download

The Integrated User Interface is closed-source and published on the private TECNALIA GitLab at: https://git.code.tecnalia.com/medina/wp5/task_5.3/integrated-ui [internal use only - authentication required].

6 Conclusions

This document reports on how the objectives for M27 related to task 5.3 have been fulfilled. First of all, the environment is kept in maintenance compared to the first version of M15, and the automation of the solution has been improved by redefining the methodology described with the use of CI/CD pipelines. The adoption of the CI/CD strategy enables the automatic release of the components in the two virtual environments of the Kubernetes cluster, "dev" and "test". The components that made up the eight building blocks of the MEDINA reference architecture have reached a high level of maturity and some components such as SSI and block five components have been added for the first time.

In addition, the document shows the seven scenarios identified in the previous version in a new way by introducing roles and their level of visibility that define the allowed actions. At the same time, integration activities have been led with the support of technical webinars and demonstrations on different topics regarding the DevOps approach integrated with the Kubernetes environment, the Keycloak integration with the component, and how to manage the authorization and filtering in MEDINA.

The next activities planned for the third and last version of this "MEDINA integrated solution" foresee improving the solution with feedback coming from the Use Cases, and improving the security pipeline by adding the MEDINA component "Codyze". A satisfactory state of completion will be reached for each component and the Integrated User Interface will be finalised with the fulfilment of all requirements, in particular regarding the look & feel. This final version will be released in M33.

7 References

[1] MEDINA Consortium, "D5.3 MEDINA integrated solution-v1," 2022.

[2] MEDINA Consortium, "D5.2 MEDINA Requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy," 2022.

[3] MEDINA Consortium, "D5.1 MEDINA Requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy," 2021.

[4] "K8s," [Online]. Available: https://kubernetes.io/docs/home/. [Accessed January 2023].

[5] "Keycloak," [Online]. Available: https://www.keycloak.org/. [Accessed January 2023].

[6] MEDINA Consortium, "D6.3 Use cases development and validation-prototypes-v1," 2022.

[7] "SPDX license," [Online]. Available: https://spdx.org/licenses/. [Accessed January 2023].

[8] "JFrog Artifactory," [Online]. Available: https://jfrog.com/artifactory/. [Accessed January 2023].

[9] "Codyze," [Online]. Available: https://www.codyze.io/?ref=https://githubhelp.com. [Accessed January 2023].

[10] MEDINA Consortium, "D2.7 Risk-based techniques and tools for Cloud Security Certification-v2".

[11] MEDINA Consortium, "D2.2 Continuously certifiable technical and organizational measures and catalogue of cloud security metrics-v2," 2023.

[12] ENISA, "EUCS -Cloud Service Scheme," Draft version provided by ENISA (August 2022) - not intended for being used outside the context of MEDINA, 2022.

[13] MEDINA Consortium, "D2.4 Specification of the Cloud Security Certification Language-v2," 2022.

[14] MEDINA Consortium, "D4.4 Methodology and tools for risk-based assessment and security control reconfiguration-v1," 2022.

[15] MEDINA Consortium, "D4.2 Tools and Techniques for the Management and Evaluation of Cloud Security Certifications - v2," 2022.

[16] MEDINA Consortium, "D3.2 Tools and techniques for the management of trustworthy evidence-v2," 2022.

[17] MEDINA Consortium, "D3.5 Tools and techniques for collecting evidence of technical and organisational measures-v2," 2022.

[18] Wazuh Inc., "Wazuh," [Online]. Available: https://wazuh.com. [Accessed January 2023].

[19] L. M. D. T. Severi Peltonen, "Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review," DAZN, London, United Kingdom and Tampere University, Tampere, Finland, 24 03 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921000549. [Accessed January 2023].

[20] Google, "Angular Material," [Online]. Available: https://material.angular.io/. [Accessed January 2023].

[21] "Nginx," [Online]. Available: https://www.nginx.com/. [Accessed January 2023].

[22] "RKE," [Online]. Available: https://rancher.com/docs/rke/latest/en/os/. [Accessed January 2023].

[23] "Rook/Ceph," [Online]. Available: https://rook.io/docs/rook/v1.8/. [Accessed January 2023].

[24] "METALLB," [Online]. Available: https://metallb.universe.tf/. [Accessed January 2023].

[25] "SSH," [Online]. Available: https://www.ssh.com/academy/ssh/protocol. [Accessed January 2023].

[26] Linux Foundation, "Helm package manager," [Online]. Available: https://helm.sh/. [Accessed January 2023].

[27] Linux Foundation, "Cert manager," [Online]. Available: https://cert-manager.io/docs/. [Accessed January 2023].

[28] "Apache Maven Project," [Online]. Available: https://maven.apache.org/. [Accessed January 2023].

[29] ENISA, "EUCS - Cloud Services Scheme," [Online]. Available: https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme. [Accessed January 2023].

[30] MEDINA Consortium, "D3.1 Tools and techniques for the management of trustworthy evidence-v1," 2021.

8 APPENDIX A: Operating Environment

The MEDINA framework functionalities are made up by the collaboration of all the micro-services, which communicate each other through REST API, are packaged in Docker images and run in Docker containers. Kubernetes orchestrates all these containers in a virtual environment running on high-available cluster.

8.1 Kubernetes Installation and Configuration

This section illustrates the container orchestration solution that is executed over the setup infrastructure described in Section 2.1.1.

Different resources are needed to proceed with the installation and configuration of the cluster. We used RKE [22] for the installation of Kubernetes [4] in the three nodes, Rook/Ceph [23] for the configuration of storage and MetalLB [24] for the network configuration.

The Kubernetes cluster is configured and managed by Rancher Kubernetes Engine (RKE) [22], an open-source distribution that simplifies the installation and operations of Kubernetes. The RKE client is installed on a console host at the cicd.medina.esilab.org VM and communicates with the nodes of the cluster through SSH (Secure Shell protocol [25]). Through RKE, we have configured each cluster node to be both Master and Worker, guaranteeing fault-tolerance and high availability. To do so, RKE creates on each of them the control plane, kubelet and kube-proxy resources in Docker containers.

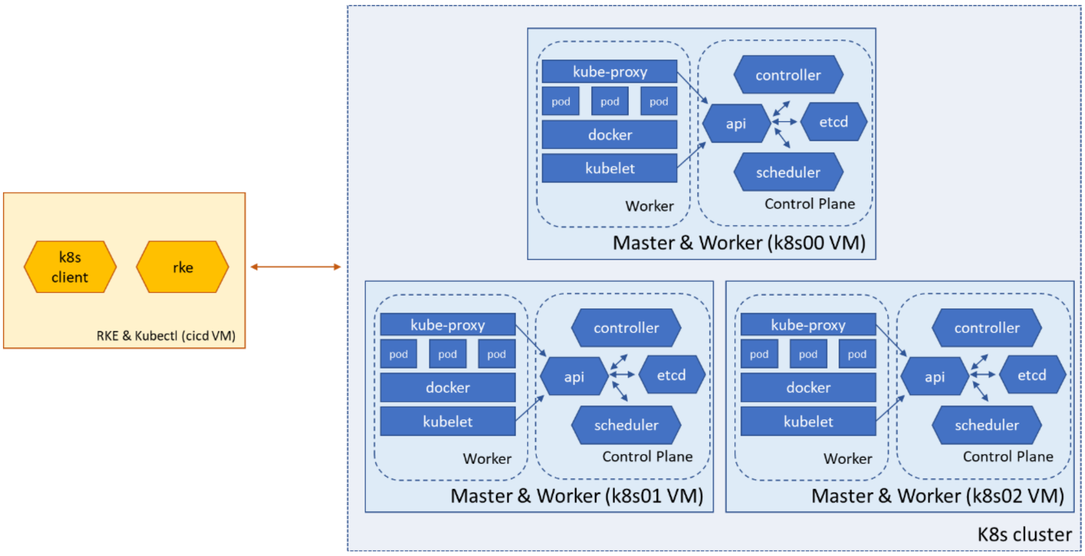


Figure 39. Kubernetes cluster installation with RKE

All the micro-services can store their data in an easy and secure way thanks to the configuration of a distributed filesystem. Indeed, each node of the cluster provides 200 GB of storage, managed by Rook/Ceph and exposed as a single, unified cluster filesystem.

Ceph is an open-source distributed storage solution for deliver block storage, object storage and shared filesystem in a single, unified system. It ensures cluster state monitoring and handles data replication, recovery and rebalancing.

Ceph is deployed to the Kubernetes cluster by Rook that is an open-source cloud-native storage orchestrator enabling Ceph to easily run on Kubernetes cluster. The Rook operator is a Kubernetes resource that automates the Ceph management and installation and turns Ceph into a self-scaling, self-managing and self-healing storage service.

Thanks to this configuration, the data are replicated across the three nodes, 200 GB of storage and fault-tolerance and high availability are assured.

The micro-services running on the Kubernetes cluster are packaged in Docker images and stored on a private Docker Registry running on Artifactory by JFrog [8].

In order to have Kubernetes access the Docker Registry, a specific integration has been done: a *secret* has been created with the registry credentials. This allows Kubernetes to pull the micro-service image and then run it on the cluster.

The images are pushed to the Docker registry according to the following structure that was agreed in the project:

<medina_registry_url>/<work_package>/<task >/<image>:<tag>

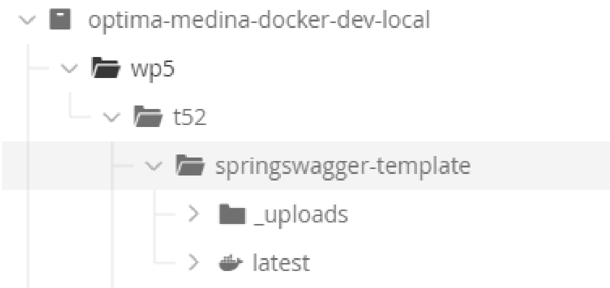


Figure 40. Excerpt of MEDINA's Docker registry

The REST API exposed by each micro-service is reachable from the Internet using the "*.k8s.medina.esilab.org" URL, corresponding to the static public IP 172.26.124.120. In particular, on the Kubernetes cluster an nginx [21] service is configured as a proxy to redirect all the requests to the correct micro-service component. The binding between the nginx service and the public IP is setup with MetalLb. MetalLb [24] is a network load-balancer implementation that associates the public IP to the nginx service and uses standard routing protocols to make available (part of) the network behind the Kubernetes cluster. It is essential for the MEDINA cluster because, unlike a public cloud provider cluster, this one has no load balancer and Kubernetes does not provide it by itself.

The user can address the environment s/he wants using this URL naming convention:

<component_name>-<environment [test or dev]>.k8s.medina.esilab.org

For example, if the user needs to refer to the API exposed by the "api-swagger" component running on the Kubernetes test environment, s/he will address it as:

api-swagger-test.k8s.medina.esilab.org

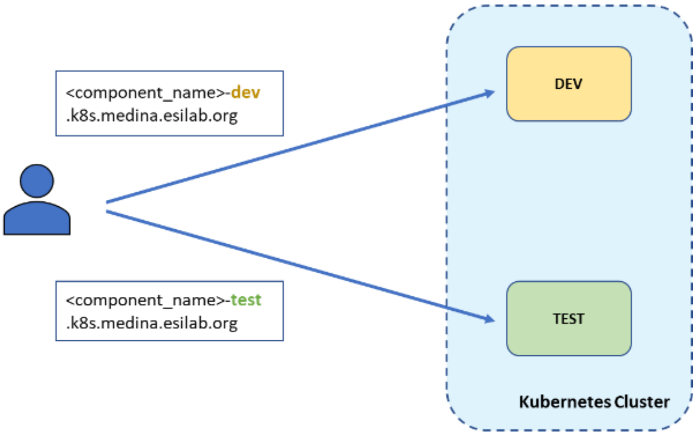


Figure 41.URL naming convention for dev/test environments

8.2 Kubernetes Dashboard

Kubernetes Dashboard is a web-based User Interface for the Kubernetes cluster. It is helpful to deploy containerized applications to a Kubernetes cluster, troubleshoot them, and manage the cluster resources. We installed K8s Dashboard using the Helm package manager [26].

To have access to the Dashboard it is needed to generate a Service Account token by creating a service account. We have two service account with different permissions: one is "dashboard-admin" that has access to all cluster resources and the other is "partner-user" for the partners access that has restricted permissions only to dev and test namespaces. We must copy the token to sign into the Dashboard.

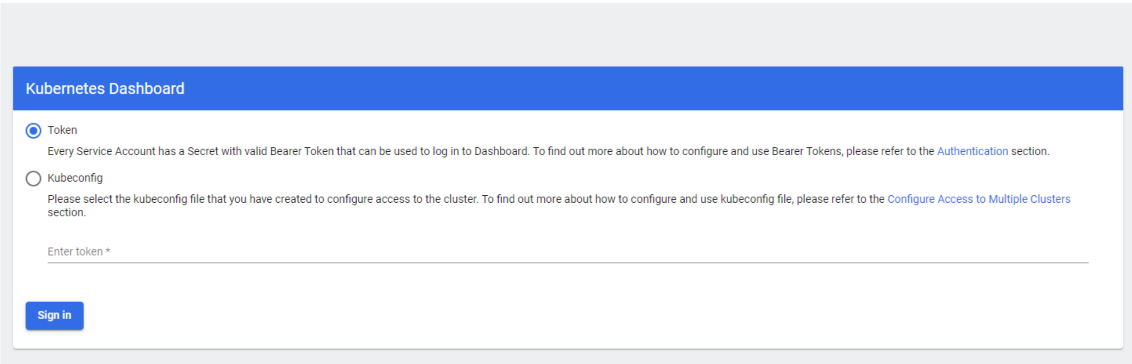


Figure 42. Service Account type used for the Kubernetes Dashboard

The Dashboard is exposed over HTTPS (see Figure 43) at https://dashboard.k8s.medina.esilab.org/#/login [internal use only - authentication required].

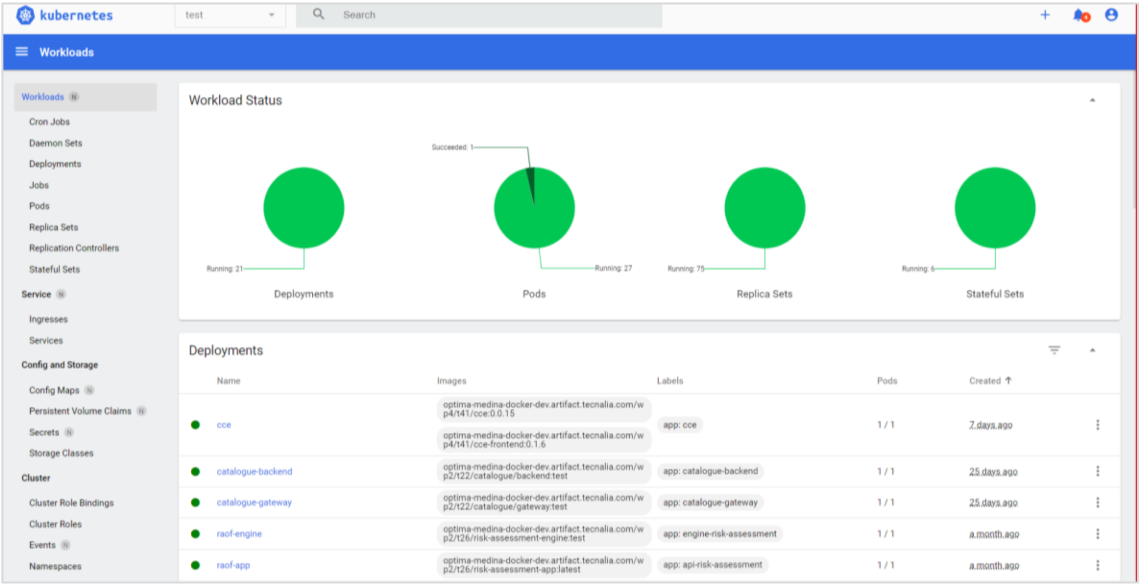


Figure 43. Kubernetes Dashboard

We have a secure Dashboard since certificates are used to expose it over HTTPS. These certificates are installed using cert-manager [27]. Cert-Manager automates the provisioning of certificates and provides a set of custom resources to issue certificates and attach them to services.

One of the most common use cases is securing web apps and APIs with SSL certificates from Let's Encrypt. Basically, we have installed Cert-Manager using the manifest file, created an issuer that uses the Let's Encrypt API for the specific domain "dashboard.k8s.medina.esilab.org" and exposed the Dashboard over HTTPS.

9 APPENDIX B: Docker and Kubernetes Webinar with Sample Component Integration example

The components' cluster integration in the first round was done manually by all partners, then it would be automated in the next MEDINA framework versions. To support all partners with this first integration, a webinar was organized in which an example project was presented.

The webinar included a part dedicated to the explanation of the main aspects and operations of Docker and Kubernetes and another part for the demonstration of all needed steps to deploy a sample project in the MEDINA environment.

The sample project, that is a spring swagger application, is available on the project's private GitLab located at TECNALIA. It exposes a REST API and stores data on PostgreSQL database while the Dockerfile, the Kubernetes manifests files and the README instructions are available on the repository.

Name	Last commit	Last update
kubernetes	Added kubernetes yaml configuration files	1 month ago
src	Initial commit	8 months ago
.gitignore	Initial commit	8 months ago
Dockerfile	Initial commit	8 months ago
LICENSE	Initial commit	8 months ago
README.md	Updated readme	1 month ago
pom.xml	Initial commit	8 months ago

Figure 44. Spring Swagger Template on GitLab

The demo of the sample project illustrates step by step all the actions to do for the correct configuration and deployment of it, starting from the build and up to its release in the k8s cluster.

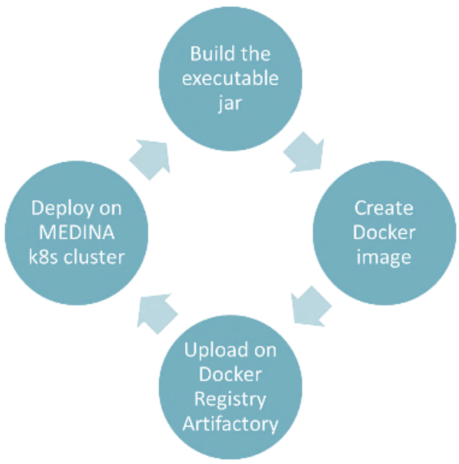


Figure 45. Sample project deployment steps

First of all, the project is packaged with Maven [28] and an executable jar is created.

This jar is included in the Dockerfile for the docker image creation. Then, after the login on the private Docker Registry Artifactory, the docker image is pushed following the path convention at:

optima-medina-docker-dev.artifact.tecnalia.com/wp5/t52/springswagger-template:latest

The final step is the deployment of the docker image in the k8s cluster through the Kubernetes Dashboard.

Once applied the Kubernetes manifests, the application is reachable from the internet according to this URL convention:

<component_name>-<namespace {dev, test}>.k8s.medina.esilab.org

For example, the access to the application in the dev environment is at:

http://api-swagger-dev.k8s.medina.esilab.org/swagger-ui/index.html#/

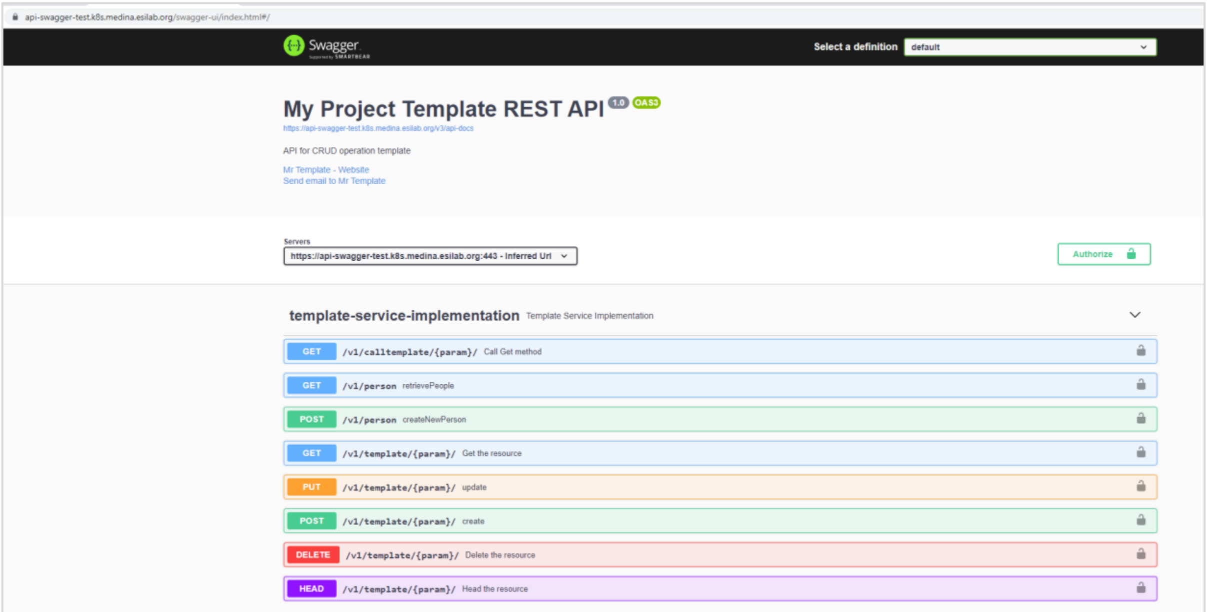


Figure 46. Demo project in the test environment

10 APPENDIX C: First integration workshop

The aim of the workshop for the first round was to release the first version of the MEDINA Framework in the development environment of the cluster. The integration and release of components was done manually by the partners which, however, would be automated through the CI/CD pipelines in the next rounds.

To carry out the integration of the components, partners were provided with access credentials to GitLab, Docker Registry Artifactory and the Kubernetes Dashboard.

During the workshop the first five actions foreseen by the defined methodology were successfully completed by all partners: first of all, each project had been uploaded to GitLab, then the Docker images had been pushed on the Artifactory registry and finally the Kubernetes manifest files had been created and applied to the development environment via the Kubernetes Dashboard.

At the end of the workshop, all components planned for this round were successfully released in the development environment (see Figure 47).

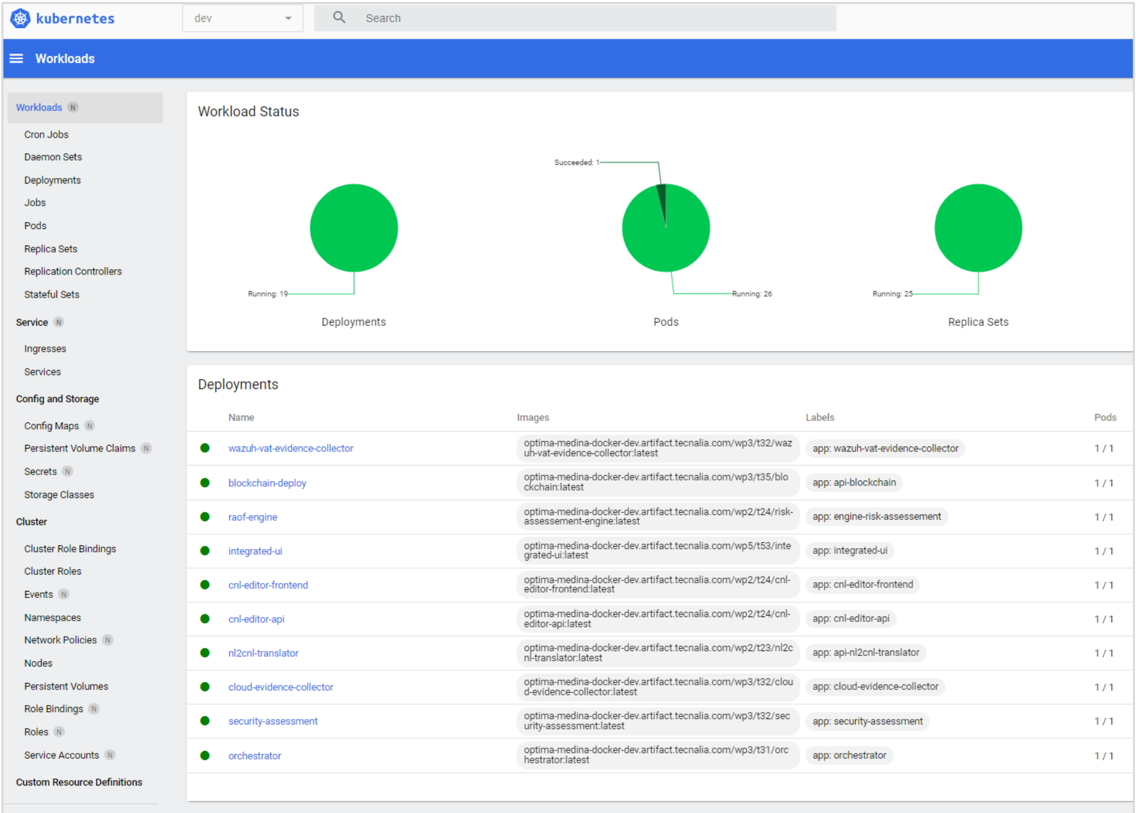


Figure 47. K8s Dashboard: Components deployed in dev environment

Figure 48 lists all the components of the MEDINA Framework: the green ones were released on the development environment, the yellow one would be deployed in the next round, and the blue ones would not be released in the Kubernetes cluster. In particular, the Codyze component would be integrated in the MEDINA Security pipeline and Wazuh and VAT would run on a dedicated standalone VM provided by TECNALIA.

INTEGRATION COMPONENTS STATUS										
				Integration Steps						
Component	**Owner (Partner)**	**Work Package**	**Task**	**TECNALIA GitLab**	**Containerization**	**K8s file**	**OpenAPI specs**	**Push to Docker Registry**	**Deploy Dev**	**Deploy Test**
CNL Editor	HPE	WP2	T2.4	yes	yes	yes	yes	yes	yes	no
Metrics and measures catalogue	TECNALIA	WP2	T2.2	yes	yes	yes	yes	yes	yes	no
NL2CNL Translator	CNR/Fabasoft	WP2	T2.3	yes	yes	yes	yes	yes	yes	no
DSL Mapper	CNR/Fabasoft	WP2	T2.5	yes	yes	yes	yes	yes	yes	no
Cloud Evidence Collector (Clouditor)	FhG	WP3	T3.2	yes	yes	yes	yes	yes	yes	no
Security Assessment (Clouditor)	FhG	WP3	T3.2	yes	yes	yes	yes	yes	yes	no
Orchestrator (Clouditor)	FhG	WP3	T3.1	yes	yes	yes	yes	yes	yes	no
Codyze	FhG	WP3	T3.3	yes (partly)	yes	\	no	yes	no (integrated Jenkins)	no
Blockchain Monitoring Tool	TECNALIA	WP3	T3.5	no (proprietary component)	yes	yes	yes (partially)	yes	yes	no
Static Risk Assessment and Optimisation Framework	CNR	WP2	T2.4	yes	yes	yes	yes	yes	yes	no
Dynamic Risk Assessment and Optimisation Framework	CNR	WP4	T4.4	no	\	\	\	\	\	\
Wazuh + VAT evidence collector (interface to sec.ass.)	XLAB	WP3	T3.2	yes	yes	no	no	yes	no	no
Wazuh & VAT proprietary	XLAB	WP3	T3.2	no (proprietary component)	no	\	no	no	no (standalone VM)	no
Continuous Certification Evaluation	XLAB	WP4	T4.1	yes	yes	yes	no	yes	yes	no
Life Cycle Manager	FhG	WP4	T4.3	yes	yes	yes	no	yes	no	no
Organisational evidence management tool	Fabasoft	WP3	T3.4	no	no	no	no	no	no	no
Integration UI	HPE	WP5	T5.3	yes	yes	yes	no	yes	yes	no

Figure 48. Status of the first integration of MEDINA components

Furthermore, partners performed point to point tests to verify the communication in pairs of the released components. Table 19 shows in green the working ones.

Table 19. Point to point communication tests

Component Name	Component Name	Status
Orchestrator	Countinuous Certification Evaluation	CONNECTED
Orchestrator	Blockchain Monitoring Tool	CONNECTED
Orchestrator	Security Assessment	CONNECTED
Orchestrator	Metrics and Measures Catalogue	NEXT ROUND
Cloud Evidence Collector	Security Assessment	CONNECTED
Security Assessment	WAZUH + VAT Evidence Collector	CONNECTED
DSL Mapper	Orchestrator	NEXT ROUND
DSL Mapper	Metrics and Measures Catalogue	NEXT ROUND
NL2CNL Translator	Metrics and Measures Catalogue	NEXT ROUND
CNL Editor	DSL Mapper	NEXT ROUND
CNL Editor	NL2CNL Translator	NEXT ROUND
CNL Editor	Metrics and Measures Catalogue	NEXT ROUND
Organisational Evidence Management Tool	Metrics and Measures Catalogue	NEXT ROUND
Static Risk Assessment and Optimisational Framework	Metrics and Measures Catalogue	NEXT ROUND
Countinuous Certification Evaluation	Metrics and Measures Catalogue	NEXT ROUND
Countinuous Certification Evaluation	Dynamic Risk Assessment and Optimisation Framework	NEXT ROUND
Dynamic Risk Assessment and Optimisation Framework	Life Cycle Manager	NEXT ROUND
Integration UI	Metrics and Measures Catalogue Keycloack	CONNECTED
Integration UI	Metrics and Measures Catalogue	CONNECTED
Integration UI	NL2CNL Translator	CONNECTED
Integration UI	Orchestrator	NEXT ROUND
Organisational Evidence Management Tool	Orchestrator	NEXT ROUND
Integration UI	Organisational Evidence Management Tool	NEXT ROUND

11 APPENDIX D: Generic Architectural Workflows

This Appendix revisits and updates the details related to the generic architectural workflows as presented in D5.3 [1]. As required, the workflows have been updated for the purposes of the present deliverable.

11.1 WF1 - Preparation of Target of Certification (ToC)

This initial workflow, despite not invoking any of the MEDINA components, is an evident pre-requisite for the CSP to fulfil before the certification process starts. Its main goal is for the CSP to prepare the Target of Certification (ToC), both from a technical (e.g., deploying the actual cloud service in the hyperscaler) and organizational (e.g., gather the operational manuals in electronic format) perspectives.

11.1.1 Related Architectural Components

As mentioned above, this workflow does not involve any of the MEDINA components. However, it setups the ToC elements in building blocks 5 and 7 from Figure 15, namely:

- ToC's organizational evidence (electronic format)
- Cloud services comprising the ToC (e.g., IaaS/PaaS/SaaS), which can be deployed in one or more hyperscaler.

11.1.2 Workflow

Table 20 describes the steps associated to this workflow.

Table 20. WF1 description

Step	Description	Role	Comments
1	Documentation related to organizational measures implemented by the Cloud Service is gathered and made available in electronic format.	CSP[34]	The documentation can be made available in portable formats like PDF.
2	All Resources that comprise the Cloud Service/ToC (VMs, SQL, Web Apps, SaaS, etc.) are assigned to an impact level, technically configured and deployed in the hyperscaler.	CSP	The impact level will be further used in subsequent workflows for the purposes of risk management. For characterizing the Resources, the current data model in D5.2 [2] considers three impacts levels corresponding to each of confidentiality, integrity and availability.

11.2 WF2 - Preparation of MEDINA Components

The second generic workflow of the architecture (WF2) refers to the actual configuration and deployment of those MEDINA components which are needed for certifying the Cloud Service. This WF2 does not perform any actual assessment, but it is a required set of deploying actions before the certification process is triggered by WF3.

11.2.1 Related Architectural Components

This workflow involves the components in building blocks 1, 2, 7 and 8 from Figure 15, namely:

- Catalogue of Controls and Metrics

[34] In this generic context, CSP means the entity responsible of the ToC (EUCS requestor).

- Organizational Evidence Gathering and Processing
- Security Assessment (CS Level and OS) – Clouditor Assessment
- Evidence Collection / Security Assessment CS level and CSP Native (Azure Policies)
- Evidence Collection / Security Assessment Application Level (Codyze)
- Evidence Collection Wazuh
- Evidence Collection VAT
- Trustworthiness Evidence Management system (DLT)
- Company Compliance Dashboard / Integrated UI

11.2.2 Workflow

Table 21 describes the steps associated to this workflow.

Table 21. WF2 description

Step	Description	Role	Comments
1	Configuring the following settings in the Company Compliance Dashboard / Integrated UI: a. SSO integration b. Setup users and roles	CSP	The Integrated UI provides the entry point to the MEDINA framework, and as such it needs to become integral part of the CSP's systems. Therefore, actions like SSO integration are needed. A role-based authorization model allows MEDINA users to only perform specific actions.
2	Setting up the Catalogue of Controls and Metrics: a. Configure the EUCS catalogue with all assurance levels, and including corresponding controls/requirements/metrics.	MEDINA[35]	The Catalogue of Controls and Metrics is prefilled with EUCS information, so it comes out-of-the-box for the CSP (see WF3).
3	Configure the Security Assessment (CS-Level and OS) – Clouditor Assessment: a. Clouditor's OS-agent is deployed in VMs Resources from the ToC b. Clouditor's CS-level is configured in PaaS Resources from the ToC	CSP	The MEDINA framework guarantees that corresponding agents can be deployed at-scale on the corresponding Resources.
4	Configuration of (Technical) Evidence Collection / Security Assessment CS level and CSP Native (Azure Policies): a. CSP-Native is configured to automatically collect compliance data from Azure	CSP	In analogy to the collector described in Step 3, this CSP-Native one is used to gather evidence from technical measures.
5	Configuration of (Technical) Evidence Collection / Security Assessment Application Level (Codyze): a. Codyze is configured	CSP	Used to gather evidence from technical measures (code-level).
6	Configuration of (Technical) Evidence Collection Wazuh: a. Wazuh is configured	CSP	Used to gather evidence from technical measures.
7	Configuration of (Technical) Evidence Collection VAT: a. VAT is configured	CSP	Used to gather evidence from technical measures.

[35] This role means the actual MEDINA framework (non-human role).

Step	Description	Role	Comments
8	Configuration / activation of the Trustworthiness Evidence Management system (DLT) for the evidence management and security assessment results management	CSP	This component is linked to the Orchestrator.

11.3 WF3 - EUCS deployment on ToC

After the ToC has been deployed on the hyperscaler (WF1) and the corresponding MEDINA components were configured/deployed by the CSP (WF2), then it is possible to use the later for certifying the Cloud Service. That is the goal of this WF3.

11.3.1 Related Architectural Components

This workflow involves the components in building blocks 1, 2, 5 and 7 from Figure 15, namely:

- Catalogue of Controls and Metrics
- CNL Editor
- Organizational Evidence Gathering and Processing
- Orchestrator / Clouditor Orchestrator

11.3.2 Workflow

Table 22 describes the steps associated to this workflow.

Table 22. WF3 description

Step	Description	Role	Comments
1	The Company Compliance Dashboard / Integrated UI is used to perform the following actions: a. Each Resource comprising the Cloud Service is registered in MEDINA as part of the ToC.	CSP	Required information from the Resource include the impact level mentioned in WF1. Additional attributes of the Resource are populated as needed and based on the MEDINA data model.
2	The Catalogue of Controls and Metrics (UI) is used to: a. Select EUCS Assurance level for the ToC to certify	CSP	The default value being “High” (which is the one requiring continuous monitoring in EUCS), but also “Basic” and “Substantial” can be selected.
3	The UI from the CNL Editor is used to: a. Select suitable **built-in Metrics** as provided by the Metrics Recommender (or accept the ones pre-selected by default) b. Customize Target Values[36] on the selected **built-in Metrics**.	CSP	Once the corresponding Obligations have been selected and configured with a Target Value (including the corresponding Metric), then they are ready to be stored along with the ToC information in MEDINA’s Orchestrator.
4	The Organizational Evidence Gathering and Processing is used to upload the collected documentation (see WF1)	CSP	These documents are stored directly on the database of the component, and not on the Orchestrator’s.
5	The Orchestrator stores the configured ToC information (see steps 1-3) in its corresponding database.	MEDINA	n/a

[36] In the form of Obligations

11.4 WF4 - EUCS Preparedness – ToC Self-Assessment

This workflow relates to the components in charge of performing the static risk management (SATRA) and the EUCS self-assessment (Catalogue of controls and metrics) as documented by D2.7 [10] and D2.2 [11] respectively. Although SATRA implements a "stand alone functionality", which does not need to be technically deployed in the Cloud Service (cf. WF3), it is integrated into the whole MEDINA framework thanks to the unified UI.

11.4.1 Related Architectural Components

This workflow involves the components in building blocks 1 and 3 from Figure 15, namely:

- Risk Assessment and Optimization Framework
- Catalogue of Controls and Metrics

11.4.2 Workflow

The related activities in WP4 are described in Table 23.

Table 23. WF4 description

Step	Description	Role	Comments

1	Catalogue of controls and metrics:
	a. Create a new questionnaire after selecting the EUCS framework and the assurance level.
	b. Load a questionnaire that has been previously stored in the Catalogue.
	c. Provide answers to the questions for each requirement, based on any of the following potential answers:
	    a. Fully supported
	    b. Partially supported
	    c. Not supported at all
	    d. Not applicable
	d. Provide some evidence to support the answers to the questionnaire
	e. Identify some non-conformities
	f. Save the questionnaire
	g. Generate the report
		CSP	The tool is based on a questionnaire interface containing requirements from EUCS, just as described in D2.2 [11] .
			
			A closed set of possible answers guarantees the computation of a degree of compliance, which represents the CSP's level of preparedness for obtaining an EUCS certificate.

2	Catalogue of controls and metrics: a. The compliance result for each requirement is calculated based on the answers provided for all its related questions. b. The compliance results are sent to the SATRA end point. c. An audit report is generated including the non-conformities defined for each requirement	MEDINA	The structure of the audit report is presented in D2.2 [11].
3	Risk Assessment and Optimization Framework: a. ToC information and Impact level (per-Resource type) are entered into the tool b. If applicable, the underlying Hyperscaler is configured as an	CSP	The ToC information required for the static risk assessment is manually entered into the tool (contrary to the automated discovery of Resources in WF3), mostly because less granular details are needed for the preparedness assessment. For

Step	Description	Role	Comments
	additional Resource (along with its associated Impact level) c. Targeted EUCS assurance level is selected, as required for the preparedness assessment		example, details about the actual Resources' configuration are not needed for this static assessment.
4	Risk Assessment and Optimization Framework: a. Implemented (CSP Responsibility) Not Implemented (CSP Responsibility) Not Applicable Unknown (Hyperscaler Responsibility) Degree of compliance for each requirement is retrieved from the Catalogue and reported to the CSP	MEDINA	The preparedness report includes the identification of major and minor non-conformities, and comparison between the ideal conformity case and the provided CSP answers. More details are presented in D2.6.

11.5 WF5 - EUCS Compliance Assessment

MEDINA proposes the notion of "continuous audit-based certification", which departs from the EUCS definition of "continuous (automated) monitoring" referring to **periodically assessing the ToC**. This WF5 describes **discrete compliance assessments**, which should then be periodically executed for the MEDINA framework to start the certification lifecycle (cf. WF6).

Further information about the underlying evidence collection mechanisms can be found in D3.2 [16].

11.5.1 Related Architectural Components

This workflow involves the components shown in building blocks 5 and 7 from Figure 15, namely:

- Organizational Evidence Gathering and Processing
- Security Assessment (CS Level and OS) – Clouditor Assessment
- Evidence Collection / Security Assessment CS level and CSP Native (Azure Policies)
- Orchestrator / Clouditor Orchestrator
- Evidence trustworthiness management (DLT)
- Evidence Collection / Security Assessment Application Level (Codyze)
- Evidence Collection / Clouditor Discovery
- Evidence Collection Wazuh
- Evidence Collection

11.5.2 Workflow

The different interactions corresponding to this WF5 are shown in Table 24.

Table 24. WF5 description

Step	Description	Role	Comments
1	Organizational Evidence Gathering and Processing: a. Automatically *assesses* the uploaded organizational documentation from the ToC based on the selected Metrics.	MEDINA	MEDINA supports EUCS auditors in their currently manual/time-consuming activity of assessing organizational evidence of the CSP (e.g., operation manuals). The automated assessment of such organizational evidence is expected to release auditors from most

Step	Description	Role	Comments
			of this time-consuming activity, although a minimum level of human interaction is still expected (e.g., to confirm the assessment results of the tool, or to provide training data which is CSP-specific).
2	Evidence Collection / Security Assessment Application Level (Codyze): a. Assesses code-level Resources from the ToC based on selected Metrics	MEDINA	D3.2 [16] already includes an analysis of the high assurance level requirements covered by the MEDINA tools. This includes not only the current coverage, but also the expected coverage once the extensions of the tools / new functionalities are included.
3	Evidence Collection / Clouditor Discovery: a. Assesses cloud service-level Resources from the ToC based on selected Metrics	MEDINA	Please refer to D3.2 [16] for further details on metrics' coverage.
4	Evidence Collection Wazuh: a. Assesses cloud service-level Resources from the ToC based on selected Metrics	MEDINA	Please refer to D3.2 [16] for further details on metrics' coverage.
5	Evidence Collection VAT: a. Assesses cloud service-level Resources from the ToC based on selected Metrics	MEDINA	Please refer to D3.2 [16] for further details on metrics' coverage.
6	Evidence Collection / Security Assessment CS level and CSP Native (Azure Policies): a. Assesses cloud service-level Resources from the ToC based on selected Metrics	MEDINA	Please refer to D3.2 [16] for further details on metrics' coverage.
7	Orchestrator / Clouditor Orchestrator: a. Assessment Results from **organizational** assessments are stored b. Evidence from **organizational** assessments is stored	MEDINA	Organizational and technical evidence are managed by MEDINA in the same manner, so they can be postprocessed homogeneously by the rest of components (cf. WF6 and WF7).
8	Evidence trustworthiness management (DLT): a. Digest/hash of relevant information related to **organizational** assessments results and evidence are stored	MEDINA	Please refer to comment above.
9	Orchestrator / Clouditor Orchestrator: a. Assessment Results from **technical** assessments are stored b. Evidence from **technical assessments** is stored c. Assessment Results are sent to Continuous Certification Evaluation	MEDINA	n/a

Step	Description	Role	Comments
10	Evidence trustworthiness management (DLT): a. Digest/hash of relevant information related to **technical** assessment results and evidence are stored	MEDINA	n/a

11.6 WF6 - EUCS – Maintenance of ToC certificate

This WF6 departs from the current definition of certificate maintenance in the EUCS core document (see Figure 49) and, for the purposes of MEDINA, adds also an initial stage of "certificate issuance". The main objective of WF6 is to take the "discrete/point in time" assessments from WF5 in order to trigger the different statuses of the corresponding EUCS certificate.

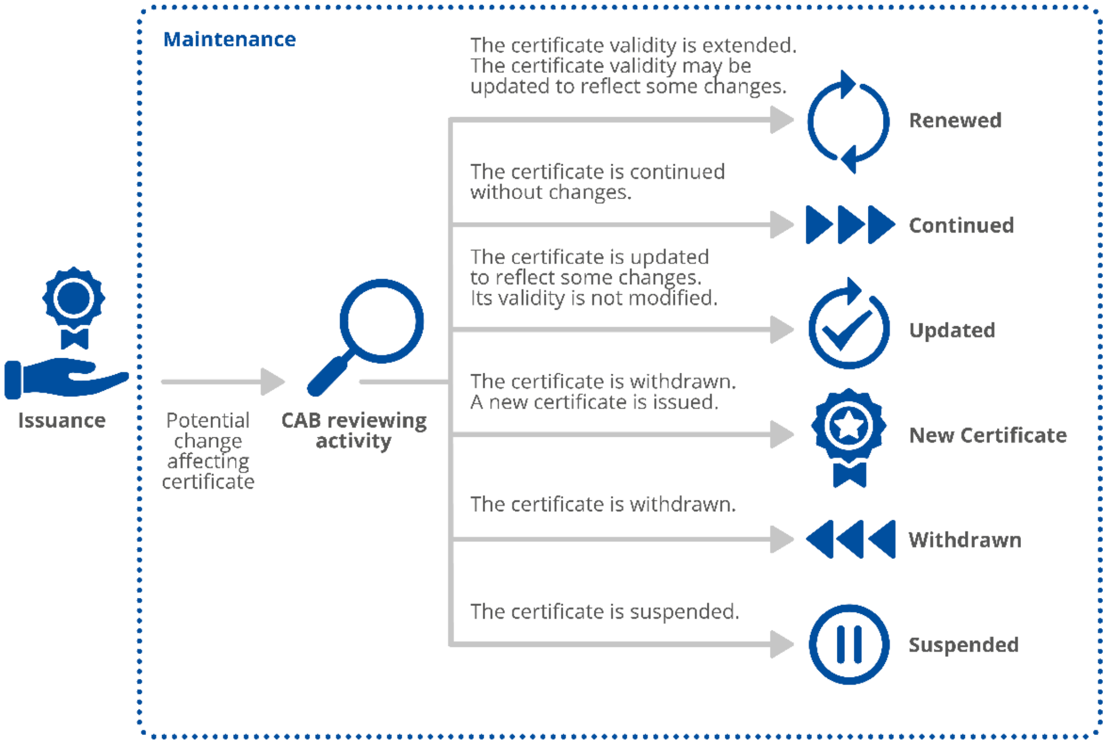


Figure 49. Certificate maintenance (source: EUCS [29])

11.6.1 Related Architectural Components

This workflow involves the components shown in building blocks 3 and 4 from Figure 15, namely:

- Continuous Certification Evaluation
- Risk Assessment and Optimization Framework
- Automated Certificate Lifecycle Management
- SSI framework

11.6.2 Workflow

The different interactions corresponding to this WF6 are shown in Table 25.

Table 25. WF6 description

Step	Description	Role	Comments
1	Continuous Certification Evaluation: a. Assessment Results (point-in-time assessment) are received from Orchestrator / Clouditor Orchestrator (push-mode) b. Tree-based evaluation is performed with received Assessment Results (which are received per-Resource) c. Tree-based evaluation results are stored in Certification Evaluation Storage d. If a non-compliance is found[37], then the Risk Assessment and Optimization Framework is invoked (RAOF, see Step 2 below)	MEDINA	This component automatizes the currently manual audit process for analysing a set of evidence (in particular when operational efficiency is in scope, like in the case of EUCS High).
2	Risk Assessment and Optimization Framework (RAOF): a. In analogy to WF4, the degree of non-compliance is computed based on the (point-in-time) assessments obtained from the Continuous Certification Evaluation b. The degree of non-compliance is communicated to the Certificate Lifecycle Manager (see Step 4 below)	MEDINA	As mentioned in WF4, the "degree on non-compliance" is computed comparing the real (e.g., based on monitored/declared status of requirements) risk level and ideal one (i.e., with all requirements satisfied). A threshold is to be set which determines if the difference is higher (major non-conformity) or lower (minor non-conformity). See D2.6 for more details.
3	Automated Certificate Lifecycle Manager: a. Based on the *Operational Effectiveness Criteria* defined by EUCS, the certificate maintenance lifecycle is triggered. b. The status of the certificate can be updated to any of New Certificate, Renewal, Continuation, Update, Withdraw, or Suspension.	MEDINA	The core EUCS document defines the basis for MEDINA to implement the automation of the certificate lifecycle management.
5	Automated Certificate Lifecycle Manager: a. Certificate status is published/updated on the MEDINA's Public Registry	MEDINA	This is a required step in EUCS to provide transparency to the certification process.
6	Automated Certificate Lifecycle Manager: a. Certificate status is notified to the CAB (emulated by an SSI-based issuer component).	MEDINA	The CAB leverages SSI techniques for issuing/updating the certificate.
7	SSI-based issuer:	CAB	The CAB leverages SSI techniques for issuing/updating the certificate.

[37] Compliances are not reported to the Risk Assessment and Optimization Framework

	a. A credential is issued to the CSP (i.e., SSI based holder) with the new certificate state. b. Previously issued credentials with different certificate status are revoked. c. Certificate status is optionally re-published on the MEDINA Public Registry.		
8	SSI-based holder: a. The credential with the updated certificate status is received and locally stored.	MEDINA	The CSP staff can check the historical certificates status.

11.7 WF7 - EUCS –Report on ToC Certificate

The goal of this WF7 is to report about the status of an EUCS certificate corresponding to the ToC and at different levels of detail, depending on the targeted audience (CAB, CSP, etc.). This WF7 consider for example, the case where a CAB needs to verify the technical/organizational evidence which resulted on the suspension of a certificate.

11.7.1 Related Architectural Components

This workflow involves the components shown in building block 4 from Figure 15, namely:

- Automated Certificate Lifecycle Management
- Evidence trustworthiness management (DLT)
- Continuous Certification Evaluation
- SSI Framework

11.7.2 Workflow

The different interactions corresponding to this WF7 are shown in Table 26.

Table 26. WF7 description

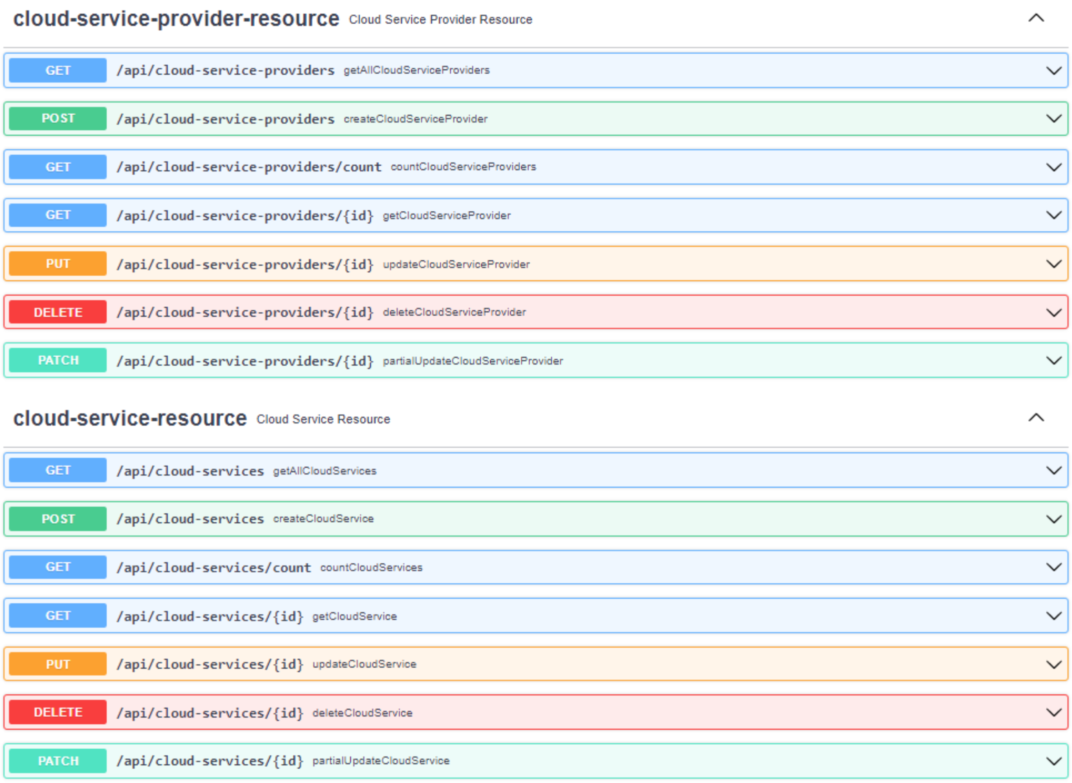
Step	Description	Role	Comments
1	Automated Certificate Lifecycle Management: a. A lookup on the Public Registry(-ies) is performed to search for a specific criterion (e.g., Certificate_ID, ToC, CSP, period of time, etc.). b. If found on the Public Registry, the corresponding certificate is shown.	CAB CSP NCCA	Details to display include certificate's history, ToC, degree of non-compliance, etc.
2	Continuous Certification Evaluation: a. For the selected certificate (see step 1 above), the details related to (non-) compliant controls/requirements/metrics/resources are displayed. b. The associated reference implementation TOM is retrieved from the Catalogue of Controls and Security Schemes and reported to the CSP. c. The associated degree of non compliance is retrieved from the Risk Assessment and Optimization Framework and reported to the CSP.	CSP	The CSP is provided with the details related to the selected certificate, in particular corresponding to the assessed controls/requirements/metrics/resources.

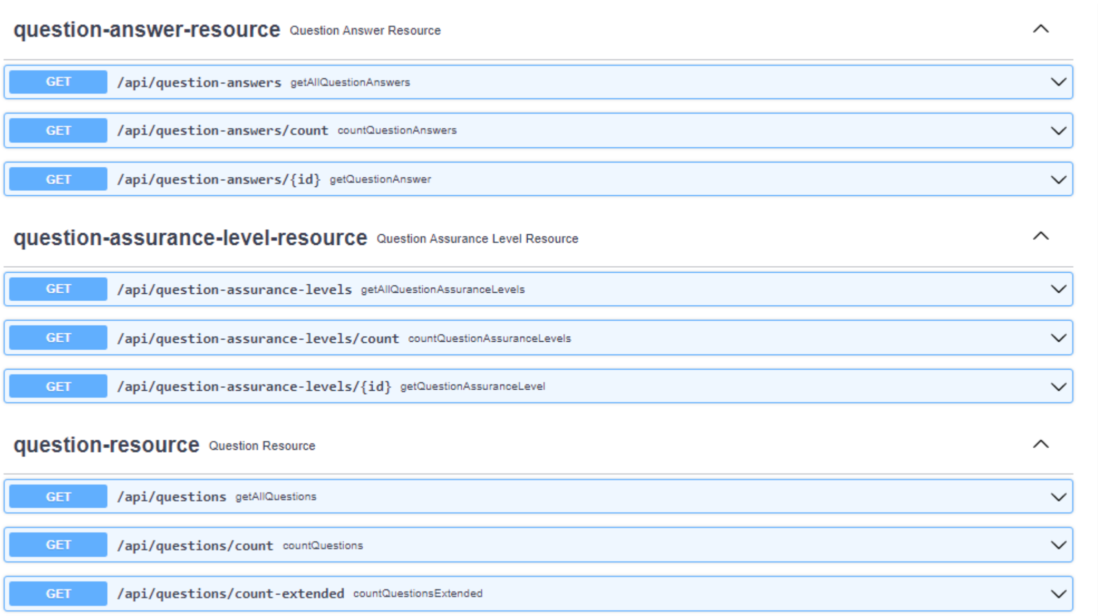
3	(Optional) Evidence trustworthiness management (DLT): a. For a selected EUCS certificate, the gathered evidence is validated and the status is then reported.	CAB CSP NCCA	A role like the CAB will have the option to check if the gathered evidence (used in the certificate's life cycle management) have not been tampered with. For this purpose, the DLT component is invoked.
4	(Optional) SSI based holder: a. The current (and previous) certificate status can be verified according to the credentials issued by the CAB.	CSP	By leveraging SSI-based techniques, the CSP verifies the historically issued certificates.
5	(Optional) SSI-based verifier: a. A potential CSP customer (or external auditor) can ask for secure proofs about the CSP certificates status.	CSP customer	The credential validity/trustworthiness can be verified.
6	(Optional) SSI based holder: a. Proofs of the current certificate status can be sent to the CSP's customer.	CSP	If requested, the CSP can send to its customers the information required to verify the certificate.
7	(Optional) SSI-based verifier: a. A potential CSP customer (or external auditor) receives the certificate status and can verify its validity/trustworthiness.	CSP customer	

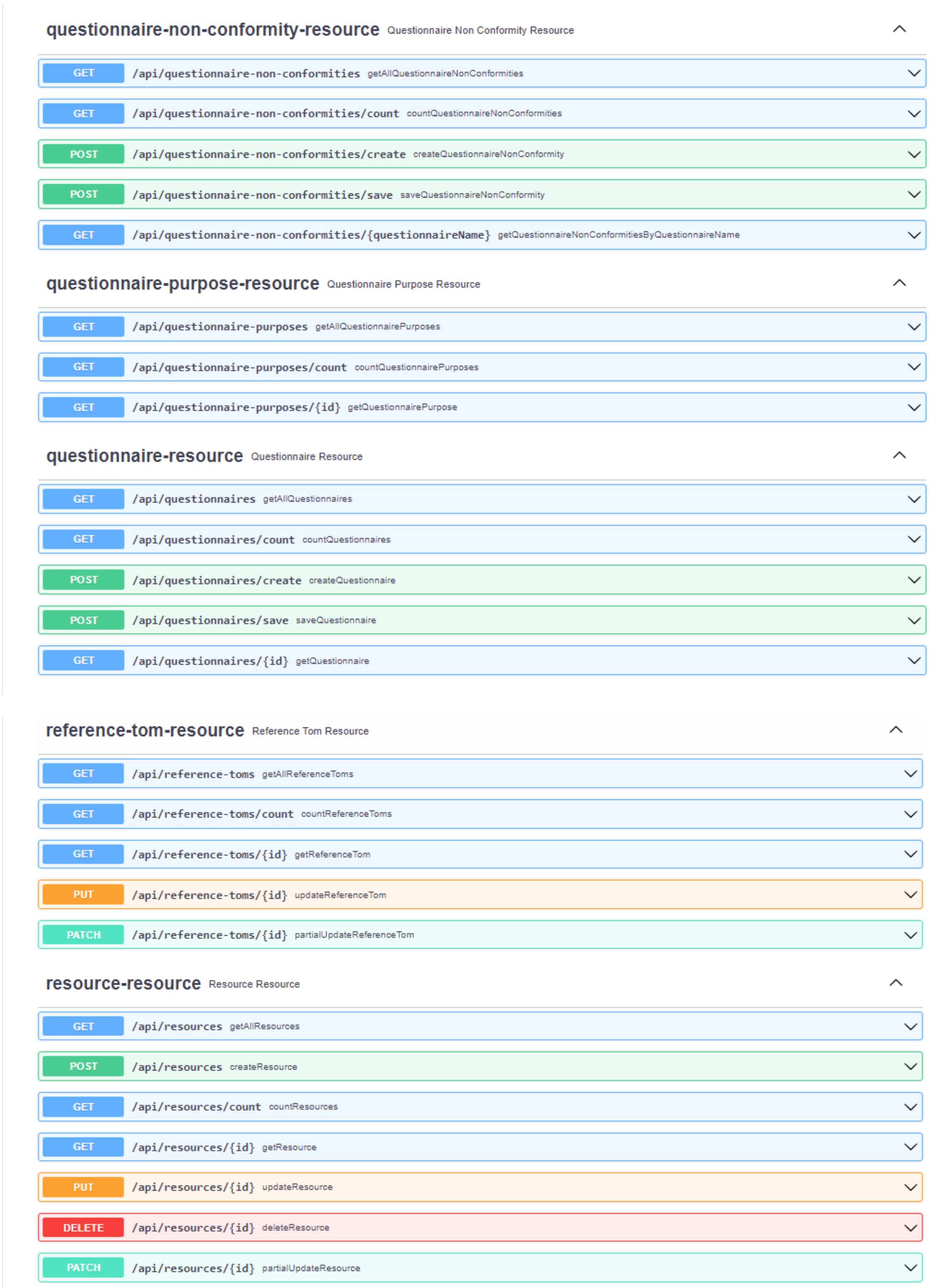
12 APPENDIX E: Published APIs

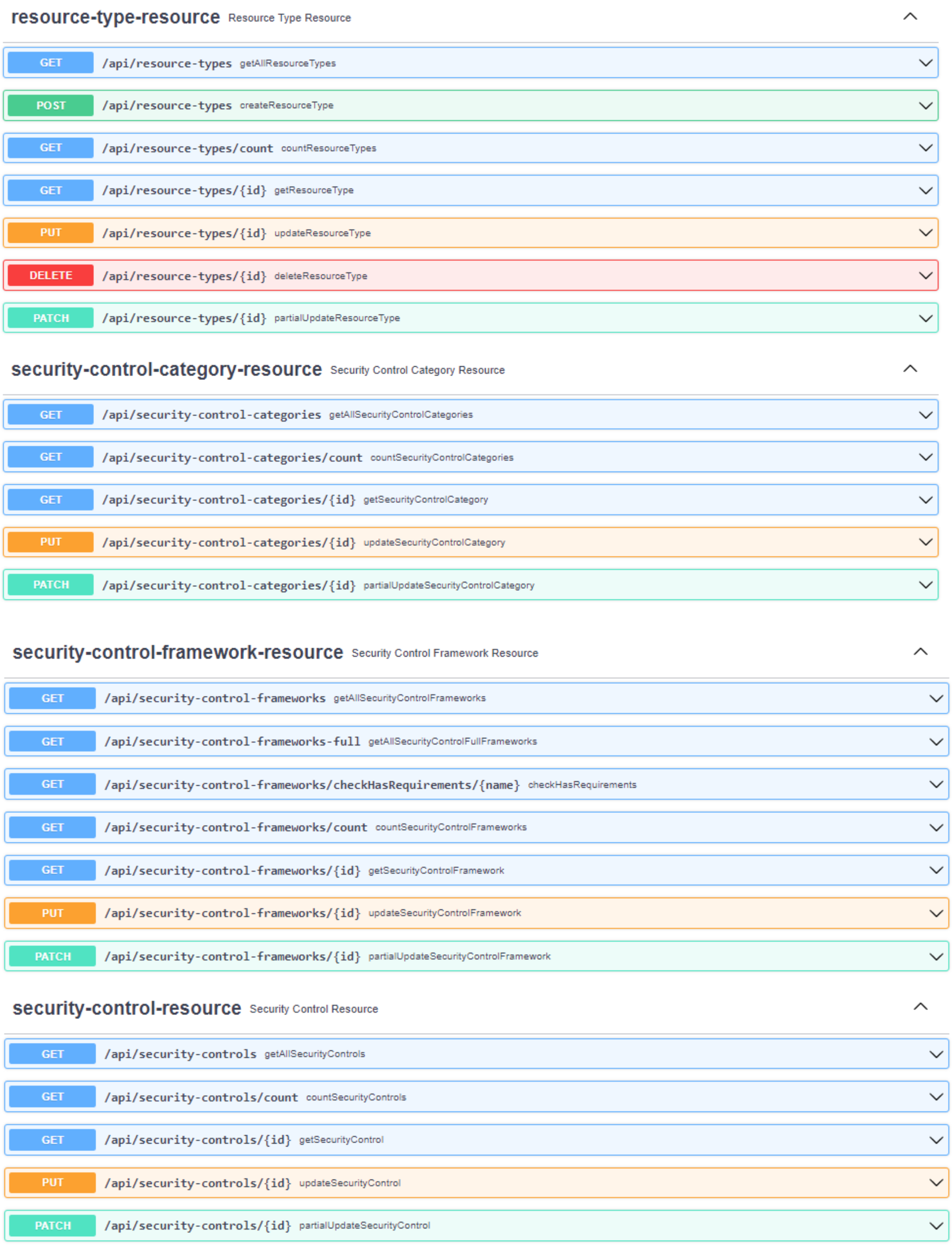
Component: Catalogue of Controls and Metrics

The following screenshot series show the list of available APIs that can be used by the components interacting with the Catalogue of Controls and Metrics.






questionnaire-non-conformity-resource Questionnaire Non Conformity Resource
GET /api/questionnaire-non-conformities getAllQuestionnaireNonConformities
GET /api/questionnaire-non-conformities/count countQuestionnaireNonConformities
POST /api/questionnaire-non-conformities/create createQuestionnaireNonConformity
POST /api/questionnaire-non-conformities/save saveQuestionnaireNonConformity
GET /api/questionnaire-non-conformities/{questionnaireName} getQuestionnaireNonConformitiesByQuestionnaireName
questionnaire-purpose-resource Questionnaire Purpose Resource
GET /api/questionnaire-purposes getAllQuestionnairePurposes
GET /api/questionnaire-purposes/count countQuestionnairePurposes
GET /api/questionnaire-purposes/{id} getQuestionnairePurpose
questionnaire-resource Questionnaire Resource
GET /api/questionnaires getAllQuestionnaires
GET /api/questionnaires/count countQuestionnaires
POST /api/questionnaires/create createQuestionnaire
POST /api/questionnaires/save saveQuestionnaire
GET /api/questionnaires/{id} getQuestionnaire
reference-tom-resource Reference Tom Resource
GET /api/reference-toms getAllReferenceToms
GET /api/reference-toms/count countReferenceToms
GET /api/reference-toms/{id} getReferenceTom
PUT /api/reference-toms/{id} updateReferenceTom
PATCH /api/reference-toms/{id} partialUpdateReferenceTom
resource-resource Resource Resource
GET /api/resources getAllResources
POST /api/resources createResource
GET /api/resources/count countResources
GET /api/resources/{id} getResource
PUT /api/resources/{id} updateResource
DELETE /api/resources/{id} deleteResource
PATCH /api/resources/{id} partialUpdateResource


resource-type-resource Resource Type Resource
GET /api/resource-types getAllResourceTypes
POST /api/resource-types createResourceType
GET /api/resource-types/count countResourceTypes
GET /api/resource-types/{id} getResourceType
PUT /api/resource-types/{id} updateResourceType
DELETE /api/resource-types/{id} deleteResourceType
PATCH /api/resource-types/{id} partialUpdateResourceType
security-control-category-resource Security Control Category Resource
GET /api/security-control-categories getAllSecurityControlCategories
GET /api/security-control-categories/count countSecurityControlCategories
GET /api/security-control-categories/{id} getSecurityControlCategory
PUT /api/security-control-categories/{id} updateSecurityControlCategory
PATCH /api/security-control-categories/{id} partialUpdateSecurityControlCategory
security-control-framework-resource Security Control Framework Resource
GET /api/security-control-frameworks getAllSecurityControlFrameworks
GET /api/security-control-frameworks-full getAllSecurityControlFullFrameworks
GET /api/security-control-frameworks/checkHasRequirements/{name} checkHasRequirements
GET /api/security-control-frameworks/count countSecurityControlFrameworks
GET /api/security-control-frameworks/{id} getSecurityControlFramework
PUT /api/security-control-frameworks/{id} updateSecurityControlFramework
PATCH /api/security-control-frameworks/{id} partialUpdateSecurityControlFramework
security-control-resource Security Control Resource
GET /api/security-controls getAllSecurityControls
GET /api/security-controls/count countSecurityControls
GET /api/security-controls/{id} getSecurityControl
PUT /api/security-controls/{id} updateSecurityControl
PATCH /api/security-controls/{id} partialUpdateSecurityControl

security-metric-resource Security Metric Resource

- GET `/api/security-metrics` getAllSecurityMetrics
- POST `/api/security-metrics` createSecurityMetric
- GET `/api/security-metrics/count` countSecurityMetrics
- GET `/api/security-metrics/{id}` getSecurityMetric
- PUT `/api/security-metrics/{id}` updateSecurityMetric
- DELETE `/api/security-metrics/{id}` deleteSecurityMetric
- PATCH `/api/security-metrics/{id}` partialUpdateSecurityMetric

similar-control-resource Similar Control Resource

- GET `/api/similar-controls` getAllSimilarControls
- GET `/api/similar-controls/count` countSimilarControls
- GET `/api/similar-controls/{id}` getSimilarControl
- PUT `/api/similar-controls/{id}` updateSimilarControl
- PATCH `/api/similar-controls/{id}` partialUpdateSimilarControl

target-value-resource Target Value Resource

- GET `/api/target-values` getAllTargetValues
- POST `/api/target-values` createTargetValue
- GET `/api/target-values/count` countTargetValues
- GET `/api/target-values/{id}` getTargetValue
- PUT `/api/target-values/{id}` updateTargetValue
- DELETE `/api/target-values/{id}` deleteTargetValue
- PATCH `/api/target-values/{id}` partialUpdateTargetValue

tom-resource Tom Resource

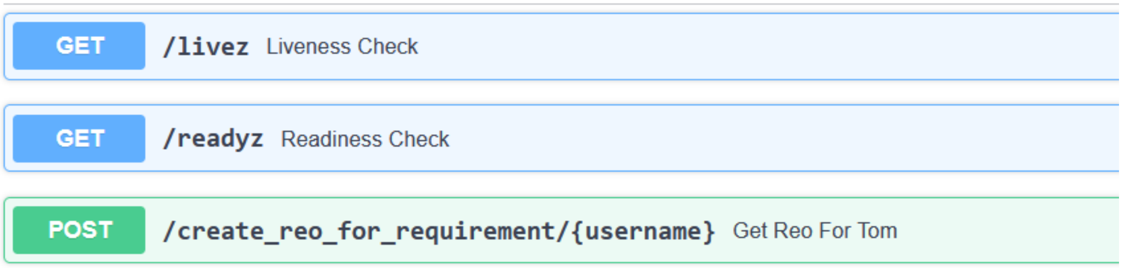
- GET `/api/toms` getAllToms
- GET `/api/toms/count` countToms
- GET `/api/toms/framework-assurance/{frameworkName}` getTomsByFrameworkName
- GET `/api/toms/framework-assurance/{frameworkName}/{assuranceLevel}` getTomsByFrameworkNameAndAssuranceLevel
- GET `/api/toms/{id}` getTom
- PUT `/api/toms/{id}` updateTom
- PATCH `/api/toms/{id}` partialUpdateTom

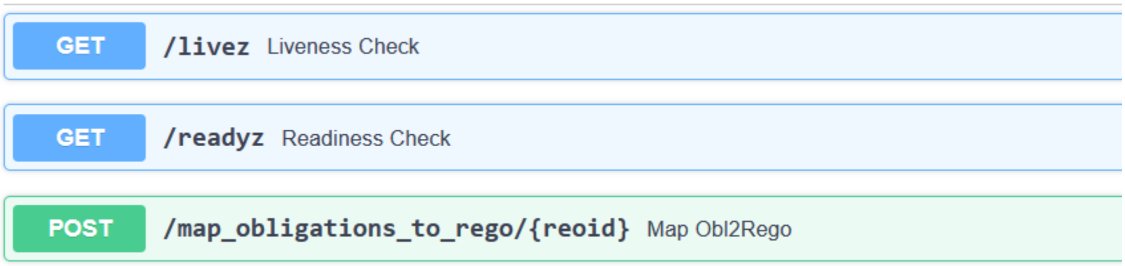
user-resource User Resource

- GET `/api/admin/users` getAllUsers
- GET `/api/admin/users/{login}` getUser

Component: NL2CNL Translator and DSL Mapper

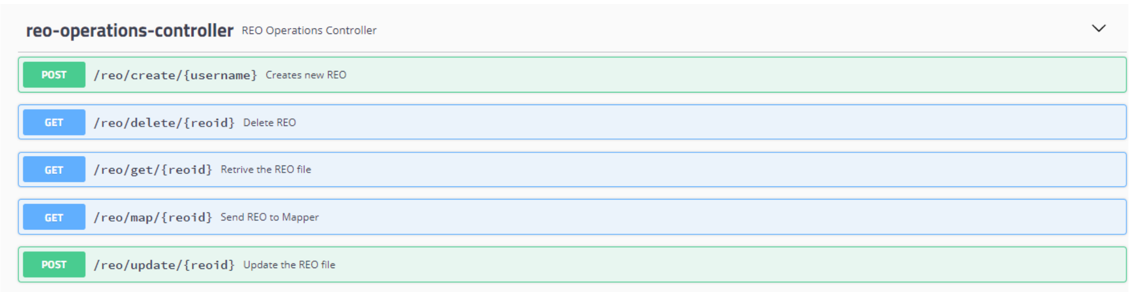
The following screenshots show available APIs that can be used by the other components to interact with the NL2CNL Translator and the DSL Mapper, respectively.





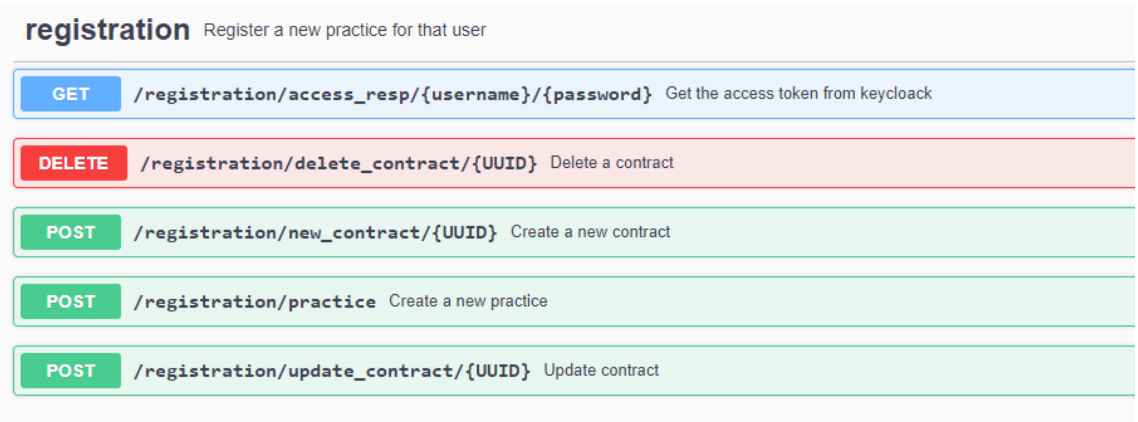
Component: CNL Editor

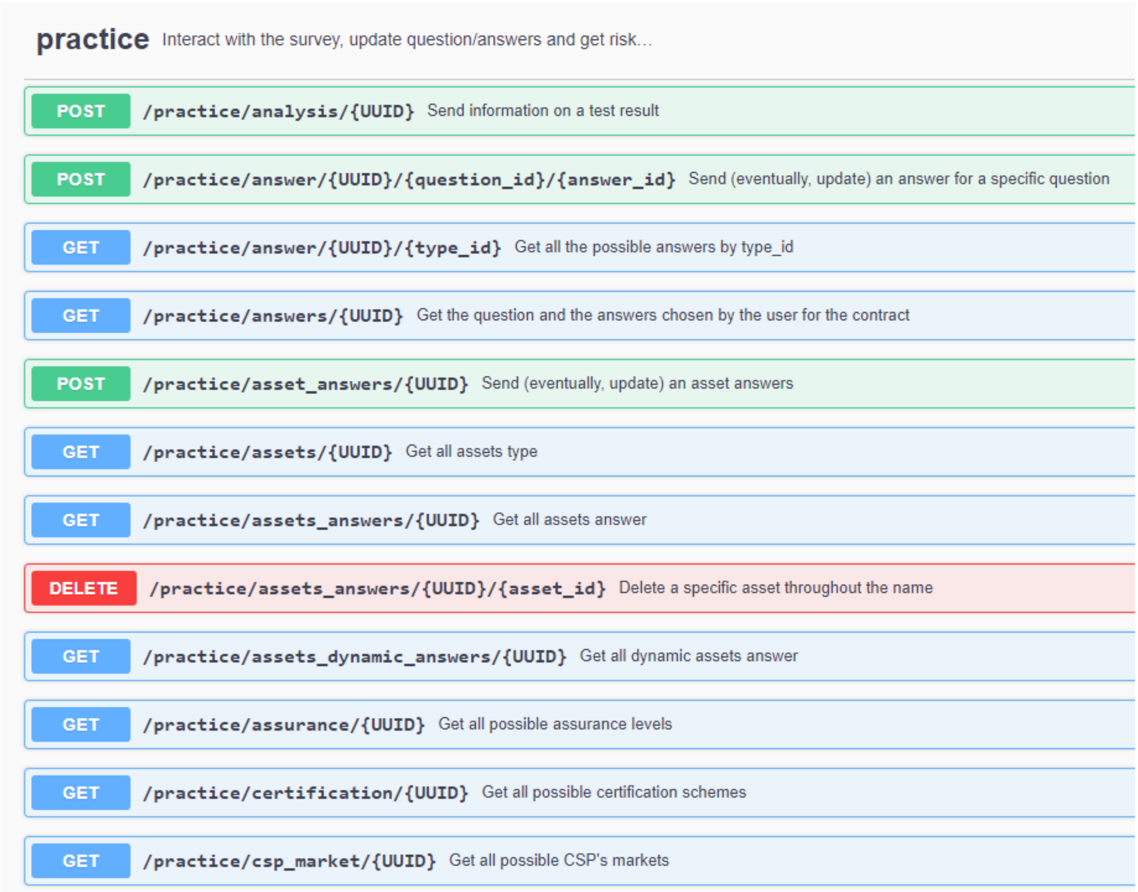
The following screenshot shows the list of available APIs that can be used by the components interacting with the CNL Editor.

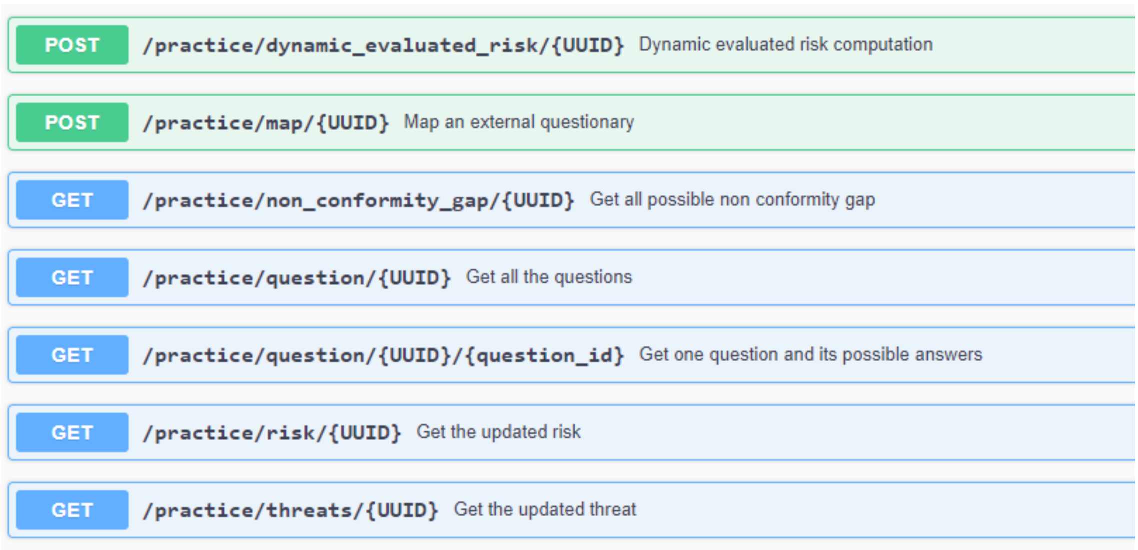


Component: Risk Assessment and Optimisation Framework

The following screenshots show the list of available APIs that can be used by the components interacting with RAOF.

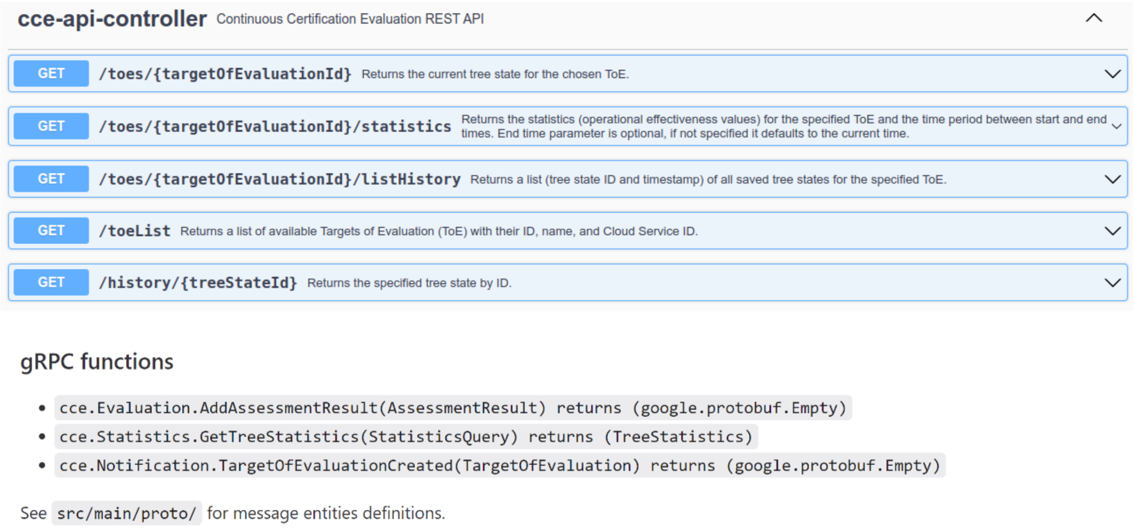






Component: Continuous Certification Evaluation

The following screenshots shows the list of available APIs that can be used by the components interacting with CCE.



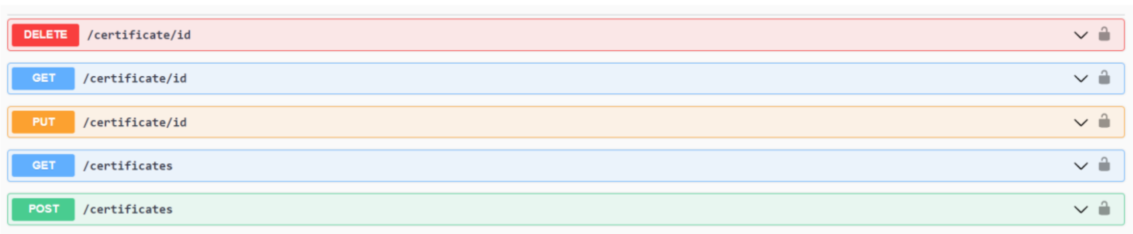
The complete technical specification (request and response parameters and types) of the gRPC API is available in the CCE repository: https://git.code.tecnalia.com/medina/public/continuous-certification-evaluation/-/tree/main/src/main/proto

Component: Life Cycle Manager

The following screenshot shows the list of available APIs that can be used by the components interacting with LCM.

Component: Automated Self-Sovereign Identity-based certificates management (SSI)

The following screenshot shows the list of available APIs that can be used by the components interacting with SSI.



Component: Assessment and Management of Organizational Evidence – AMOE

The following screenshot shows the list of available APIs that can be used by the components interacting with AMOE.

GET	/api/v1/files/{cloud_service_id}	AMOE List Files Cloud Sevice
POST	/api/v1/files/	AMOE List Files Cloud Sevices
GET	/api/v1/file/{file_id}	AMOE Get File
GET	/api/v1/evidence/list/{file_id}	AMOE Get List Evidence For File
POST	/api/v1/evidence/list_per_metric_id	AMOE Get List Evidence Per Metric
GET	/api/v1/evidence/{evidence_id}	AMOE Get Evidence
POST	/api/v1/evidence/assessment	AMOE Set Assessment Result
GET	/api/v1/evidence/send_to_orchestrator/{evidence_id}	AMOE Send Assessment Result
GET	/api/v1/evidence/file/{evidence_id}	AMOE Get HTML File
GET	/api/v1/file/pdf/{file_id}	AMOE Get PDF File
POST	/api/v1/file/{cloud_service}	AMOE Upload PDF File
GET	/api/v1/file/delete/{file_id}	AMOE Delete File And Evidence

Component: Orchestrator

The following screenshots show the list of available APIs that can be used by the components interacting with the Orchestrator.

Orchestrator ^

GET	/v1/orchestrator/assessment_results
POST	/v1/orchestrator/assessment_results
GET	/v1/orchestrator/assessment_tools
POST	/v1/orchestrator/assessment_tools
GET	/v1/orchestrator/assessment_tools/{toolId}
PUT	/v1/orchestrator/assessment_tools/{toolId}
DELETE	/v1/orchestrator/assessment_tools/{toolId}
GET	/v1/orchestrator/catalogs
POST	/v1/orchestrator/catalogs
GET	/v1/orchestrator/catalogs/{catalogId}
PUT	/v1/orchestrator/catalogs/{catalogId}
DELETE	/v1/orchestrator/catalogs/{catalogId}
GET	/v1/orchestrator/catalogs/{catalogId}/categories/{categoryName}/controls/{controlId}
GET	/v1/orchestrator/catalogs/{catalogId}/category/{categoryName}
GET	/v1/orchestrator/certificates
POST	/v1/orchestrator/certificates
GET	/v1/orchestrator/certificates/{certificateId}
PUT	/v1/orchestrator/certificates/{certificateId}
DELETE	/v1/orchestrator/certificates/{certificateId}
GET	/v1/orchestrator/cloud_services
POST	/v1/orchestrator/cloud_services

GET /v1/orchestrator/cloud_services/{cloudServiceId}

PUT /v1/orchestrator/cloud_services/{cloudServiceId}

DELETE /v1/orchestrator/cloud_services/{cloudServiceId}

GET /v1/orchestrator/cloud_services/{cloudServiceId}/catalogs/{catalogId}/toes

PUT /v1/orchestrator/cloud_services/{cloudServiceId}/catalogs/{catalogId}/toes

DELETE /v1/orchestrator/cloud_services/{cloudServiceId}/catalogs/{catalogId}/toes

GET /v1/orchestrator/cloud_services/{cloudServiceId}/metric_configurations

GET /v1/orchestrator/cloud_services/{cloudServiceId}/metric_configurations/{metricId}

PUT /v1/orchestrator/cloud_services/{cloudServiceId}/metric_configurations/{metricId}

GET /v1/orchestrator/controls

GET /v1/orchestrator/metrics

POST /v1/orchestrator/metrics

GET /v1/orchestrator/metrics/{metricId}

PUT /v1/orchestrator/metrics/{metricId}

GET /v1/orchestrator/metrics/{metricId}/implementation

PUT /v1/orchestrator/metrics/{metricId}/implementation

GET /v1/orchestrator/toes

POST /v1/orchestrator/toes

Component: Trustworthiness System

The following screenshots show the list of available APIs that can be used by the components interacting with the Trustworthiness System.

POST /client/account

GET /client/account

POST /client/wallet

GET /client/wallet

POST /client/registration

GET /client/admin

POST /client/admin

DELETE /client/admin

GET /client/adminnum

GET /client/orchestratorsnum

GET /client/orchestrator/evidence/check

GET /client/orchestrator/assessment/checkhash

GET /client/orchestrator/assessment/checkcompliance

GET /client/orchestrators

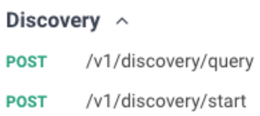
GET /client/authorizedowner

POST /client/authorizedowner

DELETE /client/authorizedowner

Component: Evidence Collection (Cloud Discovery)

The following screenshot shows the list of available APIs that can be used by the components interacting with Evidence Collection.



Component: Security Assessment (Clouditor)

The following screenshot shows the list of available APIs that can be used by the components interacting with Security Assessment.