



# MEDINA

**Deliverable D5.3**

**MEDINA integrated solution-v1**

<b>Editor(s):</b>	Debora Benedetto, Daniele Garbagnati, Mirko Manea, Claudia Zago
<b>Responsible Partner:</b>	Hewlett Packard Italiana, SRL
<b>Status-Version:</b>	Final – v1.1
<b>Date:</b>	30.09.2022
<b>Distribution level (CO, PU):</b>	PU

<b>Project Number:</b>	952633
<b>Project Title:</b>	MEDINA

<b>Title of Deliverable:</b>	MEDINA integrated solution-V1
<b>Due Date of Delivery to the EC</b>	31.01.2022

<b>Workpackage responsible for the Deliverable:</b>	WP5 - MEDINA Framework Integration
<b>Editor(s):</b>	Debora Benedetto, Daniele Garbagnati, Mirko Manea, Claudia Zago (HPE)
<b>Contributor(s):</b>	TECNALIA, Bosch, CNR, Fabasoft, FhG, XLAB
<b>Reviewer(s):</b>	Leire Orue-Echebarria, Cristina Martínez (TECNALIA)
<b>Approved by:</b>	All Partners
<b>Recommended/mandatory readers:</b>	WP2, WP3, WP4, WP6

<b>Abstract:</b>	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.
<b>Keyword List:</b>	Architecture, Workflows, Components Integration, CI/CD, Integrated UI
<b>Licensing information:</b>	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) <a href="http://creativecommons.org/licenses/by-sa/3.0/">http://creativecommons.org/licenses/by-sa/3.0/</a>
<b>Disclaimer</b>	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	17.11.2021	Initial TOC	HPE
v0.2	23.12.2021	Comments and suggestions received by consortium partners.	ALL
v0.3	14.01.2022	HPE and all the partners contributing completed their assigned section.	TECNALIA, Fabasoft, FhG, HPE, CNR, Bosch, XLAB
v0.4	17.01.2022	Integrated document ready for internal review	HPE
v0.5	19.01.2022	Internal review execution to return to HPE for final consolidation	TECNALIA
v0.9	25.01.2022	Consolidated version returned to TECNALIA for submission	HPE
v1.0	31.01.2022	Ready for submission	TECNALIA
v1.01	28.07.2022	Comments from EU review implemented. Ready for QA review	HPE, TECNALIA
v1.02	21.09.2022	Received comments from QA review	TECNALIA
v1.03	22.09.2022	Addressed all comments received in the internal QA review	HPE
v1.1	30.09.2022	Ready for submission	TECNALIA

## Table of contents

Terms and Abbreviations .....	7
Executive Summary .....	8
1 Introduction .....	9
1.1 About this deliverable .....	9
1.2 Document structure .....	9
2 MEDINA Test Bed and Secure DevOps Infrastructure .....	11
2.1 Test Bed Environment .....	11
2.1.1 Hardware Infrastructure .....	11
2.1.2 Operating Environment.....	12
2.1.3 Components Integration Methodology .....	16
2.2 Design of the CI/CD Solution .....	22
3 Generic Architectural Workflows.....	24
3.1 WF1 - Preparation of Target of Certification (ToC) .....	25
3.1.1 Related Architectural Components.....	25
3.1.2 Workflow.....	27
3.2 WF2 - Preparation of MEDINA Components.....	27
3.2.1 Related Architectural Components.....	27
3.2.2 Workflow.....	29
3.3 WF3 - EUCS deployment on ToC .....	30
3.3.1 Related Architectural Components.....	30
3.3.2 Workflow.....	32
3.4 WF4 - EUCS Preparedness – ToC Self-Assessment.....	33
3.4.1 Related Architectural Components.....	33
3.4.2 Workflow.....	35
3.5 WF5 - EUCS Compliance Assessment .....	36
3.5.1 Related Architectural Components.....	36
3.5.2 Workflow.....	38
3.6 WF6 - EUCS – Maintenance of ToC certificate .....	39
3.6.1 Related Architectural Components.....	40
3.6.2 Workflow.....	42
3.7 WF7 - EUCS –Report on ToC Certificate .....	43
3.7.1 Related Architectural Components.....	43
3.7.2 Workflow.....	45
4 MEDINA Framework Components and Integration .....	46
4.1 Catalogue (block #1).....	48
4.1.1 Catalogue of Controls & Security Schemes.....	48

4.2	NLP Technique (block #2)	53
4.2.1	CNL Translator	53
4.2.2	CNL Editor	53
4.2.3	DSL Mapper	55
4.3	Risk Assessment and Optimisation Framework (block #3)	55
4.3.1	Risk Assessment and Optimisation Framework (RAOF) (block #3)	55
4.4	Continuous Evaluation and Life Cycle Manager (block #4)	56
4.4.1	Continuous Certification Evaluation	56
4.4.2	Life Cycle Manager	56
4.5	Organizational Evidence Gathering and Processing (block #5)	57
4.6	Orchestrator and Databases (block #6)	57
4.6.1	Orchestrator and Databases	57
4.6.2	Trustworthiness System	57
4.7	Evidence Collection and Security Assessment (block #7)	61
4.7.1	Evidence Collection	61
4.7.2	Security Assessment (Clouditor)	62
4.7.3	SSI-based certificate management System	62
5	MEDINA User Interface (block #8)	64
5.1	Implementation	64
5.1.1	Functional description	64
5.1.2	Technical description	64
5.1.3	Delivery and usage	68
6	Conclusions	70
7	References	71
	APPENDIX A: Published APIs	73
	Section: Catalogue of Controls & Security Schemes	73
	Section: CNL Translator and DSL Mapper	77
	Section: CNL Editor	77
	Section: Risk Assessment and Optimisation Framework	78
	Section: Continuous Evaluation	78
	Section: Life Cycle Manager	78
	Section: Orchestrator	79
	Section: Trustworthiness System	80
	Section: Evidence Collection (Cloud Discovery)	80
	Section: Security Assessment (Clouditor)	80

---

## List of tables

---

TABLE 1. POINT TO POINT COMMUNICATION TESTS .....	21
TABLE 2. GENERIC MEDINA WORKFLOWS .....	24
TABLE 3. WF1 DESCRIPTION.....	27
TABLE 4. WF2 DESCRIPTION.....	29
TABLE 5. WF3 DESCRIPTION.....	32
TABLE 6. WF4 DESCRIPTION.....	35
TABLE 7. WF5 DESCRIPTION.....	38
TABLE 8. WF6 DESCRIPTION.....	42
TABLE 9. WF7 DESCRIPTION.....	45
TABLE 10. INTEGRATION STRATEGY FOR THE DIFFERENT MEDINA COMPONENTS.....	65
TABLE 11. INTEGRATION TESTED ENDPOINTS .....	68
TABLE 12. PACKAGE STRUCTURE .....	68

---



---

## List of figures

---

FIGURE 1. KUBERNETES CLUSTER INSTALLATION WITH RKE .....	12
FIGURE 2. EXCERPT OF MEDINA'S DOCKER REGISTRY .....	13
FIGURE 3. URL NAMING CONVENTION FOR DEV/TEST ENVIRONMENTS .....	14
FIGURE 4. SERVICE ACCOUNT TYPE USED FOR THE KUBERNETES DASHBOARD.....	14
FIGURE 5. KUBERNETES DASHBOARD .....	15
FIGURE 6. SPRING SWAGGER TEMPLATE ON GITLAB.....	17
FIGURE 7. SAMPLE PROJECT DEPLOYMENT STEPS.....	17
FIGURE 8. DEMO PROJECT IN THE TEST ENVIRONMENT.....	18
FIGURE 9. K8S DASHBOARD: COMPONENTS DEPLOYED IN DEV ENVIRONMENT .....	19
FIGURE 10. STATUS OF THE FIRST INTEGRATION OF COMPONENTS .....	20
FIGURE 11. CI/CD PIPELINES.....	22
FIGURE 12. WF1 - PREPARATION OF TARGET OF CERTIFICATION .....	26
FIGURE 13. WF2 - PREPARATION OF MEDINA COMPONENTS .....	28
FIGURE 14. WF3 - EUCS DEPLOYMENT ON TOC .....	31
FIGURE 15. WF4 - EUCS PREPAREDNESS – TOC SELF-ASSESSMENT.....	34
FIGURE 16. WF5 - EUCS COMPLIANCE ASSESSMENT.....	37
FIGURE 17. CERTIFICATE MAINTENANCE (SOURCE: EUCS VERSION 2020).....	40
FIGURE 18. WF6 - EUCS – MAINTENANCE OF TOC CERTIFICATE .....	41
FIGURE 19. WF7 - EUCS – REPORT ON TOC CERTIFICATE .....	44
FIGURE 20. MEDINA ARCHITECTURE AND DATA FLOW .....	47
FIGURE 21. WINDOW OF LIST OF SECURITY CONTROLS .....	50
FIGURE 22. LIST OF TOMS.....	51
FIGURE 23. DETAILS PAGE OF A SECURITY CONTROL .....	52
FIGURE 24. CNL EDITOR ARCHITECTURE (ADAPTED FROM [15]).....	54
FIGURE 25. TRUSTWORTHINESS SYSTEM GENERAL DASHBOARD.....	59
FIGURE 26. TRUSTWORTHINESS SYSTEM SPECIFIC DASHBOARD FOR EACH ORCHESTRATOR .....	60
FIGURE 27. MEDINA UI ARCHITECTURE.....	64
FIGURE 28. KEYCLOAK LOGIN PAGE .....	66
FIGURE 29. FULL SCREEN FRAME EMBEDDING - CATALOGUE AND INTEGRATED UI .....	67
FIGURE 30. RESPONSIVE IFRAME EMBEDDING - CATALOGUE AND INTEGRATED UI .....	67
FIGURE 31. EXAMPLE: FRAME BUTTON IS HIDDEN IN NAVIGATION BAR FOR USER WITHOUT ROLE ADMIN AND ROUTING IS INHIBITED.....	68

## Terms and Abbreviations

API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Delivery
CAB	Conformance Assessment Body
CNL	Controlled Natural Language
CSA or EU CSA	Coordination and Support Action
CSP	Cloud Service Provider
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
gRPC	Google Remote Procedure Call
GUI	Graphical User Interface
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
JWT	Java Web Token
KPI	Key Performance Indicator
KR	Key Results
NCCA	National Cybersecurity Certification Authority
NL	Natural Language
OS	Operating System
PaaS	Platform as a Service
RAOF	Risk Assessment and Optimisation Framework
RKE	Rancher Kubernetes Engine
SaaS	Software as a Service
SSL	Secure Sockets Layer
SSO	Single Sign-On
SW	Software
SWForum.eu	European forum of the software research community
ToC	Target of Certification
TOM	Technical and Organizational Measure
UI	User Interface
WF	Workflow
VM	Virtual Machine

## Executive Summary

The first version of this deliverable, D5.3, is the result of the Task 5.3 of WP5 delivered in M15. In this first version, the goal is to have an initial prototype of the MEDINA Framework integrating all components developed by the other technical WPs with the core functionalities implemented. The document contributes to pursuing the following objectives of the work package:

- The definition, set up and maintenance of the Secure DevOps infrastructure to support the CI/CD approach.
- To perform the appropriate continuous integration activities of the components and KRs developed in WP2 – WP4.

The next versions of the deliverable will provide an improved solution by following an incremental approach. The second version at M27 will present increased functionalities considering the feedback coming for the use cases and the final version at M33 will put in place revisions coming from the implementation of the use cases.

The document starts describing the Test Bed environment with hardware and operating details, its installation and configuration. The environment is based on a three-node Kubernetes Cluster that orchestrates all components in the MEDINA Framework. In this deliverable it has been defined the methodology to be adopted during the whole phase of the integration of the components in the MEDINA Framework exploiting webinars and workshops. Then, it goes on the description of MEDINA CI/CD designed solution, how it will support the automation of the processes implementing standardized pipelines. A generic workflow is presented which comprise the MEDINA architecture and data model and consisting of seven different scenarios/interactions identified. Afterwards, the document reports the status of the integration activities. Starting from the most recently version of the general architecture with building blocks, it analyses all components for each block giving a short description of their current status and their published APIs, except for the components of the block five that are still in development. The latest part is dedicated to the MEDINA User Interface, that has not been introduced in any deliverable before, and it is a component of the last building block of the MEDINA Framework.



# 1 Introduction

## 1.1 About this deliverable

The first version of this deliverable, D5.3, aims to have an initial prototype of the MEDINA Framework integrating all components developed by the other technical WPs with the core functionalities implemented by M15.

The document contains, first, the definition of the hardware equipment used to setup the Test Bed environment, and how the environment is implemented going deep in the resources needed for the installation and the configuration. The Test Bed environment is setup with a three nodes Kubernetes cluster, where Kubernetes orchestrates all MEDINA microservices running in Docker containers. The Kubernetes cluster can be reached through the Dashboard, which has been properly installed. In this document it is introduced the methodology through which a component must be integrated in the MEDINA environment. For this purpose, it has been delivered a webinar and a workshop. The webinar consisted of a theoretical explanation of the tools utilized in the Test Bed environment and an example of how to manually integrate a component in this first phase. In the workshop instead partners integrated their components in the MEDINA environment.

Secondly, the document describes the overall design of the CI/CD solution that will be put in place for supporting the MEDINA Framework development and integration activities. This solution foresees to have three pipelines of build, deploy and security to realize the automation of the integration component.

This document also includes the description of the generic Architectural Workflow with seven scenarios, while an entire chapter is dedicated to all the component blocks of the MEDINA Framework, reporting their current integration status. At the end, in this deliverable the MEDINA User Interface is introduced as component that is part of the block eight of the MEDINA Framework.

A second release of the deliverable is foreseen at the beginning of the third year of the project (M27), that will describe an infrastructure with augmented functionalities, and which will leverage the information received from the first implementation of the Use Cases as feedback to review the solution.

This deliverable is the result of Tasks 5.3 - System Continuous Integration and Optimization.

## 1.2 Document structure

The rest of the document is structured as follows:

Section 2 presents the Test Bed Environment, describing the Hardware Infrastructure used, the Kubernetes cluster configuration illustrating the several resources, the implementation of the Kubernetes Dashboard, the description of the methodology adopted for the component integration, through the Docker and Kubernetes webinar and workshop delivered in this first round.

Section 3 describes the generic workflows based on seven example scenarios with related architectural components and describing step-by-step the iterations between architectural components and the generic role(s) being involved.

Section 4 presents the MEDINA Framework component integration. There is a sub section for each block describing all component inside, except for the block five that has components in

development. In particular, the main functionalities of the component, its actual state and the APIs it exposes are reported.

Section 5 is dedicated to the MEDINA User Interface that takes part of the building block eight of the MEDINA Framework and is described as in the section before.

Finally, section 6 reports the conclusions.

## 2 MEDINA Test Bed and Secure DevOps Infrastructure

### 2.1 Test Bed Environment

The MEDINA framework is available on a Test Bed environment to test and verify all the functionalities provided before releasing it to production.

The Test Bed environment is setup with a three nodes Kubernetes [1] cluster with two different, independent and isolated virtual environments:

- **Development:** it is usually unstable, which is necessary for developers to test their modules with not fear if errors or disservice occur. This environment does not affect the end users and is used to improve the code of the MEDINA micro-services in order to deploy them to the Test environment.
- **Test:** it is more stable, used by the developers and users for integration testing and quality assessment activities. The main purpose here is to ensure that all the updates done on the different modules work as expected before releasing them to production.

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

Finally, the production environment will be hosted at Fabasoft and Bosch, each of them having their instances as reported in D5.1 [2].

#### 2.1.1 Hardware Infrastructure

This section describes the list of the hardware equipment used to setup the Test Bed environment. The environment is composed by several Virtual Machines (VM) located in the server infrastructure of TECNALIA.

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. Concretely, the list of VMs is the following:

- 3 nodes for the integration/deployment tasks (**integration, production, cicd.medina.esilab.org**) with these specifications:
  - RAM: 4G
  - Cores: 4
  - HDD: 40GB
  - OS: Ubuntu 18.04
- 3 nodes for the Kubernetes cluster (**k8s00, k8s01, k8s02.medina.esilab.org**). These VMs share the same specifications:
  - RAM: 8GB
  - Cores: 8
  - HDD: 120 GB + 200GB
  - OS: Ubuntu 20.04
- An additional VM used for the CNL Editor (**cnl.medina.esilab.org**). Characteristics:
  - RAM: 8G
  - Cores: 4
  - HDD: 60GB
  - OS: Ubuntu 20.04

These specifications can be scaled up as needed. Further VMs can also be created on-demand, according to the needs of the project.

Moreover, an additional VM used for Wazuh and VAT will be provided to simulate running these services to produce fake data for the MEDINA Framework.

## 2.1.2 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.

### 2.1.2.1 Kubernetes Installation and Configuration

This section illustrates the container orchestration solution that is executed over the setup infrastructure described previously.

Different resources are needed to proceed with the installation and configuration of the cluster. We used RKE [3] for the installation of Kubernetes [1] in the three nodes, Rook/Ceph [4] for the configuration of storage and MetalLB [5] for the network configuration.

The Kubernetes cluster is configured and managed by Rancher Kubernetes Engine (RKE) [3], 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 [6]). 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.

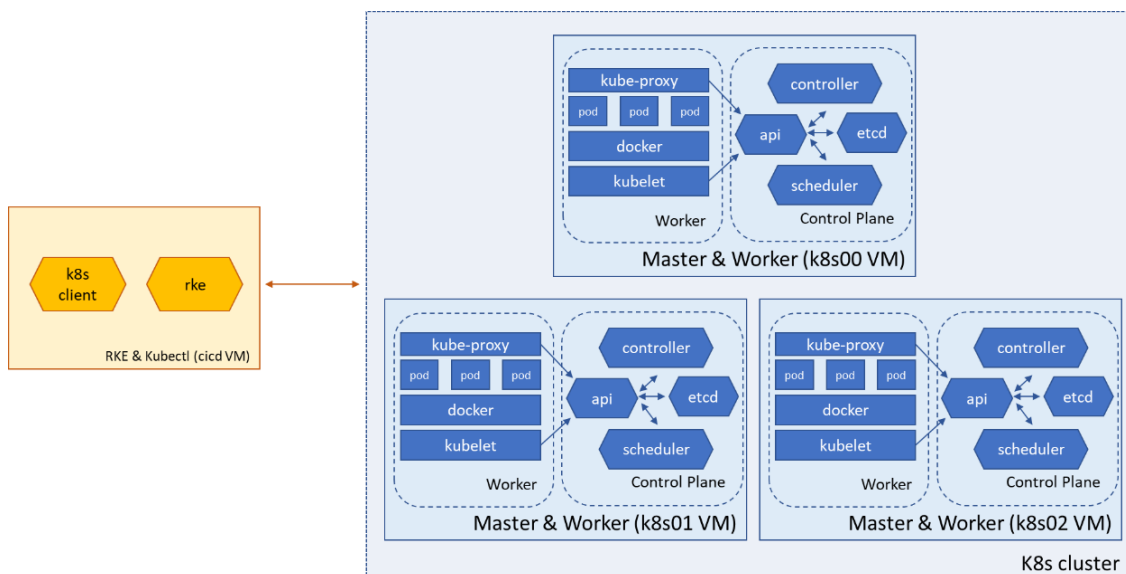


Figure 1. Kubernetes cluster installation with RKE

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 [4] and exposed as a single, unified cluster filesystem.

Ceph is an open-source distributed storage solution for delivering 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 [7].

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 we agreed for the project:

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

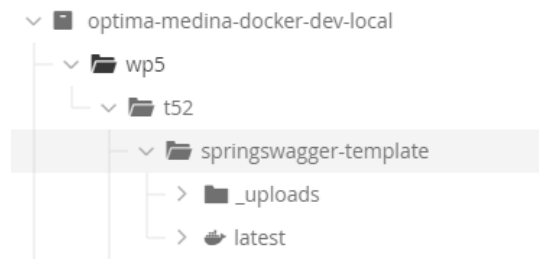


Figure 2. Excerpt of MEDINA's Docker registry

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 a nginx [8] 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 [5] 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
```

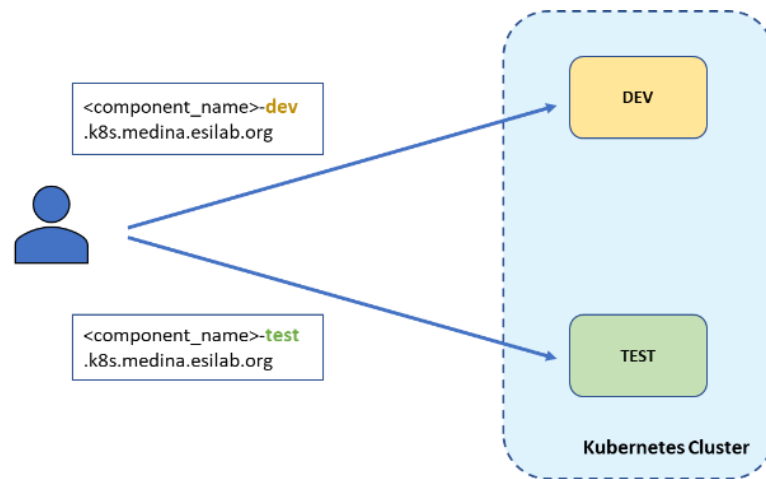


Figure 3. URL naming convention for dev/test environments

### 2.1.2.2 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 [9].

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.

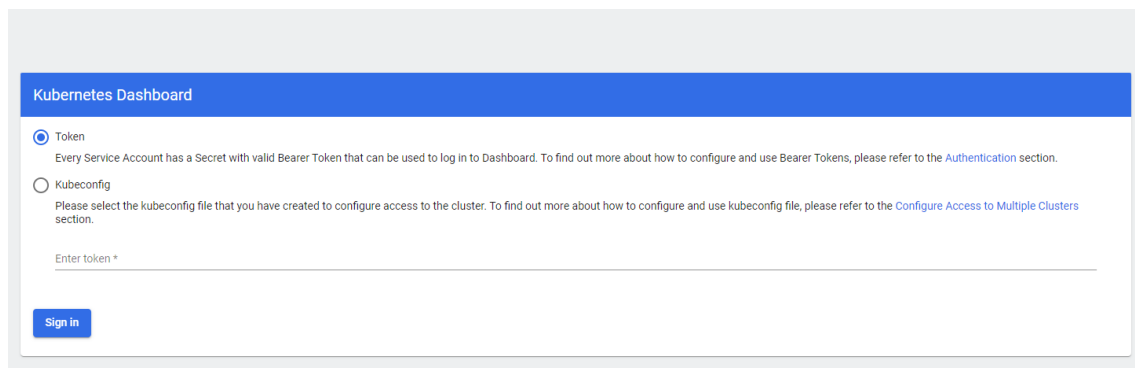


Figure 4. Service Account type used for the Kubernetes Dashboard

The Dashboard is exposed over HTTPS at <https://dashboard.k8s.medina.esilab.org/#/login> [internal use only - authentication required]

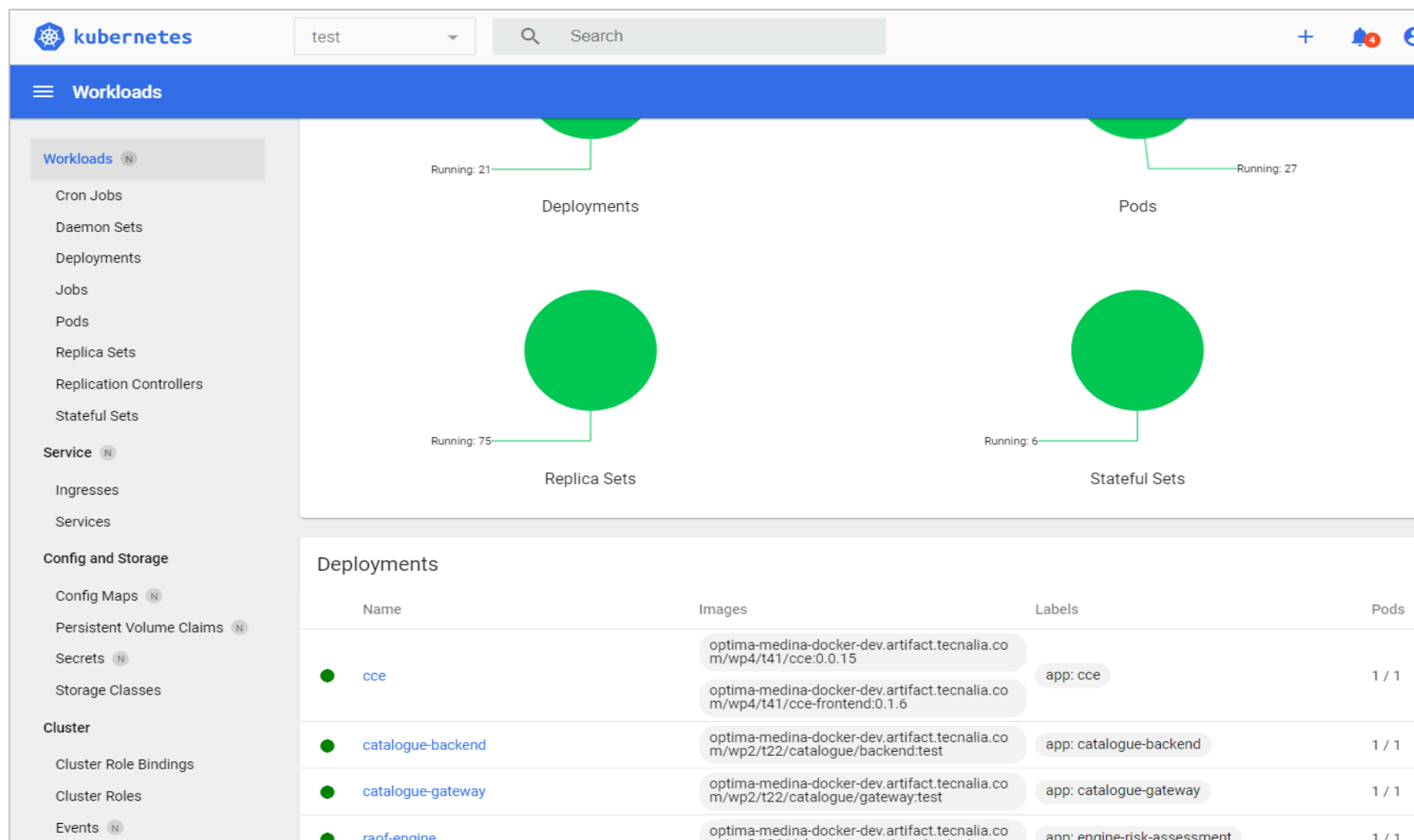


Figure 5. Kubernetes Dashboard

We have a secure Dashboard since we used certificates to expose it over HTTPS. These certificates are installed using cert-manager [10].

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 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.

### 2.1.3 Components Integration Methodology

Once the Test Bed environment has been correctly configured and all the installations needed done, the next steps is the deployment of all the components into the cluster.

In order to better organize the work of the integration we have adopted the following methodology presenting the actions to do up to 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 has 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 Chapter 3 below have been correctly implemented
7. Deploy the stable version of each component into 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 Chapter 3 below have been correctly implemented
11. Release of all the components into the production environment

The methodology is implemented through two instruments: workshops and webinars. The overall integration consists of three rounds: M15, M27, and M33. Currently we are at the first round at M15 and the components integration is done manually by each partner. For this first round, we have delivered a webinar and a workshop. During the webinar we illustrated the Docker and Kubernetes main concepts and functionalities and showed a sample project integration in the MEDINA operating environment. During the 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.

Moreover, the integration status of each component and the advancements of the methodology actions are tracked using a stylesheet available on the Fabasoft shared repository and we check and update it during the WP5 meetings.



The following sections describe the webinar and the workshop conducted in the first round.

### 2.1.3.1 Docker and Kubernetes Webinar with Sample Component Integration example

The components' cluster integration in this initial phase is done manually by all partners, then it will be automated in the next MEDINA framework versions.

To support all partners in this integration, a webinar presenting an example project has been organised. 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 the 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 6. 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 the release of it in the k8s cluster.

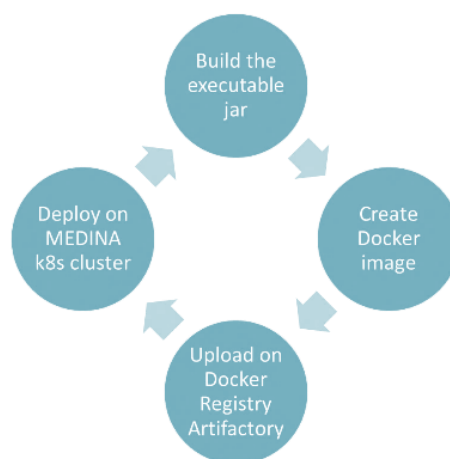


Figure 7. Sample project deployment steps

First of all, the project is packaged with Maven [11] 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#/> [public]

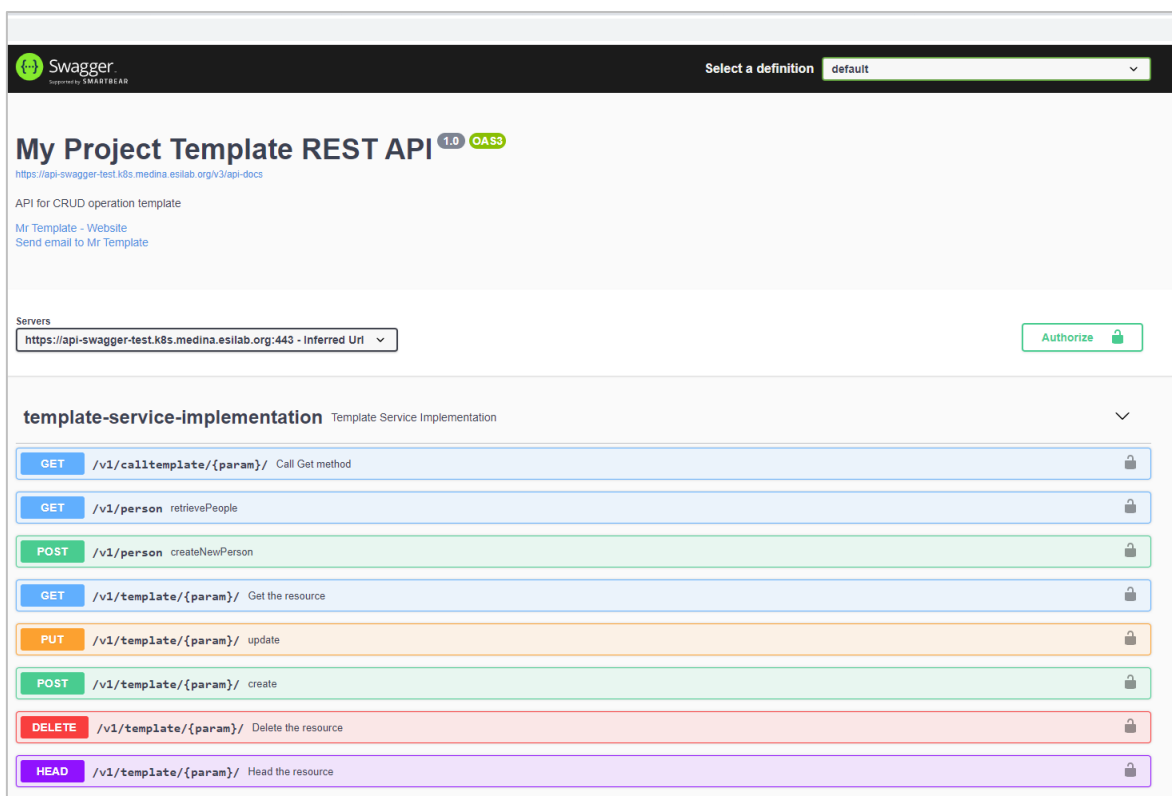


Figure 8. Demo project in the test environment

### 2.1.3.2 First integration Workshop

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

To carry out the integration of the components, the partners were provided with access credentials to GitLab, Docker Registry Artifactory and to 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 was uploaded to GitLab, then the Docker images were pushed on the Artifactory registry and finally the Kubernetes manifest files were 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, as shown in Figure 9.

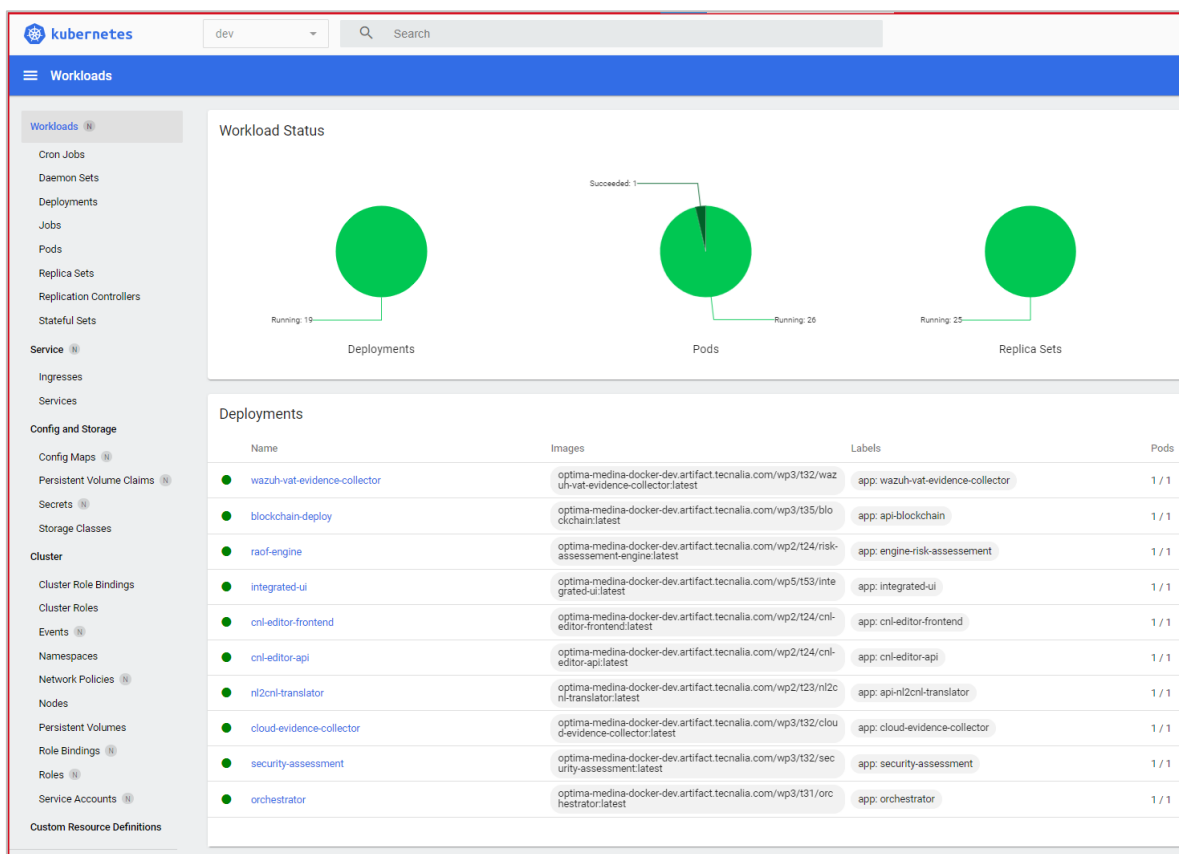


Figure 9. K8s Dashboard: Components deployed in dev environment

Figure 10 shows all the components of the MEDINA Framework: the green ones are released on the development environment, the yellow one will be deployed in the next round and the blue ones will not be released in the Kubernetes cluster.

The Codyze component will be integrated in the MEDINA Security pipeline and the Wazuh and VAT components will 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 (Cloudfitor)	FhG	WP3	T3.2	yes	yes	yes	yes	yes	yes	no	
Security Assessment (Cloudfitor)	FhG	WP3	T3.2	yes	yes	yes	yes	yes	yes	no	
Orchestrator (Cloudfitor)	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 10. Status of the first integration of components

Furthermore, partners performed point to point tests to verify the communication in pairs of the released components and Table 1 shows in green the working ones.

*Table 1. Point to point communication tests*

Component Name	Component Name	Status
Orchestrator	Continuous 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
Continuous Certification Evaluation	Metrics and Measures Catalogue	NEXT ROUND
Continuous 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 Keycloak	CONNECTED
Integration UI	Metrics and Measures Catalogue	CONNECTED
Integration UI	NL2CNL Translator	CONNECTED
Integration UI	Orchestrator	NEXT ROUND

Component Name	Component Name	Status
Organisational Evidence Management Tool	Orchestrator	NEXT ROUND
Integration UI	Organisational Evidence Management Tool	NEXT ROUND

## 2.2 Design of the CI/CD Solution

Starting from the CI/CD strategy outlined in D5.1 [2], this section describes the overall design of the CI/CD solution that will be put in place for supporting the MEDINA Framework development and integration activities.

Our infrastructure is built in a multi-node Kubernetes cluster that orchestrates all the components of the MEDINA Framework. In the first round, the integration has been done manually but in the next round all the components release steps, starting from the build of the project and up to the deployment in the Kubernetes cluster, will be automated. Our solution will use Continuous Integration (CI), Continuous Deployment (CD) practices implemented by the Build, Deploy and Security pipelines designed ad-hoc for MEDINA.

The Continuous Integration practice includes the management of the software source code through a versioning control system, and for this purpose all the MEDINA projects are available on GitLab. For the CI/CD Orchestrator, we installed the open-source Jenkins tool running on `cicd.medina.esilab.org` node.

As shown in the Figure 11, the Build pipeline can be triggered automatically at every push of the project in Gitlab and it automatizes the build of the project, the creation of the Docker image and its push on the Artifactory. Then, if the previous pipeline succeeded, the second Deploy pipeline is triggered and will automatically deploy the component to the development environment. Finally, the Security pipeline is triggered if the Build and the Deploy pipelines succeed.

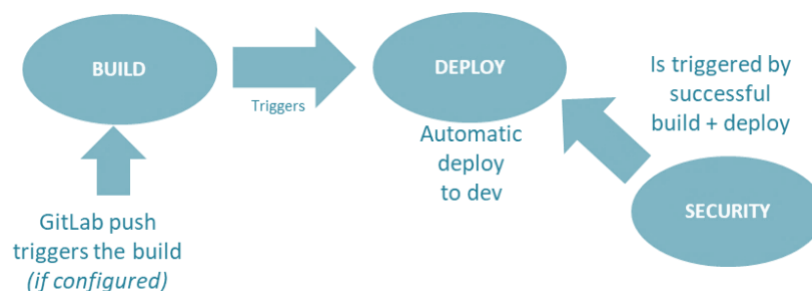


Figure 11. CI/CD pipelines

To make all the deployment process more automated, we use the Jenkins *Seed Jobs* for automating the creation of the three pipelines. This utility permits to fill out a form by entering parameters such as 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. Once these details are provided, the *Seed Job* that automatically creates the three standardized pipelines for build, deploy and security can be run.

The build pipeline will be created by default out of the packaged templates (e.g., a template for standard Java build steps), that can be further tuned with user customizations to address specific

build phases or tools, while the deployment and security pipelines typically can be used as they are.

To guarantee Quality & Assurance into the overall workflow, we will put into the security pipeline three types of security analysis: *Static Code Analysis*, *Vulnerability Analysis* and *Dynamic Analysis* that are performed by different tools.

The aforementioned pipelines will be implemented in the next version and the MEDINA component Codyze [12], described in Section 4.7.1 as tool for static code analysis by FhG partner, will be added in the Security pipeline.

### 3 Generic Architectural Workflows

This section presents the generic workflows (WF from now on) which comprise the MEDINA framework (in particular the architecture and data model), and consisting of the seven different scenarios/interactions shown in Table 2. These workflows cover different data flow paths of the architecture described in D5.1 [2] as it will be shown in the corresponding section, each of them using different components of the MEDINA framework.

Table 2. Generic MEDINA workflows

Workflow	Comment	Other/Dependency
<b>WF1 - Preparation of Target of Certification (ToC)</b>	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.	Prerequisite Mandatory workflow CSP Responsibility Dependencies: None
<b>WF2 - Preparation of MEDINA components</b>	Setup, configure and deploy the MEDINA components. Only related to those components under the responsibility of the CSP.	Prerequisite Mandatory workflow CSP Responsibility Dependencies: WF1
<b>WF3 - EUCS deployment on ToC</b>	Setup, configure and deploy the corresponding EUCS framework (for the chosen assurance level basic/substantial/high) on the ToC.	Prerequisite Mandatory workflow CSP Responsibility Dependencies: WF1, WF2
<b>WF4 - EUCS Preparedness – ToC Self-Assessment</b>	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
<b>WF5 - EUCS – compliance assessment</b>	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
<b>WF6 - EUCS – maintenance of ToC certificate</b>	Start certificate maintenance life-cycle for the ToC. Based on current EUCS, the maintenance process comprises the following stages: (issuance <sup>1</sup> ), renewal, continuation, update, re-issuance (new certificate), withdrawal, and suspension.	Mandatory workflow CAB Responsibility CSP Responsibility Dependencies: WF1, WF2, WF3, WF5
<b>WF7 - EUCS – report on ToC certificate</b>	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

In principle, more complex workflows can be built based on the seven ones presented in this section. Creating and instantiating real-world scenarios based on the generic workflows are goals to be achieved in WP6.

<sup>1</sup> 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).



The rest of this section presents further details about the generic workflows shown in Table 2, and structured in the following manner:

- Related architectural components, which are based on the MEDINA architecture at the time of writing.
- Workflow, which describes step-by-step the iterations between architectural components and the generic role(s) being involved.

### 3.1 WF1 - Preparation of Target of Certification (ToC)

This initial workflow, despite not invoking any of the MEDINA components, is an evident prerequisite 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.

#### 3.1.1 Related Architectural Components

As mentioned above, this workflow does not involve any of the MEDINA components. However, it setups the ToC elements shown in Figure 12, 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 hyperscalers

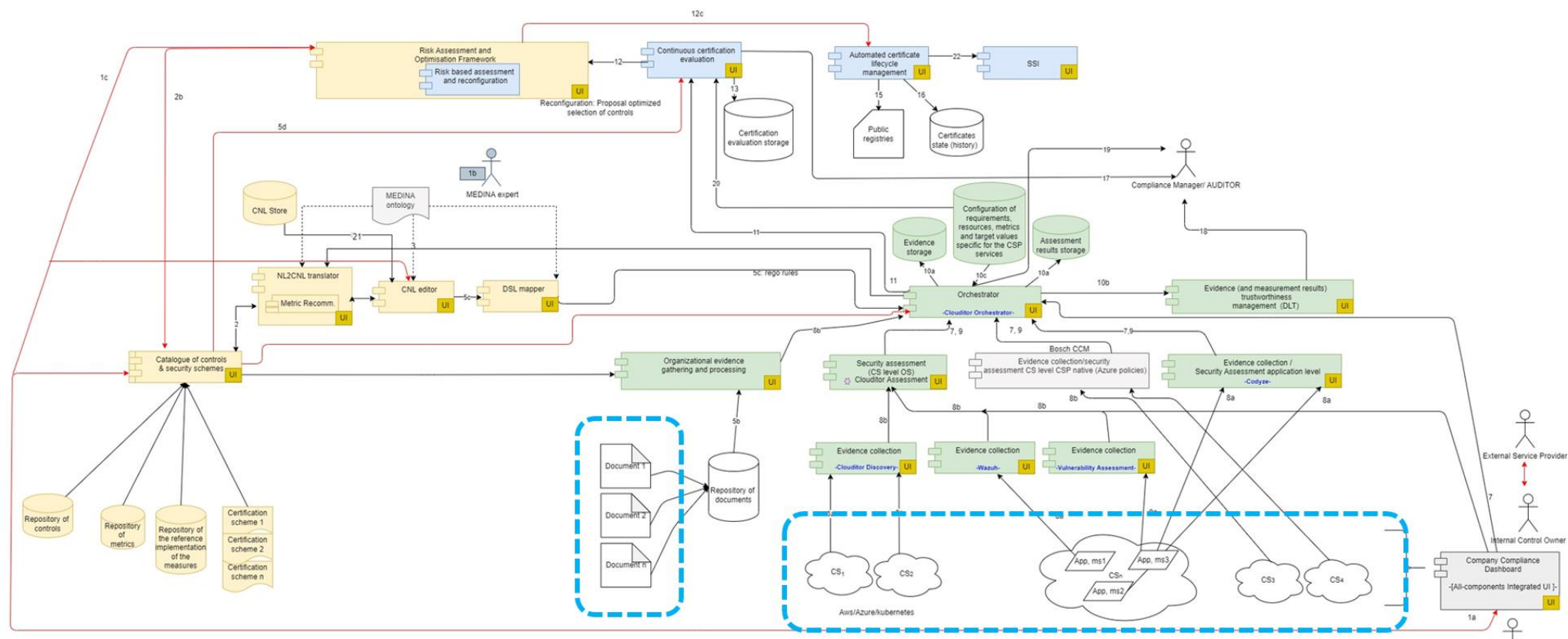


Figure 12. WF1 - Preparation of Target of Certification

### 3.1.2 Workflow

Table 3 describes the steps associated to this workflow.

*Table 3. 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 <sup>2</sup>	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 D5.1 [ref] considers three impacts levels corresponding to each of confidentiality, integrity and availability.

## 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 is a required set of deploying actions before the certification process is triggered by WF3.

### 3.2.1 Related Architectural Components

This workflow involves the components highlighted in Figure 13, namely:

- Catalogue of Controls and Security Schemes
- 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
- Company Compliance Dashboard / Integrated UI

<sup>2</sup> In this generic context, CSP means the entity responsible of the ToC (EUCS requestor).

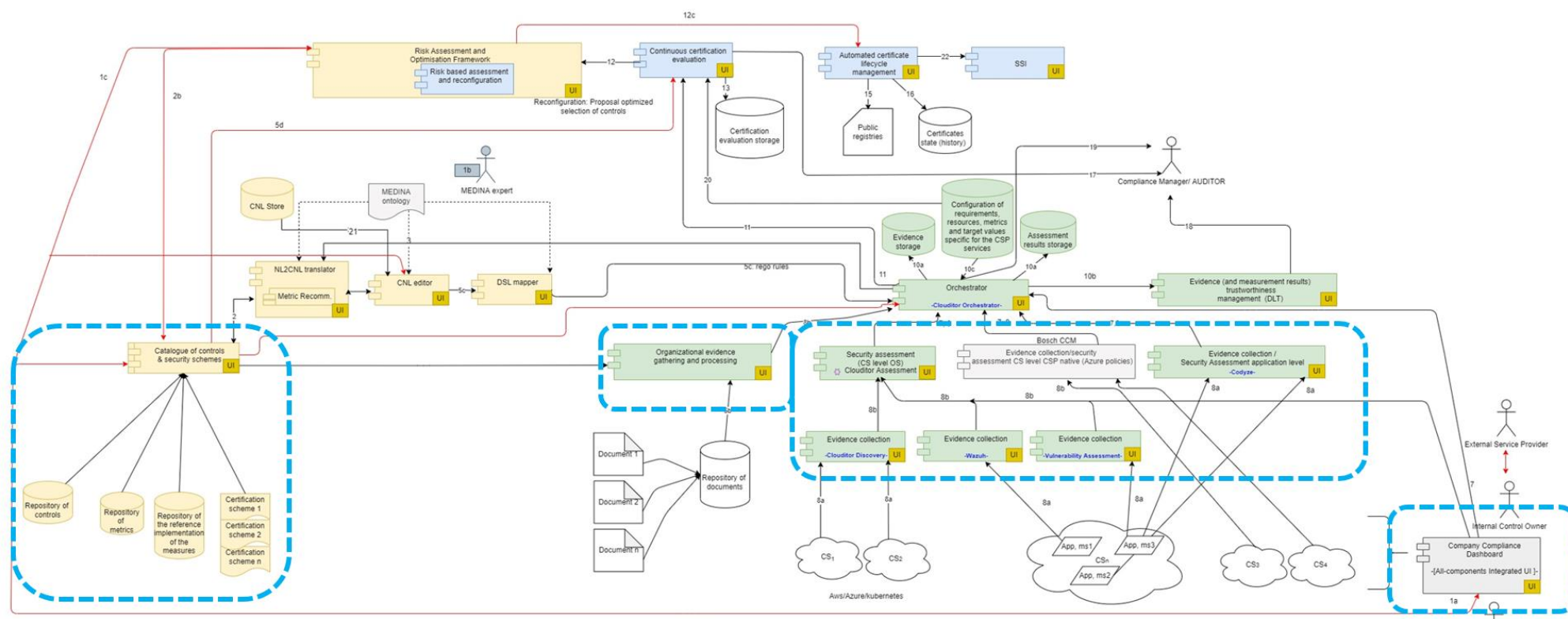


Figure 13. WF2 - Preparation of MEDINA Components

### 3.2.2 Workflow

Table 4 describes the steps associated to this workflow.

Table 4. 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 Security Schemes:  a. Configure the EUCS catalogue with all assurance levels, and including corresponding controls/requirements/metrics.	MEDINA <sup>3</sup>	The Catalogue of Controls and Security Schemes 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:	CSP	Used to gather evidence from technical measures.

<sup>3</sup> This role means the actual MEDINA framework (non-human role).

Step	Description	Role	Comments
	a. Wazuh is configured		
<b>7</b>	Configuration of (Technical) Evidence Collection VAT:  a. VAT is configured	CSP	Used to gather evidence from technical measures.
<b>8</b>	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, and only Keycloak must be configured.

### 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.

#### 3.3.1 Related Architectural Components

This workflow involves the components shown in Figure 14, namely:

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

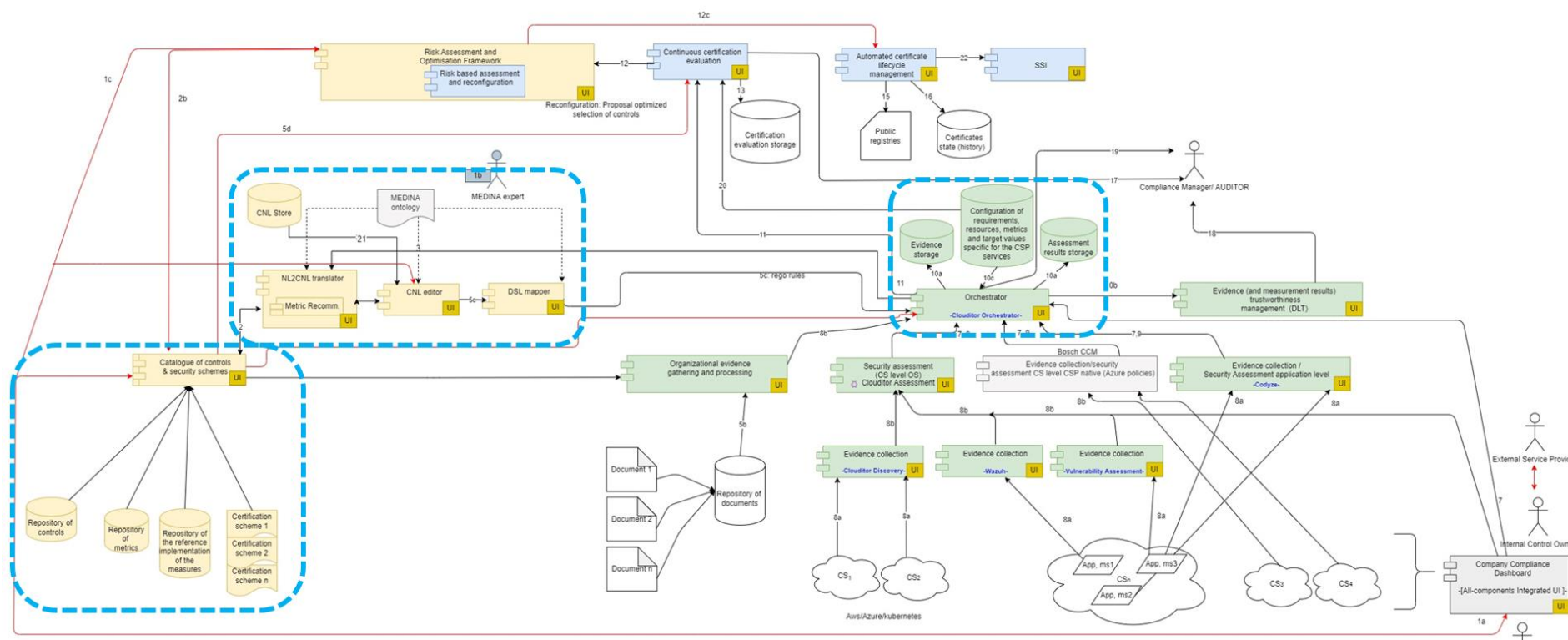


Figure 14. WF3 - EUCS deployment on ToC



### 3.3.2 Workflow

Table 5 describes the steps associated to this workflow.

Table 5. 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 Security Schemes (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 <b>built-in Metrics</b> as provided by the Metrics Recommender (or accept the ones pre-selected by default) b. Customize Target Values <sup>4</sup> on the selected <b>built-in Metrics</b> . c. Add <b>CSP-custom Metrics</b> as needed	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

<sup>4</sup> In the form of Obligations



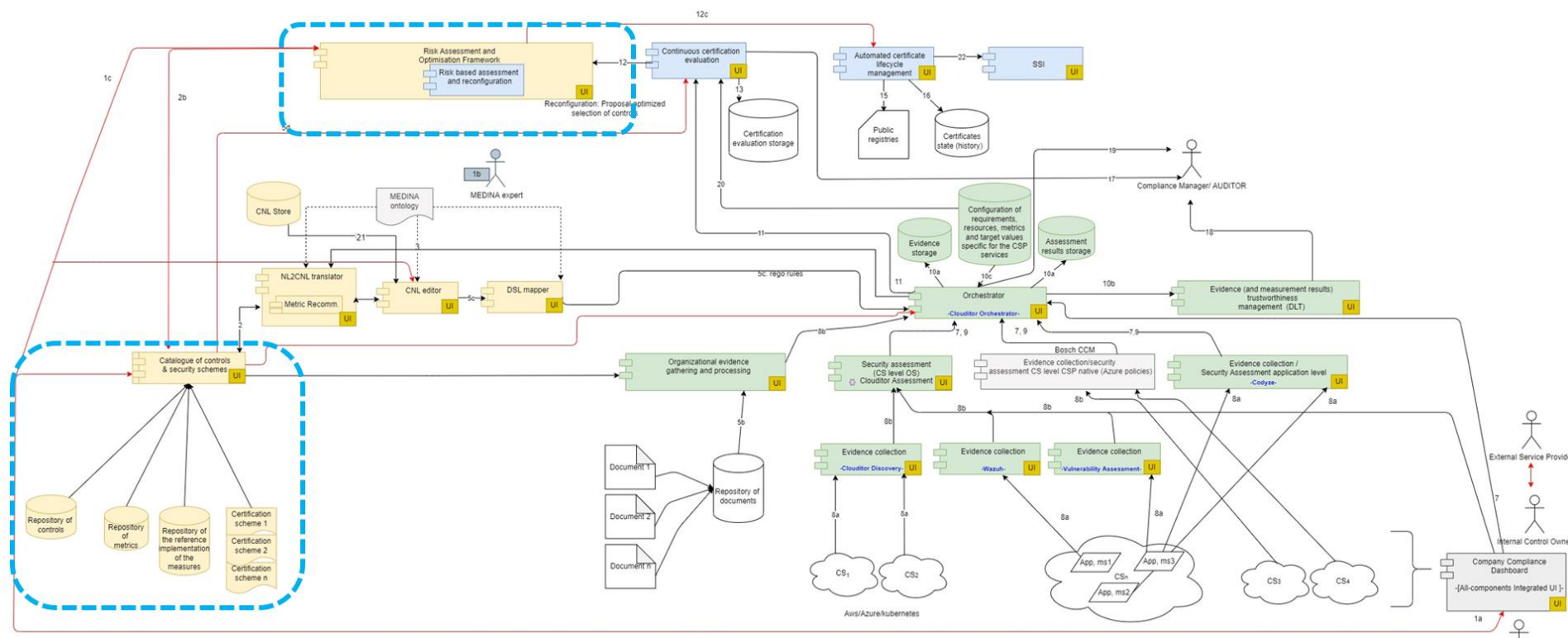
### 3.4 WF4 - EUCS Preparedness – ToC Self-Assessment

This workflow relates to the component in charge of performing the static risk management as documented by “D2.6 - Risk-based techniques and tools for Cloud Security Certification-v1”. Although this component 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.

#### 3.4.1 Related Architectural Components

This workflow involves the components shown in Figure 15, namely:

- Risk Assessment and Optimization Framework
- Catalogue of Controls and Security Schemes



### 3.4.2 Workflow

The related activities in WP4 are described in Table 6.

Table 6. WF4 description

Step	Description	Role	Comments
1	<p>Risk Assessment and Optimization Framework:</p> <ul style="list-style-type: none"> <li>a. ToC information and Impact level (per-Resource type) are entered into the tool</li> <li>b. If applicable, the underlying Hyperscaler is configured as an additional Resource (along with its associated Impact level)</li> <li>c. Targeted EUCS assurance level is selected, as required for the preparedness assessment</li> </ul>	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 example, details about the actual Resources' configuration are not needed for this static assessment.
2	<p>Catalogue of Controls and Security Schemes:</p> <ul style="list-style-type: none"> <li>a. Based on selected EUCS, the self-assessment questionnaire is retrieved and shown to the CSP, so it can proceed to answer about the implementation of requirements</li> </ul>	MEDINA	The preparedness tool is based on a questionnaire interface containing requirements from EUCS, just as described in the referenced D2.6
3	<p>Risk Assessment and Optimization Framework:</p> <ul style="list-style-type: none"> <li>a. CSP provides answers to the questionnaire (cf. Step 2), based on any of the following potential answers: <ul style="list-style-type: none"> <li>a. Implemented (CSP Responsibility)</li> <li>b. Not Implemented (CSP Responsibility)</li> <li>c. Not Applicable</li> </ul> </li> </ul>	CSP	A close 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. 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

Step	Description	Role	Comments
	d. Unknown (Hyperscaler Responsibility) b. Degree of compliance is automatically computed and reported to the CSP		

### 3.5 WF5 - EUCS Compliance Assessment

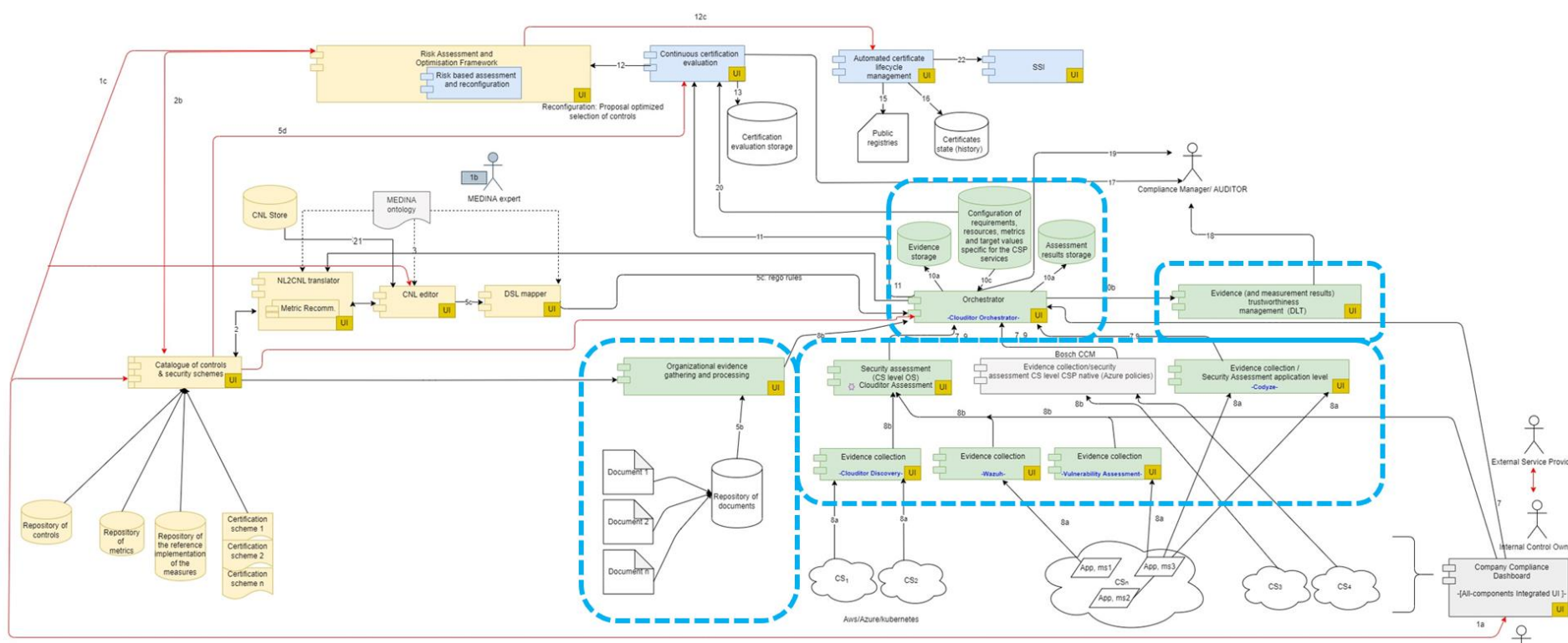
MEDINA proposes the notion of “continuous audit-based certification”, which departs from the EUCS definition of “continuous (automated) monitoring” referring to **periodically assessing the ToC**. This WF5 describes **discrete compliance assessments**, which should then be periodically executed for the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

Further information about the underlying evidence collection mechanisms can be found in “D3.1 Tools and techniques for the management of trustworthy evidence-v1” [13].

#### 3.5.1 Related Architectural Components

This workflow involves the components shown in Figure 16, 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 VAT



### 3.5.2 Workflow

The different interactions corresponding to this WF5 are shown in Table 7.

Table 7. 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 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	D3.1 [13] 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 D3.1 [13] 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 D3.1 [13] 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 D3.1 [13] for further details on metrics' coverage.
6	Evidence Collection / Security Assessment CS level and CSP Native (Azure Policies):	MEDINA	Please refer to D3.1 [13] for further details on metrics' coverage.

Step	Description	Role	Comments
	a. Assesses cloud service-level Resources from the ToC based on selected Metrics		
7	Orchestrator / Clouditor Orchestrator:  a. Assessment Results from <b>organizational</b> assessments are stored b. Evidence from <b>organizational</b> 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 <b>organizational</b> assessments results and evidence are stored	MEDINA	Please refer to comment above.
9	Orchestrator / Clouditor Orchestrator:  a. Assessment Results from <b>technical</b> assessments are stored b. Evidence from <b>technical assessments</b> is stored c. Assessment Results are sent to Continuous Certification Evaluation	MEDINA	n/a
10	Evidence trustworthiness management (DLT):  a. Digest/hash of relevant information related to <b>technical</b> assessment results and evidence are stored	MEDINA	n/a

### 3.6 WF6 - EUCS – Maintenance of ToC certificate

This WF6 departs from the current definition of certificate maintenance in the EUCS core document (see Figure 17) and, for the purposes of MEDINA, it also adds 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.

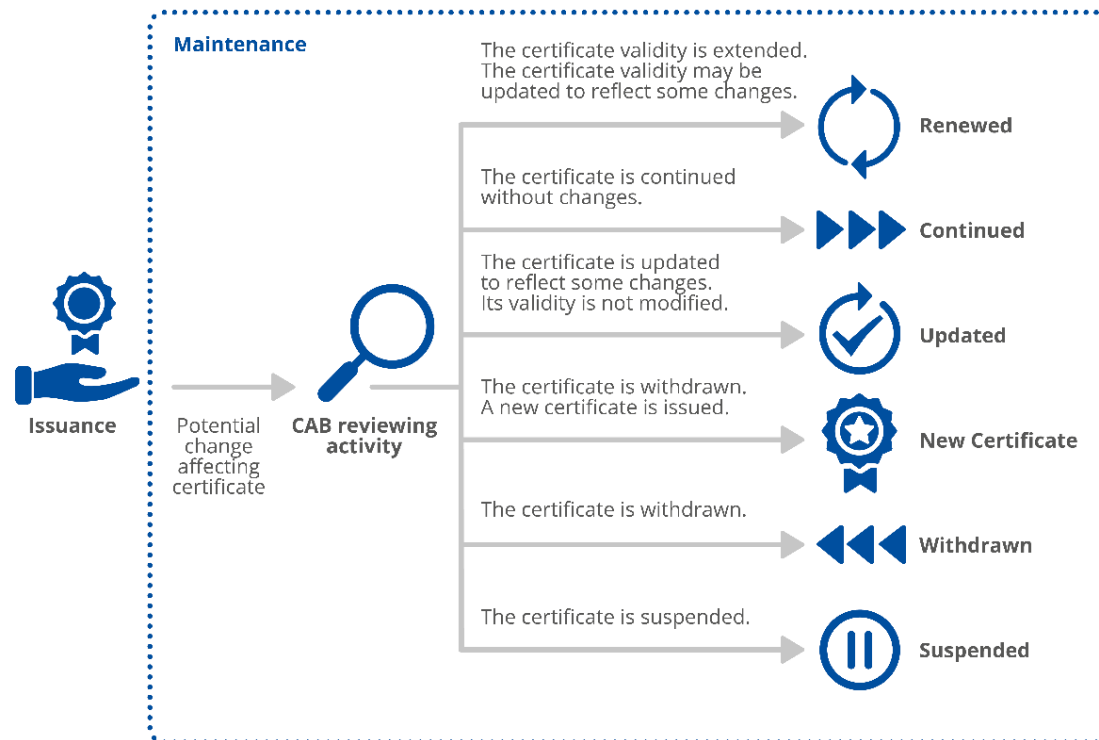


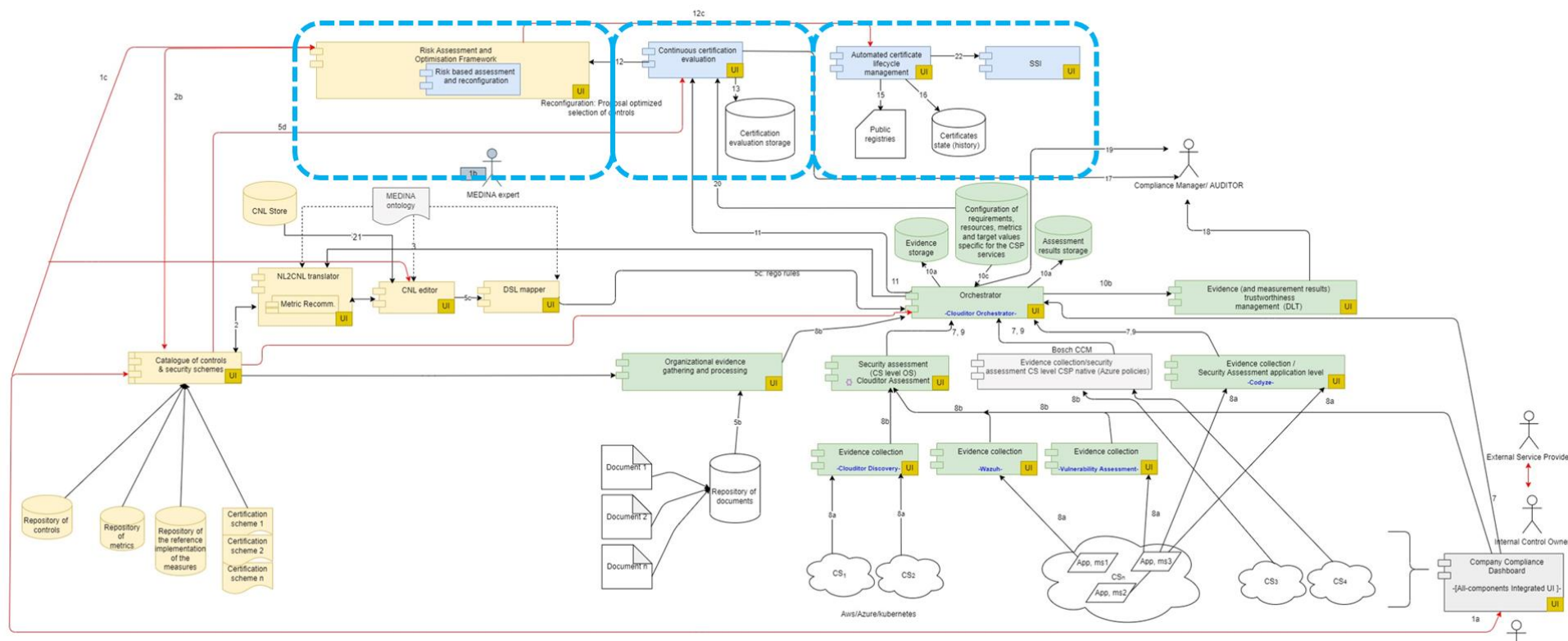
Figure 17. Certificate maintenance (source: EUCS version 2020)

### 3.6.1 Related Architectural Components

This workflow involves the components shown in Figure 18, namely:

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





### 3.6.2 Workflow

The different interactions corresponding to this WF6 are shown in Table 8.

Table 8. WF6 description

Step	Description	Role	Comments
<b>1</b>	<p>Continuous Certification Evaluation:</p> <ul style="list-style-type: none"> <li>a. Assessment Results (point-in-time assessment) are received from Orchestrator / Clouditor Orchestrator</li> <li>b. Tree-based evaluation is performed with received Assessment Results (which are received per-Resource)</li> <li>c. Tree-based evaluation results are stored in Certification Evaluation Storage</li> <li>d. If a non-compliance is found<sup>5</sup>, then the Risk Assessment and Optimization Framework is invoked (see Step 2 below)</li> </ul>	MEDINA	This component automatizes the currently manual audit process for analysing set of evidence (in particular when operational efficiency is in scope, like in the case of EUCS High).
<b>2</b>	<p>Risk Assessment and Optimization Framework:</p> <ul style="list-style-type: none"> <li>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</li> <li>b. The degree of non-compliance is communicated to the Certificate Lifecycle Manager (see Step 4 below)</li> </ul>	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.
<b>3</b>	Automated Certificate Lifecycle Manager:	MEDINA	The core EUCS document defines the basis for MEDINA to implement the

<sup>5</sup> Compliances are not reported to the Risk Assessment and Optimization Framework

Step	Description	Role	Comments
	a. Based on the <i>Operational Effectiveness Criteria</i> 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.		automation of the certificate lifecycle management.
5	Automated Certificate Lifecycle Manager: a. Certificate status is published/updated on the Public Registry	MEDINA	This is a required step in EUCS to provide transparency to the certification process.

### 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.). 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.

#### 3.7.1 Related Architectural Components

This workflow involves the components shown in Figure 19, namely:

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

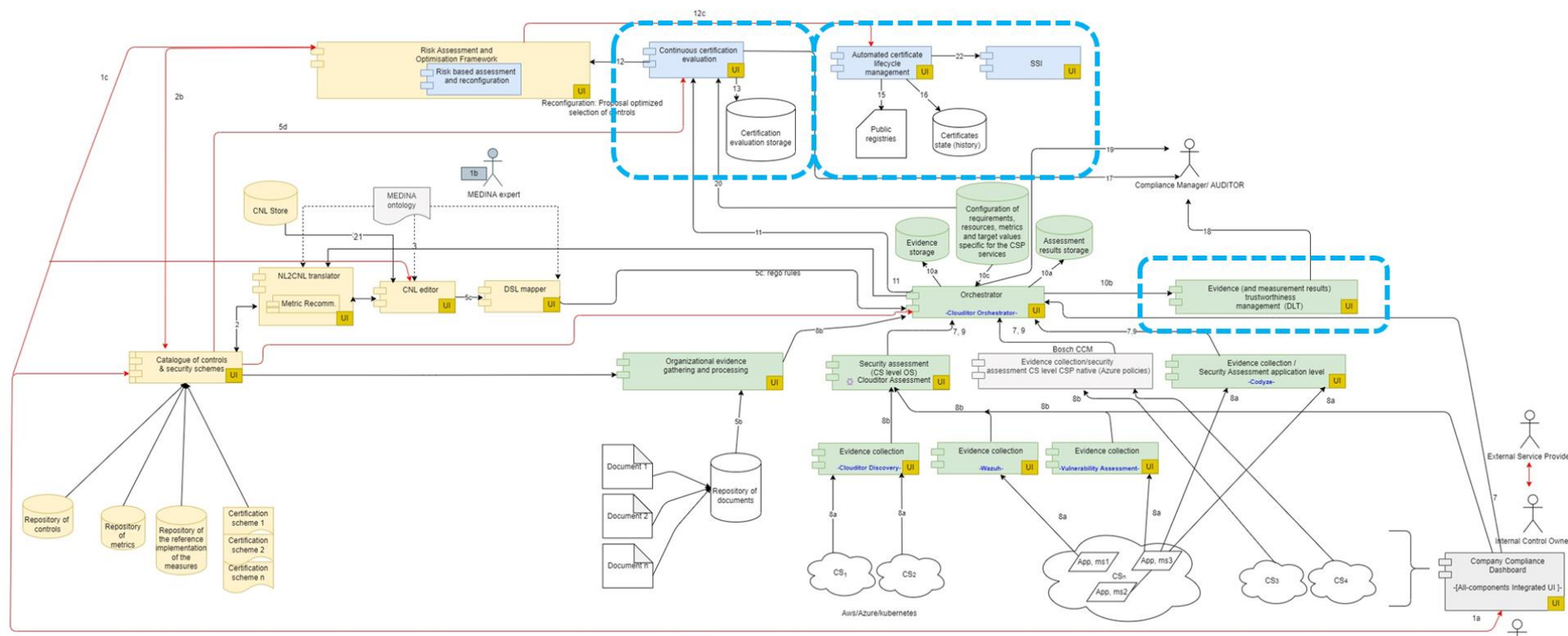


Figure 19. WF7 - EUCS – Report on ToC Certificate

### 3.7.2 Workflow

The different interactions corresponding to this WF7 are shown in Table 9.

Table 9. WF7 description

Step	Description	Role	Comments
<b>1</b>	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 EUCS certificate is shown.	CAB  CSP  NCCA	Details to display include certificate's history, ToC, degree of non-compliance, etc.
<b>2</b>	(Optional) Evidence trustworthiness management (DLT):  a. For a selected EUCS certificate, the gathered evidence are 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.
<b>3</b>	Continuous Certification Evaluation:  a. For the selected EUCS certificate/ToC, the degree of non-compliance is reported.	CAB  CSP  NCCA	The degree of non-compliance is further discussed in D2.6 [14]

## 4 MEDINA Framework Components and Integration

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

Figure 20 represents an evolution of the architecture shown in D5.1 [2]. It depicts the status of the architecture in M15. Other changes can be made during the course of the project, and they will be reported in subsequent versions.

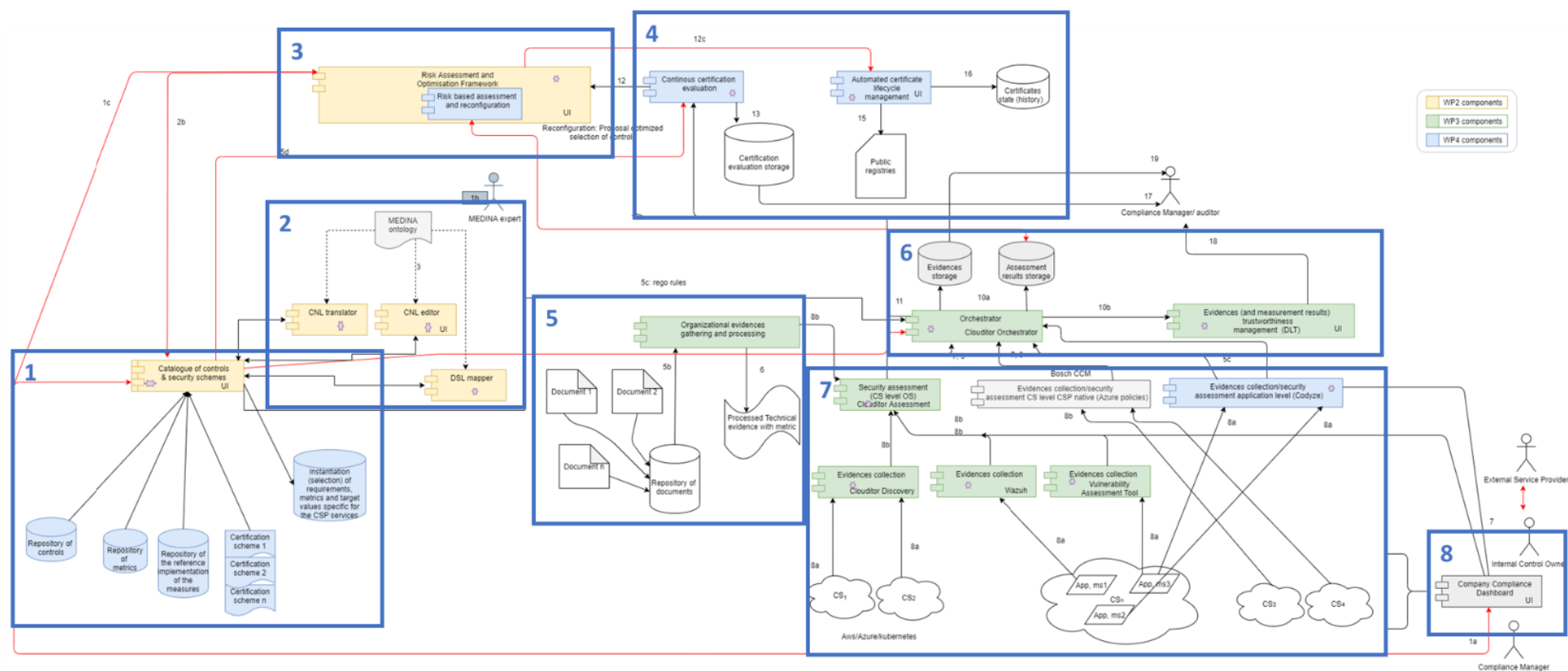


Figure 20. MEDINA Architecture and data flow

## 4.1 Catalogue (block #1)

### 4.1.1 Catalogue of Controls & Security Schemes

The component provides the following functionalities [15]:

- Endorsement of Security Control Frameworks and related attributes: Security requirements, categories, controls, reference TOMs, metrics, evidence types and assurance levels.
- Provision of guidance for the (self-)assessment of the requirements.
- Filtering of the information based on some values for the attributes:
  - Selection of requirements of a certain assurance level
  - Selection of requirements from a certain framework
  - Selection of metrics related to reference TOM
- Homogenization of the certification schemes: Provision of information about related requirements from different frameworks especially referenced to the EUCS.

The catalogue is composed by the following components:

- **Registry:** The Catalogue registry will store the available list of Frameworks and the related info for a specific CSP. This component will also include the corresponding database. A MySQL database provides the data persistence.
- **Back-end:** The Catalogue backend is the core sub-component of the Catalogue. It will perform the actual discovery of the requirements, evidence, etc. from the Catalogue registry, considering the set of filters established by the user through the UI/ API.
- **Frontend:** This sub-component is the graphical user interface of the Catalogue. This frontend will allow the user to indicate his requirements to filter and select a set of information related to the existing frameworks, i.e., requirements of a certain assurance level, requirements from a certain framework, metrics related to reference TOM, references TOMs, guidance, etc.

#### 4.1.1.1 Implementation and Integration Status

A first version of the Catalogue has been implemented at M15. The code is uploaded to the corresponding repository<sup>6</sup> in the project GitLab, and a docker-compose file for deployment is provided. A preliminary deployment was done in order to test the software and the deployment process.

The Catalogue provides a GUI for end users, as well as a RESTful API to interact with it.

#### 4.1.1.2 Published APIs

The Catalogue has implemented all the functionality to access and modify the database elements as a REST API, so the number of interfaces and endpoints is quite large. The Appendix A contains a graphical presentation of the available APIs, that can be used by the components interacting with the Catalogue.

The complete API is also available at the repository<sup>7</sup> as a json file (OpenAPI definition).

<sup>6</sup> <https://git.code.tecnalia.com/medina/public/catalogue-of-controls>

<sup>7</sup> <https://git.code.tecnalia.com/medina/public/catalogue-of-controls/-/blob/main/openapi.json>



#### 4.1.1.3 *Graphical interface*

The Catalogue offers a GUI to access and manipulate the different entities that compose the database. The typical screens for a CRUD (Create/Retrieve/Update/Delete) interface have been developed.

As a sample of the interface, some screenshots are presented here. Figure 21 and Figure 22 show the list of Security Controls and TOMs, while Figure 23 shows the detail of a specific Security Control (this last window is very similar to the one used to edit/create the entity, just changing the fields from “read” mode to “edit” mode).

**Medina SFC v0.0.1-SNAPSHOT**

Home Search requirements Entities Language Account

## Security Controls

ID	Code	Name	Objective	Description	Guidance	Risk Reduction Weight	Security Control	Actions
1	OIS-01	OIS-01	INFORMATION SECURITY MANAGEMENT SYSTEM	The CSP operates an information security management system (ISMS). The scope of the ISMS covers the CSP's organisational units, locations and processes for providing the cloud service.	PENDING	0	Security Control Framework	View Edit Delete
2	OIS-02	OIS-02	SEGREGATION OF DUTIES	Conflicting tasks and responsibilities are separated based on an RM-01 risk assessment to reduce the risk of unauthorised or unintended changes or misuse of cloud customer data processed, stored or transmitted in the cloud service.	PENDING	0	Security Control Category	View Edit Delete
3	OIS-03	OIS-03	CONTACT WITH AUTHORITIES AND INTEREST GROUPS	The CSP stays informed about current threats and vulnerabilities by maintaining the cooperation and coordination of security-related aspects with relevant authorities and special interest groups. The information flows into the procedures for handling risks (cf. RM-01) and vulnerabilities (cf. OPS-17).	PENDING	0	Security Control	View Edit Delete
4	OIS-04	OIS-04	INFORMATION SECURITY IN PROJECT MANAGEMENT	Information security is considered in project management, regardless of the nature of the project.	PENDING	0	Similar Control	View Edit Delete
5	ISP-01	ISP-01	GLOBAL INFORMATION SECURITY POLICY	The top management of the Cloud Service Provider has adopted an information security policy and communicated it to internal and external employees as well as cloud customers.	PENDING	0	Tom	View Edit Delete
6	ISP-02	ISP-02	SECURITY POLICIES AND PROCEDURES	Policies and procedures are derived from the information security policy, documented according to a uniform structure, communicated and made available to all internal and external employees of the Cloud Service Provider in an appropriate manner.	PENDING	0	Reference Tom	View Edit Delete
7	ISP-03	ISP-03	EXCEPTIONS	Exceptions to the policies and procedures for information security as well as respective controls are explicitly listed.	PENDING	0	Security Metric	View Edit Delete

ORGANISATION OF INFORMATION SECURITY

Figure 21. Window of list of Security Controls

Medina SFC v0.0.1-SNAPSHOT

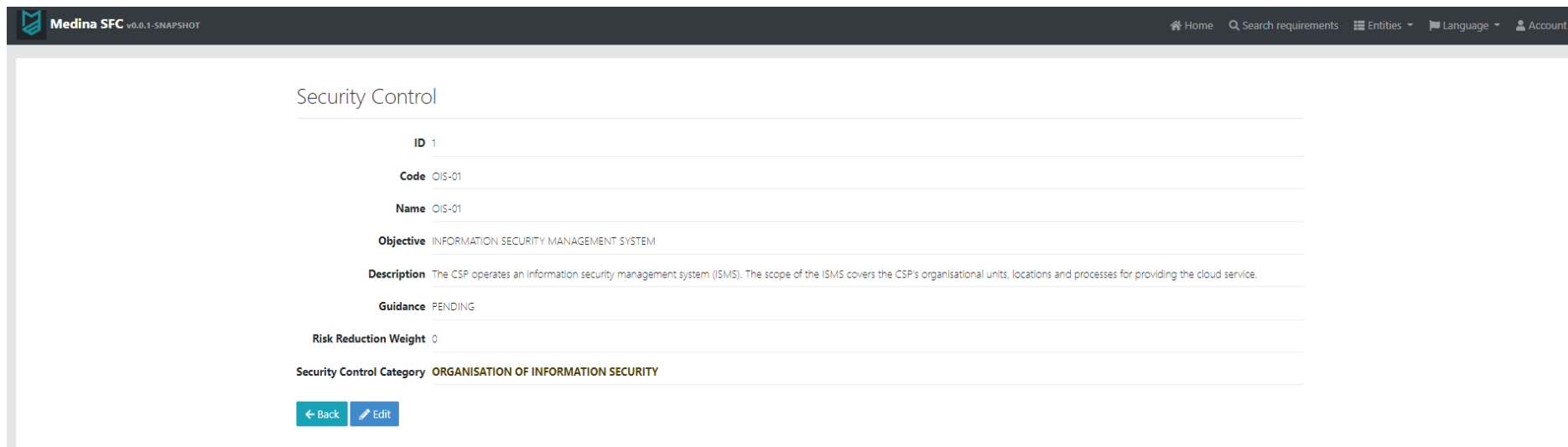
HomeSearch requirementsEntitiesLanguageAccount

Toms

Refresh listCreate a new Tom

ID	Code	Name	Description	Assurance Level	Type	Security Control	
1	OIS-01.1	OIS-01.1	The CSP shall define, implement, maintain and continually improve an information security management system (ISMS), covering at least the operational units, locations and processes for providing the cloud service	Basic	Organizational	OIS-01	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
2	OIS-01.2	OIS-01.2	The ISMS shall be in accordance to ISO/IEC 27001	Substantial	Organizational	OIS-01	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
3	OIS-01.3	OIS-01.3	The ISMS shall have a valid certification according to ISO/IEC 27001 or to national schemes based on ISO 27001	High	Organizational	OIS-01	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
4	OIS-01.4	OIS-01.4	The CSP shall document the measures for documenting, implementing, maintaining and continuously improving the ISMS	Basic	Organizational	OIS-01	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
5	OIS-01.5	OIS-01.5	The documentation shall include at least: " Scope of the ISMS (Section 4.3 of ISO/IEC 27001); " Declaration of applicability (Section 6.1.3), and " Results of the last management review (Section 9.3).	Substantial	Organizational	OIS-01	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
6	OIS-02.1	OIS-02.1	The CSP shall perform a risk assessment as defined in RM-01 about the accumulation of responsibilities or tasks on roles or individuals, regarding the provision of the cloud service	Basic	Organizational	OIS-02	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
7	OIS-02.2	OIS-02.2	The risk assessment shall cover at least the following areas, insofar as these are applicable to the provision of the cloud service and are in the area of responsibility of the CSP: "Administration of rights profiles, approval and assignment of access and access authorisations (cf. IAM-01); "Development, testing and release of changes (cf. DEV-01, CCM-01); and "Operation of the system components.	Basic	Organizational	OIS-02	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
8	OIS-02.3	OIS-02.3	The CSP shall implement the mitigating measures defined in the risk assessment, privileging separation of duties, unless impossible for organisational or technical reasons, in which case the measures shall include the monitoring of activities in order to detect unauthorised or unintended changes as well as misuse and the subsequent appropriate actions	Basic	Organizational	OIS-02	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
9	OIS-02.4	OIS-02.4	The CSP shall automatically monitor the assignment of responsibilities and tasks to ensure that measures related to segregation of duties are enforced.	High	Organizational	OIS-02	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
10	OIS-03.1	OIS-03.1	The CSP shall stay informed about current threats and vulnerabilities	Basic	Organizational	OIS-03	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>
11	OIS-03.2	OIS-03.2	The CSP shall maintain contacts with the competent authorities in terms of information security and relevant technical groups to stay informed about current threats and vulnerabilities	Substantial	Organizational	OIS-03	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a>

Figure 22. List of TOMs



The screenshot shows the 'Security Control' details page in the Medina SFC v0.0.1-SNAPSHOT application. The page has a dark header with the application name and version on the left, and navigation links (Home, Search requirements, Entities, Language, Account) on the right. The main content area is titled 'Security Control' and displays the following details for a control with ID 1:

- ID**: 1
- Code**: OIS-01
- Name**: OIS-01
- Objective**: INFORMATION SECURITY MANAGEMENT SYSTEM
- Description**: The CSP operates an information security management system (ISMS). The scope of the ISMS covers the CSP's organisational units, locations and processes for providing the cloud service.
- Guidance**: PENDING
- Risk Reduction Weight**: 0
- Security Control Category**: ORGANISATION OF INFORMATION SECURITY

At the bottom of the form, there are two buttons: 'Back' (with a left arrow icon) and 'Edit' (with a pencil icon).

Figure 23. Details page of a Security Control

## 4.2 NLP Technique (block #2)

### 4.2.1 CNL Translator

The NL2CNL Translator [16] is the MEDINA component used to map EUCS NL (Natural Language) requirements into their MEDINA CNL translation and consists of two modules: a recommender system and a translator. It interacts with the MEDINA User Interface, with the Catalogue of Controls and Security Schemes, with the CNL Store, and with the CNL Editor. The first module, the recommender system, takes as input a requirement and returns as output one or more metrics associated with it. The second module, the translator, takes as input the result obtained by the recommendation and translates the metrics in the MEDINA CNL.

#### 4.2.1.1 Implementation and Integration Status

At M15 a preliminary version of the CNL Translator has been implemented and it is to be deployed to the MEDINA Kubernetes cluster during the integration workshop<sup>8</sup>. The CNL Translator provides a set of RESTful APIs to interact with.

##### 4.2.1.1 Published APIs

The Appendix A - Section: CNL Translator and DSL Mapper depicts the available APIs that can be used by the components that interact with the NL2CNL Translator.

### 4.2.2 CNL Editor

The CNL Editor is the MEDINA component used to associate requirements with metrics, to manually refine output of CNL Translator, to define Obligations and to change TargetValues for Metrics.

The CNL Editor has a Web GUI Interface usable by a User e.g. a CSP who wants to define Requirement and Obligations (next abbreviated in Req&Obl or R&O in diagrams) association instances. For each Requirement instance exists an XML (Extensible Mark-up Language) file that is initially created, with only Requirement metadata, by Metric Recommender and CNL Editor acts on this xml file adding metrics and defining or refining obligations on these. CNL Editor can choose Metrics from the MEDINA Catalogue.

The structure of Req&Obl xml file is fundamentally divided in two parts:

- Requirement metadata: a unique identifier, a title, the security framework, Category, Assurance Level, Metrics List.
- Obligations (on Metrics): Obligations that express rules that must be checked encoded in a Controlled Natural Language (CNL), based on a predefined vocabulary.

When a Req&Obl is completed and is ready to be applied/used, the user with Editor role can invoke CNL Mapper to translate Obligations into Rego Rules.

The Req&Obl Objects are stored in a database, named CNL Store, that is used internally from the Editor calling CNL Store specific APIs.

CNL Editor is composed by different modules:

- CNL Editor Interface: the web GUI for access CNL Editor.
- Vocabulary: a file in RDF format with .owl extension where are defined Ontology structures and terms needed for Editor control of user changes on Obligations.

---

<sup>8</sup> The MEDINA integration workshop took place during January 18<sup>th</sup> – January 20<sup>th</sup>, 2022

- CNL Editor REST API: APIs used by Editor and eventually from other Certification Languages tools like CNL Translator and DSL Mapper for basic operations (see 4.2.2.2).
- CNL Store: database with Req&Obl xml files.
- Back Store Interface: REST APIs for access to CNL Store used by CNL Editor.

Figure 24 shows a picture of the internal CNL Editor Architecture and the modules listed above.

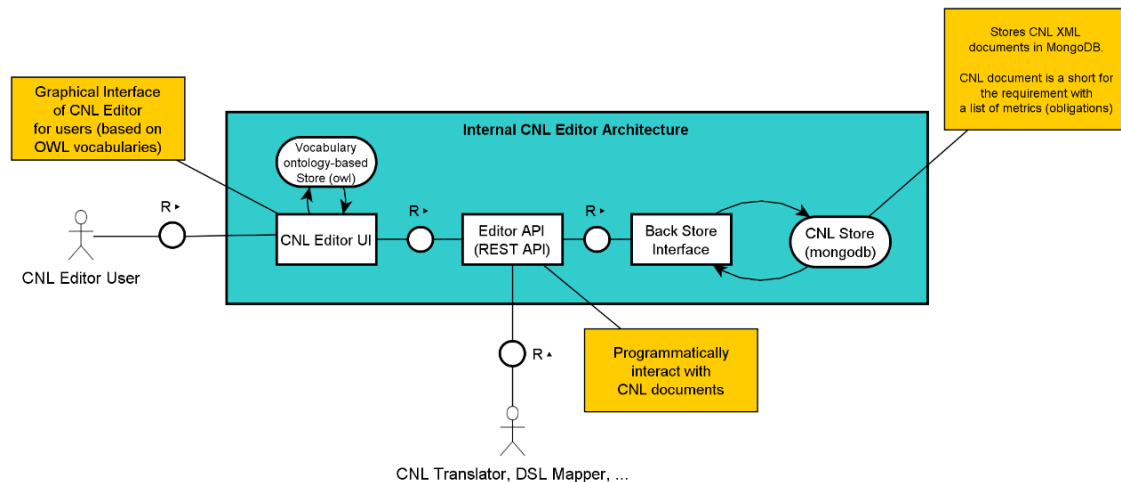


Figure 24. CNL Editor architecture (adapted from [16])

Vocabulary and CNL Editor are customised to allow the writing of obligations in the format agreed with the other Certification Languages tools:

*A ResourceType MUST MetricName TargetValueType(Operator, TargetValue)*

and just as an example:

Storage MUST EncryptionAtRestEnabled Boolean(=, true)

#### 4.2.2.1 Implementation and Integration Status

At M15 the CNL Editor is implemented in a first preliminary version and has been partially deployed to the MEDINA Kubernetes cluster. It provides both a GUI for end users and a set of RESTful APIs to interact with it. It has a basic vocabulary used to start defining R&Os and to understand how to better integrate CNL Editor into the MEDINA framework.

CNL Editor is hosted on [cnl.medina.esilab.org](https://cnl.medina.esilab.org) and can be invoked with the link:

[https://cnl.medina.esilab.org/DSAEditor/](https://cnl.medina.esilab.org/DSAEEditor/) [internal use only - authentication required]

At the moment it can be accessed by local users stored in an internal database MySQL, but the idea is to integrate it with the MEDINA access system.

Req&Obl xml structure is in a finalising stage and could be extended for MEDINA needs.

#### 4.2.2.2 Published APIs

In the Appendix A there are the APIs published by the CNL Editor.

### 4.2.3 DSL Mapper

The DSL Mapper is the MEDINA component that maps the rigorous, yet not executable MEDINA CNL into the MEDINA Domain Specific Language (DSL), whose statements are instead machine-readable. It interacts with the CNL Editor, with the CNL Store, and with the Orchestrator.

#### 4.2.3.1 Implementation and Integration Status

At M15 a preliminary version of the DSL Mapper has been implemented and is to be deployed to the MEDINA Kubernetes cluster during the integration workshop. At this stage of the project, the CNL Mapper has been implemented as a module of the NL2CNL Translator, since they share common resources and libraries. Thus, the APIs provided by this component have been included with the ones provided by the NL2CNL Translator.

#### 4.2.3.1 Published APIs

The APPENDIX A - Section: CNL Translator and DSL Mapper depicts the available APIs that can be used by the components that interact with the DSL Mapper.

## 4.3 Risk Assessment and Optimisation Framework (block #3)

### 4.3.1 Risk Assessment and Optimisation Framework (RAOF) (block #3)

The Risk Assessment and Optimisation Framework (RAOF) provides a risk-based analysis for the evaluation of non-conformities and will support the CSP in optimising of non-compliance reduction effort. The Framework is realised with a tool/service called Self-Assessment Tool for Risk Analysis (SATRA).

#### 4.3.1.1 Implementation and Integration Status

The first version of RAOF has been implemented and deployed. This version has the main focus on implementing the model defined in D5.1 [2], to make the tool to provide the core functionality and be able to integrate with other components. The tool provides GUI and API ways to operate with RAOF.

The current version is fixed for EUCS-based risk assessment only. The flexibility to select a certification scheme (which requires implementation of the functionality to connect with *Catalogue of Controls & Certification Schemes*) is planned for the future. The integration with *Continuous Certification Evaluation* and *Automatic Certificate Life-Cycle Manager* is in a rudimentary state and only provides a simplistic functionality to ensure that the defined workflow is executed. Implementation of this functionality is planned for M18, the due date of D4.4 [17], dedicated to the dynamic risk assessment. The optimisation support (i.e., help in risk-based selection of requirements to cover) is planned to be implemented during the next phase of the project.

#### 4.3.1.2 Published APIs

Appendix A displays the currently available APIs for other MEDINA Components.

#### 4.3.1.3 Graphical interface

The RAOF provides a GUI for a user to interact with the tool directly during preparation for certification. A screenshot of the tool is shown in D2.6 [14] .

## 4.4 Continuous Evaluation and Life Cycle Manager (block #4)

### 4.4.1 Continuous Certification Evaluation

The Continuous Certification Evaluation component (CCE) collects assessment results gathered by Security Assessment components through the Orchestrator (block #7) and continuously builds an evaluation tree representing the aggregation of assessment results to determine compliance with the different certification elements.

Beside the assessment results that CCE receives from the Orchestrator, other required inputs come from the Catalogue of Controls & Security Schemes (block #1). Data gathered from the Catalogue include a database of resources present in the observed CSP's infrastructure and the structure of the evaluation scheme itself (metrics, requirements, controls, control groups).

Output of the CCE is consumed by the Risk Assessment and Optimisation Framework (RAOF): CCE sends RAOF a list of all negatively evaluated resource-requirement pairs to determine whether they represent major or minor unconformities.

#### 4.4.1.1 Implementation and Integration Status

In the current implementation state, the CCE completely implements the basic evaluation aggregation (the methodology is described in D4.1 [18]). Because the integration with the MEDINA Catalogue to obtain the real certification schema is not yet established, CCE initializes its structure with a hard-coded sample tree. The tree is updated according to the assessment results received through the gRPC interface.

The CCE exposes an API that serves the evaluated certification values in all parts of the evaluation tree. The API is ready to be used by the front-end components and the RAOF.

Currently, the evaluation tree and its values are stored in-memory.

CCE source code is available on the project's GitHub repository<sup>9</sup>. A Dockerfile is available for simple deployment.

#### 4.4.1.2 Published APIs

Appendix A displays the currently REST APIs published by Continuous Certification Evaluation.

Data is returned in the format of JSON objects.

The interface for receiving assessment result is implemented using gRPC. Definition of the RPC service can be found in the corresponding Protocol Buffer source file<sup>10</sup>.

### 4.4.2 Life Cycle Manager

The Life-Cycle Manager component implements a state machine that tracks a certificate's state. The states it defines are based on how the EUCS defines them, e.g., *new*, *continued*, or *suspended* (see WF6 for additional information). As such, it takes inputs from the risk assessment and optimization framework to process them and translate them into the appropriate certificate state – possible after manual review by an auditor.

<sup>9</sup> <https://git.code.tecnalia.com/medina/public/continuous-certification-evaluation/>

<sup>10</sup> <https://git.code.tecnalia.com/medina/public/continuous-certification-evaluation/-/tree/main/src/main/proto>



#### 4.4.2.1 *Implementation and Integration Status*

The Life-Cycle Manager currently implements a state machine, a certificate model based on the ISO/IEC 17021 standard ("Requirements for bodies providing audit and certification of management systems"), and several APIs to interact with it. It also includes test cases, as well as build and deployment configurations for Docker and Kubernetes. It is therefore in a state where it can be deployed, and its communication with the RAOF is currently being tested. Note that it does not further forward data to other components.

#### 4.4.2.2 *Published APIs*

Appendix A displays the currently REST APIs published by the Life-Cycle Manager.

### 4.5 Organizational Evidence Gathering and Processing (block #5)

### 4.6 Orchestrator and Databases (block #6)

#### 4.6.1 Orchestrator and Databases

The Orchestrator [13] [19] is the central management component in the MEDINA framework. For example, it receives assessment results from the Security Assessment, and forwards them to the Trustworthiness System and to the Continuous Evaluation Component.

##### 4.6.1.1 *Implementation and Integration Status*

The Orchestrator mainly consists of APIs; their implementation status is described below. The Orchestrator does furthermore include the necessary configuration files for building Docker images and deploying them in a Kubernetes cluster.

By default, the Orchestrator stores all data to be stored in an ephemeral in-memory storage. A PostgreSQL database, however, can be used as well, which only requires to configure the respective connection parameters.

##### 4.6.1.2 *Published APIs*

The Orchestrator implements many APIs, since it is connected to several components. Please see the Appendix A for an overview.

#### 4.6.2 Trustworthiness System

The trustworthiness system [13] provides a common service of trustworthy records to be able to carry out automated inspections if needed guaranteeing information integrity.

It is composed of five elements:

- **Blockchain network** where the information will be saved guaranteeing its integrity due to the security features of the Blockchain technology.
- **Smart Contracts** with the trustworthiness management system functionalities: registration of data in the Blockchain (evidence and assessment results) to be verified, as well as the use of this previously registered data for integrity verification.
- **Blockchain client**, required to interact with the Blockchain and the Smart Contracts deployed on the Blockchain.
- **Blockchain viewer**, for receiving, normalizing, and categorizing the information recorded on the Blockchain to be consumed by users in a user-friendly way being Blockchain technology transparent for them.
- **Blockchain viewer client**, which provides a graphical interface to consume the information gathered by the Blockchain viewer.

#### 4.6.2.1 *Implementation and Integration Status*

A first version of the Trustworthiness system has been implemented and deployed for validation purposes.

- The Blockchain network, the Smart Contracts, the Blockchain viewer and the Blockchain viewer client are provided as a service from TECNALIA premises.
- An instance of the Blockchain client is provided as a service from TECNALIA premises for validation purposes. A docker image has been generated to be deployed at the orchestrator premises, which is the only MEDINA component interacting with the trustworthiness system.

#### 4.6.2.2 *Published APIs*

Appendix A describes the available API that can be used by the components interacting with the Blockchain client from the Trustworthiness System. The specific details are available at: <https://medina.bclab.dev/doc> [internal use only - authentication required].

#### 4.6.2.3 *Graphical interface*

The trustworthiness system includes a graphical interface accessible at: <https://medina.bclab.dev> [internal use only - authentication required]. The considered dashboards are shown in Figure 25 and Figure 26.

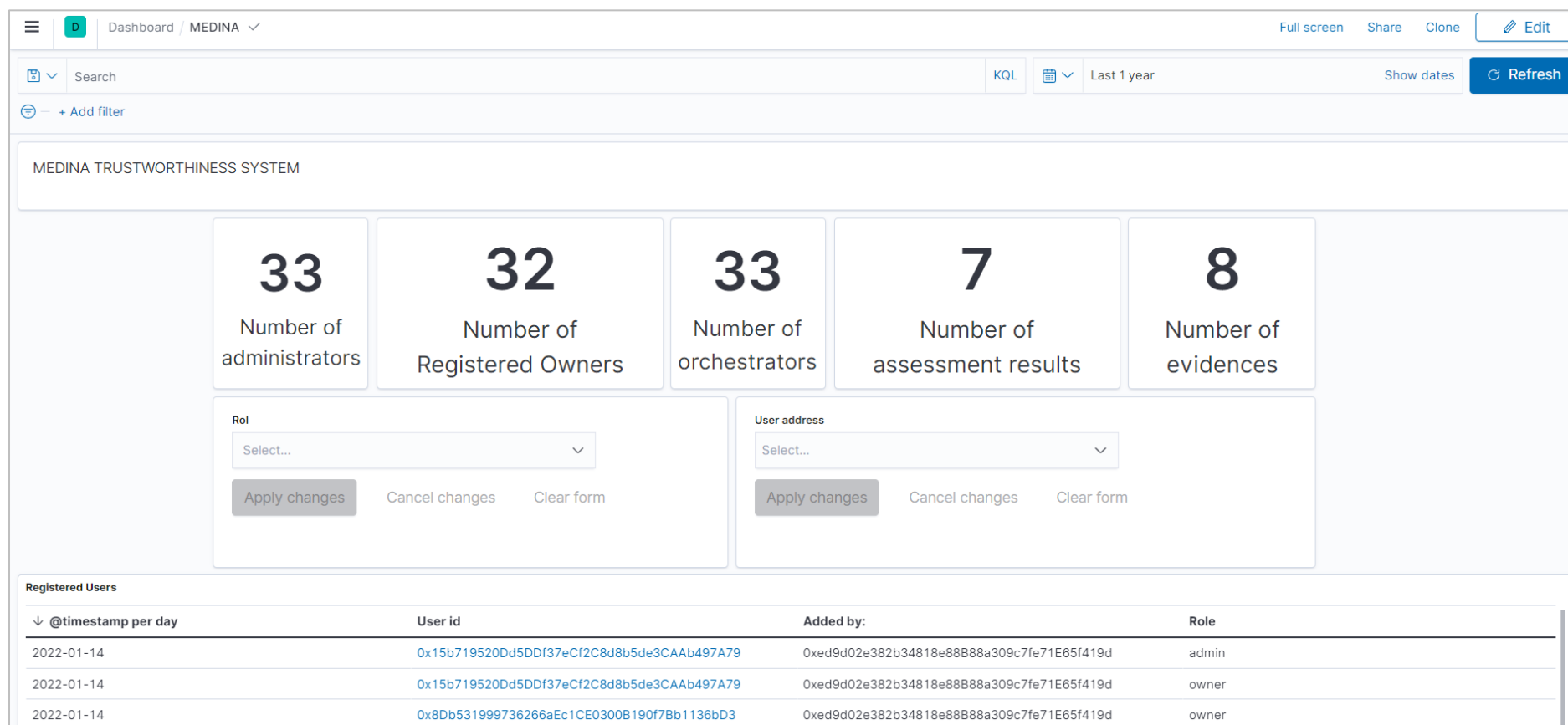


Figure 25. Trustworthiness System General Dashboard

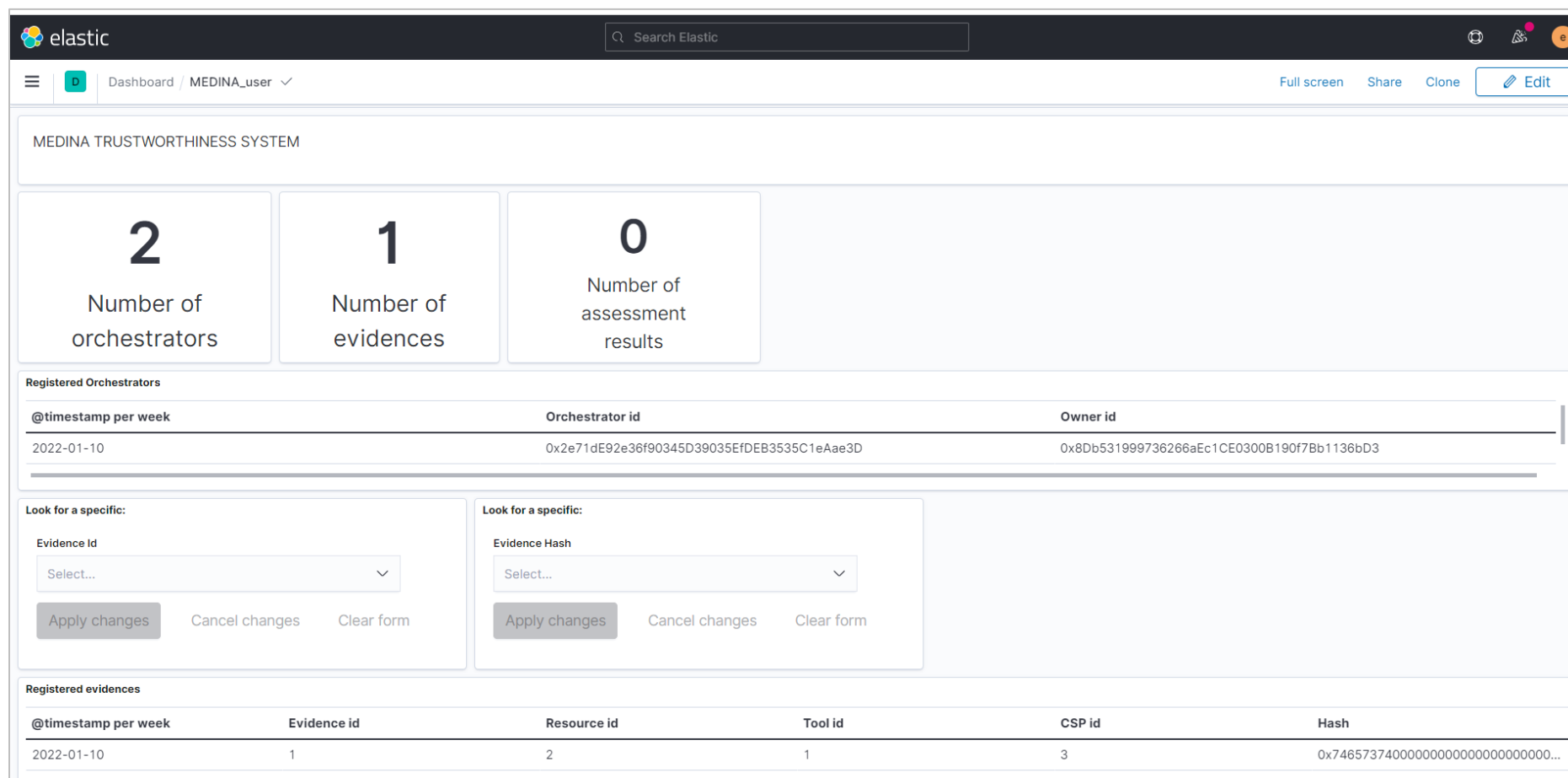


Figure 26. Trustworthiness System Specific Dashboard for each orchestrator

## 4.7 Evidence Collection and Security Assessment (block #7)

Evidence collection represents the starting point of the evidence flow in the MEDINA framework. Different components responsible for evidence collection discover cloud resources, measure their properties, and forward results (Evidence) to the Security Assessment component.

### 4.7.1 Evidence Collection

#### 4.7.1.1 Evidence Collection (Clouditor Discovery)

##### 4.7.1.1.1 Implementation and Integration Status

Currently, the Evidence Collector [13] [19] can discover resources in different cloud systems (Microsoft Azure, Amazon Web Services), and for different services in these systems, including computing, storage, and networking services.

##### 4.7.1.1.2 Published APIs

Appendix A displays the currently REST APIs published by the Evidence Collector.

#### 4.7.1.2 Evidence Collection from Wazuh and VAT

Wazuh [13] [19] is an open-source security monitoring tool for threat detection, integrity monitoring, incident response and basic compliance monitoring. In MEDINA, Wazuh is used to satisfy and verify security controls related to malware protection, logging, threat analytics and automatic monitoring. Vulnerability Assessment Tools (VAT) [13] [19] is a vulnerability scanning and detection framework that includes several vulnerability scanning tools. It can be configured to periodically scan the CSP's infrastructure and detect vulnerabilities and potential threats. VAT can be used to satisfy or gather evidence for controls related to vulnerability detection, use of encrypted communication, detection of new devices, etc.

Both Wazuh and Vulnerability Assessment Tools are integrated with other MEDINA components through a common component that collects evidence from both tools: Wazuh & VAT Evidence Collector, which forwards the gathered evidence to the Security Assessment component of Clouditor. The architecture of these components is described in further detail in D3.4 [19].

##### 4.7.1.2.1 Implementation and Integration Status

Deployment scripts are implemented to setup a demo environment including Wazuh, the Evidence Collector component, and a sample infrastructure to gather evidence from. The Evidence Collector currently implements gathering evidence from Wazuh for a limited number of metrics.

The basic integration with Clouditor's Security Assessment is implemented, sending evidence through the gRPC protocol. Other types of communication between the Wazuh & VAT Evidence collector and Clouditor include component registration and configuration instructions and are currently being developed and tested.

##### 4.7.1.2.2 Published APIs

Wazuh & VAT Evidence Collector is integrated with Clouditor through gRPC. No other APIs are exposed.

#### 4.7.1.1 Evidence Collection (Vulnerability Assessment Tool)

A further type of evidence collection is implemented in Codyze<sup>11</sup> [13] [19] which is a static code analyse that can verify security policies in source code. Thus, it complements the other evidence collection tools.

##### 4.7.1.1.1 Implementation and Integration Status

The current design foresees Codyze as a tool that both collects evidence and assesses them. It then forwards the assessment results directly to the Orchestrator. To that end, a wrapper for Codyze has been implemented which constructs assessment results according to the MEDINA data model and sends them to the Orchestrator.

##### 4.7.1.1.2 Published APIs

Codyze is integrated with the Orchestrator via the existing REST interfaces. No other APIs are exposed.

#### 4.7.2 Security Assessment (Cloudfitor)

The Security Assessment [13] [19] components assess the incoming evidence to create Assessment Results and send them to the Orchestrator. The main Security Assessment component provided by MEDINA is described here and implemented as part of Cloudfitor. If required by special evidence collection tools, other security assessment components (provided by CSPs) can be used as well.

##### 4.7.2.1 Implementation and Integration Status

The Security Assessment is implemented with an integrated policy engine, the Open Policy Agent (OPA)<sup>12</sup>, which assesses incoming evidence according to pre-defined metrics.

The Evidence Collector and the Security Assessment are integrated via gRPC calls and include the necessary configuration files for building Docker images and deploying them in a Kubernetes cluster.

##### 4.7.2.2 Published APIs

Appendix A displays the currently APIs published by the Security Assessment.

#### 4.7.3 SSI-based certificate management System

The SSI-based certificate management system [18] provides the necessary tools for digitalizing the conformity assessment results report based on the information gathered by MEDINA framework. It is an additional component to the general MEDINA framework.

In this sense, it is formed by the required Blockchain-based and the necessary services for the CAB, CSP and a potential CSP client:

The CAB needs the following services:

- A service for listening to the events from the MEDINA framework and know when to issue/update/revoke a certificate.
- A service for issuing, updating and/or revoking the public and private attestations (verifiable credentials) about the CSP based on the input from the MEDINA framework.

<sup>11</sup> <https://github.com/Fraunhofer-AISEC/codyze>

<sup>12</sup> <https://www.openpolicyagent.org/>

- A service for automatically saving the public attestations in a public registry.

The CSP needs the following services:

- A service for receiving public and private attestations (verifiable credentials) from the CAB and locally save them.
- A service for generating verifiable proofs to share with their clients based on the verifiable credentials from the CAB.

The CSP clients need the following services:

- A service for asking the CSP for specific proofs (both associated to private or public attestations).
- A service for verifying signatures from the verifiable proofs.

#### ***4.7.3.1 Implementation and Integration Status***

At the time of writing this deliverable, it is still under development.

#### ***4.7.3.2 Published APIs***

At the time of writing this deliverable, it is still under development.

## 5 MEDINA User Interface (block #8)

This section describes the MEDINA User Interface while the previous section (Chapter 4) is dedicated to components' integration interfaces only.

### 5.1 Implementation

#### 5.1.1 Functional description

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

##### 5.1.1.1 Fitting into overall MEDINA Architecture

GUIs in the MEDINA Framework are separated. Final users need a leading thread that makes it easier to navigate through content.

#### 5.1.2 Technical description

In order to facilitate independent team frontend development of functionalities, the architecture chosen for this implementation is “micro-frontends”. [20] This kind of architecture allows us to embed into a main frontend component (Integrated UI) any other UI in the framework independently of the underlying technology.

##### 5.1.2.1 Prototype architecture

The following diagram describes a simplified architecture from an Integrated UI perspective.

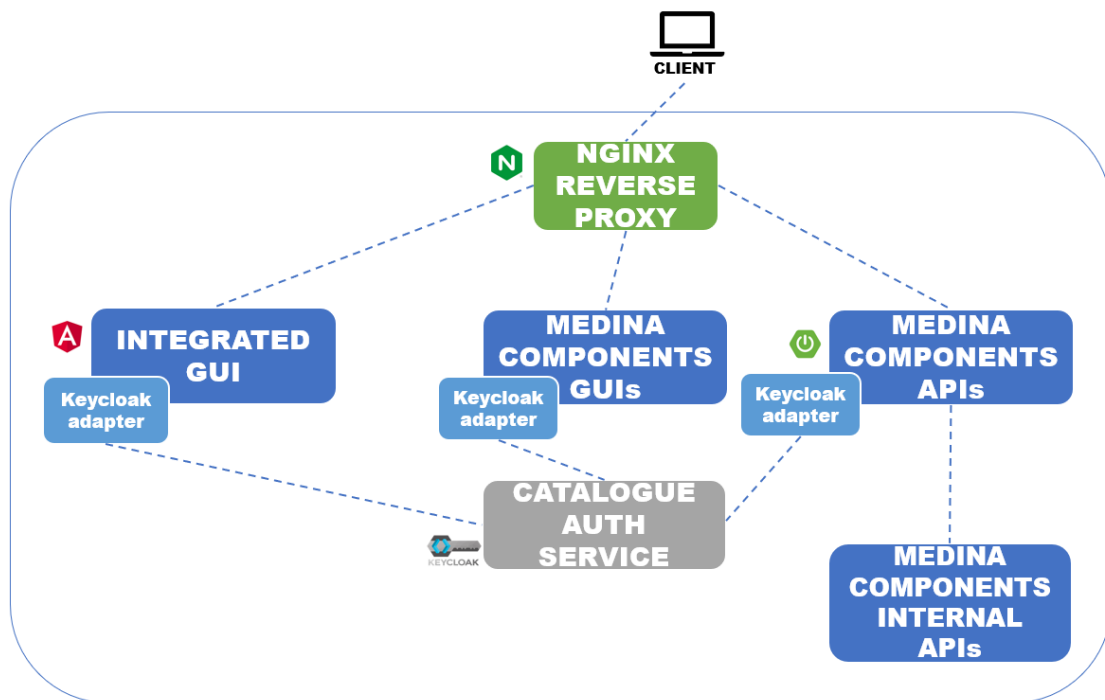


Figure 27. MEDINA UI Architecture



### 5.1.2.2 Description of Components

#### 5.1.2.2.1 Authentication and authorization

Authentication is managed by the MEDINA Catalogue's Keycloak<sup>13</sup>, 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 the 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.

#### 5.1.2.2.2 Integration of components

The following list comprises the components we integrate at the moment and the chosen strategy. It will be subject of further evolution in next iteration processes. Workflows involving reported components can be found in chapter 3 of this document. Current integration status has a technical enablement only purpose.

Table 10. Integration strategy for the different MEDINA components

Component name	Integration strategy
Metrics and Measures Catalogue Keycloak	REST API
Metrics and Measures Catalogue	REST API & Iframe
NL2CNL Translator	REST API
Orchestrator*	REST API
*This integration has been set up, but will be finalized in the next phase	

#### 5.1.2.3 Technical specifications

The prototype is developed using Angular 12, a modern typescript framework that allows us to build high-performance, scalable, component-based single page web applications. [21] The framework is enriched with Angular Material 2 library, a set of high quality animated responsive components that follow Material Design UI specifications. [22] [23] The application runs on a Nginx web server. [8]

Integration of micro-frontends is obtained through iframes and REST API. In particular, since REST APIs are following OpenAPI specifications, we are able to generate and update automatically referred services in the application, with great benefits in regards to productivity

<sup>13</sup> <https://www.keycloak.org/>

and bug-free implementation. [24] Ngx-charts is used to render animated graphical content (e.g. Histograms). [25]

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 current UIs interactions. It will be both expanded and refined in the next iteration processes.

##### 5.1.2.4.1 Login, authentication and iframe embedding

Unauthenticated users that try to access Integrated-UI are redirected to keycloak's login page.

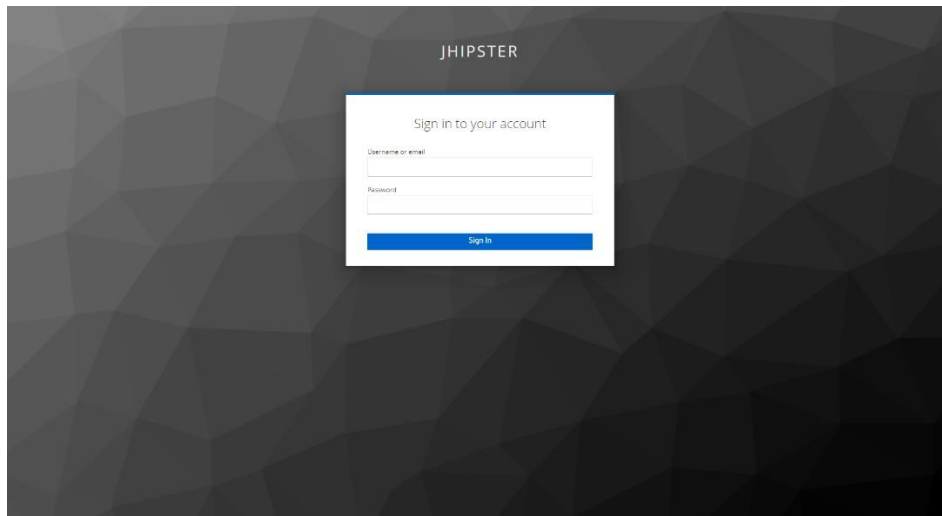


Figure 28. Keycloak Login Page

After inserting correct credentials, users are redirect to the page that the request was originated from.

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 Integrated-UI embeds the Catalogue dashboard. As intended, the authentication is received correctly by the embedded component, without the need to log-in again.

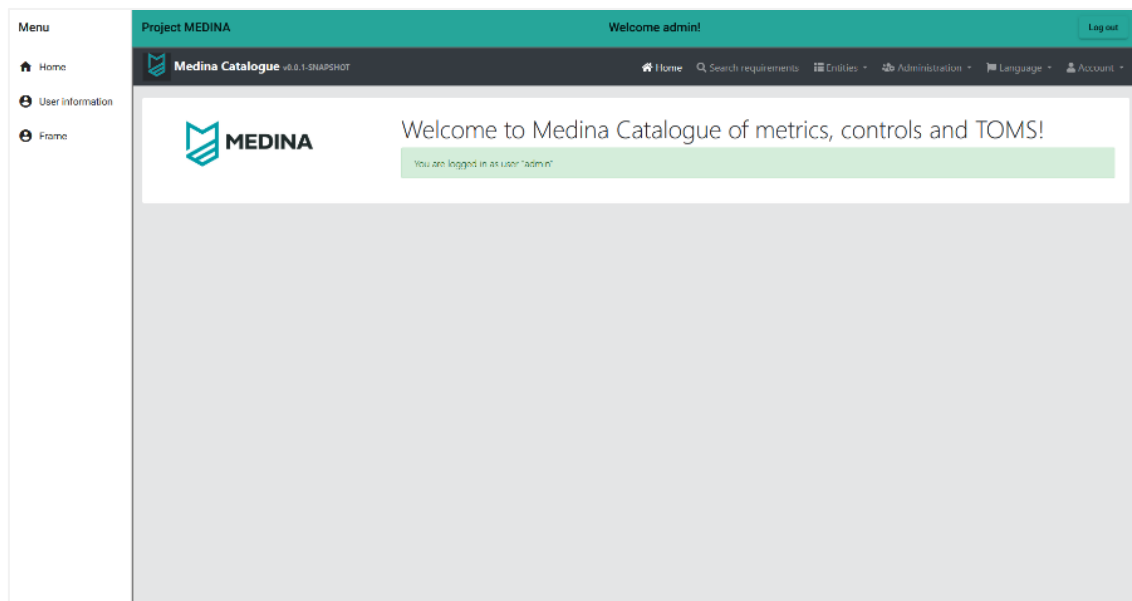


Figure 29. Full Screen Frame Embedding - Catalogue and Integrated UI

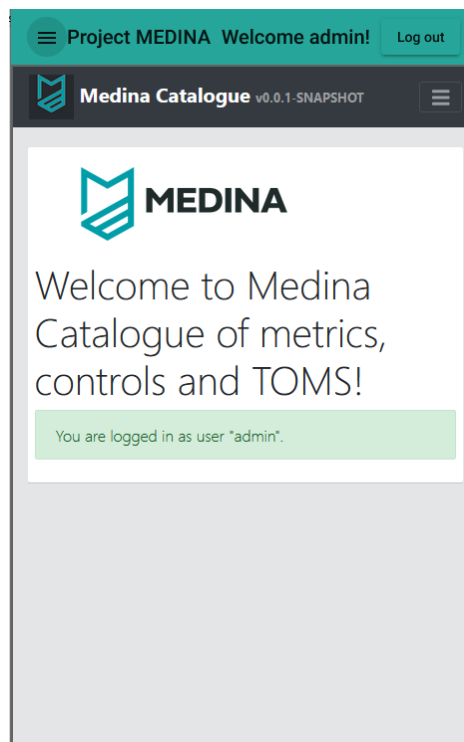


Figure 30. Responsive IFrame Embedding - Catalogue and Integrated UI

#### 5.1.2.4.2 Conditional formatting and guard

A Guard component is implemented which allows to define under what conditions the component can be rendered by the router. Basic set-up requires authentication only, but roles can be defined as parameters to implement authorization. Authorization can be implemented taking advantage of conditional formatting too, allowing us to hide/change certain features based on user role.

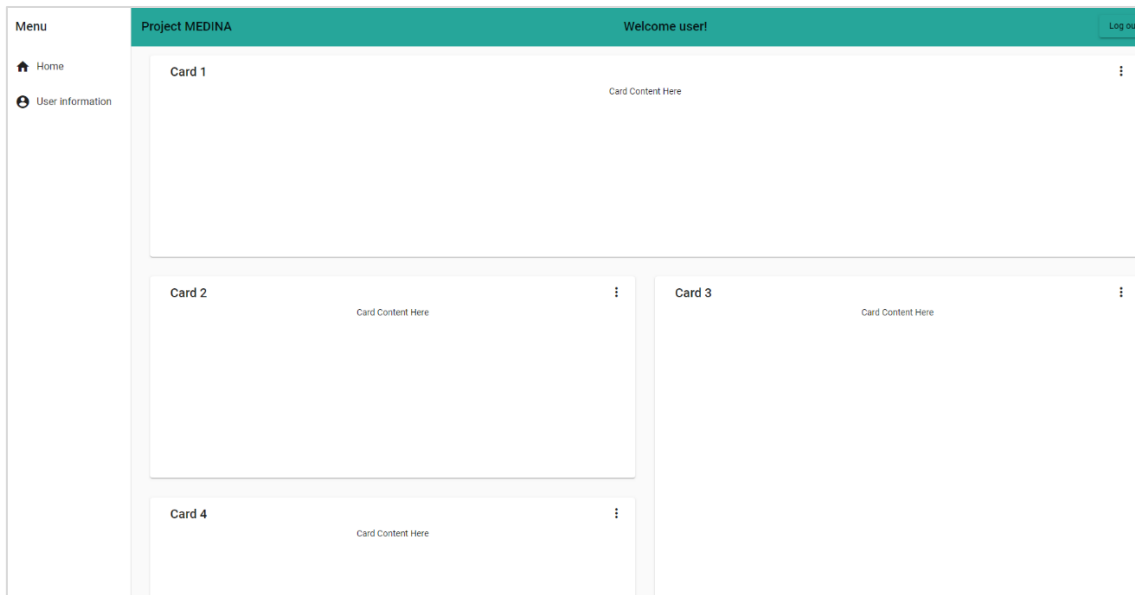


Figure 31. Example: Frame button is hidden in navigation bar for User without role ADMIN and routing is inhibited

#### 5.1.2.4.3 Services

For integration testing purposes, a set of buttons have been implemented in home component that simulate a REST GET call to the following endpoints:

Table 11. Integration tested endpoints

Component name	Endpoint
Metrics and Measures Catalogue	<a href="https://catalogue-dev.k8s.medina.esilab.org/services/cocbackend/api/security-controls/count">https://catalogue-dev.k8s.medina.esilab.org/services/cocbackend/api/security-controls/count</a> [internal use only - authentication required]
NL2CNL Translator	<a href="https://nl2cnl-translator-dev.k8s.medina.esilab.org/ids">https://nl2cnl-translator-dev.k8s.medina.esilab.org/ids</a> [internal use only - authentication required]

### 5.1.3 Delivery and usage

#### 5.1.3.1 Package information

The package has the following structure:

Table 12. 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
/kubernetes	Contains kubernetes configuration files for deployment
/node_modules	Contains installed npm modules

/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 prod)
/src/app/services	Contains services that are generated via OpenAPI specs in order to integrate with other applications in Medina Framework
/src/app/	Contains application main components

### 5.1.3.2 Download

[https://git.code.tecnalia.com/medina/wp5/task\\_5.3/integrated-ui](https://git.code.tecnalia.com/medina/wp5/task_5.3/integrated-ui) [internal use only - authentication required].

## 6 Conclusions

The document covers the objectives of the Task 5.3 at M15. On the one hand, the environment has been defined and setup to support CI/CD approach, that have to be further optimized. On the other, integration activities are initiated, and a webinar and a workshop are organized in order to do it appropriately according to the methodology defined in this deliverable.

The Test Bed environment has been realized on a three-node Kubernetes Cluster that orchestrates all components in the MEDINA Framework. All the components on the Test Bed environment are containerized and communicate each other with RESTful API over HTTPS secure protocol. A Kubernetes Dashboard, that is a web-based User Interface, is created to help to deploy containerized applications, and manage the cluster resources.

During the workshop the actual environment was illustrated in order to do the integration activities along with the partners: following the methodology all components have been deployed in the MEDINA cluster, meanwhile through the webinar are transmitted to the partners the main notions on Kubernetes and Docker.

Next activities will foresee the implementation of the pipelines that automatize all deployment process. A new version of the components in the MEDINA Framework will be provided and in addition, the components of the block five will be added. Also, creating and instantiating real-world scenarios based on the generic workflows are goals to be achieved in WP6. Lastly, a workshop on CI/CD is planned and others will be held if necessary.

A second version of this document will be delivered at M27, containing updated versions of the components in MEDINA Framework and the setup of the build, deploy and security pipelines.

## 7 References

- [1] “K8s,” [Online]. Available: <https://kubernetes.io/docs/home/>.
- [2] MEDINA Consortium, “D5.1 MEDINA Requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy,” 2021.
- [3] “RKE,” [Online]. Available: <https://rancher.com/docs/rke/latest/en/os/>.
- [4] “Rook/Ceph,” [Online]. Available: <https://rook.io/docs/rook/v1.8/>.
- [5] “METALLB,” [Online]. Available: <https://metallb.universe.tf/>.
- [6] “SSH,” [Online]. Available: <https://www.ssh.com/academy/ssh/protocol>.
- [7] “JFrog Artifactory,” [Online]. Available: <https://jfrog.com/artifactory/>.
- [8] “Nginx,” [Online]. Available: <https://www.nginx.com/>.
- [9] Linux Foundation, “Helm package manager,” [Online]. Available: <https://helm.sh/>.
- [10] Linux Foundation, “Cert manager,” [Online]. Available: <https://cert-manager.io/docs/>.
- [11] “Apache Maven Project,” [Online]. Available: <https://maven.apache.org/>.
- [12] “Codyze,” [Online]. Available: <https://www.codyze.io/?ref=https://githubhelp.com>.
- [13] MEDINA Consortium;, “D3.1-Tools and techniques for the management of trustworthy evidence-v1,” 2021.
- [14] MEDINA Consortium, “D2.6 - Risk-based techniques and tools for Cloud Security Certification-v1,” 2022.
- [15] MEDINA Consortium;, “D2.1 – Continuously certifiable technical and organizational measures and catalogue of cloud security metrics-v1,” 2021.
- [16] MEDINA Consortium, “D2.3 Specification of the Cloud Security Certification Language-v1,” 2021.
- [17] MEDINA Consortium, “D4.4 - Methodology and tools for risk-based assessment and security control reconfiguration-v1,” 2022.
- [18] MEDINA Consortium;, “D4.1 Tools and Techniques for the Management and Evaluation of Cloud Security Certifications,” 2021.
- [19] MEDINA Consortium;, “D3.4-Tools and techniques for collecting evidence of technical and organisational measures-v1,” 2021.
- [20] 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 01 11 2021].

- [21] Google, “Angular,” [Online]. Available: <https://angular.io/>.
- [22] Google, “Material Design,” [Online]. Available: <https://material.io/design>.
- [23] Google, “Angular Material,” [Online]. Available: <https://material.angular.io/>.
- [24] Linux Foundation, “OpenAPI,” [Online]. Available: <https://www.openapis.org/>.
- [25] Swimlane, “Ngx-charts,” [Online]. Available: <https://swimlane.github.io/ngx-charts>.
- [26] S. Madsen, “How to Prioritize with the MoSCoW Technique,” October 2017. [Online]. Available: <https://www.projectmanager.com/training/prioritize-moscow-technique> . [Accessed March 2018].



## APPENDIX A: Published APIs

### Section: Catalogue of Controls & Security Schemes

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

**cocBackend API** 0.0.1 OAS3  
/medina/vp2/task\_2.2/catalogue-deploy/-raw/main/openapi.json  
 cocBackend API documentation  
 unlicensed

Servers  
 /services/cocbackend/ - added by global filter

**cloud-service-provider-resource** Cloud Service Provider Resource
 

- GET** /api/cloud-service-providers getAllCloudServiceProviders
- POST** /api/cloud-service-providers createCloudServiceProvider
- GET** /api/cloud-service-providers/count countCloudServiceProviders
- GET** /api/cloud-service-providers/{id} getCloudServiceProvider
- PUT** /api/cloud-service-providers/{id} updateCloudServiceProvider
- DELETE** /api/cloud-service-providers/{id} deleteCloudServiceProvider
- PATCH** /api/cloud-service-providers/{id} partialUpdateCloudServiceProvider

**cloud-service-resource** Cloud Service Resource
 

- GET** /api/cloud-services getAllCloudServices
- POST** /api/cloud-services createCloudService
- GET** /api/cloud-services/count countCloudServices
- GET** /api/cloud-services/{id} getCloudService
- PUT** /api/cloud-services/{id} updateCloudService
- DELETE** /api/cloud-services/{id} deleteCloudService
- PATCH** /api/cloud-services/{id} partialUpdateCloudService

**public-user-resource** Public User Resource
 

- GET** /api/authorities getAuthorities
- GET** /api/users getAllPublicUsers

**reference-tom-resource** Reference Tom Resource ^

GET	/api/reference-toms	getAllReferenceToms	▼
POST	/api/reference-toms	createReferenceTom	▼
GET	/api/reference-toms/count	countReferenceToms	▼
GET	/api/reference-toms/{id}	getReferenceTom	▼
PUT	/api/reference-toms/{id}	updateReferenceTom	▼
DELETE	/api/reference-toms/{id}	deleteReferenceTom	▼
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	▼
POST	/api/security-control-categories	createSecurityControlCategory	▼
GET	/api/security-control-categories/count	countSecurityControlCategories	▼
GET	/api/security-control-categories/{id}	getSecurityControlCategory	▼
PUT	/api/security-control-categories/{id}	updateSecurityControlCategory	▼
DELETE	/api/security-control-categories/{id}	deleteSecurityControlCategory	▼
PATCH	/api/security-control-categories/{id}	partialUpdateSecurityControlCategory	▼

**security-control-framework-resource** Security Control Framework Resource ^

GET	/api/security-control-frameworks	getAllSecurityControlFrameworks	▼
POST	/api/security-control-frameworks	createSecurityControlFramework	▼
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	▼
DELETE	/api/security-control-frameworks/{id}	deleteSecurityControlFramework	▼
PATCH	/api/security-control-frameworks/{id}	partialUpdateSecurityControlFramework	▼

**security-control-resource** Security Control Resource ^

GET	/api/security-controls	getAllSecurityControls	▼
POST	/api/security-controls	createSecurityControl	▼
GET	/api/security-controls/count	countSecurityControls	▼
GET	/api/security-controls/{id}	getSecurityControl	▼
PUT	/api/security-controls/{id}	updateSecurityControl	▼
DELETE	/api/security-controls/{id}	deleteSecurityControl	▼
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	▼
POST	/api/similar-controls	createSimilarControl	▼
GET	/api/similar-controls/count	countSimilarControls	▼
GET	/api/similar-controls/{id}	getSimilarControl	▼
PUT	/api/similar-controls/{id}	updateSimilarControl	▼
DELETE	/api/similar-controls/{id}	deleteSimilarControl	▼
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	▼
POST	/api/toms	createTom	▼
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	▼
DELETE	/api/toms/{id}	deleteTom	▼
PATCH	/api/toms/{id}	partialUpdateTom	▼

**user-resource** User Resource ^

GET	/api/admin/users	getAllUsers	▼
GET	/api/admin/users/{login}	getUser	▼

## Section: CNL Translator and DSL Mapper

### NL2CNL Translator and DSL Mapper 0.0.1 OAS3

/openapi.json

NL2CNL Translator and DSL Mapper APIs for MEDINA Project. 🚀

#### Description

You can use this APIs to:

- Retrieve MEDINA metric recommendations for a Security Control Requirement.
- Translate Security Control Requirements and Metrics from Natural Language to a Controlled Natural Language (obligations).
- Map obligations into Rego Rules and push them to the MEDINA Orchestrator.

Contact Franz Berger

default		^
GET	/ids Get Ids	▼
GET	/recommend/{req_id} Recommend	▼
GET	/recommend_metrics/{req_id} Recommend Metrics	▼
GET	/cnl_string/{m_id} Get Cnl	▼
GET	/cnl/{m_id} Get Cnl	▼
GET	/dsl_data/{m_id} Dsl Data	▼
GET	/dsl_rego_rule/{m_id} Dsl Rego Rule	▼

## Section: CNL Editor

/copyreq&obltemp/{req&obltempid}/user/{userid} Create a Req&Obl from Req&Obl XML Template

/createreq&obl store a new Req&Obl in the CNL Store. The API returns a unique Req&Obl identifier

/deletereq&obl/{req&oblid} delete a Req&Obl, by its identifier

/fetchdatapolicy/{req&oblid} Retrieve Req&Obl DSL part (Obligations) from the DSL Mapper

/fetchusagepolicy/{req&oblid} Retrieve Req&Obl DSL part (Obligations) from the DSL Mapper

/getreq&obl/{req&oblid} retrieve a Req&Obl by its identifier

/getreq&obldetails/{userid} Fetch the details of the Req&Obl

/getuserreq&oblslst/{username} List Available Req&ObIs

/mapreq&obl/{req&oblid} Send Req&Obl to DSL Mapper

/updatereq&obl modify the content of an already existing Req&Obl, by its identifier

## Section: Risk Assessment and Optimisation Framework

**SATRA - Self-Assessment Tool for Risk Analysis** <sup>63</sup>

[ Base URL: /api/v1 ]  
[api/v1/swagger.json](#)

Manage interaction with the SATRA engine

**practice** Interact with the survey, update question/answers and get risk. ▼

POST	/practice/analysis/{UUID}	Send information on a test result
POST	/practice/answer/{UUID}/{question_id}/{answer_id}	Send (eventually, update) an answer for a specific question
GET	/practice/answer/{UUID}/{type_id}	Get all the possible answers by type_id
GET	/practice/answers/{UUID}	
GET	/practice/assets/{UUID}	
POST	/practice/assets_answers/{UUID}	
DELETE	/practice/assets_answers/{UUID}/{asset_name}	
GET	/practice/assurance/{UUID}	
GET	/practice/certification/{UUID}	
GET	/practice/cspmarket/{UUID}	
POST	/practice/dynamic_evaluated_risk/{UUID}	
GET	/practice/evaluated_risk/{UUID}	
POST	/practice/json/{UUID}	
GET	/practice/question/{UUID}	Get all the questions
GET	/practice/question/{UUID}/{question_id}	Get one question and its possible answers
GET	/practice/risk/{UUID}	Get the updated risk
GET	/practice/threats/{UUID}	Get the updated threat

## Section: Continuous Evaluation

- /tree, GET: Returns the complete structure and values of the currently-evaluated certification tree.
- /resourcenodes, GET: Returns an array of all resource-requirement pairs (fulfilment of a specific requirement by a specific resource) with their evaluated values. Additional parameter “conformant” can be used to return only positively or only negatively evaluated nodes.

## Section: Life Cycle Manager

- /new, POST: Creates a new certificate that is initialized with the state *new*
- /evaluation, POST: Processes a new evaluation result which can be positive or negative; the state of the respective certificate is then changed accordingly, e.g. to *continued* or *suspended*
- /update, POST: Processes a request to update the certificate’s information, e.g. the scope (without changing its compliance state)
- /remediation, POST: Processes a remediative evaluation result that changes a *suspended* certificate back to *continued*
- /statechange, GET: Provides information about the last state change of a certificate

## Section: Orchestrator

POST	/v1/assessment/evidences	Assesses the evidence sent by the discovery. Part of the public API, also exposed as REST.	✓ ↩
GET	/v1/assessment/results	List all assessment results. Part of the public API, also exposed as REST.	✓ ↩
POST	/v1/orchestrator/assessment_results	Stores the assessment result provided by an assessment tool	✓ ↩
GET	/v1/orchestrator/assessment_tools	Lists all assessment tools assessing evidences for the metric given by the passed metric id	✓ ↩
POST	/v1/orchestrator/assessment_tools	Registers the passed assessment tool	✓ ↩
GET	/v1/orchestrator/assessment_tools/{tool_id}	Returns assessment tool given by the passed tool id	✓ ↩
PUT	/v1/orchestrator/assessment_tools/{tool_id}	Updates the assessment tool given by the passed id	✓ ↩
DELETE	/v1/orchestrator/assessment_tools/{tool_id}	Remove assessment tool with passed id from the list of active assessment tools	✓ ↩
GET	/v1/orchestrator/cloud_services	Lists all target cloud services	✓ ↩
POST	/v1/orchestrator/cloud_services	Registers a new target cloud service	✓ ↩
GET	/v1/orchestrator/cloud_services/{service_id}	Retrieves a target cloud service	✓ ↩
PUT	/v1/orchestrator/cloud_services/{service_id}	Registers a new target cloud service	✓ ↩
DELETE	/v1/orchestrator/cloud_services/{service_id}	Removes a target cloud service	✓ ↩
GET	/v1/orchestrator/cloud_services/{service_id}/metric_configurations	Lists all a metric configurations for a specific service and metric ID	✓ ↩
GET	/v1/orchestrator/cloud_services/{service_id}/metric_configurations/{metric_id}	Retrieves a metric configuration for a specific service and metric ID	✓ ↩
GET	/v1/orchestrator/metrics	List all metrics provided by the metric catalog	✓ ↩
GET	/v1/orchestrator/metrics/{metric_id}	Returns the metric with the passed metric id	✓ ↩

## Section: Trustworthiness System

Swagger  
REST API (1.0.0)  
[ Base URL: localhost:3000/ ]

Schemes  
HTTP

default

- POST /client/account
- GET /client/account
- POST /client/wallet
- GET /client/wallet
- GET /client/privatekey
- 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
- GET /client/authorizedowners
- POST /client/orchestrator
- POST /client/orchestrator/evidence
- POST /client/orchestrator/assessment
- GET /client/orchestrator/evidence/{id}
- GET /client/orchestrator/assessment/{id}
- GET /client/orchestrator/evidences
- GET /client/orchestrator/assessments
- GET /client/orchestrator/owner
- GET /client/orchestrator/creationtime
- GET /client/orchestrator/id

## Section: Evidence Collection (Cloud Discovery)

- /v1/discovery/start, POST: Starts discovering the cloud resources
- /v1/discovery/query, POST: Lists all evidences collected in the last run

## Section: Security Assessment (Clouditor)

- /v1/assessment/evidences, POST: Assesses the evidence sent by the Evidence Collector
- /v1/assessment/results, GET: Lists all assessment results