Deliverable D3.5

Evidence assessment and Certification — Integration v1

Editor(s):	Nico Haas
Responsible Partner:	Fraunhofer AISEC
Status-Version:	Final – v1.0
Date:	31.01.2025
Type:	OTHER
Distribution level:	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	Evidence assessment and Certification -Integration-v1
Due Date of Delivery to the EC	31.01.2025

Workpackage responsible for the Deliverable:	WP3 - Evidence assessment and Certification
Editor(s):	Fraunhofer AISEC
Contributor(s):	Nico Haas (FHG) Cristina Regueiro, Iñaki Etxaniz (TECNALIA) Marinella Petrocchi (CNR)
Reviewer(s):	Michela Fazzolari (CNR) Cristina Martínez, Juncal Alonso (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP1, WP2, WP4, WP5, WP6

Abstract:	Interim integration of the WP3 components in the EMERALD system.
Keyword List:	Integration, WP3, Evidence Assessment, Assessment Evaluation, Certification, Trustworthiness, Orchestration, Controls and Metrics
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 4.0 International (**CC BY-SA 4.0 DEED** https://creativecommons.org/licenses/by-sa/4.0/)
Disclaimer	Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	12.12.2024	TOC	FHG
v0.2	20.01.2025	First draft	FHG, TECNALIA, CNR
v0.3	27.01.2025	Internal QA Review	CNR
v0.4	28.01.2025	Address internal QA Review	FHG
v0.5	29.01.2025	Final reviewed version	TECNALIA
v1.0	31.01.2025	Submitted to the European Commission	TECNALIA

Table of contents

List of tables

List of figures

Terms and abbreviations

AI	Artificial Intelligence
AMOE	Assessment and Management of Organisational Evidence
API	Application Programming Interface
CaaS	Certification-as-a-Service
CI/CD	Continuous Integration/Continuous Deployment
CPU	Central Processing Unit
DoA	Description of Action
EBSI	European Blockchain Services Infrastructure
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
GUI	Graphical User Interface
KB	Kilobyte
KR	Key Result
NLP	Natural Language Processing
gRPC	Google Remote Procedure Calls
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
KR	Key Result
MARI	Mapping Assistant for Regulations with Intelligence
OAuth	open authorization
OSCAL	Open Security Controls Assessment Language
RCM	Repository of Controls and Metrics
SDLC	Software Development Life Cycle
TOM	Technical and Organizational Measure
TWS	Trustworthiness System
WSGI	Web Server Gateway Interface
YAML	Yet Another Markup Language

Executive Summary

This deliverable, the first version of "Evidence Assessment and Certification – Integration", provides an initial report on the integration details of the WP3 components within the EMERALD framework. The goal of WP3 is to serve as the central integration point for evidence collection and knowledge extraction tools, contributing to the development of a Certification-as-a-Service (CaaS) framework for continuous certification of harmonized cybersecurity schemes, by assessing the provided evidence to make appropriate certificate decisions. In particular, WP3 and its deliverables address the key result CERTGRAPH (KR2) by implementing the evidence store as a graph database, OPTIMA (KR3) by providing the optimal set of metrics for a given control of a security scheme, MULTICERT (KR4) by providing certification decisions for multiple schemes, and INTEROP (KR7) by providing an interoperability layer for trustworthy systems, assessment results, and catalogue data [1].

This deliverable provides a comprehensive overview of the current integration status of the WP3 components within the EMERALD framework. The components are the following: *Clouditor-Orchestrator*, *Clouditor-Assessment*, *Clouditor-Evidence Store*, *Clouditor-Evaluation*, *Mapping Assistant for Regulations with Intelligence* (*MARI*), *Repository of Controls and Metrics* (*RCM*), and *Trustworthiness System* (*TWS*). The primary objective is to ensure that these components work cohesively to support the Certification-as-a-Service (CaaS) framework. This document outlines the integration strategy, contributions to non-functional requirements, published APIs, and the specific integration status of each component. WP1 has already defined the overall DevOps Methodology and CI/CD strategies [2] and established several requirements for components and the entire framework, focusing on non-functional aspects [3]. WP3 aims to apply these principles by focusing on the continuous integration part, which outlines the principles guiding the integration of different components of the CaaS Framework before their deployment to pilots.

This deliverable is the third WP3 deliverable, after D3.1 and D3.3. While D3.1 focused on the evidence assessment and certification concepts (first version) [4] and D3.3 addressed the implementation (initial prototypes) [5], D3.5 focuses on the integration of these components (first version). Resembling the Software Development Life Cycle (SDLC), the next three WP3 deliverables will complete the process: D3.2 will complete the theoretical phase, D3.4 will document the implementation of components (final version), and D3.6 will finalize the integration of components.

1 Introduction

This introduction outlines the purpose and structure of Deliverable D3.5, focusing on the interim integration of WP3 components within the EMERALD framework. It includes details about the deliverable in section 1.1 and on the document's structure in section 1.2.

1.1 About this deliverable

The EMERALD project is dedicated to establishing a Certification-as-a-Service (CaaS) framework that facilitates continuous certification of cybersecurity schemes, including the EUCS. The project addresses the vital need for improving transparency, accountability, and trust in cloud services across Europe, by building strong evidence management components and exploring AI certification schemes.

In this context, WP3 is crucial, as it integrates the evidence collection and knowledge extraction tools from WP2 and serves as an interface for auditors and pilots via the user interface developed in WP4. WP3's primary objective is to enhance the CaaS framework by evaluating evidence to support informed certification decisions.

This deliverable, D3.5, focuses on the interim integration of WP3 components within the EMERALD framework. It aims to detail the current integration status of each WP3 component, offering insights into their functional and technical aspects, as well as their alignment with the overall project objectives.

To summarize, this document provides the integration strategy used in WP3 and the current status, setting the baseline for future WP3 deliverables.

1.2 Document structure

This document is structured to provide a comprehensive overview of the WP3 components' implementation. The rest of this document is structured as follows:

Section 2 presents the integration strategy that is applied in WP3. The groundwork of this integration strategy is done in WP1. We adhere to the principles of continuous integration and DevOps methodology to ensure a cohesive integration of the components into the EMERALD framework.

Section 3 through 9 delve into the integration details of the WP3 components, which include *Clouditor-Orchestrator*, *Clouditor-Assessment*, *Clouditor-Evidence Store*, *Mapping Assistant for Regulations with Intelligence* (*MARI*), *Clouditor-Evaluation*, *Repository of Controls and Metrics* (*RCM*), and *Trustworthiness System* (*TWS*). Each chapter discusses the component-specific integration status, contributions to non-functional requirements, published APIs, and current interactions with other components.

Finally, Section 10 provides the conclusion, summarizing the key findings and outlining the next steps for further integration and refinement of the WP3 components within the EMERALD project.

2 Integration strategy

WP3 components constitute a significant portion of the EMERALD framework and occupy a central role from an architectural perspective. Therefore, having a well-defined integration strategy is crucial. WP1 has already established the overall DevOps Methodology and CI/CD strategies [2] and has defined several requirements for both individual components and the entire framework, including non-functional aspects [3]. This deliverable emphasises continuous integration, which encompasses the principles guiding the integration of various components within the CaaS Framework before they are deployed to the pilots. WP3 is committed to applying these principles, and this deliverable represents the current state of integration. The approach to integration in WP3 is informed by the foundational work completed in WP1 and is elaborated upon in this section.

This section begins with an overview of the test bed environment, the DevOps methodology and the continuous integration practices used in EMERALD [2] [3]. We leverage the work done in WP1 and put it into the context of the WP3 components. Then, we elicit from the WP1 requirements the ones that are crucial for the integration of the WP3 components. Section 2.5 provides the structure of the component-specific integration status that is used in the following sections, offering insights into the current progress of each component's integration.

2.1 Test Bed Environment

The EMERALD CaaS Framework is supported by two different environments: integration and production (see Figure 1). Each of them consists in a four-node Kubernetes cluster over a vSphere platform. The components are first deployed in the Integration environment in containerized form. Once the integration tests have been passed, the components are promoted to the production environment.

These environments have been developed following an Infrastructure as Code (IaC) approach, so that the deployment on another vSphere[1] platform is easier (vSphere is a cloud computing virtualization platform from VMware[2]). For the creation and configuration of the cluster we use a set of OpenTofu[3] and Ansible[4] scripts that can be easily adjusted for use on other nodes.

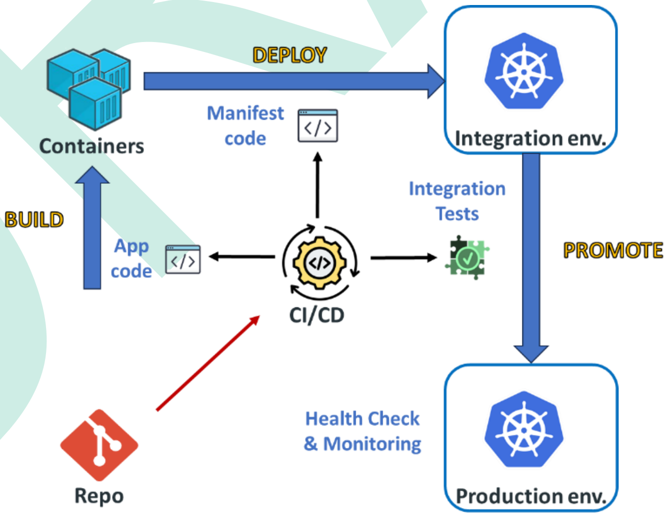


Figure 1. Integration and Production environments in the CaaS framework

[1] http://vmware.com/products/cloud-infrastructure/vsphere
[2] https://www.vmware.com/
[3] https://opentofu.org/
[4] https://www.ansible.com/

2.2 DevOps Methodology

In WP3, we face the challenge of managing multiple components developed by different partners, making it prudent to follow a common approach. WP1 identified several relevant risks, such as the potential inability to fully integrate EMERALD components, and challenges, like integrating and testing all components effectively.

The DevOps methodology emphasizes key characteristics that WP3 is also responsible for, particularly those that play a crucial role in this deliverable: "Integrate as soon as" and "Keep requirement traceability".

2.3 Continuous Integration

WP1 has outlined four key aspects for the continuous integration of components into the EMERALD framework [2], with WP3 responsible for specific areas. WP3's responsibilities are highlighted in the following, while the remaining aspects are managed entirely by WP1:

- **Packaging components as containers**,
- defining the environment defined with Infrastructure as code (IaC),
- **ensuring progressive verification,** and
- automating the integration process.

For WP3, these requirements entail that each component must provide Dockerfile(s) in the root of the GitLab repository, to automatically build and deploy images when the code changes. Utilizing Dockerfiles bridges the development of individual components (in WP2 and WP3) with operations (in WP1), promoting synergy. While Dockerfiles are preferred, Docker Compose or custom scripting can be used as alternatives when necessary. Additionally, a *gitlab-ci.yml* file must be included in the root of the component's GitLab repository to specify instructions for the GitLab CI pipeline, such as defining stages and scripts. It is also encouraged to use integration tests, ranging from simple health checks to more complex verifications of component interactions. At this stage of the project, the focus is on the initial deployment in the test bed, with further integration tests from developers and pilots to follow in the coming months, which will be part of the subsequent integration deliverable D3.6. The addition of integration tests is expected to be a collaborative effort between the DevOps team of WP1 and the component owners. Nonetheless, we will provide an integration status to show how one component is interacting with others.

2.4 Requirements for WP3 Integration

In this section, we outline the requirements for WP3 integration, derived from the work completed in WP1 [3]. The EMERALD project encompasses several types of requirements, including functional and non-functional requirements. The non-functional ones have been classified as business-driven requirements, UI/UX-related requirements, and requirements reported by WP1. We have already mapped relevant business-driven and UI/UX requirements to the functional requirements of individual components. Such functional requirements, e.g. "ORCH.01 Final certificate decision", have been addressed in previous WP3 deliverables [4] [5] and will continue to be discussed in upcoming deliverables. However, since the primary focus of this deliverable is on the integration of WP3 components into the entire EMERALD framework, we will only consider a subset of the eight non-functional requirements elicited in WP1, which were defined in D1.3 [3]:

- **WP1.01 Performant Network**
- **WP1.02 Portability**
- WP1.03 Scalability

- WP1.04 Installability
- WP1.05 Documentation
- **WP1.06 Agile development**
- WP1.07 Observability
- **WP1.08 Security**

Our objective is to provide information on how much WP3 currently contributes to these requirements on a component basis. As this deliverable deals with the integration of WP3 components, we narrow the focus to WP1.01, WP1.02, WP1.06, and WP1.08 (marked in bold in the list above). These requirements are described in more detail below.

- **WP1.01: Performant framework**. The EMERALD framework should be as performant as possible. The response time for a user action in normal conditions should not be larger than a few seconds. The framework components will have to pass automatic integration tests by the CI/CD pipeline before being integrated into the framework. The validation task in WP5 will validate both the functionality and the performance of the EMERALD framework. Apart from these controls, the framework infrastructure will be continuously monitored, and the implemented environment will allow flexibility to upgrade the resources if they are falling short (e.g., adding more memory or CPUs to the Kubernetes nodes or providing extra nodes).
- **WP1.02: Portability**. The EMERALD framework should be portable and work in any typical business environment. The framework components will be packaged as containers, which are portable technology by definition. We will use the Docker ecosystem to build and share images. For image building, we will support both Docker and Docker Compose.
- **WP1.06: Agile development.** The EMERALD framework will be constructed using an agile methodology, with several cycles of Design, Build, Test, and Deploy. The project management has already foreseen three incremental releases -V1, V2, V3- in months M12, M24 and M33. The WP1 team will provide several tools to make this possible, for example:
  - **Source control**: The *GitLab* tool allows code management and implementation of CI/CD processes that help to speed up the development.
  - **CI/CD processes**: *GitLab CI* allows continuous integration and deployment tasks to be implemented.
  - **Integration automation**. A *GitLab Agent* for *Kubernetes* monitors the framework repository and will allow deploying and testing new versions of the components directly, checking the health of the components.
- **WP1.08: Security.** The EMERALD framework must be secure. This implies correct user authentication and authorization, secret management, preventing intrusion, etc. For user management, a specific tool such as Keycloak[5] will be installed, which is specifically designed to manage identity and access. Keycloak supports OpenID Connect, single-sign-on for all the components and allows synchronization with external identity sources. The framework will implement role-based access control as an authorization mechanism to prevent all users from having access to all functionalities. Sensitive information, such as API keys, certificates, and passwords will be securely stored in the cluster. Additionally, network policies will be established, utilizing an inverse proxy, to control communication between containers and services within the cluster, thereby minimizing potential attack vectors.

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

2.5 Component-specific Integration Status

In the following sections, we will dive into the current integration status of each WP3 component. Below, we outline the structure and content of these sections to provide a clear understanding of what each will cover.

2.5.1 Contribution to Non-Functional Requirements

Each component should provide information on its contribution to the following four non-functional requirements outlined by WP1:

- **WP1.01 Performant Network:** This section should describe how the component contributes to network performance, including whether it operates with sufficient efficiency.
- **WP1.02 Portability:** Here, contributions to portability are detailed, such as the availability and use of Dockerfiles or Docker Compose, to ensure the component can be easily deployed across different environments.
- **WP1.06 Agile development:** This requirement focuses on contributions to agile development, including the presence of a *gitlab-ci.yml* file and the existence of tests. While automated integration tests for more realistic scenarios are still in development, the current focus is on ensuring the initial deployment of all components. Integration tests will be added in the coming months and detailed in the final version of this deliverable, D3.6.
- **WP1.08 Security:** Contributions to security are discussed, such as the proper configuration of Keycloak and the functioning of authentication mechanisms within the deployed component in the Kubernetes cluster.

2.5.2 Published APIs

This section provides detailed information about the API endpoints for each component. It includes a description of available endpoints, their functionalities, and how they facilitate interaction between components.

2.5.3 Integration Status

This section describes the current status of the integration of each component with other components with which it is designed to interact. The status can be categorized into several stages, each reflecting the progress made in integrating the component within the EMERALD framework:

- **Not Started**: The integration process has not yet begun. No work has been done to establish connections or interfaces between this component and others.
- **Developing API**: The component is currently in the process of developing its API. This stage involves defining how the component will interact with others, but the API is not yet finalized or available for use.
- **API Finished**: The API development has been completed and is ready for testing. However, the component has not yet undergone local tests or integration tests with other components.
- **Tested Locally**: The component has undergone local testing, verifying its functionality in isolation. While it works as intended on its own, it has not yet been tested in conjunction with other components.
- **Connected**: This status indicates that the component has successfully established connections with other components. Data exchange can occur, but further testing is needed to ensure full compatibility and performance.

- **Testing**: The integration of the component with others is currently being tested. This phase involves checking interactions and data flow between the components to identify any issues that may arise.
- **Integrated**: The component has been fully integrated into the framework. It has passed all necessary tests, and is working as expected, interacting seamlessly with other components in the EMERALD system.

3 Integration of Clouditor-Orchestrator

This section reports on the integration of the *Clouditor-Orchestrator* (*Orchestrator* from now on) into the EMERALD framework. The *Orchestrator* plays a central role in the EMERALD framework by orchestrating several components and providing final certificate decisions.

We provide current information about the contribution to relevant non-functional requirements of WP1, the API endpoints and the current integration status with other components within the CaaS framework.

3.1 Contribution to Non-Functional Requirements

The contribution to the relevant WP1 non-functional requirements defined in section 2.5.1 is listed below.

WP1.01 Performant Network: The *Orchestrator* provides REST APIs for regular and occasional data exchange, e.g. for user interaction with the EMERALD user interface. However, it also provides highly efficient connections via gRPC for endpoints where large amounts of data are expected, e.g. for assessment results sent from the assessment. Therefore, it contributes very well to the requirement of having an overall performant network in the *EMERALD UI*.

WP1.02 Portability: We provided a Dockerfile[6] at the root of the component in the GitLab repository. For more information about installation and usage, see D3.3 [5]. Providing a Dockerfile enables to easily use the *Orchestrator* in other business environments which use other CI/CD pipeline technologies or another container orchestration.

WP1.06 Agile development: By using the provided GitLab instance for further development of the component and providing the *gitlab-ci.yml* file in addition to the Dockerfile, we highly contribute to an agile style of development. Every time a change is done in the main branch, a new image is built and deployed into the testbed. Currently, we leverage unit tests to make the code more robust. Automated integration tests will be added in the following months.

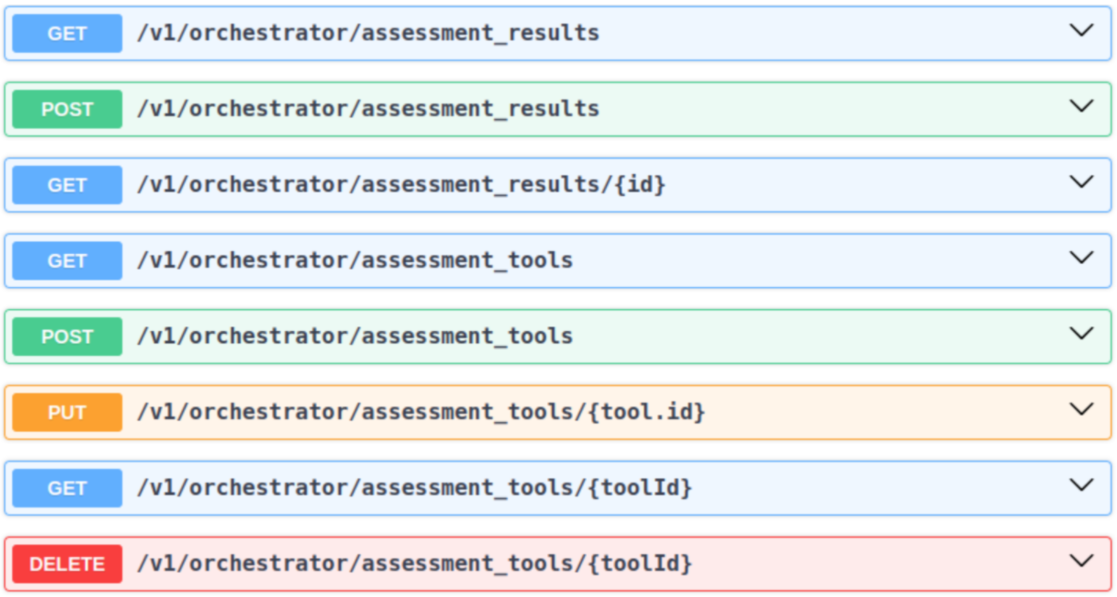
WP1.08 Security: Like all Clouditor components, the *Orchestrator* supports OAuth[7] to only allow interaction with the components that are defined in the KeyCloak. Therefore, it refuses connections to/from components that are not allowed, e.g. from components outside the cluster. By supporting this zero-trust approach, security is strengthened throughout the framework.

3.2 Published APIs

The following shows the *Orchestrator* API endpoints that other components can use (if they are authenticated). For more details, see the *openapi.yaml* file in the Clouditor repository[8].

[6] https://git.code.tecnalia.com/emerald/public/components/orchestrator/orchestrator/-/blob/master/Dockerfile
[7] https://datatracker.ietf.org/wg/oauth/about/
[8] https://github.com/clouditor/clouditor/blob/main/openapi/orchestrator/openapi.yaml

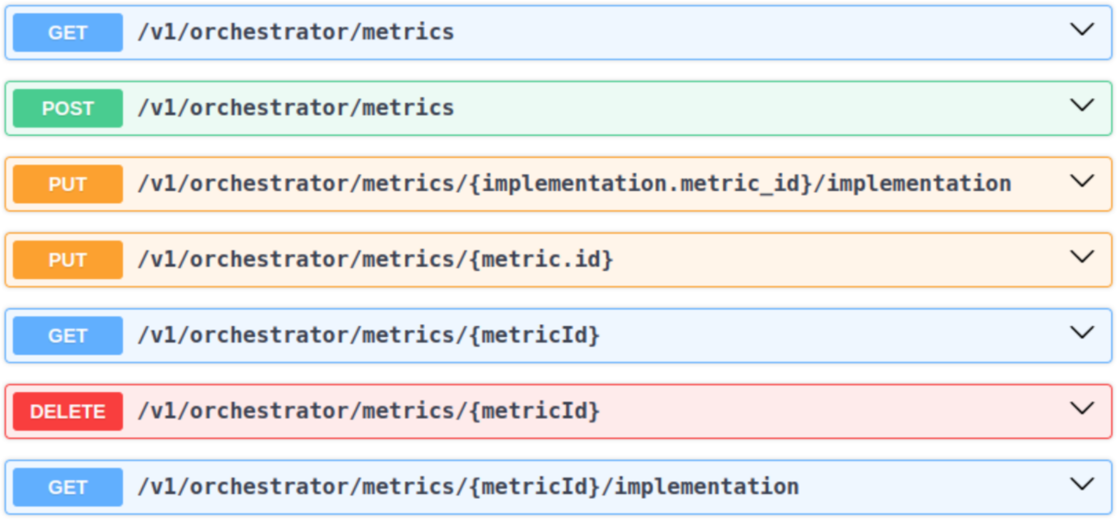
API endpoints for handling assessment results and tools:



GET /v1/orchestrator/assessment_results
POST /v1/orchestrator/assessment_results
GET /v1/orchestrator/assessment_results/{id}
GET /v1/orchestrator/assessment_tools
POST /v1/orchestrator/assessment_tools
PUT /v1/orchestrator/assessment_tools/{tool.id}
GET /v1/orchestrator/assessment_tools/{toolId}
DELETE /v1/orchestrator/assessment_tools/{toolId}

Figure 2. Orchestrator API endpoints for assessment results

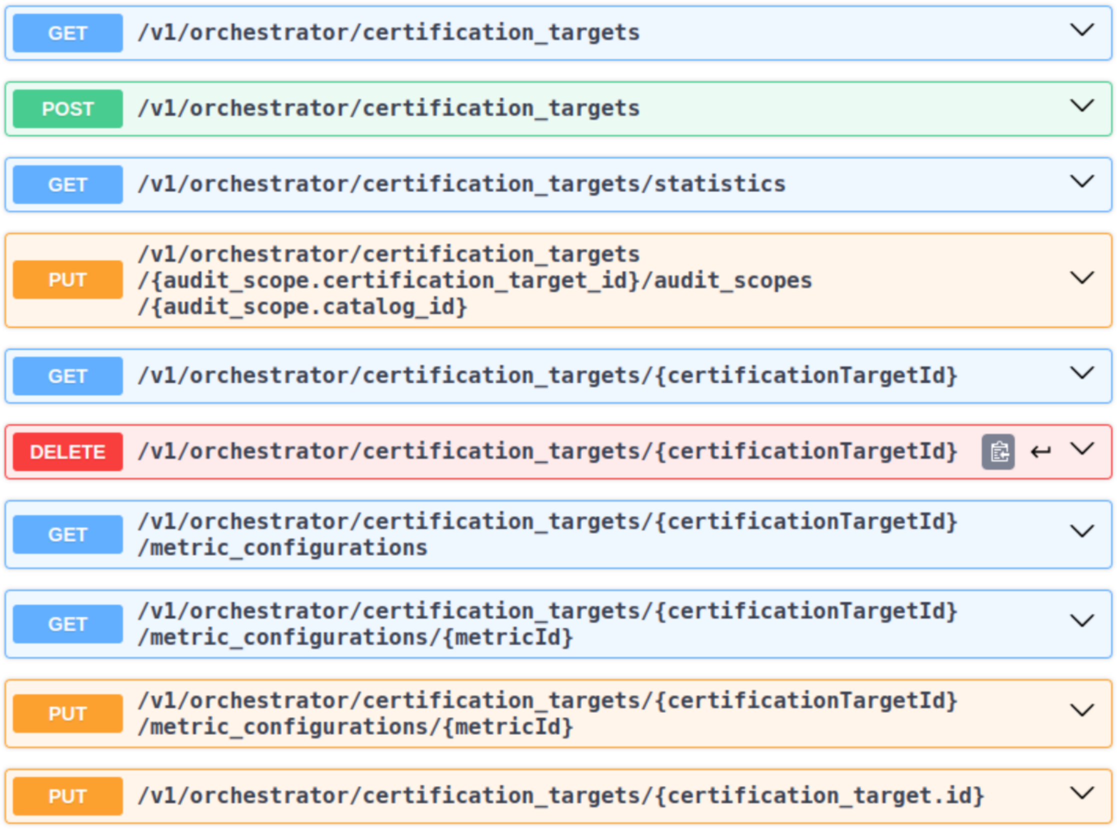
API endpoints for handling metrics:



GET /v1/orchestrator/metrics
POST /v1/orchestrator/metrics
PUT /v1/orchestrator/metrics/{implementation.metric_id}/implementation
PUT /v1/orchestrator/metrics/{metric.id}
GET /v1/orchestrator/metrics/{metricId}
DELETE /v1/orchestrator/metrics/{metricId}
GET /v1/orchestrator/metrics/{metricId}/implementation

Figure 3. Orchestrator API endpoints for metrics

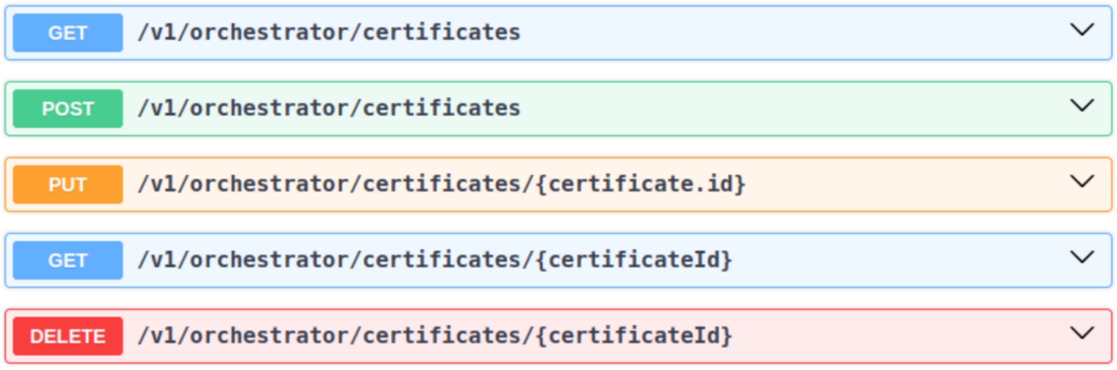
API endpoints for handling certification targets:



GET /v1/orchestrator/certification_targets
POST /v1/orchestrator/certification_targets
GET /v1/orchestrator/certification_targets/statistics
PUT /v1/orchestrator/certification_targets/{audit_scope.certification_target_id}/audit_scopes/{audit_scope.catalog_id}
GET /v1/orchestrator/certification_targets/{certificationTargetId}
DELETE /v1/orchestrator/certification_targets/{certificationTargetId}
GET /v1/orchestrator/certification_targets/{certificationTargetId}/metric_configurations
GET /v1/orchestrator/certification_targets/{certificationTargetId}/metric_configurations/{metricId}
PUT /v1/orchestrator/certification_targets/{certificationTargetId}/metric_configurations/{metricId}
PUT /v1/orchestrator/certification_targets/{certification_target.id}

Figure 4. Orchestrator API endpoints for certification targets

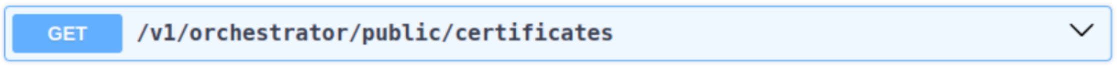
API endpoints for handling certificates:



GET /v1/orchestrator/certificates
POST /v1/orchestrator/certificates
PUT /v1/orchestrator/certificates/{certificate.id}
GET /v1/orchestrator/certificates/{certificateId}
DELETE /v1/orchestrator/certificates/{certificateId}

Figure 5. Orchestrator API endpoints for certificates

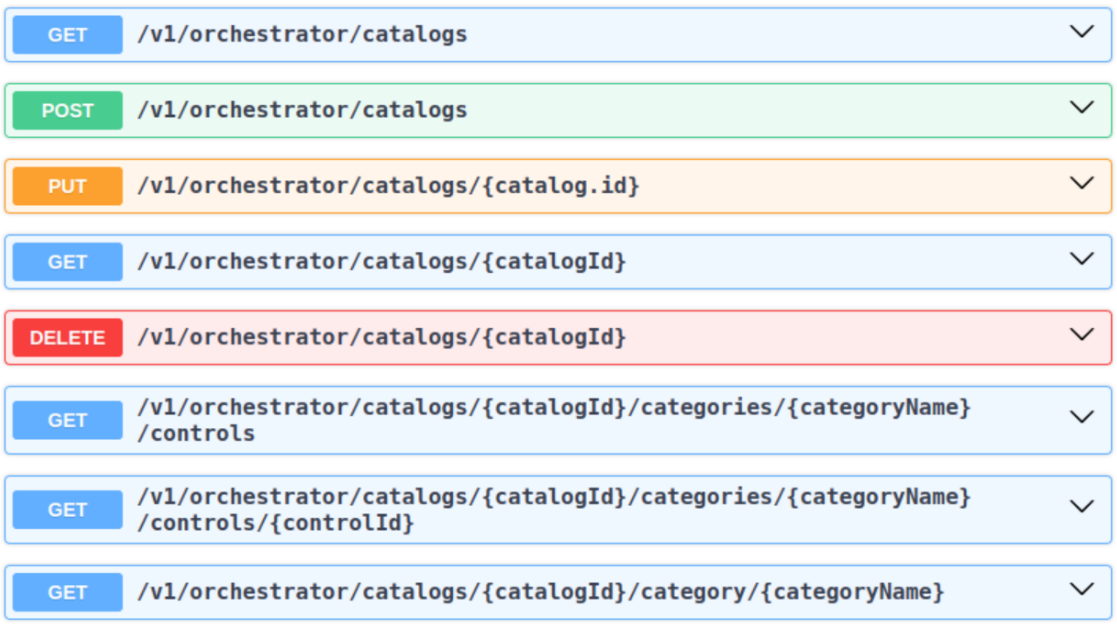
API endpoint for listing all target certificates without state history:



GET /v1/orchestrator/public/certificates

Figure 6. Orchestrator API endpoints for certificates (publicly available)

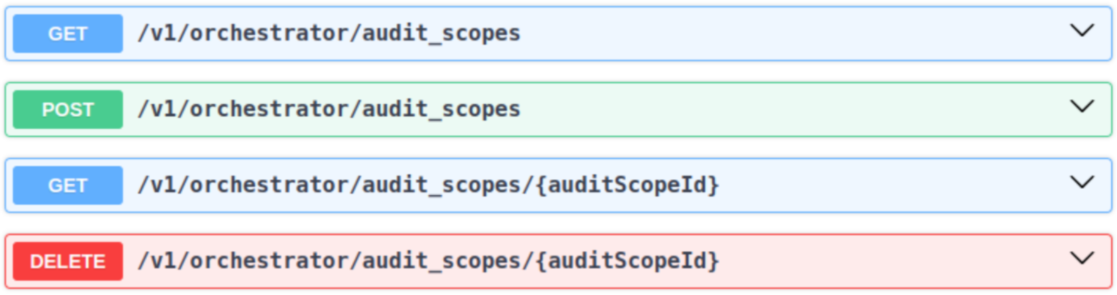
API endpoints for handling catalogues:



```
GET     /v1/orchestrator/catalogs
POST    /v1/orchestrator/catalogs
PUT     /v1/orchestrator/catalogs/{catalog.id}
GET     /v1/orchestrator/catalogs/{catalogId}
DELETE  /v1/orchestrator/catalogs/{catalogId}
GET     /v1/orchestrator/catalogs/{catalogId}/categories/{categoryName}
        /controls
GET     /v1/orchestrator/catalogs/{catalogId}/categories/{categoryName}
        /controls/{controlId}
GET     /v1/orchestrator/catalogs/{catalogId}/category/{categoryName}
```

Figure 7. Orchestrator API endpoints for catalogues

API endpoints for handling audit scopes:



```
GET     /v1/orchestrator/audit_scopes
POST    /v1/orchestrator/audit_scopes
GET     /v1/orchestrator/audit_scopes/{auditScopeId}
DELETE  /v1/orchestrator/audit_scopes/{auditScopeId}
```

Figure 8. Orchestrator API endpoints for audit scopes

3.3 Integration Status

Currently, the *Orchestrator* has been successfully deployed on the Kubernetes cluster. The foundation of this integration lies in the code base located in the EMERALD repository for the *Orchestrator*[9]. To facilitate the utilization of this code base, a Dockerfile was provided in deliverable D3.3 [5], which has undergone some modifications to ensure the successful deployment of the *Orchestrator*. Furthermore, the configuration for continuous integration is found in the DevOps repository, where the *kustomization.yaml* file has been updated to include several manifest files essential for the deployment and operation of the *Orchestrator* and its dependencies. These manifest files define various services, deployments, and ingress configurations for the Postgres database and the *Orchestrator* itself, in addition to scripts in configuration maps that handle the Postgres setup.

The *EMERALD UI* relies on several endpoints provided by the *Orchestrator* for communication. The *Orchestrator* has been previously implemented in MEDINA, including its API endpoints [6] . It is anticipated that minor modifications might be necessary to align these endpoints with the

[9] https://git.code.tecnalia.dev/emerald/public/components/orchestrator/orchestrator

specific requirements of the EMERALD project, and any adjustments will be discussed and coordinated with WP4 soon.

The *Evaluation*, *Assessment*, and *Evidence Store* components have been well-tested locally and in other projects since they are part of the Clouditor toolbox. However, their communication must be validated in the EMERALD development cluster, e.g. to check if authentication is correctly configured. The ideal integration test will involve workflows that start with creating evidence from the collectors, sending assessment results to the *Evaluation*, aggregating these results, and sending them back to the *Orchestrator* for making final certificate decisions.

Regarding the *RCM*, the API has been completed and requires testing in the EMERALD cluster. Some refinements may be needed after testing, but the initial set of endpoints related to the *RCM* is available.

Table 1 provides a quick summary of the current state of the connections that the *Orchestrator* is supposed to interact with.

Table 1. Integration Status of Orchestrator with other EMERALD Components

Component	**Status**	**Comment**
EMERALD UI	API Finished	Requires coordination with WP4 and testing.
Evaluation	Tested Locally	Communication needs testing in the EMERALD Kubernetes cluster.
RCM	API Finished	Ready for integration testing.
Assessment	Tested Locally	Communication needs testing in the EMERALD Kubernetes cluster.
Evidence Store	Tested Locally	Communication needs testing in the EMERALD Kubernetes cluster.

4 Integration of Clouditor-Assessment

The *Clouditor-Assessment* component (in the following "*Assessment*") is responsible for assessing evidence sent from the *Evidence Store*. It utilizes metrics (i.e. rules) to effectively assess evidence. By leveraging the ontology (which is being further developed in WP2), the assessment process is decoupled from evidence collection. For more details on the metrics and other technical aspects, see the previous implementation deliverable of WP3 [5]. Additionally, the *Assessment* interacts with the *Trustworthiness System* (*TWS*) by sending assessment results to enhance integrity. It is also connected to the *Orchestrator* by forwarding these results, which are stored and processed and are essential for making final decisions on certificates.

4.1 Contribution to Non-Functional Requirements

The contribution to the relevant WP1 non-functional requirements defined in section 2.5.1 is listed below.

WP1.01 Performant Network: The *Assessment* component provides only gRPC APIs since its API is exclusively used by the Clouditor components that communicate via gRPC. This ensures efficient data exchange and contributes significantly to the requirement of having an overall performant network in the EMERALD framework.

WP1.02 Portability: Dockerfile[10] is provided in the root of the *Assessment* component’s GitLab repository. This enables the *Assessment* to be easily used in various business environments that utilize different CI/CD pipeline technologies or container orchestration methods.

WP1.06 Agile development: By utilizing the provided GitLab instance for the development of the *Assessment* component and providing a *gitlab-ci.yml* file in addition to the Dockerfile, we support an agile development style. Changes in the main branch trigger automatic image builds and deployments into the testbed. Currently, we leverage unit tests to enhance code robustness, with automated integration tests planned for the upcoming months.

WP1.08 Security: Like all Clouditor components, the *Assessment* component supports OAuth to restrict interactions to the components defined in Keycloak. This ensures that only authorized components can connect, thereby strengthening security throughout the EMERALD framework.

4.2 Published APIs

The *Assessment* component only provides one endpoint, namely for assessing evidence that is sent to it.

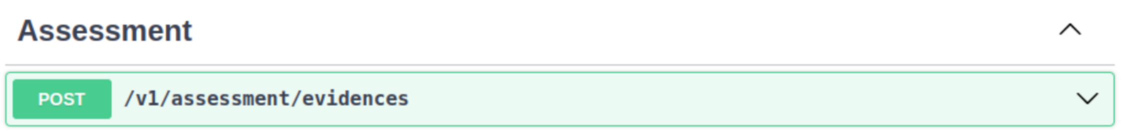


Figure 9. Assessment API endpoint for evidence

4.3 Integration Status

The *Assessment* component is in the process of being deployed in the Kubernetes cluster. The foundation of this integration lies in the code base located in the EMERALD repository for the *Assessment* component. To facilitate the utilization of this code base, a Dockerfile was provided in deliverable D3.3 [5], which has undergone modifications to ensure the successful deployment

[10]https://git.code.tecnalia.dev/emerald/public/components/assessment/assessment/-/blob/master/Dockerfile

of the *Assessment*. Furthermore, the configuration for continuous integration is found in the DevOps repository, where the *kustomization.yaml* file was updated to include several manifest files essential for the deployment and operation of the *Assessment* and its dependencies.

The communication with the *Orchestrator* and the *Evidence Store* components is well-tested locally and in other projects since they are part of the Clouditor toolbox. However, their communication must be validated in the EMERALD development cluster, e.g., to check if authentication is correctly configured.

Currently, the *TWS* API is under development (see section 9.3). When this is finished and the *Assessment* is being deployed as well, the connection to *TWS* (sending assessment results) must be tested. Potentially, adaptations in the code within the *Assessment* repository might be necessary to ensure that assessment results are forwarded to both the *Orchestrator* and the *TWS*.

Table 2 provides a quick summary of the current state of the connections that the *Assessment* component is supposed to interact with.

Table 2. Integration Status of Assessment with other EMERALD Components

Component	**Status**	**Comment**
Orchestrator	Tested Locally	Communication needs testing in the EMERALD Kubernetes cluster.
Evidence Store	Tested locally	Communication needs testing in the EMERALD Kubernetes cluster.
TWS	Developing API (*TWS*)	To be tested.

5 Integration of Clouditor-Evidence Store

This section reports on the integration of the *Clouditor-Evidence Store* (in the following "*Evidence Store*") within the EMERALD framework. The *Evidence Store* plays a vital role in the overall architecture by serving as the central repository for evidence collected from various sources. Its primary tasks include retrieving evidence from the evidence collectors, saving them in a graph-based database, and forwarding evidence to the *Assessment* and the *TWS* to enhance the integrity of the evidence.

We provide current information on the contributions of the *Evidence Store* to relevant non-functional requirements defined in WP1, the published API endpoints, and the current integration status with other components in the EMERALD ecosystem.

The *Evidence Store* is designed to support various evidence collectors, ensuring flexibility in how evidence is gathered and stored.

5.1 Contribution to Non-Functional Requirements

The contribution to the relevant WP1 non-functional requirements defined in section 2.5.1 is listed below.

WP1.01 Performant Network: The *Evidence Store* provides both REST APIs and gRPC endpoints to support various evidence collectors, accommodating different usage scenarios. For instance, *Clouditor-Discovery* utilizes gRPC [7], while *AMOE* will use the REST API endpoints [8]. For forwarding the evidence to the *Assessment*, the *Evidence Store* will utilize the *Assessment*'s gRPC endpoint, ensuring highly efficient data transmission of evidence that can scale up significantly in complex systems such as clouds. Additionally, we plan to leverage the certification graph by implementing the *Evidence Store* as a graph database, which could lead to performance increases.

WP1.02 Portability: A Dockerfile[11] is provided in the root of the *Evidence Store*'s GitLab repository. This enables the *Evidence Store* to be easily utilized in various business environments that employ different CI/CD pipeline technologies or container orchestration methods.

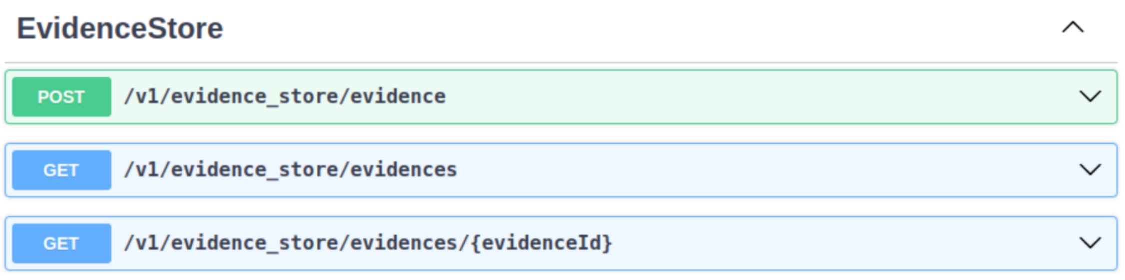
WP1.06 Agile development: By utilizing the provided GitLab instance for the development of the *Evidence Store* and including a *gitlab-ci.yml* file in addition to the Dockerfile, we support an agile development style. Changes in the main branch trigger automatic image builds and deployments into the testbed. Currently, we leverage unit tests to enhance the code's robustness, with automated integration tests planned for future implementation.

WP1.08 Security: Like all Clouditor components, the *Evidence Store* supports OAuth to restrict interactions to the components defined in Keycloak. This ensures that only authorized components can connect, thus strengthening security throughout the EMERALD framework.

[11]https://git.code.tecnalia.dev/emerald/public/components/evidence-store/evidence-store/-/blob/master/Dockerfile

5.2 Published APIs

The *Evidence Store* provides the following three endpoints for storing evidence, listing all evidence and getting specific evidence, respectively.


EvidenceStore

POST /v1/evidence_store/evidence

GET /v1/evidence_store/evidences

GET /v1/evidence_store/evidences/{evidenceId}

Figure 10. Evidence Store API endpoints

5.3 Integration Status

Currently, the *Evidence Store* is being deployed on the Kubernetes cluster. The foundation of this integration lies in the code base located in the EMERALD repository for the *Evidence Store*. To facilitate the utilization of this code base, a Dockerfile was provided in deliverable D3.3 [5], which has undergone modifications to ensure the deployment of the *Evidence Store*, including the manifest files that encompass the database.

The connections between the *Evidence Store* and the *Assessment* and *Clouditor-Discovery* [7] have been well-tested locally and in other projects since they are part of the Clouditor toolbox. However, their communication must be validated in the EMERALD development cluster, e.g., to check if authentication is correctly configured. The other evidence collectors, i.e. *AMOE* [8], *Codyze* [9], *eknows-e3* [9], and *AI-SEC* [10], will need to be tested once all the components, including the *Evidence Store*, are successfully deployed. These tests may lead to updates in the ontology and, consequently, updates in the database of the *Evidence Store* as well.

Table 3 provides a quick summary of the current state of the connections that the *Evidence Store* is supposed to interact with.

Table 3. Integration Status of Evidence Store with other EMERALD Components

Component	Status	Comment
Assessment	Tested Locally	Communication needs testing in the EMERALD Kubernetes cluster.
Clouditor-Discovery	Tested Locally	Communication needs testing in the EMERALD Kubernetes cluster.
AMOE	Not tested	Needs to be tested after deployment.
Codyze	Not tested	Needs to be tested after deployment.
Eknows-e3	Not tested	Needs to be tested after deployment.
AI-SEC	Not tested	Needs to be tested after deployment.

6 Integration of MARI

This section reports on the integration of the *Mapping Assistant for Regulations with Intelligence* (*MARI*) component into the EMERALD framework. *MARI* facilitates certification management by leveraging its intelligent capabilities to identify relevant metrics for each control, link controls across multiple certification schemes, and improve both performance and accuracy.

MARI is powered by Natural Language Processing (NLP) techniques that enable automatic associations between:

- A security control and one or more security metrics.
- Security controls across different certification schemes.

At its core, *MARI* uses a sentence transformer model to generate embeddings. These allow *MARI* to effectively represent the semantic meaning of controls and metrics in a vector space. Associations are then performed based on the similarity between these embeddings.

This intelligent approach not only saves time and reduces errors but also enhances the overall performance of certification management. For a more detailed explanation, please refer to Deliverable D3.3 [5].

6.1 Contribution to Non-Functional Requirements

The contribution to the relevant WP1 non-functional requirements defined in section 2.5.1 is listed below.

WP1.01 Performant Network: *MARI* offers REST APIs for control-to-control mappings and control-to-metric association results. Using NLP tools to obtain associations can be resource-intensive, but we have implemented optimizations to tackle this challenge. *MARI* runs its association processes only when needed and saves the resulting mappings in a dedicated repository (*RCM*). By avoiding repeated executions, this approach reduces computational load and boosts efficiency. As a result, it helps ensure an overall performance network in the *EMERALD UI*.

WP1.02 Portability: The Dockerfile[12] on the root directory serves as a blueprint for building containerized environments and streamlining the deployment process for the *MARI* system. This file specifies all necessary dependencies, libraries, configurations, and runtime instructions required to package *MARI* into a consistent and portable container. This approach enhances scalability, simplifies integration with CI/CD pipelines, and enables seamless migration to cloud or on-premises environments.

WP1.06 Agile development: The project's modular architecture and the included Dockerfile lays a strong foundation for agile development. Although we do not yet have a CI/CD pipeline in place, we plan to integrate one, such as GitLab CI, in future updates. This will automate the process of building and deploying new changes for integration and production environments. Unit tests have not yet been implemented, but they are planned to be introduced soon to enhance the project's stability and reliability.

WP1.08 Security: Future steps will also include the integration of OAuth for secure interactions, ensuring that only components registered in systems like Keycloak can access the framework. Once complete, the system will adopt a zero-trust model, blocking unauthorized connections—

[12]https://git.code.tecnalia.com/emerald/public/components/mari/mari/-/blob/master/Dockerfile?ref_type=heads

such as those from components outside the cluster—and greatly improving the framework's overall security.

6.2 Published APIs

The logic of mapping controls and metrics is fully implemented and ready for deployment. However, the APIs have not been published yet. These will handle two types of mappings: *metric to control* mapping and *control-to-control* mapping.

The *metric-to-control* mapping will link metrics to controls of a schema. The input for this will include a list of controls and metrics, and the output will provide a mapping between each control and its associated metrics. Here's an example:

Input (Example):

```
[# array of Controls
schema_id: "1",
schema_name: "schema1",
controls:  [ # Array of controls of the schema
   {# control 1 info
      id: “controlId1”,
      …
   },
   {# control 2 info
      id: “controlId2”,
      …
   },
   {
   …
   }
],
[ # Array of metrics
   {# metric 1 info
       Id: “metricId1”
       …
    },
    {# metric 2 info
       Id: “metricId2”
       …
    },
    {
       …
    }
}]
```

Figure 11. MARI input example for metric-to-control mapping

Output (Example):

```
[ # array of linked controls
   {
   # Control 1 info
   control_id: “controlId1”,
   …
   metrics_mapped:[ metricId22, metricId43, …]
   },
   {
   # Control 2 info
   control_id: “controlId2”,
   …
   metrics_mapped:[ metricId32, metricId21, …]
   },
   {
      …
   }
]
```

Figure 12. MARI output example for metric-to-control mapping

The *control-to-control* mapping, which maps controls between two different schemas, will follow a similar structure.

Input (Example):

```
{
schema_id: "1",
schema_name: "schema1",
controls:  [ # Array of controls of the schema
   {# control 1 info
      id: “controlId11”,
      …
   },
   {# control 2 info
      id: “controlId54”,
      …
   },
   {
   …
   }
},
{
schema_id: "2",
schema_name: "schema2",
controls:  [ # Array of controls of the schema
   {# control 1 info
      id: “controlId12”,
      …
   },
   {# control 2 info
      id: “controlId33”,
      …
   },
   {
   …
   }
}
```

Figure 13. MARI input example for Control-to-Control mapping

Output (Example):

```
[ #array of linked controls
   {# Control info (schema 1)
      control_id: “controlId11”,
      mapped_controls:  #  Control  list  (schema  2)  [  controlId12,
controlId33, … idn ]
},
   {# Control info (schema 1)
      control_id: “controlId54”,
      mapped_controls:   #  Control  list(schema  2)[  controlId33,
controlId12, … idn ]
   }
   …
]
```

Figure 14. MARI output example for Control-to-Control mapping

Although the endpoints are not yet available, the underlying logic is fully functional. The *src/mock_tool.py* script simulates these requests and demonstrates how the API will respond once published.

The application, built with Flask[13], will be served via Gunicorn[14]. This robust, production-grade WSGI server is widely used for deploying Python web applications. It was chosen for its ability to handle multiple concurrent requests efficiently through its pre-fork worker model, which improves reliability and scalability[15][16].

Once deployed, the API can handle requests as described above. Future integration through Kubernetes and Keycloak will ensure secure and authorized access to the API.

6.3 Integration Status

MARI is being deployed on a Kubernetes cluster and interacts solely with the *RCM* through a predefined API. The API has been defined, though it has not yet been published. Table 4 provides a quick summary of the current state of the connections that *MARI* is supposed to interact with.

Table 4. Integration Status of MARI with other EMERALD Components

Component	Status	Comment
RCM	Developing API	Ready for publishing and testing.

[13] https://flask.palletsprojects.com/en/stable/
[14] https://gunicorn.org/
[15] https://docs.gunicorn.org/en/latest/design.html
[16] https://flask.palletsprojects.com/en/stable/deploying/gunicorn/

7 Integration of Clouditor-Evaluation

In this section, we report on the integration of the *Clouditor-Evaluation* (in the following "*Evaluation*") within the EMERALD framework. The *Evaluation* component aggregates multiple assessment results and serves as the last step before the final certificate decision is made by the *Orchestrator* after receiving the evaluation results.

We provide current information on the contributions of the *Evaluation* component to relevant non-functional requirements defined in WP1, the published API endpoints, and the current integration status with other components in the EMERALD ecosystem.

The *Evaluation* component is designed to communicate exclusively with the *Orchestrator*, ensuring efficient *gRPC* connections that optimize performance.

7.1 Contribution to Non-Functional Requirements

The contribution to the relevant WP1 non-functional requirements defined in section 2.5.1 is listed below.

WP1.01 Performant Network: The *Evaluation* component communicates exclusively with the *Orchestrator*, utilizing pure *gRPC* connections to maximize performance. This design choice ensures efficient data exchange and minimizes latency when delivering assessment results, thereby contributing significantly to the overall performance of the EMERALD framework.

WP1.02 Portability: A Dockerfile[17] is provided in the root of the *Clouditor-Evaluation*'s GitLab repository. This allows for an easy deployment of the *Evaluation* component across various business environments that utilize different CI/CD pipeline technologies or container orchestration methods. The Dockerfile includes all necessary dependencies and configurations to facilitate seamless integration.

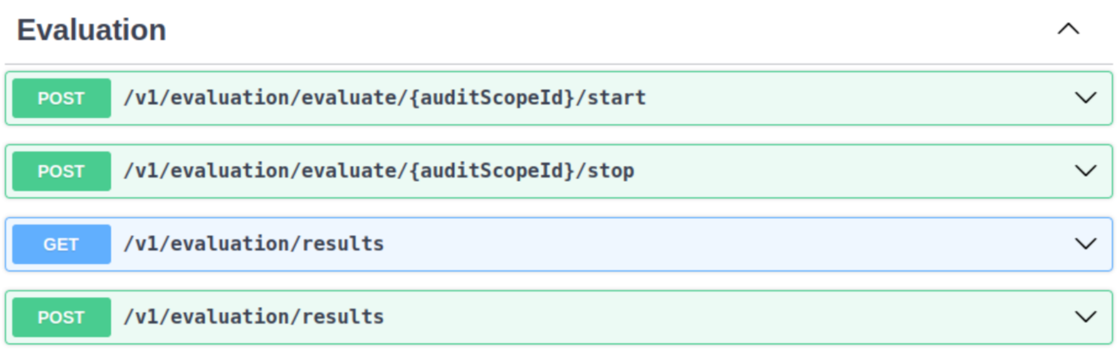
WP1.06 Agile development: The *Clouditor-Evaluation* component adopts agile development practices, utilizing a GitLab repository for code management and including a *gitlab-ci.yml* file alongside the Dockerfile. This setup supports continuous integration and deployment, allowing for rapid iterations and enhancements based on stakeholder feedback. Automated unit tests are implemented to ensure code reliability, with plans for comprehensive integration tests in future updates.

WP1.08 Security: Like all Clouditor components, the *Evaluation* component incorporates OAuth to restrict access to authenticated components registered in Keycloak. This security measure ensures that only authorized users and components can interact with the *Evaluation* component, thereby enhancing the overall security posture of the EMERALD framework.

7.2 Published APIs

The *Evaluation* component provides the following three endpoints for starting/stopping an evaluation, listing evaluation results, or creating an evaluation result manually, respectively.

[17] https://git.code.tecnalia.dev/emerald/public/components/evaluation/evaluation/-/blob/master/Dockerfile



Evaluation

POST /v1/evaluation/evaluate/{auditScopeId}/start

POST /v1/evaluation/evaluate/{auditScopeId}/stop

GET /v1/evaluation/results

POST /v1/evaluation/results

Figure 15. Evaluation API endpoints

7.3 Integration Status

Currently, the *Evaluation* component is being deployed on the Kubernetes cluster. The foundation of this integration lies in the code base located in the EMERALD repository for the *Evaluation* component. To facilitate the utilization of this code base, a Dockerfile has been provided, which has undergone modifications to ensure the successful deployment of the *Evaluation*.

The connections between the *Evaluation* component and the *Orchestrator* have been well-tested locally and in other projects since they are part of the Clouditor toolbox. However, their communication must be validated in the EMERALD development cluster, e.g., to check if authentication is correctly configured.

Table 5 provides a quick summary of the current state of the connections that the *Evaluation* component is supposed to interact with.

Table 5. Integration Status of the Evaluation with other EMERALD Components

Component	**Status**	**Comment**
Orchestrator	Tested Locally	Communication needs testing in the EMERALD Kubernetes cluster.

8 Integration of RCM

This section reports on the integration of the *Repository of Controls and Metrics* (*RCM*) into the EMERALD framework.

The *RCM* serves as a smart catalogue of controls and metrics that provides a central resource in the EMERALD framework where the certification schemes are stored and managed. Through the *RCM*, Compliance Managers and Auditors can obtain all the information related to security certification schemes. The *RCM* supports multi-scheme and multi-level certification and incorporates the definition of the metrics used in EMERALD to assess evidence.

The *RCM* will provide mechanisms to update the catalogues and maintain a versioning system and will foster interoperability using OSCAL as exchange format. This feature will allow importing and exporting catalogues into/from the *RCM*. The *RCM* will manage also the mappings generated by *MARI*, which provides (i) a list of mapped controls from different security schemes, and (ii) a list of the metrics mapped for each control.

Regarding its internal design, the *RCM* is implemented as a microservice that offers a REST API for external access. Details of the *RCM* implementation have already been provided in D3.3 [5].

8.1 Contribution to Non-Functional Requirements

The contribution to the relevant WP1 non-functional requirements defined in section 2.5.1 is listed below.

WP1.01 Performant Network: The *RCM* provides a REST API for data exchange. The main data sent to other components is the schema data (controls) and the defined metrics. While the data contained in a call can be large in some cases (when the entire schema is required), these calls are very few and probably made at the beginning of the user workflow. In general, the *EMERALD UI* is the component that request the most data to the *RCM*. In most of the cases the information required is of limited extent, and in other cases it can be paginated for a fast UI refresh.

WP1.02 Portability: The *RCM* is currently dockerized and deployed on Kubernetes. Providing a Dockerfile[18] makes it easier to port the *RCM* to other environments or use it in another container orchestration.

WP1.06 Agile development: As with the rest of the components, the *RCM* code is stored in GitLab and controlled by a GitLab CI pipeline. Also, its deployment details in Kubernetes are defined in the corresponding YAML file, so that when any change is done in the main branch, a new image is built and automatically deployed into the testbed.

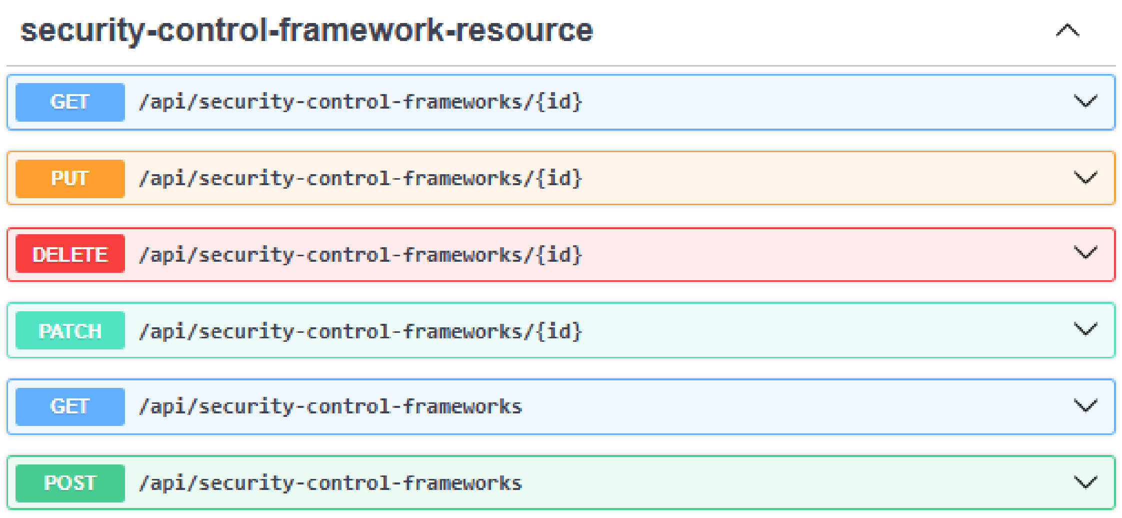
WP1.08 Security: the *RCM* supports OAuth, which is the protocol used in EMERALD for authentication and authorization. Interaction with the Keycloak component will be provided. This way, it only interacts with authenticated components and allows only the interactions (calls) defined for the role of the user logged.

8.2 Published APIs

In the following, we show the principal API endpoints developed so far for the *RCM*. More complete information about the API can be obtained by accessing the *RCM* component with an administrator account.

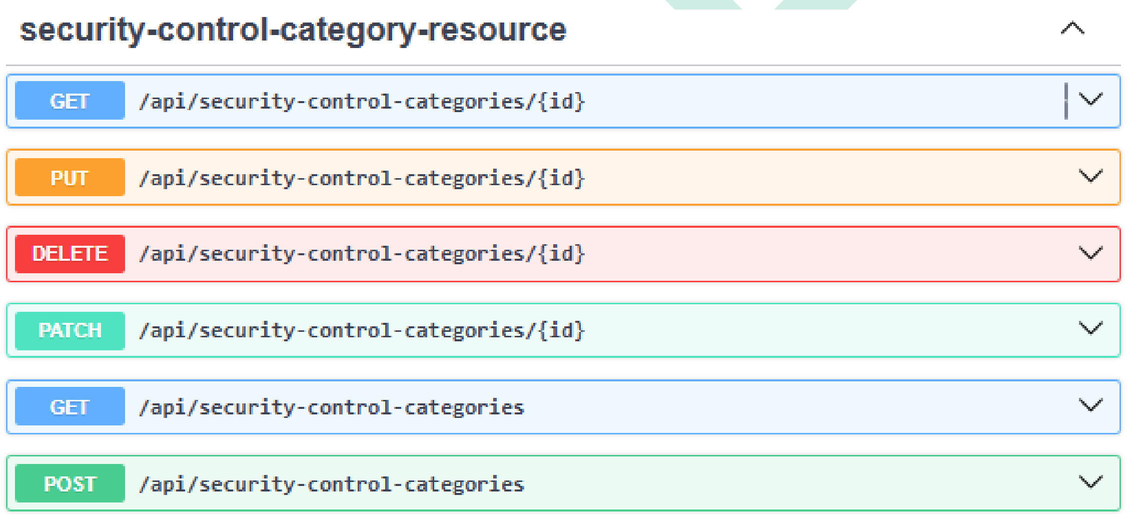
[18] Each *RCM* sub-component has its own Dockerfile until they are all unified into a docker-compose. For example, this is the Dockerfile for the backend component: https://git.code.tecnalia.com/emerald/public/components/rcm/backend/-/blob/master/Dockerfile

API endpoints for getting schema info:


security-control-framework-resource

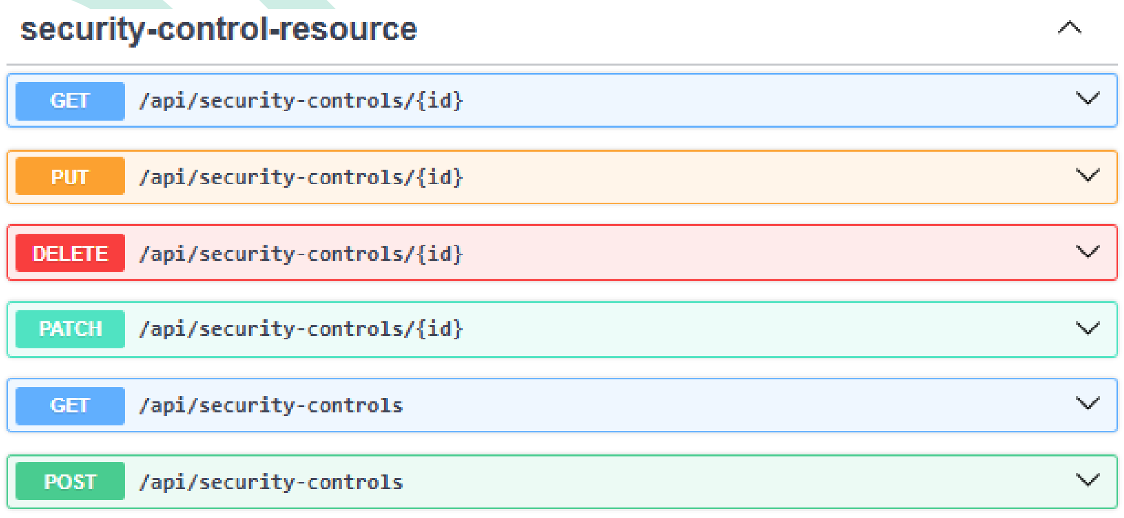
GET /api/security-control-frameworks/{id}
PUT /api/security-control-frameworks/{id}
DELETE /api/security-control-frameworks/{id}
PATCH /api/security-control-frameworks/{id}
GET /api/security-control-frameworks
POST /api/security-control-frameworks

Figure 16. RCM API Endpoints for Security Control Framework Resources


security-control-category-resource

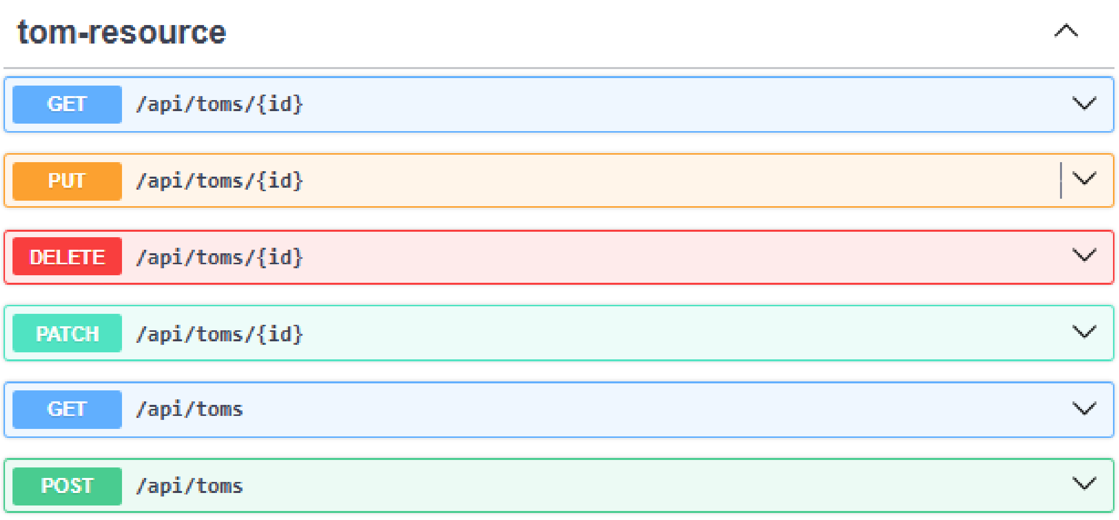
GET /api/security-control-categories/{id}
PUT /api/security-control-categories/{id}
DELETE /api/security-control-categories/{id}
PATCH /api/security-control-categories/{id}
GET /api/security-control-categories
POST /api/security-control-categories

Figure 17. RCM API Endpoints for Security Control Category Resources


security-control-resource

GET /api/security-controls/{id}
PUT /api/security-controls/{id}
DELETE /api/security-controls/{id}
PATCH /api/security-controls/{id}
GET /api/security-controls
POST /api/security-controls

Figure 18. RCM API Endpoints for Security Control Resources

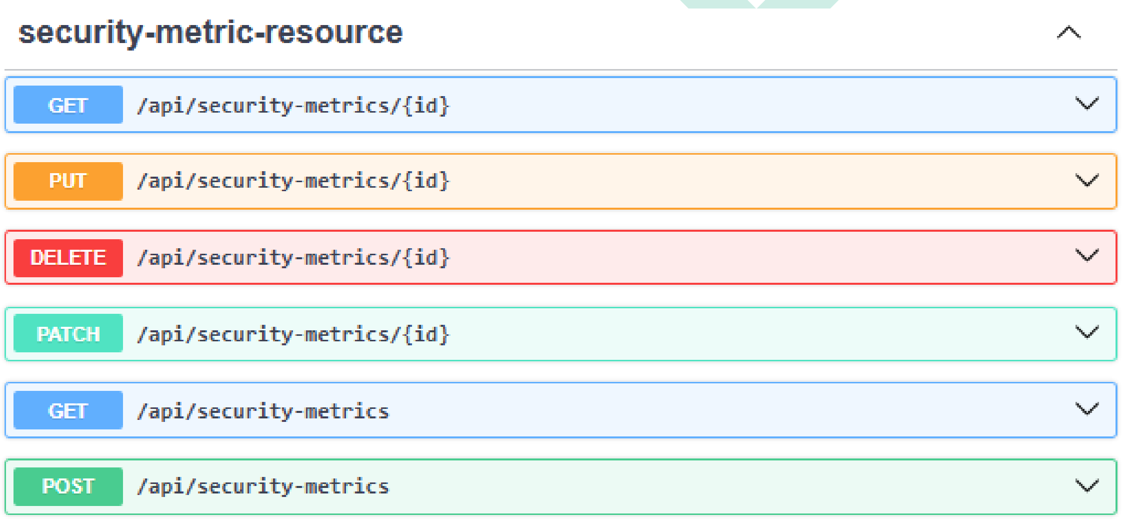


tom-resource

GET /api/toms/{id}
PUT /api/toms/{id}
DELETE /api/toms/{id}
PATCH /api/toms/{id}
GET /api/toms
POST /api/toms

Figure 19. RCM API Endpoints for TOM Resources

API endpoints for getting metrics info:

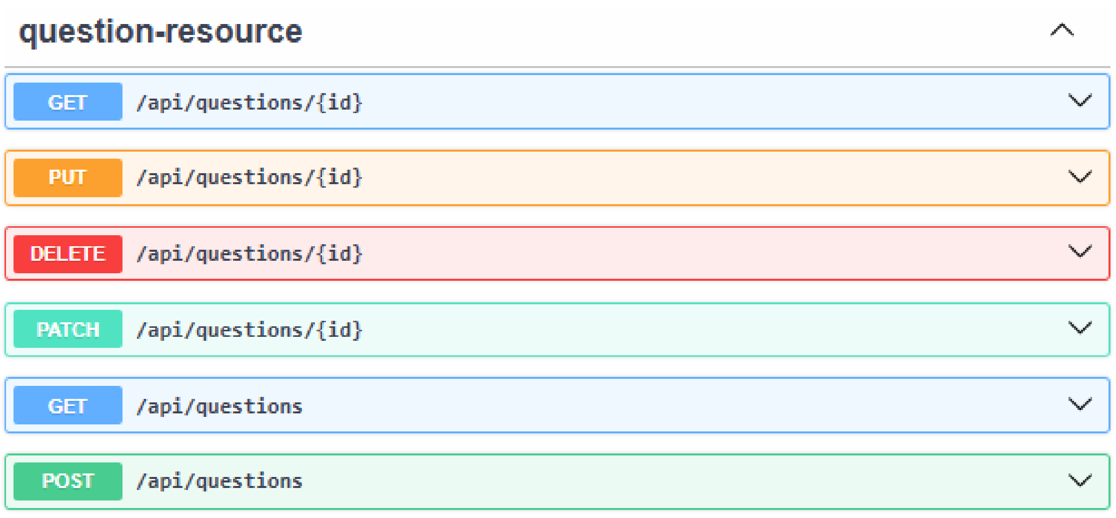


security-metric-resource

GET /api/security-metrics/{id}
PUT /api/security-metrics/{id}
DELETE /api/security-metrics/{id}
PATCH /api/security-metrics/{id}
GET /api/security-metrics
POST /api/security-metrics

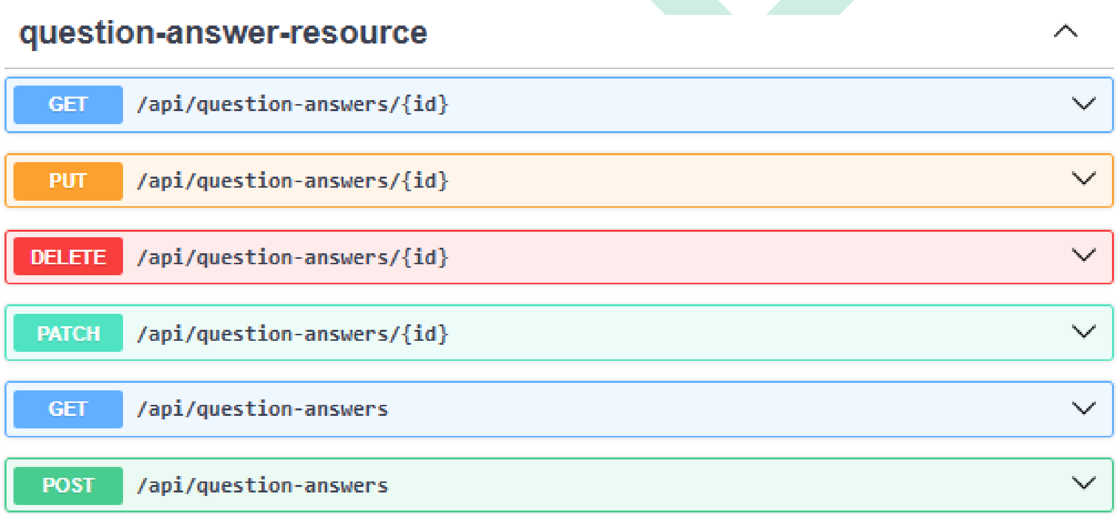
Figure 20. RCM API Endpoints for Security Metric Resources

API endpoints for getting questionnaire info:


question-resource

GET /api/questions/{id}
PUT /api/questions/{id}
DELETE /api/questions/{id}
PATCH /api/questions/{id}
GET /api/questions
POST /api/questions

Figure 21. RCM API Endpoints for Questionnaire Resources


question-answer-resource

GET /api/question-answers/{id}
PUT /api/question-answers/{id}
DELETE /api/question-answers/{id}
PATCH /api/question-answers/{id}
GET /api/question-answers
POST /api/question-answers

Figure 22. RCM API Endpoints for Questionnaire Answer Resources

8.3 Integration Status

The *RCM* has already been already deployed on the Kubernetes cluster. Table 6 provides the current state of connections the *RCM* is supposed to interact with.

Table 6. Integration Status of RCM with other EMERALD components

Component	Status	Comment
EMERALD UI	Developing API/Connected *	Updating / extending the API
Clouditor-Orchestrator	Tested Locally	
MARI	Not Started	API defined
AMOE	Tested Locally	

(*) At the moment, the API is connected with the component-owned UI. As the *EMERALD UI* is still under development, it is expected that some changes to the API will be necessary.

9 Integration of TWS

This section reports about the integration of the *Trustworthiness System (TWS)* into the EMERALD framework. The *TWS* provides trustworthiness, fairness and transparency to the evidence and assessment results stored in EMERALD, guaranteeing the integrity and authenticity of the recorded information. Its main functionality is to allow secure evidence and assessment results proofs of integrity registration and verification processes.

The *TWS* is backboned by a Blockchain network to provide information security features, such as integrity, trustworthiness, and transparency. Initially, it was based on a Quorum[19] network, but Hyperledger Besu[20] has been finally considered due to its interoperability with existing semi-public Blockchain ecosystems such as the European Blockchain Services Infrastructure (EBSI) and Alastria[21]. Due to some internal legal matters, EBSI is not currently onboarding any projects and, consequently, Alastria has been finally considered. A Blockchain Viewer is included to enhance usability and allow non-technicians to use the system. Besides, an automatic evidence verification service has been also considered to improve automatization and facilitate integration in the EMERALD solution, as manual interaction is not required in this way. More details can be found in D3.3 [5].

9.1 Contribution to Non-Functional Requirements

The contribution to the relevant WP1 non-functional requirements defined in section 2.5.1 is listed below.

WP1.01 Performant Network: The *TWS* exposes REST APIs for regular evidence and assessment results proofs of integrity registration from the *Evidence Store* or the evidence collectors. It also facilitates regular or occasional integrity verification operations through the EMERALD UI. In both scenarios, the information exchanged is minimal (these are JSON structures of a few KB). However, the involvement of a Blockchain network usually means lower performance levels. To address this, the Blockchain network is being configured in the most optimal way in terms of performance, by adjusting factors such as consensus algorithms, block sizes, etc. to improve the registration processes. Additionally, users will have access to filters that help reduce the amount of information to be verified on the Blockchain, further optimizing performance.

WP1.02 Portability: The *TWS* is composed of two types of components:

- The semi-public Alastria Blockchain network with the deployed Smart Contracts and the Blockchain Viewer, which are provided as a service for all the EMERALD users.
- The Blockchain client and the verification service, which are locally deployed and have been dockerized[22]. More details about installation and usage can be found in D3.3 [5].

WP1.06 Agile development: GitLab is used for the *TWS* code management and implementation of CI/CD processes. Additionally, YAML files have been created for the *TWS* components to be deployed on the Kubernetes testbeds (Blockchain client and verification service) to automatize redeployments. Every time a change is done in the main branch, a new image is built and deployed on the testbed.

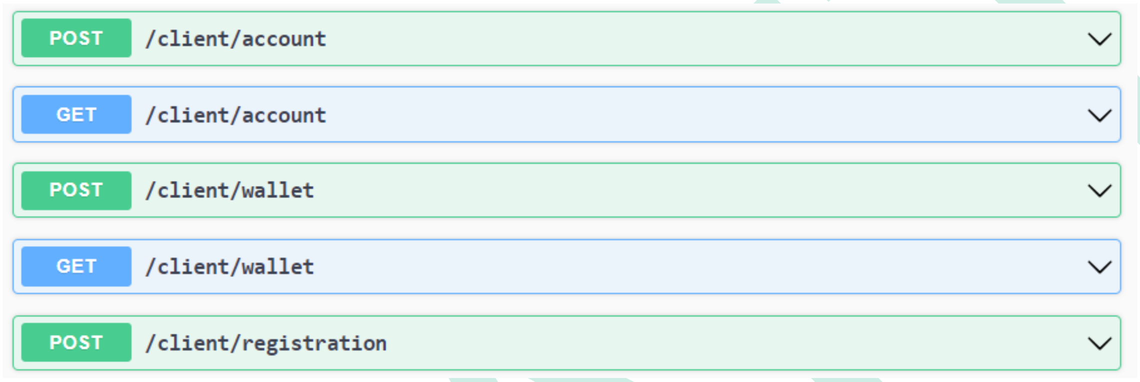
[19] Quorum Blockchain | Build & Deploy Networks Quickly
[20] Welcome | Besu documentation
[21] Plataforma de Blockchain
[22] Please note that Docker images are not publicly available as the *TWS* is a proprietary licensed component ©Tecnalia.

WP1.08 Security: The *TWS* supports Oauth to restrict interactions to authenticated users and components through Keycloak. Additionally, it is deployed on the Kubernetes testbed where security is well managed.

9.2 Published APIs

In the following, we show the current API endpoints that can be used by other components (if they are authenticated). However, we are currently under an updating process due to the migration process from Quorum to Hyperledger Besu (Alastria) and the new requirements that come with it. Updated details will be provided in the following deliverable D3.6.

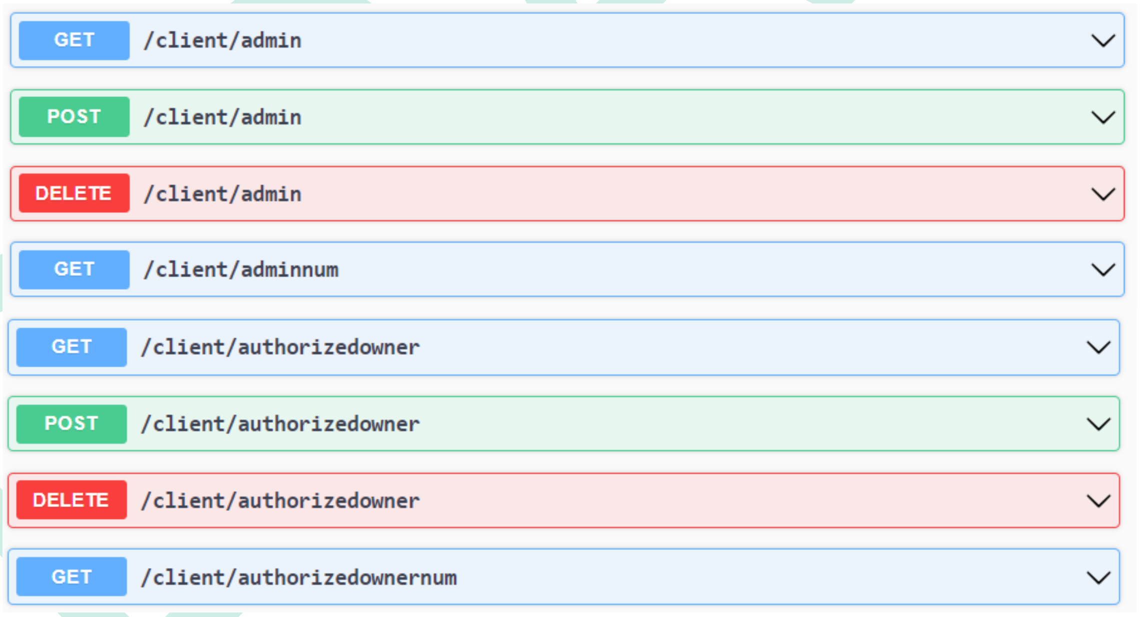
API endpoints for Blockchain account management:



```
POST    /client/account
GET     /client/account
POST    /client/wallet
GET     /client/wallet
POST    /client/registration
```

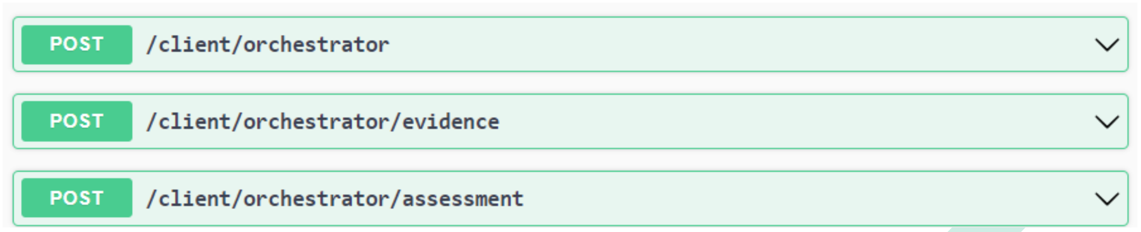
Figure 23. TWS API Endpoints for Account Management

API endpoints for user roles management:



```
GET     /client/admin
POST    /client/admin
DELETE  /client/admin
GET     /client/adminnum
GET     /client/authorizedowner
POST    /client/authorizedowner
DELETE  /client/authorizedowner
GET     /client/authorizedownernum
```

Figure 24. TWS API Endpoints for Roles Management I



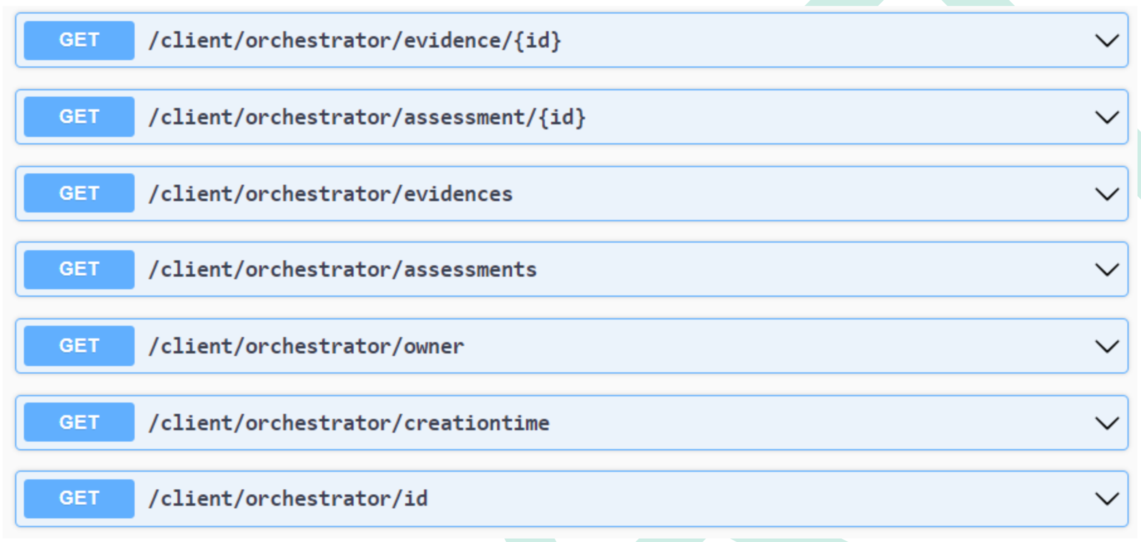
POST /client/orchestrator

POST /client/orchestrator/evidence

POST /client/orchestrator/assessment

Figure 25. TWS API Endpoints for Roles Management II

API endpoints for evidence and assessment results proof of integrity information access:



GET /client/orchestrator/evidence/{id}

GET /client/orchestrator/assessment/{id}

GET /client/orchestrator/evidences

GET /client/orchestrator/assessments

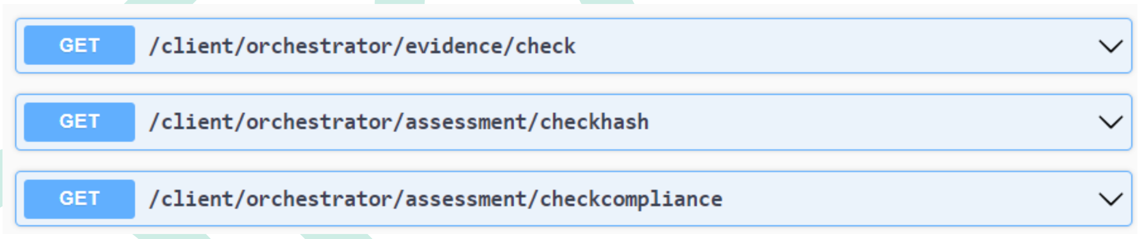
GET /client/orchestrator/owner

GET /client/orchestrator/creationtime

GET /client/orchestrator/id

Figure 26. TWS API Endpoints for Information Access

API endpoints for evidence and assessment results proof of integrity verification:



GET /client/orchestrator/evidence/check

GET /client/orchestrator/assessment/checkhash

GET /client/orchestrator/assessment/checkcompliance

Figure 27. TWS API Endpoints for Integrity Access

9.3 Integration Status

Currently, the *TWS* has been successfully deployed on the Kubernetes cluster. However, it is under an ongoing migration process from the Quorum to the Alastria Blockchain network. This migration has slightly delayed the integration process with other EMERALD components. Table 7 provides the current state of connections the *TWS* is supposed to interact with.

Table 7. Integration Status of TWS with other EMERALD components

Component	Status	Comment
EMERALD UI	Developing API	Currently updating the API details due to the migration process.
Evidence Store	Developing API	
Evidence collectors (Codyze)	Developing API	

10 Conclusions

In this deliverable, we have provided an initial report on the integration of the WP3 components within the EMERALD framework. This includes detailed integration strategies, contributions to non-functional requirements, published APIs, and the current integration status for each component: *Clouditor-Orchestrator*, *Clouditor-Assessment*, *Clouditor-Evidence Store*, *Clouditor-Evaluation*, *Mapping Assistant for Regulations with Intelligence* (*MARI*), *Repository of Controls and Metrics* (*RCM*), and *Trustworthiness System* (*TWS*).

The primary objective of this deliverable is to document the interim integration status of the WP3 components, ensuring that they effectively interact and function cohesively within the EMERALD system. This integration is crucial for supporting the development of a Certification-as-a-Service (CaaS) framework for continuous certification of harmonized cybersecurity schemes.

Key contributions of this deliverable include the integration of the WP3 components, supporting the fulfilment of the key results such as CERTGRAPH (KR2), OPTIMA (KR3), MULTICERT (KR4), and INTEROP (KR7).

Looking ahead, the next steps will involve refining and enhancing the integration of the WP3 components, culminating in the final versions of the concepts and integrations outlined in the subsequent deliverables (D3.2, D3.4, and D3.6). These efforts will ensure continuous improvement and alignment with the project's overarching objectives, ultimately strengthening the robustness and effectiveness of the EMERALD framework.

11 References

[1] EMERALD Consortium, "EMERALD - Annex 1- Description of Action - GA101120688," 2022.

[2] EMERALD Consortium, "D1.5 DevOps methodology and CI/CD strategy for EMERALD-v1," 2024.

[3] EMERALD Consortium, "D1.3 EMERALD solution architecture-v1," 2024.

[4] EMERALD Consortium, "D3.1 Evidence assessment and Certification–Concepts-v1," 2024.

[5] EMERALD Consortium, "D3.3 Evidence assessment and Certification–Implementation-v1," 2024.

[6] MEDINA Consortium, "D5.5 – MEDINA integrated solution-v3," 2023.

[7] EMERALD Consortium, "D2.8 Runtime Evidence Extractor – v1," 2024.

[8] EMERALD Consortium, "D2.4 AMOE - v1," 2024.

[9] EMERALD Consortium, "D2.2 Source Evidence Extractor – v1," 2024.

[10] EMERALD Consortium, "D2.6 ML Model Certification – v1," 2024.