# Deliverable D3.6

# Evidence assessment and Certification – Integration - v2

| | |
|---|---|
| **Editor(s):** | Nico Haas |
| **Responsible Partner:** | Fraunhofer AISEC |
| **Status-Version:** | Final – v1.0 |
| **Date:** | 30.01.2026 |
| **Type:** | OTHER |
| **Distribution level:** | PU |

| | |
|---|---|
| **Project Number:** | 101120688 |
| **Project Title:** | EMERALD |

| | |
|---|---|
| **Title of Deliverable:** | Evidence assessment and Certification-Integration-v2 |
| **Due Date of Delivery to the EC** | 31.01.2026 |

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

| | |
|---|---|
| **Abstract:** | Final 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. |

DRAFT

# Document Description

| Version | Date | Modifications Introduced | |
|---|---|---|---|
| | | Modification Reason | Modified by |
| v0.1 | 17.12.2025 | TOC | Nico Haas (FHG) |
| v0.2 | 16.01.2026 | First draft | Nico Haas (FHG)<br>Cristina Regueiro, Iñaki Etxaniz (TECNALIA)<br>Marinella Petrocchi, John Bianchi (CNR) |
| v0.3 | 23.01.2026 | Internal QA Review | Michela Fazzolari (CNR) |
| v0.4 | 26.01.2026 | Address internal QA Review | Nico Haas (FHG) |
| v0.5 | 28.01.2026 | Final reviewed version | Cristina Martínez, Juncal Alonso (TECNALIA) |
| v1.0 | 30.01.2026 | Submitted to the European Commission | Cristina Martínez, Juncal Alonso (TECNALIA) |

DRAFT


CC BY-SA

# Table of contents




(cc) BY-SA

DRAFT

# List of tables



# List of figures




CC BY-SA

DRAFT


(cc) BY-SA

# Terms and abbreviations

| | |
|---|---|
| ABE | Attribute-Based Encryption |
| AI | Artificial Intelligence |
| AI-SEC | AI Security Evidence Collector |
| 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 |
| gRPC | Google Remote Procedure Calls |
| GUI | Graphical User Interface |
| IaC | Infrastructure as Code |
| JSON | JavaScript Object Notation |
| KB | Kilobyte |
| KR | Key Result |
| MARI | Mapping Assistant for Regulations with Intelligence |
| NLP | Natural Language Processing |
| OAuth | open authorization |
| OSCAL | Open Security Controls Assessment Language |
| PR | Pull Request |
| RCM | Repository of Controls and Metrics |
| SDLC | Software Development Life Cycle |
| SSI | Self-Sovereign Identity |
| TWS | Trustworthiness System |
| WSGI | Web Server Gateway Interface |
| YAML | Yet Another Markup Language |


(cc) BY-SA

# Executive Summary

This deliverable, the final version of "Evidence Assessment and Certification – Integration", provides the 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 results: CERTGRAPH (KR2) by implementing the evidence store allowing graph-based queries, 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 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] [3] and established several requirements for components and the entire framework, focusing on non-functional aspects [4] [5]. 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 the pilots.

This deliverable, D3.6, is the final WP3 integration deliverable within the EMERALD project. D3.1 and D3.2 completed the theoretical phase [6] [7], D3.3 and D3.4 documented the implementation of the components [8] [9], and D3.5 presented the first integrated version [10]. D3.6 now reports the final integration status of the WP3 components and their readiness for the pilot phase.


(cc) BY-SA

# 1 Introduction

This introduction outlines the purpose and structure of Deliverable D3.6, which reports on the final integration of WP3 components within the EMERALD framework. This introduction outlines the purpose and structure of Deliverable D3.6, which reports on the final integration of WP3 components within the EMERALD framework. It provides details about the deliverable in Section 1.1, describes the document's structure in Section 1.2, explains the advances from the MEDINA project in Section 1.3 and summarises the updates introduced in this deliverable compared to D3.5 in Section 1.4.

## 1.1 About this deliverable

The EMERALD project is dedicated to establishing a Certification-as-a-Service (CaaS) framework that facilitates the continuous certification of cybersecurity schemes, including the European Cybersecurity Certification Scheme for Cloud Services (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 plays a crucial role, as it integrates the evidence collection and knowledge extraction tools developed in 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, D3.6, focuses on the final integration of WP3 components within the EMERALD framework. It aims to detail the 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 adopted in WP3 and the integration status of the WP3 components.

## 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 established 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.

Sections 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 application programming interfaces (APIs), and interactions with other components.

Finally, Section 10 provides the conclusion, summarizing the key findings and outlining the next steps for refinement of the EMERALD framework during the pilot phase.


(cc) BY-SA

## 1.3 Technological advances from the MEDINA project

EMERALD builds upon the outcomes of the MEDINA project[1]: *Security framework to achieve a continuous audit-based certification in compliance with the EU-wide cloud security certification scheme* (GA 952633). While MEDINA only reached TRL5, the goal of EMERALD is to enhance the system and elevate its TRL from prototype (TRL5) to product (TRL7). The advances from the MEDINA project of the WP3 components which integration is described in this deliverable have been described in D3.4 [9] section 1.3.

## 1.4 Updates from D3.5

The deliverable builds upon D3.5 [10] and provides an updated version of that document. It includes sections of content from D3.5 that have not changed, alongside newly added information. To help track the updates and enhancements introduced since the previous document, Table 1 outlines the modifications and new additions for each section of the document.

*Table 1. Overview of the Changes to D3.5*

| Section | Changes |
|---|---|
| **1 Introduction** | Minor text changes due to the fact that this deliverable is D3.6 and not D3.5. Technological advances from MEDINA have been added |
| **2 Integration Strategy** | Minor changes |
| **3 Integration of Clouditor-Orchestrator** | We updated the figures for the API endpoints, since starting from D3.2 the term “target of evaluation” is used instead of “certification target” or “cloud service”.<br>Table 2 has been updated; the status of each connection to other components is now “ integrated”. |
| **4 Integration of Clouditor-Assessment** | Table 3 has been updated; the status of each connection to other components is now “ integrated”. |
| **5 Integration of Clouditor-Evidence Store** | Table 4 has been updated; the status of each connection to other components is “integrated”. |
| **6 Integration of MARI** | Table 5 has been updated; the status of the connection to the *RCM* component is “integrated”. |
| **7 Integration of Clouditor-Evaluation** | Table 6 has been updated; the status of the single connection to the *Orchestrator* is now “ integrated”. |
| **8 Integration of RCM** | Mapping interaction with *MARI* and management of results in the *RCM* database are included in the Published APIs section.<br>The Integration Status has been updated, being the *MARI* and AMOE APIs completely Integrated.<br>The interaction with the Public Metrics Repository has been included in the Published APIs and Integration Status sections. |
| **9 Integration of Trustworthiness System** | Table 8 has been updated; the status of the connection to EMERALD UI is now “Testing” and the others are now fully “integrated”. The extension of the *TWS* based on ABE and SSI has been detailed. |

[1] https://medina-project.eu/

# 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 [4]. 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 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] [4]. 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 progress of each component's integration that has been achieved during the course of the project.

## 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[2] platform is easier (vSphere is a cloud computing virtualization platform from VMware[3]). For the creation and configuration of the cluster we use a set of OpenTofu[4] and Ansible[5] scripts that can be easily adjusted for use on other nodes.

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


(cc) BY-SA
DRAFT

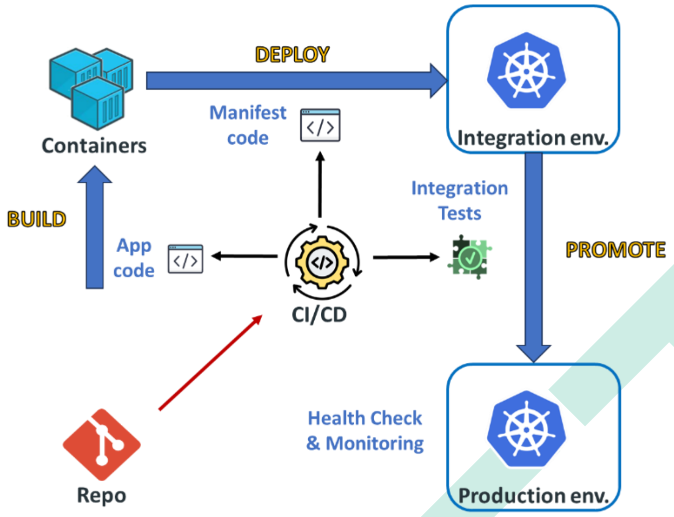


*Figure 1. Integration and Production environments in the EMERALD CaaS framework*

## 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, as well as 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 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[6] 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 integration of all WP3 components into the EMERALD framework has been completed. The current focus is on their stable deployment in the integration and production environments and on supporting their use in the pilots, including more comprehensive tests conducted jointly by the different WPs and the component owners.

[6] https://docs.docker.com/compose/


BY-SA

## 2.4 Requirements for WP3 Integration

In this section, we outline the requirements for WP3 integration, derived from the work completed in WP1 [4]. 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 specified and addressed in previous WP3 concept and implementation deliverables [7] [9]. However, since the primary focus of this deliverable is on the integration of WP3 components into the entire EMERALD framework, we will consider a subset (highlighted in the following list) of the eight non-functional requirements elicited in WP1, which were defined in D1.4 [5]:

- **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 WP3 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.


CC BY-SA

  - **Integration automation**. A *GitLab Agent* for *Kubernetes* monitors the framework repository and allows 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[7] has been 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.

## 2.5 Component-specific Integration Status

The following sections present the integration status of each WP3 component at the end of WP3. 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 sections provides information on its contribution to the following four non-functional requirements outlined by WP1:

- **WP1.01 Performant Network:** This section describes 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 more advanced automated integration tests for realistic end-to-end scenarios are being added during the pilot phase, the WP3 components have been deployed in the integration and production environments, and their current focus is on supporting stable operation and pilot usage. In this deliverable, we report the integration status of all WP3 components at the end of WP3.
- **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

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


CC BY-SA

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.
- **Locally Tested**: 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.

DRAFT


CC BY-SA

# 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 information about the contribution to relevant non-functional requirements defined in WP1, the API endpoints and the 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 (Google Remote Procedure Calls) 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[8] at the root of the component in the GitLab repository. For more information about installation and usage, see D3.4 [9]. Providing a Dockerfile enables easy use of the *Orchestrator* in other business environments that rely on different CI/CD pipeline technologies or container orchestration solutions.

**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 made 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. Additional integration tests are planned in the context of the pilot deployments.

**WP1.08 Security:** Like all Clouditor components, the *Orchestrator* supports OAuth[9] to only allow interaction with the components that are defined in Keycloak. Therefore, it refuses connections to/from components that are not authorized, e.g. from components outside the cluster. By adopting this zero-trust approach, security is strengthened throughout the framework.

## 3.2 Published APIs

The following figures show the *Orchestrator* API endpoints that other components can use (provided they are authenticated). For more details, see the `openapi.yaml` file in the Clouditor repository[10].

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


(cc) BY-SA
DRAFT

Figure 2 illustrates the *Orchestrator* API endpoints for handling assessment results and tools.

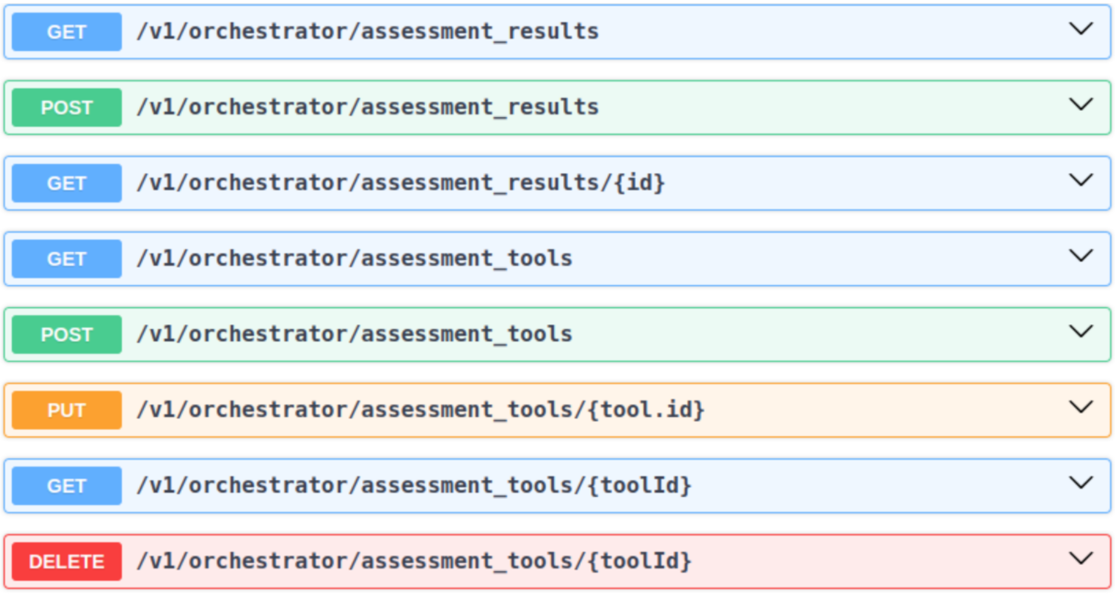


*Figure 2. Orchestrator API endpoints for assessment results and tools*

Figure 3 illustrates the *Orchestrator* API endpoints for handling metrics.

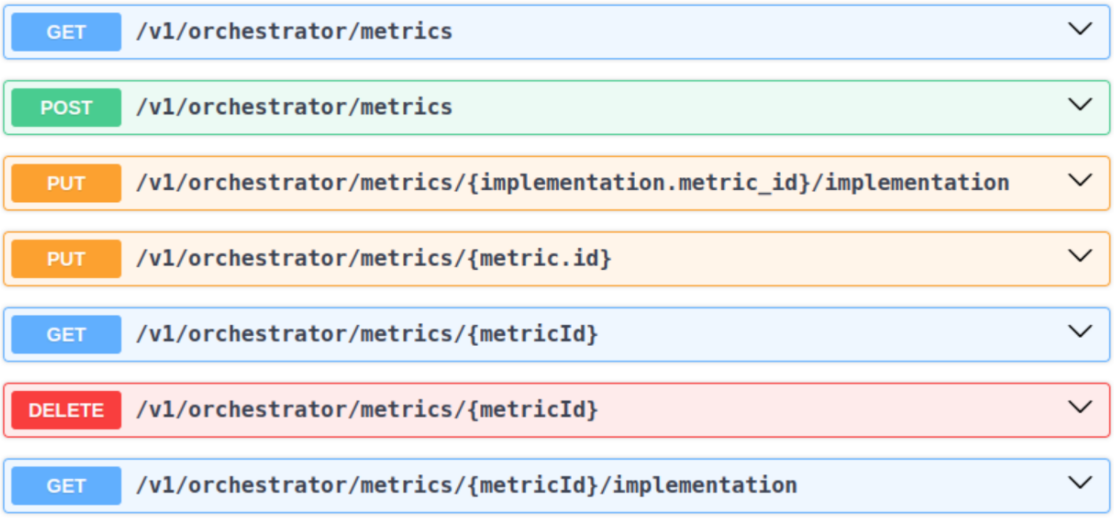


*Figure 3. Orchestrator API endpoints for metrics*


(cc) BY-SA

Figure 4 illustrates the *Orchestrator* API endpoints for handling targets of evaluation.

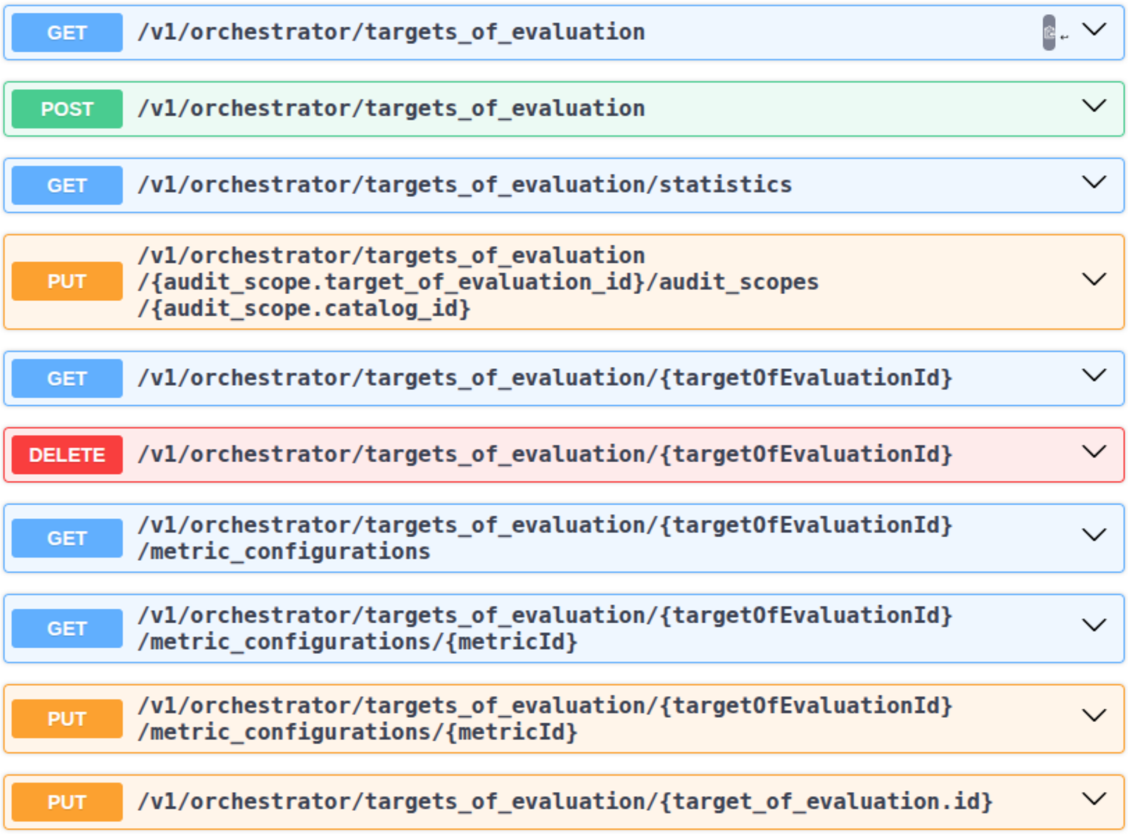


*Figure 4. Orchestrator API endpoints for targets of evaluation*

Figure 5 illustrates the *Orchestrator* API endpoints for handling certificates.

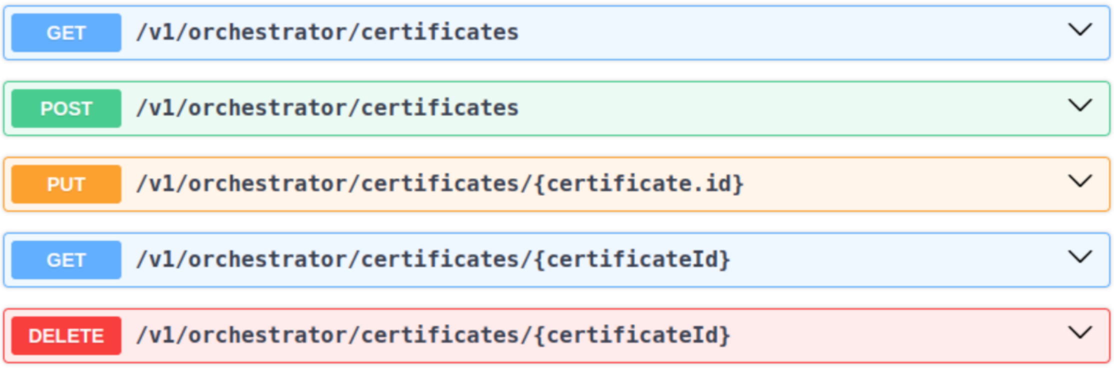


*Figure 5. Orchestrator API endpoints for certificates*

Figure 6 illustrates the *Orchestrator* API endpoint for listing all target certificates without state history. This endpoint is publicly available by design, as it only exposes certificates that have been issued and explicitly marked as publicly visible by the certificate owner, and does not return any sensitive evidence, assessment results, or internal identifiers.


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


(cc) BY-SA

Figure 7 illustrates the *Orchestrator* API endpoints for handling catalogues.

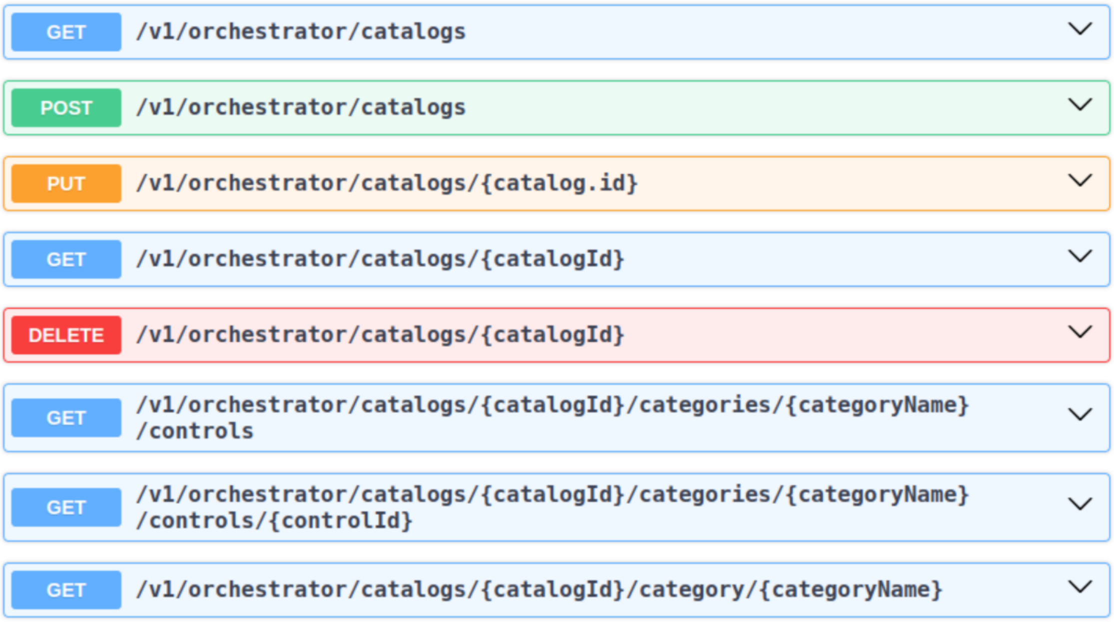


*Figure 7. Orchestrator API endpoints for catalogues*

Figure 8 illustrates the *Orchestrator* API endpoints for handling audit scopes.

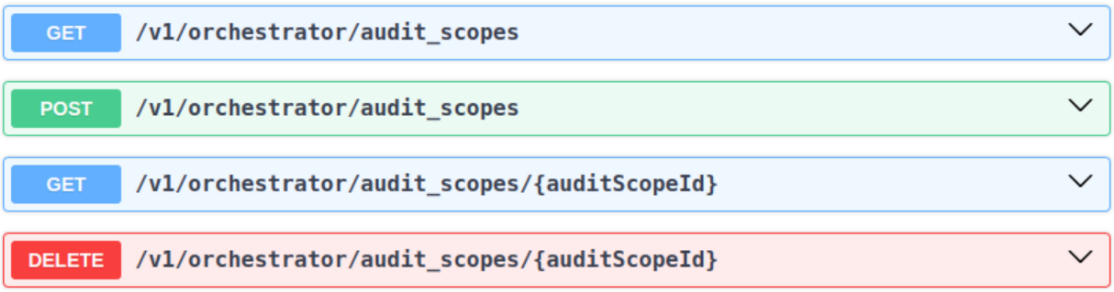


*Figure 8. Orchestrator API endpoints for audit scopes*

## 3.3 Integration Status

The *Orchestrator* has been successfully deployed on the Kubernetes cluster. The foundation of this integration lies in the codebase located in the EMERALD repository for the *Orchestrator*[11]. To facilitate the utilization of this codebase, a Dockerfile was provided in deliverable D3.4 [9], which has been modified 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. Using this UI, the different personas (e.g. the auditor) can retrieve specific information that is received from the *Orchestrator* through these API endpoints, such as showing the controls relevant for a given target of evaluation. Although these API endpoints are final for this

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


(cc) BY-SA

deliverable, minor changes can occur in the future. The possible need for changes would result from the pilot testing phase, which will continue until the end of the project.

The *Evaluation*, *Assessment*, and *Evidence Store* components have been well-tested locally and in other projects since they are part of the Clouditor toolbox. In addition, their communication has been validated in the EMERALD development cluster. The integration tests involved workflows that start with creating evidence in the evidence extractors, forwarding this evidence to the central *Evidence Store*, which stores the evidence and then sends it to the *Assessment* component. The resulting assessment results are forwarded to the *Evaluation* component, where they are aggregated and finally sent back to the *Orchestrator* for the final certificate decision.

Regarding the *RCM*, the API has been completed, and the components are connected. However, the connection requires further testing in the pilot testing phase, and some refinements may be needed after these tests.

Table 2 provides a quick summary of the state of the connections that the *Orchestrator* interacts with.

*Table 2. Integration Status of Orchestrator with other EMERALD Components*

| **Component** | **Status** | **Comment** |
|---|---|---|
| *EMERALD UI* | Integrated* | The EMERALD UI is integrated and can use the main *Orchestrator* endpoints. This integration will be further exercised and refined during the pilot phase, where feedback from real pilot usage may lead to adjustments. |
| *Evaluation* | Integrated | Minor changes may occur in the pilot testing phase. |
| *RCM* | Integrated | Minor changes may occur in the pilot testing phase. |
| *Assessment* | Integrated | Minor changes may occur in the pilot testing phase. |
| *Evidence Store* | Integrated | Minor changes may occur in the pilot testing phase. |

(*) The API is connected with the *EMERALD UI*. Since this component is still under development, some changes in the *Orchestrator* and its API might still be necessary.


BY-SA

# 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 [9]. 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:** a Dockerfile[12] 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 by 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, while more advanced automated integration tests for realistic end-to-end scenarios are being added and exercised in the context of the pilot deployments.

**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 (see Figure 9).

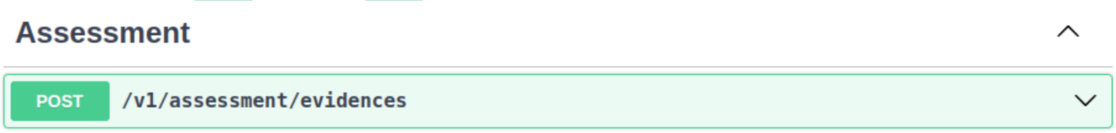


*Figure 9. Assessment API endpoint for evidence*

## 4.3 Integration Status

The *Assessment* component has been deployed in the Kubernetes cluster. The foundation of this integration lies in the codebase located in the EMERALD repository for the *Assessment* component. To facilitate the utilization of this codebase, a Dockerfile was provided in deliverable

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


CC BY-SA

D3.4 [9], which has been modified to ensure the successful deployment 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 has been well-tested locally and in other projects, since they are part of the Clouditor toolbox. In addition, their integration has been validated in the EMERALD cluster.

The *Assessment* is connected to *TWS* (sending assessment results). This connection has been tested with example assessment results. Further testing will be done in the pilot testing phase.

Table 3 provides a quick summary of the status of the connections that the *Assessment* component interacts with.

*Table 3. Integration Status of Assessment with other EMERALD Components*

| Component | Status | Comment |
| --- | --- | --- |
| *Orchestrator* | Integrated | Minor changes may occur in the pilot testing phase. |
| *Evidence Store* | Integrated | Minor changes may occur in the pilot testing phase. |
| *TWS* | Integrated | Minor changes may occur in the pilot testing phase. |

DRAFT


(cc) BY-SA

# 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 extractors, storing it in a database that allows graph-based queries, and forwarding evidence to the *Assessment* as well as to the *TWS* to enhance the integrity of the evidence.

We provide 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 extractors, 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 extractors, accommodating different usage scenarios. For instance, *Clouditor-Discovery* utilizes gRPC [11], while the evidence extractor *AMOE* (*Assessment and Management of Organisational Evidence*) will use the REST API endpoints [12]. 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.

**WP1.02 Portability:** A Dockerfile[13] 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 code robustness, while more advanced automated integration tests for realistic end-to-end scenarios are being added and exercised in the context of the pilot deployments.

**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.

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


(cc) BY-SA

## 5.2 Published APIs

The *Evidence Store* provides the three endpoints for storing evidence, listing all evidence and getting specific evidence, respectively (see Figure 10).

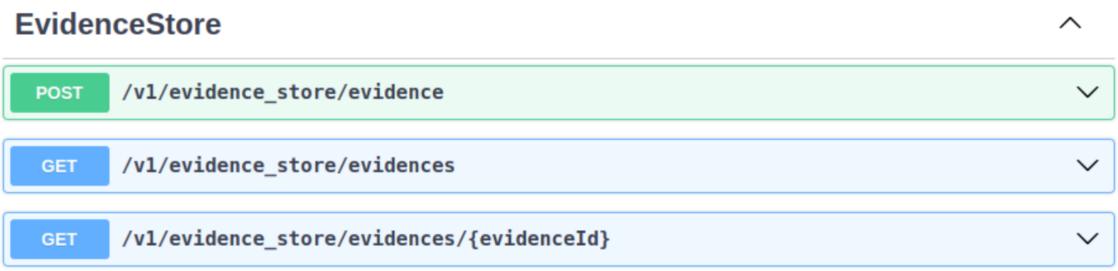


*Figure 10. Evidence Store API endpoints*

## 5.3 Integration Status

The *Evidence Store* has been deployed in the Kubernetes cluster. The foundation of this integration lies in the codebase located in the EMERALD repository for the *Evidence Store*. To facilitate the use of this codebase, a Dockerfile was provided in deliverable D3.4 [9], 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 the *Clouditor-Discovery* [13] have been well-tested locally and in other projects, since they are part of the Clouditor toolbox. In addition, their communication has been validated in the EMERALD development cluster. The other evidence extractors, i.e. *AMOE* [14], *Codyze* [15], *eknows-e3* [15], and *AI-SEC* [16], are integrated as well. New metrics are still being developed, i.e. new evidence will be created as well. Nevertheless, changes that may occur in this process will be minor and will not affect the connection in general.

Table 4 provides a quick summary of the status of the connections that the *Evidence Store* is supposed to interact with.

*Table 4. Integration Status of Evidence Store with other EMERALD Components*

| Component | Status | Comment |
|---|---|---|
| *Assessment* | Integrated | Minor changes may occur in the pilot testing phase. |
| *Clouditor-Discovery* | Integrated | Minor changes may occur in the pilot testing phase. |
| *AMOE* | Integrated | Further metrics and, thus, new evidence are still being developed. Minor changes may occur. |
| *Codyze* | Integrated | Further metrics and, thus, new evidence are still being developed. Minor changes may occur. |
| *eknows-e3* | Integrated | Further metrics and, thus, new evidence are still being developed. Minor changes may occur. |
| *AI-SEC* | Integrated | Further metrics and, thus, new evidence are still being developed. Minor changes may occur. |

DRAFT


(cc) BY-SA

# 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* utilizes the all-MiniLM-L6-v2 sentence transformer model[14] to generate embeddings. These allow *MARI* to effectively represent the semantic meaning of controls and metrics in a vector space. Associations are then computed using cosine similarity, allowing results to be filtered based on a configurable similarity threshold.

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 D3.4 [9].

## 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[15] in 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, separated into distinct controllers and services, combined with containerization via Docker, provides a solid framework for Agile development. This structure enables rapid iteration and seamless updates within the Kubernetes environment. The application is designed to be easily integrated into CI/CD pipelines (e.g., GitLab CI), automating the build and deployment processes for both integration and production environments, while the modular codebase facilitates the execution of unit and integration tests to ensure stability.

[14] *all-MiniLM-L6-v2*: lightweight sentence embedding model for semantic similarity tasks. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[15] https://git.code.tecnalia.dev/emerald/public/components/mari/mari/-/blob/master/Dockerfile?ref_type=heads

**WP1.08 Security:** Secure interactions are fully enforced through the integration with Keycloak, ensuring that only authenticated and authorized components can access the framework. The system adopts a zero-trust model, where the API operates within a protected Kubernetes cluster. This configuration effectively blocks unauthorized connections—such as those from components outside the trusted network—thereby guaranteeing the framework's overall security and data integrity.

## 6.2 Published APIs

The logic of mapping controls and metrics is fully implemented and deployed. The APIs supports two types of mappings: *metric-to-control* and *control-to-control*.

The *metric-to-control* mapping links metrics to controls of a scheme. To focus on the most relevant matches, the input includes a threshold parameter that defines the minimum similarity score required for a mapping to be returned (see Figure 11). The output provides a mapping where each item incorporates the similarity (%) value (see Figure 12).

**Input (Example):**

```
{
  "similarity_threshold": 50,
  "controls": [
    {
      "id": "controlId1",
      "framework_id": "schema1",
      "category_id": "cat1",
      "description_title":      "Access      control
policy...",
      "requirements": []
    },
    {
      "id": "controlId2",
      "framework_id": "schema1",
      "category_id": "cat2",
      "description_title":             "Cryptography
controls...",
      "requirements": []
    }
  ],
  "metrics": [
    {
      "Id": "metricId1",
      "description": "Percentage of users..."
    },
    {
      "Id": "metricId2",
      "description": "Number of incidents..."
    }
  ]
}
```

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

The following is an example of the JSON response returned by the API, where each control is associated with a list of metrics and their relative similarity scores.

**Output (Example):**


```
[
 {
   "control_id": "controlId1",
   "framework_id": "schema1",
   "category_id": "cat1",
   "requirement_id": null,
   "metrics_mapped": [
     "metricId1;85",
     "metricId2;60"
   ]
 },
 {
   "control_id": "controlId2",
   "framework_id": "schema1",
   "category_id": "cat2",
   "requirement_id": null,
   "metrics_mapped": []
 }
]
```


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

The *control-to-control* mapping, which maps controls between two different schemes, follows a similar structure, accepting *schemaA*, *schemaB*, and a *similarity_threshold* (see Figure 13).

**Input (Example):**


```
{
 "similarity_threshold": 60,
 "schemaA": {
   "schema_id": "1",
   "schema_name": "schema1",
   "controls": [
     {
       "id": "controlId11",
       "description_title":    "Information    Security
Policy..."
     }
   ]
 },
 "schemaB": {
   "schema_id": "2",
   "schema_name": "schema2",
   "controls": [
     {
       "id": "controlId12",
       "description_title": "Security Planning..."
     },
     {
       "id": "controlId33",
       "description_title": "Policy maintenance..."
     }
   ]
```



CC BY-SA

```
  }
}
```

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

The resulting output provides the matched controls from the target schema along with their calculated similarity percentages:

**Output (Example):**

```
[
  {
    "control_id": "controlId11",
    "mapped_controls": [
      "controlId12;90",
      "controlId33;75"
    ]
  }
]
```

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

The API endpoints described above are fully implemented and operational. The core logic for semantic mapping is handled directly by the application services defined in the source code, processing requests in real-time. The application, built with Flask[16], is served via Gunicorn[17]. This robust, production-grade Web Server Gateway Interface (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[18].

The component is fully containerized and integrated into the Kubernetes cluster, utilizing native liveness and readiness probes for automated orchestration and high availability. Furthermore, access to the API is secured via Keycloak, which enforces strict authentication and authorization policies, ensuring that only verified components can interact with the service in a zero-trust environment.

Figure 15 shows the *MARI* API endpoints that other components can use for semantic mapping (if they are authenticated). These two endpoints allow for linking metrics to controls and mapping controls between different schemes, respectively.

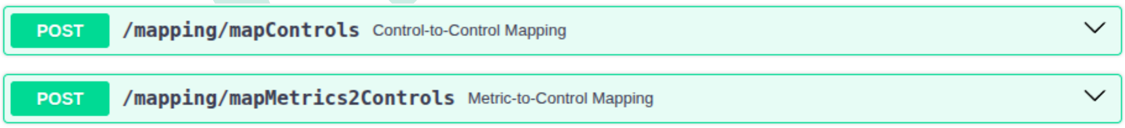


Figure 15. *MARI API endpoints for mapping*

[16] https://flask.palletsprojects.com/en/stable/
[17] https://gunicorn.org/
[18] https://flask.palletsprojects.com/en/stable/deploying/gunicorn/


CC BY-SA

## 6.3 Integration Status

*MARI* is fully deployed on the Kubernetes cluster and interacts with the *RCM* through the defined REST API. The API has been published and is currently operational and secured by Keycloak authentication. Table 5 provides a quick summary of the status of the connections that *MARI* interacts with.

*Table 5. Integration Status of MARI with other EMERALD Components*

| Component | Status | Comment |
|---|---|---|
| *RCM* | Integrated | API fully implemented, deployed on Kubernetes, and secured via Keycloak. |

DRAFT


(cc) BY-SA

# 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 information on the contributions of the *Evaluation* component to relevant non-functional requirements defined in WP1, the published API endpoints, and the 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, as described in more detail in Deliverable D3.4 [9].

## 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[19] 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 Currently, we leverage unit tests to enhance code robustness, while more advanced automated integration tests for realistic end-to-end scenarios are being added and exercised in the context of the pilot deployments.

**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 four endpoints for starting/stopping an evaluation, listing evaluation results, or creating an evaluation result manually, respectively (see Figure 16).

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


CC BY-SA

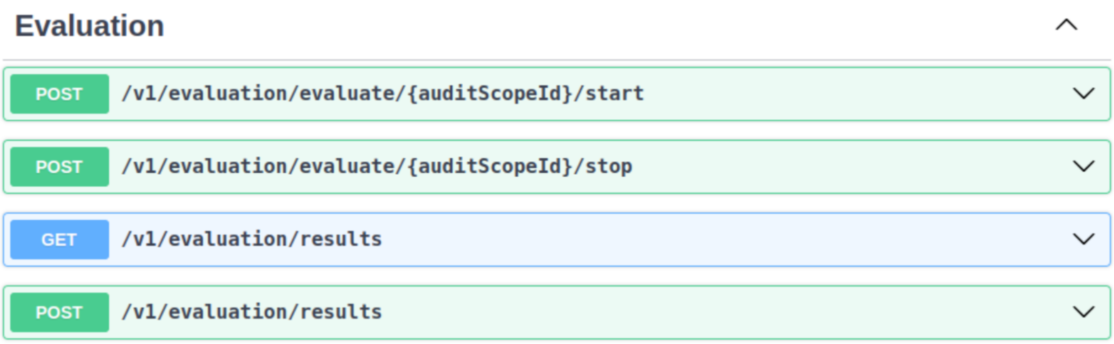


*Figure 16. Evaluation API endpoints*

## 7.3 Integration Status

The *Evaluation* component has been 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. In addition, their communication has been validated in the EMERALD development cluster.

Table 6 provides a quick summary of the state of the connections that the *Evaluation* component is supposed to interact with.

*Table 6. Integration Status of the Evaluation with other EMERALD Components*

| Component | Status | Comment |
|---|---|---|
| *Orchestrator* | Integrated | Minor changes may occur in the pilot testing phase. |

DRAFT


(cc) BY-SA

# 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 the Open Security Controls Assessment Language (OSCAL) as exchange format. This feature will allow importing and exporting catalogues into/from the *RCM*. The *RCM* will also manage 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 D3.4 [9].

## 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 scheme data (controls) and the defined metrics. While the data contained in a call can be large in some cases (when the entire scheme is required), these calls are very few and are made at the beginning of the user workflow. In general, the *EMERALD UI* is the component that requests the most data from the *RCM* and the information it requires is generally either of limited extent or can be paginated for a fast UI refresh.

**WP1.02 Portability**: The *RCM* is currently dockerized and deployed on Kubernetes. Providing a Dockerfile[20] makes it easier to port the *RCM* to other environments or to 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 (Yet Another Markup Language) files, so that when any change is made 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 is provided, so that the *RCM* can only interact with authenticated components and only allows the interactions (calls) defined for the logged-in user's role.

[20] 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.dev/emerald/public/components/rcm/backend/-/blob/master/Dockerfile


(cc) BY-SA

## 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.

Figure 17 illustrates the *RCM* API endpoints for the Security Control Framework.

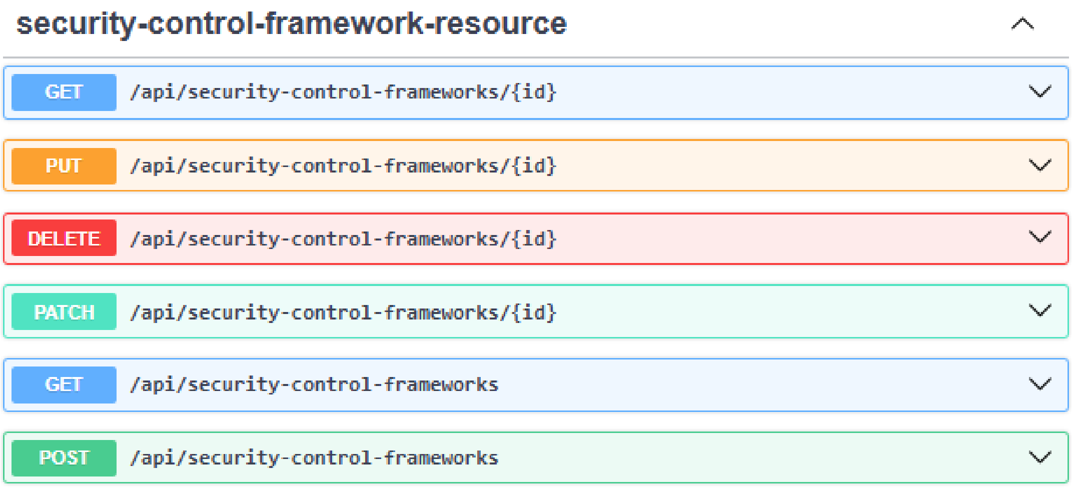


*Figure 17. RCM API Endpoints for Security Control Framework*

Figure 18 illustrates the *RCM* API endpoints for security controls.

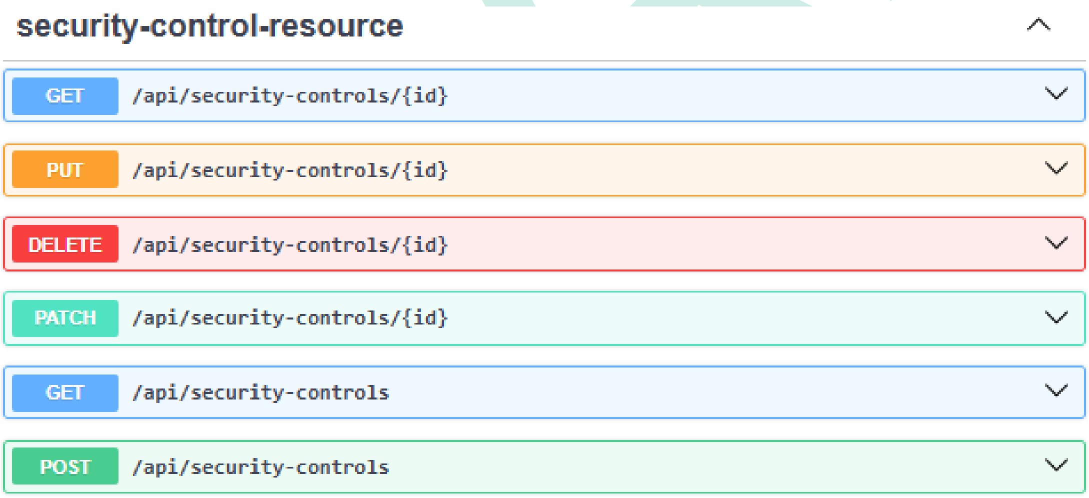


*Figure 18. RCM API Endpoints for Security Control*

Figure 19 illustrates the *RCM* API endpoints for security metrics.


(cc) BY-SA

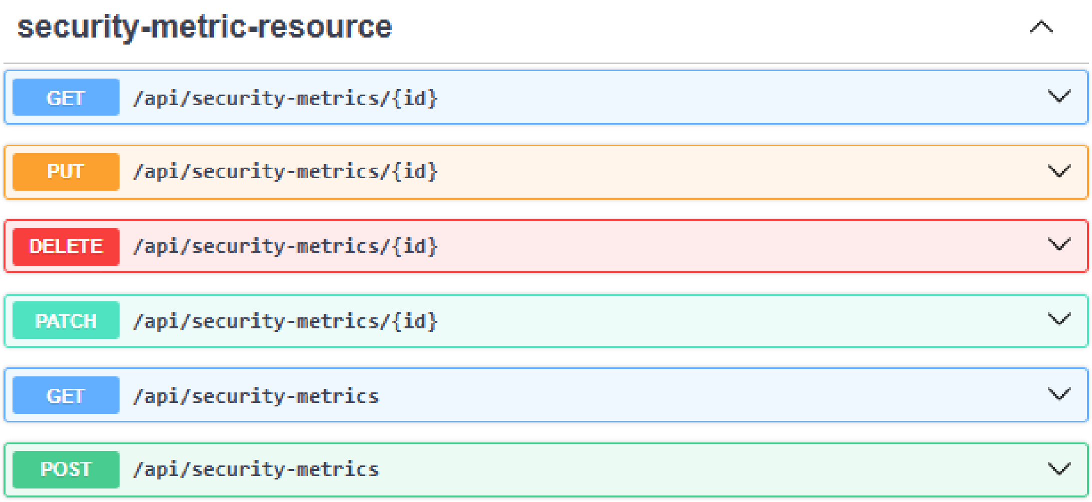


*Figure 19. RCM API Endpoints for Security Metrics*

Figure 20 and Figure 21 illustrate the *RCM* API endpoints for managing the mappings calling to the *MARI* component for getting control-to-control and metrics-to-control mapping info.

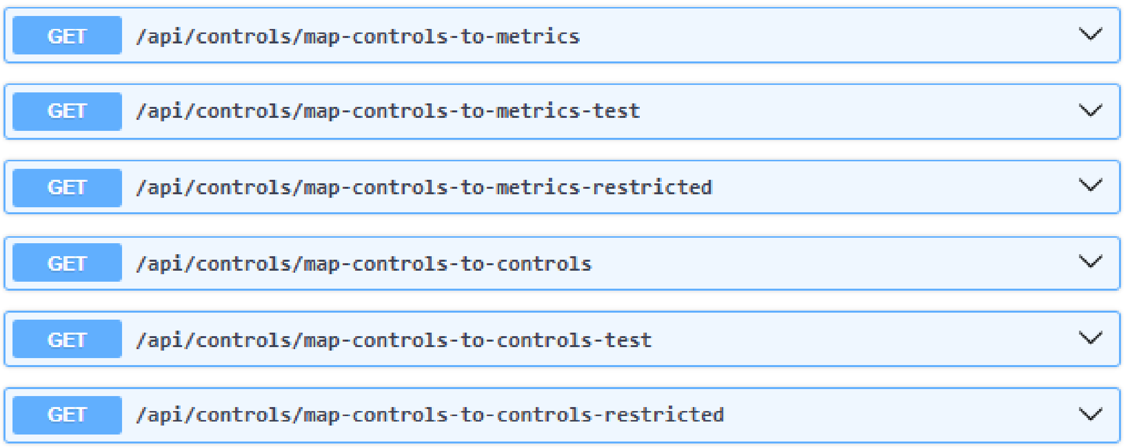


*Figure 20. RCM API Endpoints for Mapping management with the MARI component*

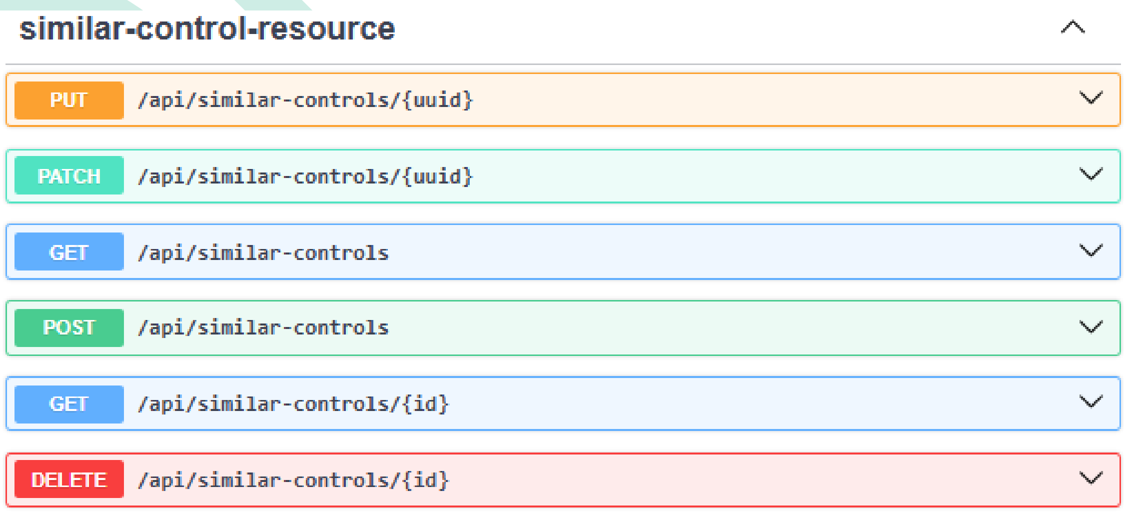


*Figure 21. RCM API Endpoints for similar controls*

Figure 22 illustrates the *RCM* API endpoints regarding security metrics.


(cc) BY-SA

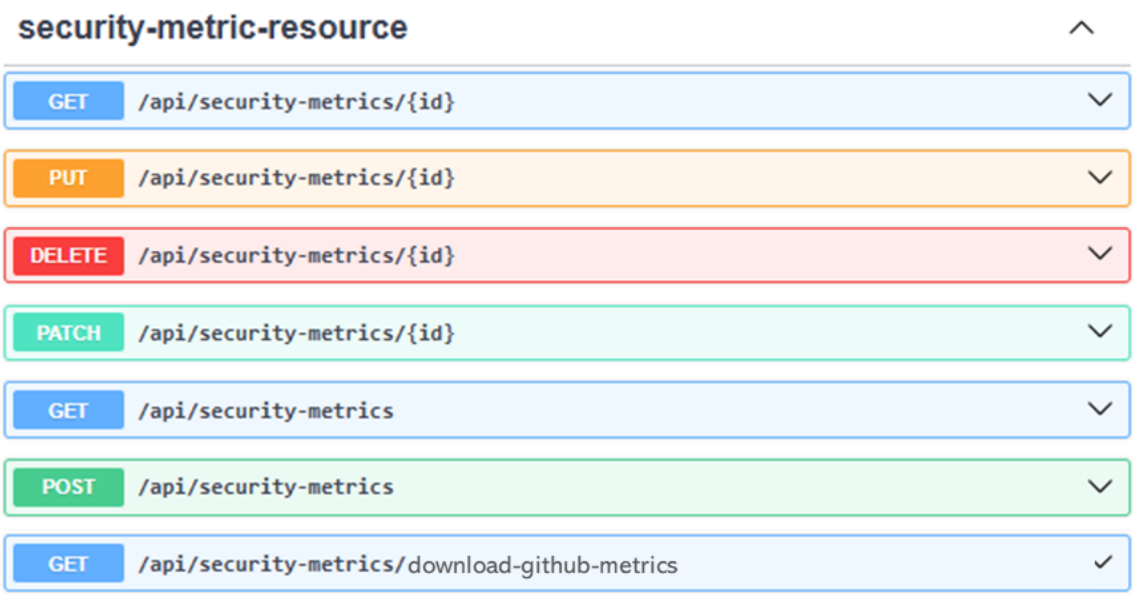


*Figure 22. RCM API Endpoints for security metrics*

API endpoints for managing questionnaire info are illustrated in Figure 23 and Figure 24.

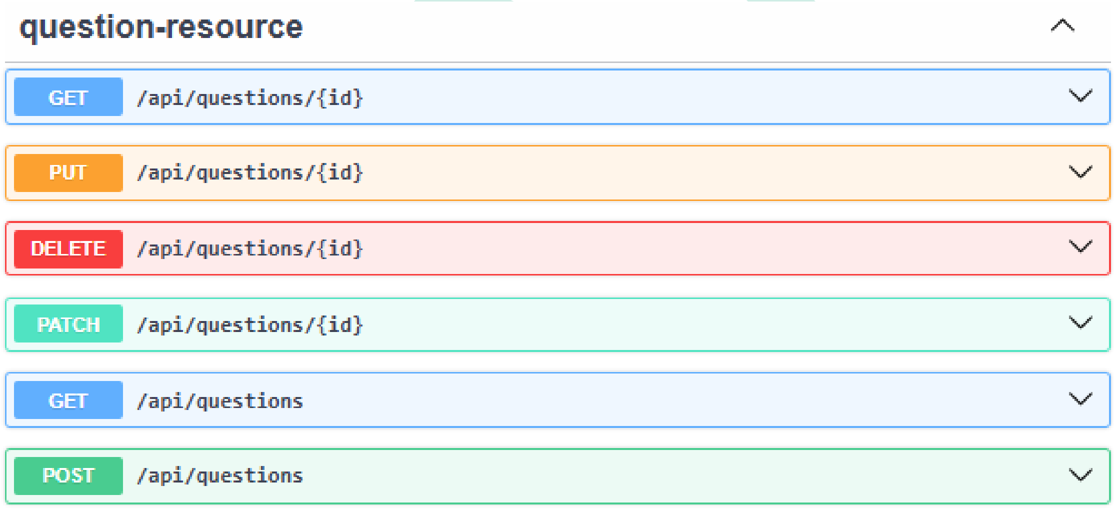


*Figure 23. RCM API Endpoints for questionnaire resources*


(cc) BY-SA

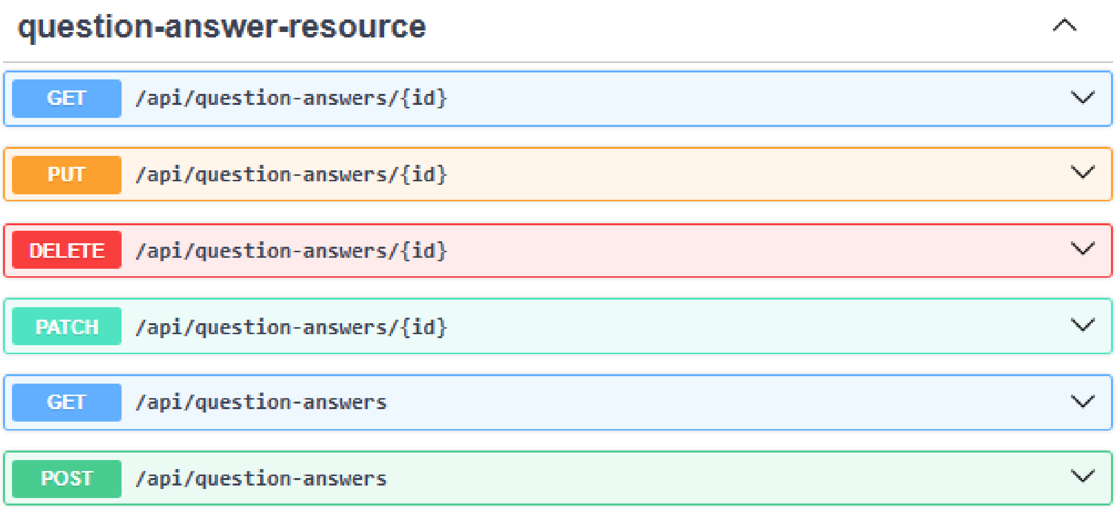


*Figure 24. RCM API Endpoints for questionnaire answer resources*

## 8.3 Integration Status

The *RCM* has already been deployed on the Kubernetes cluster. Table 7 provides the current status of connections the *RCM* interacts with.

*Table 7. Integration Status of RCM with other EMERALD components*

| Component | Status | Comment |
|---|---|---|
| *EMERALD UI* | Connected* | Updating and extending the API |
| *Clouditor-Orchestrator* | Connected* | New interactions have been added for the questionnaire |
| *MARI* | Integrated | Some improvements pending |
| *AMOE* | Integrated | |
| *Collaborative Metrics Repository* | Connected | The API needs to be integrated into the EMERALD UI |

(*) The API is connected with the *EMERALD UI and the Orchestrator*. Since these components are still under development, some changes in the *RCM* and its API might still be necessary.

### 8.3.1 Integration with the Collaborative Metrics Repository

Introduced in the deliverable D3.4 [9], the *Collaborative Metrics Repository* is a joint GitHub repository[21] created by EMERALD and contributed by several parallel Horizon-Europe projects – pursuing comparable certification objectives and therefore needing identical metrics– under the umbrella of the European Cluster for Cybersecurity Certification (EC3)[22]. There, the metrics are defined and maintained collaboratively.

This joint approach turns metric authoring into a collaborative, cross-project activity, and ensures that every component within EMERALD and within related Horizon-Europe projects *operates* on an identical, quality-controlled metric set.

The *RCM* component has been provided with the capability of reading this repository, so that is able to download the metrics defined there. The basic idea is to pull the metrics in an initial configuration phase to complete the *RCM* setup. Afterwards, the pulling can be done by the user

[21] https://github.com/Cybersecurity-Certification-Hub
[22] https://cybersecuritycertcluster.eu/


CC BY-SA

to update the set of metrics. Initially, the *RCM* provided an API endpoint to download all the metrics in the repository, that are organized in folders that represent categories.

Afterwards, we implemented a mechanism to distinguish the EMERALD metrics among all the metrics in the repository (some could have been defined by other teams and not intended for use in EMERALD). We decided to use a label in the pull request (PR) to support this. A new API endpoint has been defined in the *RCM* to allow this operation. It is able to read any specified label and download only the intended metrics. Basically, it makes use of the GitHub API to:

1. Scan all the metric categories (folders).
2. Detect which metrics have been defined in the EMERALD project (identifying first the last commit and then its associated pull request, which should contain the "EMERALD" label).
3. Update the *RCM* database accordingly.

When downloading a metric from the GitHub repository, and depending on the actual content of the *RCM* database, different situations can occur:

- **New metric**: the metric is included in the *RCM*. It could be a new version of an existing metric, as metrics are versioned , i.e., for all intents and purposes, it is treated as a new metric in the EMERALD CaaS framework.
- **Updated metric**: the metric properties are updated.
- **Missing metric** (i.e., an existing metric is not present in the downloaded set): the existing metric is not removed for the *RCM*.

DRAFT


(cc) BY-SA

# 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[23] network, but Hyperledger Besu[24] has been finally considered due to its interoperability with existing semi-public Blockchain ecosystems such as the European Blockchain Services Infrastructure (EBSI) and Alastria[25]. Due to some internal legal matters, EBSI is not currently onboarding any projects and, consequently, Alastria has been finally considered.

The *TWS* includes a Blockchain Viewer to enhance usability and allow non-technicians to use the system. Besides, an automatic evidence verification service has also been considered to improve automation and facilitate integration into the EMERALD framework, as this way no manual interaction is required. More details can be found in D3.4 [9].

## 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 registration of evidence and assessment results proofs of integrity from the *Evidence Store* or the evidence extractors. 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 (including the verification service), which is locally deployed and has been dockerized[26]. More details about installation and usage can be found in D3.4 [9].

**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) to automatize redeployments. Every time a change is done in the main branch, a new image is built and deployed on the testbed.

[23] https://www.kaleido.io/blockchain-platform/quorum
[24] https://besu.hyperledger.org/
[25] https://alastria.io/
[26] Please note that Docker images are not publicly available as the *TWS* is a proprietary licensed component ©Tecnalia.


(cc) BY-SA

**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 final API endpoints that can be used by other components (if they are authenticated).

Figure 25 illustrates the *TWS* API endpoints for administrators.

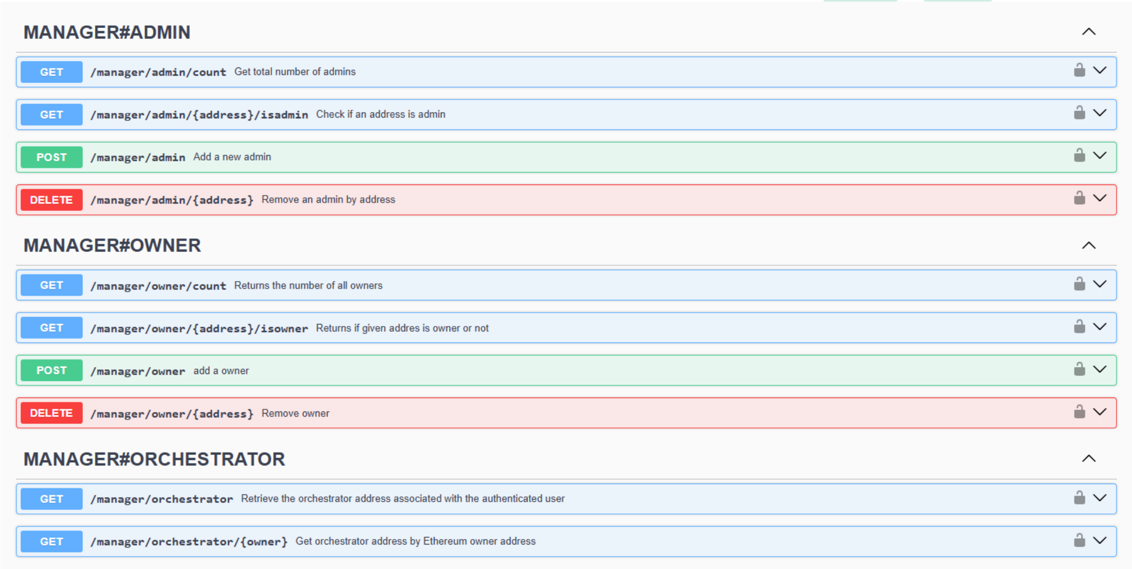


*Figure 25. TWS API Endpoints for administrators*

Figure 26 illustrates the *TWS* API endpoints for the *Orchestrator* or *Assessment* component management (this could be also applicable to other evidence sources).

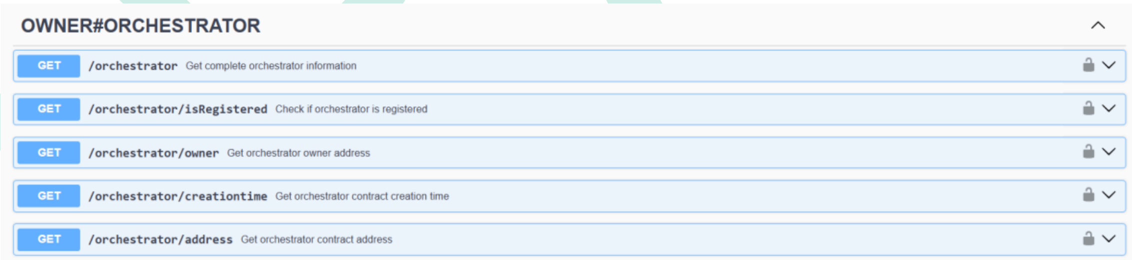


*Figure 26. TWS API Endpoints for assessment management*

Figure 27 illustrates the *TWS* API endpoints for evidence management (record, access and verification).


CC BY-SA

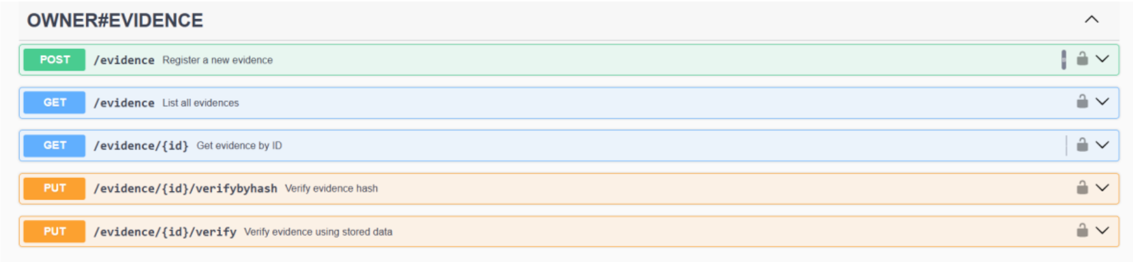


*Figure 27. TWS API Endpoints for evidence management*

Figure 28 illustrates the *TWS* API endpoints for assessment results management (record, access and verification).


*Figure 28. TWS API Endpoints for assessment results management*

## 9.3 Integration Status

The final version of the *TWS* has been successfully deployed on the Kubernetes cluster. Table 8 provides the current state of connections the *TWS* is supposed to interact with.

*Table 8. Integration Status of TWS with other EMERALD components*

| Component | Status | Comment |
|---|---|---|
| *EMERALD UI* | Testing* | Testing on-going |
| *Evidence Store* | Integrated | |
| *Evidence extractors (Codyze)* | Integrated | |

(*) Since the *EMERALD UI* is still under development, some changes in the API might be necessary.

## 9.4 Extension with encrypted data

As it has been introduced in the deliverable D3.4 [9], a proof of concept for an extended *TWS* that allows direct registration of evidence and assessment results in a secure and private way has been developed. The main objective is to record the information (evidence and assessment results) in the *TWS* in an encrypted way to avoid confidential information disclosure. This extension was identified when the architecture, data models, and operational flows had already been described in EMERALD. Therefore, it has been designed as a future extension for the *TWS*, but it will not be integrated during the EMERALD project's lifetime, as it would involve significant design changes in the information flow and data models that would affect several components. This extension is therefore documented as a separate part of the integration of the *TWS* with the other EMERALD components.


CC BY-SA

This extension considers **Attribute-Based Encryption (ABE)** [17] as the encryption methodology for evidence and assessment results registered in the *TWS*. **Self-Sovereign Identity (SSI)** [18] is also considered as a proof of the attributes required in the ABE operation.

### 9.4.1 Background

Attribute-based Encryption (ABE) is a cryptographic system that encrypts the data in a way that users who have a set of identifying non-unique information, also referred to as attributes, can decrypt it. ABE achieves this by making use of bilinear curve cryptography, which allows for the use of various private keys to decrypt data that has been encrypted through a specific public key. ABE systems make use of access policies, access structures that dictate which set of attributes is valid. The access policy separates the different attributes with logical operators (AND, OR) to declare which combination of the attributes is acceptable. Figure 29 shows an example of an access policy considering three different attributes (organization, project and division).

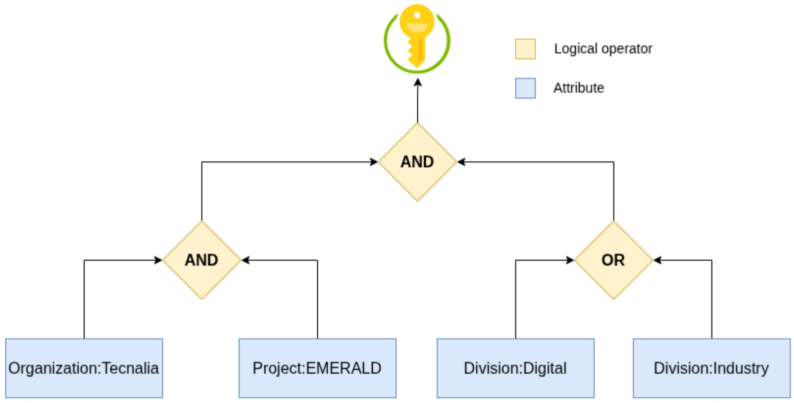


*Figure 29. TWS* Example of an access policy

ABE schemes are separated into two groups based on their usage of the access policy. Key-Policy Attribute-Based Encryption (KP-ABE) uses the access policy for decryption key generation and a set of attributes for data encryption. Conversely, Ciphertext-Policy Attribute-Based Encryption (CP-ABE) uses the attributes of the user for decryption key generation and access policy for data encryption [19].

Usually, ABE systems rely on a trusted Attribute Authority (AA) for facilitating the decryption keys to users, so that non-authorized users cannot generate untruthful decryption keys to gain unauthorized access to encrypted data. Different ABE designs [20] discuss the usage of multiple AAs to avoid bottlenecks and dependency over a single entity. However, the coordination between these AAs needs to be clarified and designed beforehand.

The attributes of a user must be associated with an identity that can be validated. Self-Sovereign Identity (SSI) is a technology that allows the users to present their identity through claims and, at the same time, associate them with already existing identities like Ethereum addresses. Verifiable Credentials (VC), the main elements used in SSI for identity presentation, have elements called "properties", which can be treated as attributes, allowing an identity verifier to act as the Attribute Authority required in ABE ecosystems and generate attribute decryption keys through the credentials of the user.

Verifiable credentials also contain data associated with the validity of the credential. This data specifies aspects like the date from which the credential is valid or the expiration date.

Revocation or suspension of the credential can also be indexed so that it can be consulted by an identity verifier.

### 9.4.2 Extended TWS architecture

As already introduced, ABE is an advanced cryptographic paradigm that enables fine-grained, policy-based access control over encrypted information. Unlike traditional public-key encryption, where data is encrypted for specific users, ABE associates both ciphertexts and decryption keys with descriptive attributes. Access to the encrypted data is granted only if the attributes of a user's key satisfy the logical access policy embedded within the ciphertext. This allows encryption to be directly tied to user properties such as roles, departments, or security clearance levels, rather than to individual identities (i.e., only users whose job is to be an auditor could access confidential evidence). However, in a basic ABE system, there is no guarantee that the user (person or system) who receives an attribute decryption key will not use it once the attribute to which it is associated is no longer valid. ABE has no mechanisms to check whether an attribute is valid or not at a specific time. This raises the following question: ***How are we sure that the user does not use a decryption key that should not be valid to access encrypted data?***

The solution to this question involves the combination of ABE with SSI. The proposed solution assumes that the used ABE scheme is a CP-ABE scheme, due to its higher compatibility with SSI credentials compared with KP-ABE. Figure 30 shows the architecture of the proposed solution.

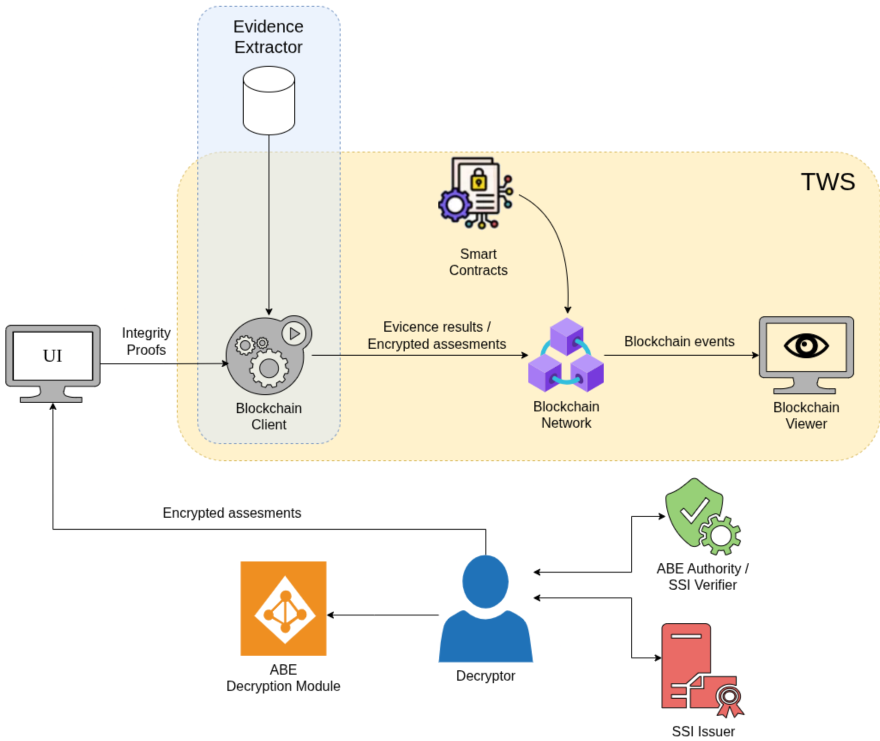


*Figure 30. Proposed architecture for the TWS extension with encrypted data*

Some of the involved components in the extension are common to the initial *TWS* implementation:

- **EmeraldUI**: An extended automatic verification service is integrated into the EMERALD UI, allowing auditors to easily verify the trustworthiness of evidence and assessment results. The extended version allows the access, decryption and verification of evidence and assessment results.


(cc) BY-SA

- **Evidence extractors and *Assessment* component**: The same components as in the initial *TWS* implementation are considered to provide evidence and assessment results through a Blockchain client.
- **Blockchain client**: An extended Blockchain client is needed to allow the direct registration of encrypted evidence and assessment results (instead of their hashes).
- **Blockchain Network**: The same Blockchain network as in the initial *TWS* implementation is considered.
- **Smart Contracts**: Extended smart contracts for the auditing functionalities of the *TWS* are needed to be deployed on the Blockchain network. These Smart Contracts handle the registration of data (encrypted evidence results and encrypted assessments) on the Blockchain and its consumption for integrity validation (decryption and verification). They also generate Blockchain-based events to feed the Blockchain viewer and allow user-friendly access to the information recorded on the *TWS*.
- **Blockchain Viewer**: The same Blockchain Viewer as in the initial *TWS* implementation is considered. In this case, the evidence and assessment results raw information will be visualized in an encrypted way.

However, some new components are needed in the proposed extension with encrypted data:

- **Decryptor**: The user who wants to access the encrypted information (evidence or assessment results). It acts as an ABE user and an SSI Holder, having a wallet with their own verifiable credentials.
- **ABE Decryption Module**: This is the module with the software to execute the different ABE operations for the Decryptor.
- **ABE Authority**: The ABE Attribute Authority that generates the decryption keys. For attribute verification, it acts as an SSI verifier and generates the attributes based on the verifiable credentials handed to them by the Decryptor.
- **SSI Issuer**: The SSI issuer that creates and emits the credentials for the user.

The proposal ingrains the data associated with revocation and time related validity of the credentials used for the credential creation inside the decryption key. The key will also have the signature of the Attribute Authority that created the key to certify that the revocation information has not been tampered with. A process that checks the validity of the revocation data and its contents before executing the decryption process has been designed to ensure that the data is only decrypted by users whose attributes are still valid. Figure 31 shows the sequence diagram of the operation of the extension with encrypted data.


(cc) BY-SA

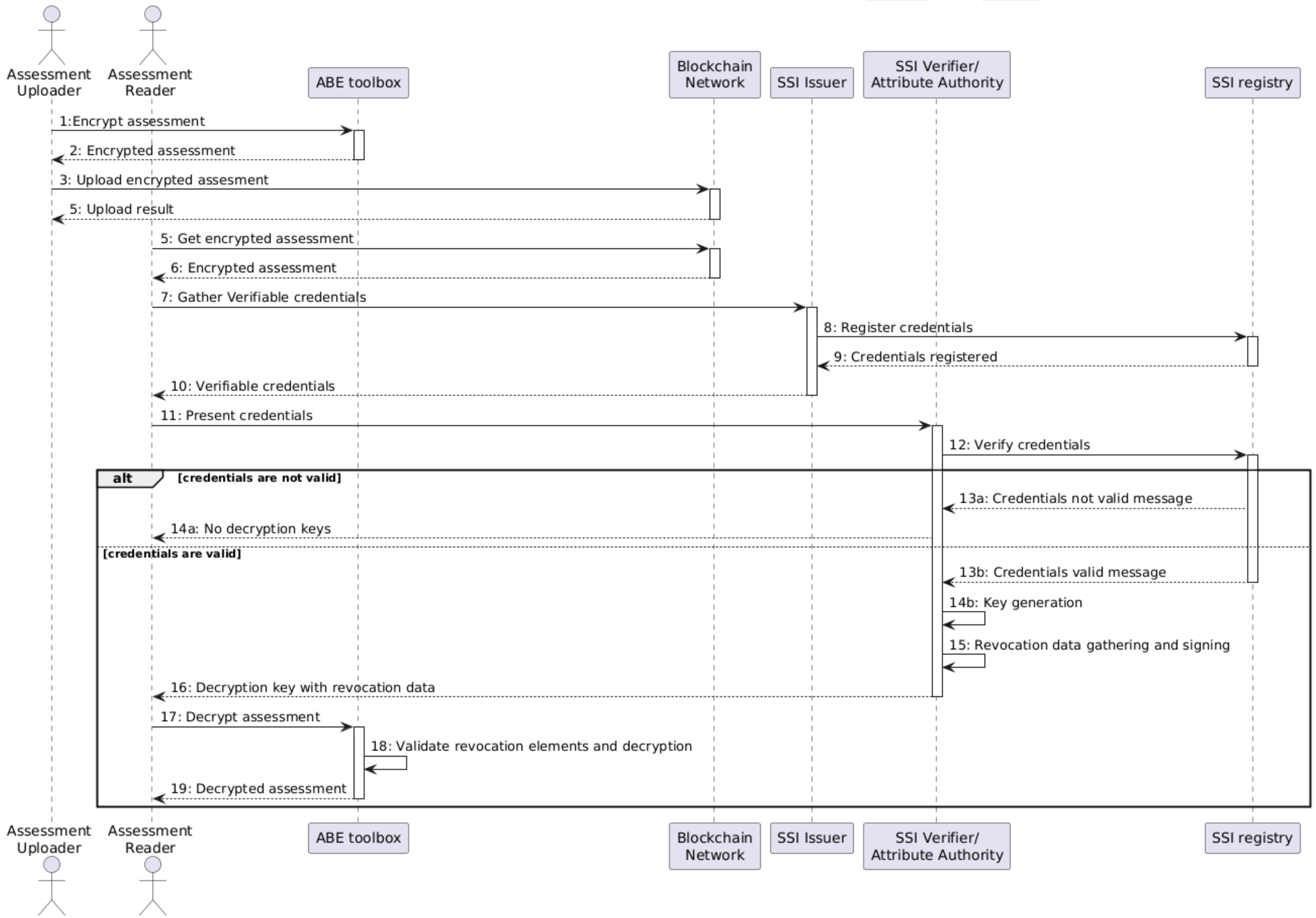


*Figure 31. Proposed sequence diagram for the TWS extension with encrypted data*

### 9.4.3 Proof of Concept

The implemented proof of concept considers a decryptor with verifiable credentials associated with the attributes requested to be able to evaluate or read the evidence or assessment results. Figure 32 shows an example of properties for the required credential.


```
{
 "credentialSubject": {
  "id": "example:did:4198151871587",
  "auditor": {
   "type": "AuditorProfile",
   "name": "John Doe",
   "AuditorType": "AssesmentAuditor"
  }
 }
 "issuanceDate": "2025-12-20T06:38:06.202581908Z",
 "expirationDate": "2026-07-20T06:38:06.202607447Z"
}
```


*Figure 32. Proposed credential for the TWS extension with encrypted data*

The attributes will be generated from the properties selected by the Attribute Authority, avoiding or ignoring data that may be unique like name and surname or identifiers. Different credentials with different expiration and revocation values will be generated to ensure that the users cannot access the data when their credentials, and consequently their attributes, are not valid but that they can access while those credentials are still valid. All of this, while still having the decryption keys in their device.

The authority that will sign the key will have an Ethereum address, which can be contrasted in the *TWS* Blockchain network used for the evidence and assessment results storage.

WaltID[27] SSI environment has been considered for SSI credentials, configured to use the user's Ethereum address and signature system for authentication. The considered credential format is the Verifiable Credential standard made by the World Wide Web Consortium[28]. For ABE schemes and operations, the Charm Cryptography library[29] has been considered.

### 9.4.4 Next Steps

Possible next steps include the integration of the *TWS* extension into the EMERALD framework (beyond the duration of the project). In addition, the verification operations should be integrated in a way that the credential revocation information is integrated inside the key itself instead of having it included with the key. Additionally, current revocation systems require consulting the status of the credential in an external server, not being able to do the verification locally in the whole process. Future lines could be focused on the possibility of attribute validation through local validations. Current publications try to approach this[30], but inevitably there are elements that rely on being updated from external sources and malicious users could knowingly exploit this factor.

[27] https://walt.id/
[28] https://www.w3.org/TR/vc-data-model-2.0/
[29] https://github.com/JHUISI/charm
[30] https://ceur-ws.org/Vol-3962/paper12.pdf


(cc) BY-SA

# 10 Conclusions

In this deliverable, we have provided the final 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 final 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 CERTGRAPH (KR2), OPTIMA (KR3), MULTICERT (KR4), and INTEROP (KR7).

This is the final deliverable of WP3, so no further publications are planned. The next phase of the EMERALD project involves the validation of pilots. Therefore, minor changes may occur in the implementation and integration parts.

DRAFT


(cc) BY-SA

# 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, "D1.6 DevOps methodology and CI/CD strategy for EMERALD-v2," 2025.

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

[5] EMERALD Consortium, "D1.4 EMERALD solution architecture - V2," 2025.

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

[7] EMERALD Consortium, "D3.2 Evidence assessment and Certification - Concepts - v2," 2025.

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

[9] EMERALD Consortium, "D3.4 Evidence assessment and Certification–Implementation-v2," 2025.

[10] EMERALD Consortium, "D3.5 Evidence assessment and Certification - Integration - v1," 2025.

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

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

[13] EMERALD Consortium, "D2.9 Runtime Evidence Extractor – v2," 2025.

[14] EMERALD Consortium, "D2.5 AMOE - v2," 2025.

[15] EMERALD Consortium, "D2.3 Source Evidence Extractor – v2," 2025.

[16] EMERALD Consortium, "D2.7 ML Model Certification – v2," 2025.

[17] Qiao, Z., Liang, S., Davis, S., & Jiang, H., «Survey of attribute-based encryption,» de *15th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD)*, 2014.

[18] Ahmed, M. R., Islam, A. M., Shatabda, S., & Islam, S., «Blockchain-based identity management system and self-sovereign identity ecosystem: A comprehensive survey,» *Ieee Access,* vol. 10, pp. 113436-113481, 2022.

[19] Selar, G. D., & Apoorva, P., «Comparative study on KP-ABE and CP-ABE algorithm for secure data retrieval in military network,» de *International Conference on Intelligent Computing and Control (I2C2)*, 2017.



BY-SA

[20] Rasori, M., La Manna, M., Perazzo, P., & Dini, G., «A survey on attribute-based encryption schemes suitable for the internet of things,» *EEE Internet of Things Journal,* vol. 9, nº 11, pp. 8269-8290, 2022.

DRAFT


(cc) BY-SA