



EMERALD

Deliverable D1.9

EMERALD Integrated solution – v3

Editor(s):	Iñaki Etxaniz, Gorka Benguría
Responsible Partner:	TECNALIA Research & Innovation
Status-Version:	Final-v1.0
Date:	31.08.2026
Type:	Other (SW)
Distribution level (SEN, PU):	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	EMERALD Integrated solution – v3
Due Date of Delivery to the EC	31.08.2026

Workpackage responsible for the Deliverable:	WP1 - Concept and methodology of EMERALD
Editor(s):	Iñaki Etxaniz, Gorka Benguría (TECNALIA)
Contributor(s):	FABAR, TECNALIA, Fraunhofer, CNR, SCCH
Reviewer(s):	Christian Banse (Fraunhofer) Cristina Martínez, Juncal Alonso (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP1, WP2, WP3, WP4, WP5

Abstract:	Third (final) integrated solution of the EMERALD audit suite
Keyword List:	Architecture, Integration, CaaS, Docker, Kubernetes, platform, API, environments, development, production
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	09.06.2026	ToC defined	TECNALIA
v0.2	09.06.2026	First draft version	TECNALIA
v0.3	09.07.2026	Updated Sections 1 and 2	TECNALIA
v0.4	30.07.2026	Contributions by consortium partners to Section 3	FABAR, TECNALIA, Fraunhofer, CNR, SCCH
v0.5	03.08.2026	Conclusions and Executive Summary. Sent to QA review	TECNALIA
v0.6	14.08.2026	Reviewed with comments	Fraunhofer
v0.7	17.08.2026	Addressed recommendations from QA review. Sent to final review.	TECNALIA
v1.0	31.08.2026	Final version submitted to the European Commission	TECNALIA

Table of contents

Terms and Abbreviations	7
Executive Summary	9
1 Introduction.....	10
1.1 About this deliverable.....	10
1.2 Document structure.....	10
1.3 Technological advances from the MEDINA project	11
1.4 Updates from D1.8.....	11
2 Integration Overview.....	12
2.1 Architecture Overview.....	12
2.1.1 Workflows.....	14
2.1.1 Design of the CI/CD Solution	15
2.2 Components Integrated into the EMERALD framework v3.....	16
2.3 Test Bed Environments	17
2.3.1 Container orchestration	18
2.3.2 Storage.....	19
2.3.3 Docker registry	19
2.3.4 Network.....	20
2.3.5 Dashboard	20
2.3.6 Certificates.....	21
2.3.7 Deployment view.....	21
2.4 Steps to Integrate a Component	22
2.5 Overall Status of the Integration	23
2.5.1 Requirements Summary.....	27
3 Integration of Components	32
3.1 Evidence Extractors	32
3.1.1 AI-SEC.....	32
3.1.2 AMOE.....	33
3.1.3 Clouditor-Discovery	35
3.1.4 Codyze	35
3.1.5 eknows-e3	36
3.2 Evidence Assessment and Certification	38
3.2.1 TWS.....	38
3.2.2 MARI	41
3.2.3 RCM	42
3.2.4 Orchestrator	45
3.2.5 Evidence Store	51
3.2.6 Assessment.....	52
3.2.7 Evaluation	53
3.3 EMERALD UI	53
3.3.1 Expected behaviour (inputs/outputs)	54
3.3.2 Published APIs	54
3.3.3 Integration Status	54

4	Conclusions.....	56
5	References.....	57

List of tables

TABLE 1. SUMMARY OF UPDATES WITH RESPECT TO D1.8	11
TABLE 2. COMPONENTS IN THE EMERALD FRAMEWORK V3	16
TABLE 3. INTEGRATION STATUS	24
TABLE 4. COMPONENTS' POINT-TO-POINT INTEGRATION STATUS	25
TABLE 5. REQUIREMENTS PRIORITIZATION MATRIX AND STATUS AT M34	27
TABLE 6. SUMMARY OF REQUIREMENTS STATUS AT M34 (BY COMPONENT)	30
TABLE 7. INTEGRATION STATUS OF AI-SEC WITH OTHER EMERALD COMPONENTS.....	33
TABLE 8. INTEGRATION STATUS OF AMOE WITH OTHER EMERALD COMPONENTS.....	35
TABLE 9. INTEGRATION STATUS OF CLOUDITOR-DISCOVERY WITH OTHER EMERALD COMPONENTS	35
TABLE 10. INTEGRATION STATUS OF CODYZE WITH OTHER EMERALD COMPONENTS	36
TABLE 11. INTEGRATION STATUS OF EKOWS-E3 WITH OTHER EMERALD COMPONENTS	38
TABLE 12. INTEGRATION STATUS OF TWS WITH OTHER EMERALD COMPONENTS.....	40
TABLE 13. INTEGRATION STATUS OF MARI WITH OTHER EMERALD COMPONENTS	41
TABLE 14. INTEGRATION STATUS OF THE RCM WITH OTHER EMERALD COMPONENTS	45
TABLE 15. INTEGRATION STATUS OF ORCHESTRATOR WITH OTHER EMERALD COMPONENTS	50
TABLE 16. INTEGRATION STATUS OF EVIDENCE STORE WITH OTHER EMERALD COMPONENTS.....	52
TABLE 17. INTEGRATION STATUS OF ASSESSMENT WITH OTHER EMERALD COMPONENTS	53
TABLE 18. INTEGRATION STATUS OF EVALUATION WITH OTHER EMERALD COMPONENTS	53
TABLE 19. INTEGRATION STATUS OF EMERALD UI WITH OTHER EMERALD COMPONENTS.....	55

List of figures

FIGURE 1. EMERALD COMPONENTS.....	12
FIGURE 2. INVOLVEMENT OF THE FRAMEWORK'S COMPONENTS IN THE EMERALD BLUEPRINT FOR AUDIT PREPARATION	15
FIGURE 3. MERGE REQUEST AND GENERIC CI/CD PIPELINES	15
FIGURE 4. INTEGRATION AND PRODUCTION ENVIRONMENTS IN THE EMERALD CAAS FRAMEWORK	17
FIGURE 5. URL NAMING CONVENTION FOR INTEGRATION/PRODUCTION ENVIRONMENTS	18
FIGURE 6. LONGHORN DASHBOARD IN RANCHER.....	19
FIGURE 7. EMERALD DOCKER REGISTRY.....	20
FIGURE 8. RANCHER DASHBOARD.....	21
FIGURE 9. DEPLOYMENT DIAGRAM OF EMERALD CAAS FRAMEWORK V3.....	22
FIGURE 10. MAIN STEPS OF THE COMPONENT INTEGRATION	23
FIGURE 11. NUMBER OF REQUIREMENTS BY STATUS AT M34	29
FIGURE 12. REQUIREMENT PROGRESS DISTRIBUTION AT M34.....	29
FIGURE 13. REQUIREMENTS COMPLETION PER COMPONENT	31
FIGURE 14. EVIDENCE EXTRACTORS IN THE EMERALD FRAMEWORK ARCHITECTURE	32
FIGURE 15. EVIDENCE ASSESSMENT AND CERTIFICATION TOOLS IN THE EMERALD ARCHITECTURE	38
FIGURE 16. EMERALD UI IN THE EMERALD ARCHITECTURE.....	54

List of listings

LISTING 1. AMOE API OVERVIEW	34
LISTING 2. TWS API ENDPOINTS FOR ACCOUNT MANAGEMENT	39
LISTING 3. TWS API ENDPOINTS FOR EVIDENCE SOURCES MANAGEMENT	40

LISTING 4. TWS API ENDPOINTS EVIDENCE MANAGEMENT	40
LISTING 5. TWS API ENDPOINTS FOR ASSESSMENTS RESULTS MANAGEMENT	40
LISTING 6. MARI API ENDPOINTS FOR MAPPING	41
LISTING 7. RCM API ENDPOINTS FOR SCHEMA INFORMATION	42
LISTING 8. RCM API ENDPOINTS FOR CONTROL INFORMATION	43
LISTING 9. RCM API ENDPOINTS FOR METRIC INFORMATION	43
LISTING 10. RCM API ENDPOINTS FOR SIMILAR CONTROLS	44
LISTING 11. RCM API ENDPOINTS FOR RELATED METRICS	44
LISTING 12. RCM API ENDPOINTS FOR QUESTIONNAIRE RESOURCES	45
LISTING 13. ORCHESTRATOR API ENDPOINTS FOR ASSESSMENT RESULTS.....	46
LISTING 14. ORCHESTRATOR API ENDPOINTS FOR AUDIT SCOPES	46
LISTING 15. ORCHESTRATOR API ENDPOINT FOR AUDIT TRAIL EVENTS	46
LISTING 16. ORCHESTRATOR API ENDPOINTS FOR CATALOGUES	47
LISTING 17. ORCHESTRATOR API ENDPOINTS FOR HANDLING CERTIFICATES	47
LISTING 18. ORCHESTRATOR API ENDPOINTS FOR CERTIFICATES (PUBLICLY AVAILABLE)	47
LISTING 19. ORCHESTRATOR API ENDPOINTS FOR CONTROLS	48
LISTING 20. ORCHESTRATOR API ENDPOINTS FOR EVALUATION RESULTS.....	48
LISTING 21. ORCHESTRATOR API ENDPOINTS FOR METRICS	48
LISTING 22. ORCHESTRATOR API ENDPOINT FOR RUNTIME INFO.....	49
LISTING 23. ORCHESTRATOR API ENDPOINTS FOR TARGETS OF EVALUATION	49
LISTING 24. ORCHESTRATOR API ENDPOINTS FOR USERS	50
LISTING 25. EVIDENCE STORE API ENDPOINTS.....	51
LISTING 26. ASSESSMENT API ENDPOINT FOR EVIDENCE.....	52
LISTING 27. EVALUATION API ENDPOINTS.....	53

Terms and Abbreviations

AI	Artificial Intelligence
AI-SEC	AI Security Evidence Collector
AIC4	AI Cloud Service Compliance Criteria Catalogue
AMOE	Assessment and Management of Organizational Evidence
API	Application Programming Interface
AWS	Amazon Web Services
BSI	Bundesamt für Sicherheit in der Informationstechnik
CaaS	Compliance-as-a-Service ¹
CI/CD	Continuous Integration / Continuous Delivery
CLI	Command Line Interface
CM	Compliance Manager
CSP	Cloud Service Provider
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
GB	Gigabyte
gRPC	Google Remote Procedure Call
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
IaC	Infrastructure as Code
IP	Internet Protocol
JSON	JavaScript Object Notation
KR	Key Result
MARI	Mapping Assistant for Regulations with Intelligence
ML	Machine Learning
MS	Milestone
NLP	Natural Language Processing
OSCAL	Open Security Controls Assessment Language
OS	Operating System
PaaS	Platform as a Service
RAM	Random Access Memory
RBAC	Role-Based Access Control
RCM	Repository of Controls and Metrics
REST	Representational State Transfer
RKE	Rancher Kubernetes Engine
SSH	Secure Shell
SSL	Secure Sockets Layer
TB	Terabyte
TWS	Trustworthiness System
UI/UX	User Interface / User Experience
URL	Uniform Resource Locator
VCS	Version Control System

¹ Please note that in previous deliverables and in the DoA, the term Certification-as-a-Service was used to stand for CaaS. Compliance has now been introduced to clarify that EMERALD can be used to assess both normative models and internal organizational models.

VM	Virtual Machine
WP	Work Package

DRAFT

Executive Summary

The deliverable D1.9 comprises the software we defined as “EMERALD CaaS framework v3”, which incorporates the components developed in the project's technical work packages. This document accompanies the software deliverable and provides an overview of the integration approach, the status of the components involved and their final integration.

This deliverable is part of Work Package 1 (WP1), which focuses on the project's concept and methodology, and more specifically of the task T1.3 “Continuous integration and optimization”. The integrated solution is a crucial part of the project, as it assembles the components developed in other technical work packages and enables collaboration and communication between them.

The component integration approach, described in the first version of this deliverable [1], is based on two main pipelines that automate the process: the first one retrieves the code from the *GitLab* repository and builds the project, creating *Docker* images and pushing them to an *Artifactory*-based repository. The second one deploys the components to a test bed environment and verifies them. The test bed is based on *OpenStack* Virtual Machines. On top of all this, a *Kubernetes* cluster has been created, composed by three nodes at the first version, that has been gradually escalated to the actual six nodes. The cluster provides five environments: one for integration and four others for the pilots.

For the integration of a component in the EMERALD framework, an eight-step procedure was defined. The integration plan includes three phases to be completed in months M18, M30, and M34, respectively. This deliverable corresponds to the third phase. In earlier phases, workshops were delivered by WP1 to the rest of the consortium to introduce the main concepts of *Docker* and *Kubernetes* and to explain the integration procedures to developers.

The document details the level of integration of each component in the framework, including the list of published APIs, and the status of the interconnection with the other components. Ten of the components have reached 100% implementation, and all components are above 95% completion. From version 2 of this deliverable onwards, we have included, as a bonus, a summary of the status of the requirements for each component; based on this, we can state that, at the time of writing this report, the average completion rate for the project stands at 99%.

The main outcomes of this deliverable include the third and final prototype of the EMERALD CaaS framework, which consolidates the first and second versions; documentation detailing the integration status of each component; and the deployment of the framework in the four EMERALD pilots.

The EMERALD framework provided is the latest deliverable of Task T1.3, but work on the task will continue until the end of the project, providing support to the pilots during the evaluation phase and incorporating last-minute changes and improvements.

1 Introduction

1.1 About this deliverable

This is the companion document of the software deliverable D1.9, which aims to produce a third prototype of the EMERALD Compliance-as-a-Service² (CaaS) framework that integrates the components developed by the other technical work packages. This third version of the integrated solution corresponds to the Milestone *MS8 – Integrated Audit Suite v3* and is mainly based on the version v2 of the EMERALD components (M24), with the additional developments carried out up to M34. All the referred software is available in the project's public *Gitlab* (<https://git.code.tecnalia.dev/emerald/public>).

The document begins with an overview of the integration approach, with the aim of providing the reader with a general understanding of which components are integrated, where and how, as well as the status of integration task as a whole. It also describes the hardware and software used to setup the test bed³ and its general conformation. Furthermore, it outlines the methodology used to integrate a component into the framework. The document also includes a description of the main workflow across various scenarios and briefly outlines the CI/CD solution that has been implemented to support the development and integration activities of the EMERALD framework. Finally, the document provides a detailed overview of the current status of the integration of all components of the EMERALD framework.

This third –and final– version of the deliverable incorporates the advancements and improvements made since last release, based on users' feedback, testing and discussions regarding the previous version.

1.2 Document structure

The remainder of the document is organized as follows:

The rest of the “Introduction” section is dedicated to summarizing the advances of EMERALD in relation to the MEDINA project, and the updates from the previous version of the deliverable.

Section 2 presents a general description of the integration strategy and the tools used. It gives an overview of the EMERALD CaaS framework, the resources used for the test bed environments, the integration steps for each component, and the CI/CD implementation supporting the integration of the EMERALD framework. An overall status of the integration is also included.

Section 3 details the integration status of each EMERALD component, describing the APIs implemented and how they connect to the other components of the framework.

Section 4 presents the conclusions, including a summary of the main outcomes of the deliverable.

² Please note that in previous deliverables and in the DoA, the term Certification-as-a-Service was used to stand for CaaS. Compliance has now been introduced to clarify that EMERALD can be used to assess both normative models and internal organizational models.

³ A “Test Bed” refers to the setup where the testing activities take place. It includes the combination of hardware, software, network configurations, and other necessary components that provide the infrastructure which aims to simulate the real-world conditions under which the software will operate. (see more at <https://testingfundamental.com/test-bed>)

1.3 Technological advances from the MEDINA project

EMERALD builds upon the outcomes of the MEDINA project⁴: *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).

As a result, the architectural and integration approaches in both projects are quite similar. We are supported by the same technologies (*Kubernetes, Docker, GitLab, Artifactory*) which have proven to be robust and scalable. In some cases, for the sake of robustness, we have selected different tools in EMERALD (e.g., *nginx, Rancher* or *Longhorn*) to replace equivalent ones used in MEDINA. The integration workflow is similar to that used in MEDINA, and is based on the same *GitLab* CI pipelines for automating build and deployment. The main difference in terms of infrastructure is that EMERALD runs on an *OpenStack*-based virtualisation platform at TECNALIA, whereas MEDINA used *VMware*.

The advance of each EMERALD tool has already been detailed in the corresponding deliverable, as identified in Section 2.2.

1.4 Updates from D1.8

The deliverable evolves from D1.8 and, with the goal of making the document self-contained and easier to follow, some of its content remains unchanged, while other parts are new. To simplify tracking progress from the previous version, Table 1 provides a summary of the changes and additions made to the document.

Table 1. Summary of updates with respect to D1.8

Section	Changes
2. Integration Overview	Minor changes have been made to reflect updates to the test bed environment. Section 2.5 - <i>Overall Status of the Integration</i> has been updated to reflect the overall requirement and integration status in M34.
3. Integration of Components	The status of the integration per component has been updated.
4. Conclusions	Updated to reflect the changes in the document.

⁴ <https://medina-project.eu/>

2 Integration Overview

The integration strategy set out in EMERALD is aimed at two main audiences: developers and end-users. On the one hand, it should provide means for **developers** to:

- Finetune the configuration of the deployment: parameters, auxiliary services, IAM configuration, ingress configuration, etc.
- Be able to debug the behaviour of a component, either by accessing the logs or by logging into the container in which the component is running.
- Have rights to destroy a component and deploy it again with changes or fixes.
- Automatically deploy new versions of a component as it is released.

On the other hand, the **end-users**, impersonated by the pilots, should be supported by providing:

- An easy-to-deploy CaaS framework that facilitates on premise installation.
- A stable release that guarantees support for the base workflows.
- A production environment to test the latest version of the framework, or to be used by pilots for demonstration purposes.

2.1 Architecture Overview

Figure 1 provides an updated general view of the EMERALD components and their data flow, superseding the architecture diagram originally in D1.2 [2]. The colour of a component indicates its function within the framework, while dashed/full lines denote pull/push of data.

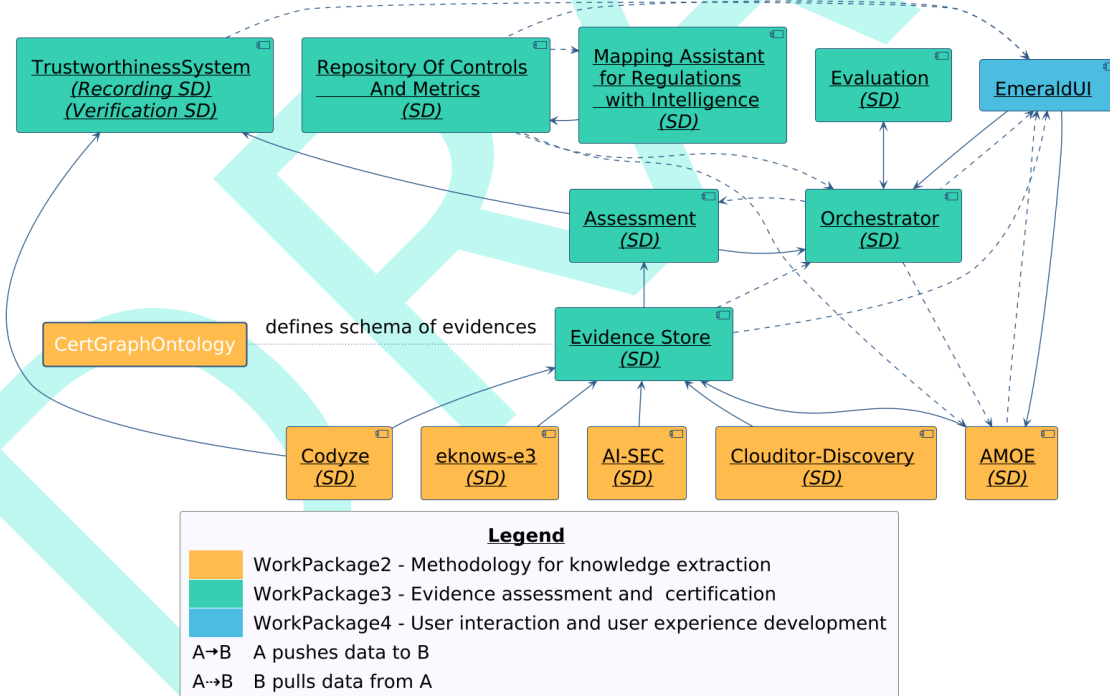


Figure 1. EMERALD Components

The following evidence extractors (in **orange**) collect different types of data and extract evidence, which is then shared within the EMERALD framework:

- **AMOE – Assessment and Management of Organisational Evidence** – extracts evidence from policy PDF documents. The component stores the uploaded files, as well as the relevant metadata relating to the documents and metrics.
- **Codyze** is a static source code analysis tool which analyses source code of applications comprising cloud services and assesses security-relevant implementation details according to specified security requirements.
- **eknows-e3** contributes to increase the coverage of code-related security requirements by extracting evidence from source code files and code-related artifacts (e.g., build system configuration and metadata from version control systems) collected from the cloud service environment. It offers language-independent static codes analysis and automates code reviews.
- **AI-SEC** is an evidence collection tool that extracts various security and robustness information from AI models.
- **Clouditor-Discovery** is an evidence collection tool which extracts cloud configurations for different cloud resources (e.g., virtual machines, storage, networks) from different cloud providers via API calls.

The evidence assessment and compliance components (in green) are the next step in the EMERALD workflow, namely:

- The **Evidence Store** functions as a centralized repository for storing evidence from the evidence extraction components during the compliance process. It utilizes a graph-based database to organize and manage evidence in an efficient and accessible manner.
- The **Assessment** component is responsible for assessing the evidence and providing the *Orchestrator* with assessment results. It calculates the assessment results using the metrics provided by the *Repository of Controls and Metrics (RCM)*.
- The **Evaluation** component is responsible for combining assessment results of individual metrics relevant to a specific control of a certification scheme to create an evaluation result for this control.
- The **Orchestrator's** main purpose is to hold all dynamic information relating to an audit process, such as the target of evaluation, the assessment results and the compliance state. It includes the Compliance Graph, providing a snapshot of the target of evaluation's state.
- The **Trustworthiness System (TWS)** component ensures that all actions and data within the compliance process are tamper-proof and verifiable. It securely stores the information and associated metadata of evidence and assessment results on a general-purpose blockchain network.
- The **Mapping Assistant for Regulations with Intelligence (MARI)** component is an intelligent system using AI techniques and NLP processing to select suitable metrics for demonstrating compliance with certification schemes. It can also associate security controls of two different certification schemes.
- The **Repository of Controls and Metrics (RCM)** component serves as a smart catalogue of controls and metrics. The repository supports different schemes, with the corresponding categorization. It also provides import/export mechanisms to facilitate the reuse and composition of catalogue elements.

Finally, there is the User Interface component (in blue), which is essential for implementing the business cases for the various user roles in EMERALD:

- The **EmeraldUI** wraps all the functionality of the components into a single user interface. It calls the APIs provided by the components to exchange information and present it to users.

2.1.1 Workflows

The work processes for the EMERALD framework have been reported for the different pilots and user roles in the WP4 deliverables, which detail the various steps in the processes; based on these, a blueprint has been developed for a universal application designed to implement EMERALD within audit preparation workflows.

A condensed version of these steps is presented below, which is then used to illustrate the involvement of the components in the processes in Figure 2 (for more detailed information, see D4.4 [3]):

- **Phase 1: Certification Scheme:** The Compliance Manager (CM) uploads/creates a certification scheme. The EMERALD framework automatically assigns metrics to controls.
- **Phase 2: Check controls and metrics:** The CM checks/changes the metrics automatically assigned to a control.
- **Phase 3: Setup Target of Evaluation and Audit Scope:** The CM needs to set up a new target of evaluation; and the CM uploads the policy documents in EMERALD. The CM sets up a new audit scope using the newly created target of evaluation and the respective certification scheme.
- **Phase 4: Audit Scope:** EMERALD will automatically collect evidence from a target of evaluation for all controls and assigned metrics and will create assessment results. The CM can browse through all controls/metrics and is able to filter between compliant and noncompliant assessment and evaluation results.
- **Phase 5: Check controls and assessment results:** The CM checks the corresponding assessment results/evidence. If the check is ok, the CM can set the control/metric to compliant or assign it to another person. If a control cannot be automatically assessed, the user can add evidence manually and set the control/metric as compliant.
- **Phase 6: Reporting & Validation:** EMERALD provides different types of outcomes, such as Audit Report, Track Record of Evidence, Compliance Status, Categorization of the Service, and Validity Check.

The components of the EMERALD framework are the basis for the implementation of these workflows, providing their functionality when required. As shown in Figure 2, the *Orchestrator* acts as the central coordinator across all workflow steps. The *RCM* and *MARI* components are involved in the early phases of the workflow. The evidence extractors and the *Assessment* component are involved in the intermediate phases, and the *TWS*, *Evaluation* and *Evidence Store* components come into play towards the end of the process. The *EMERALD UI*, for its part, takes part in the entire workflow.

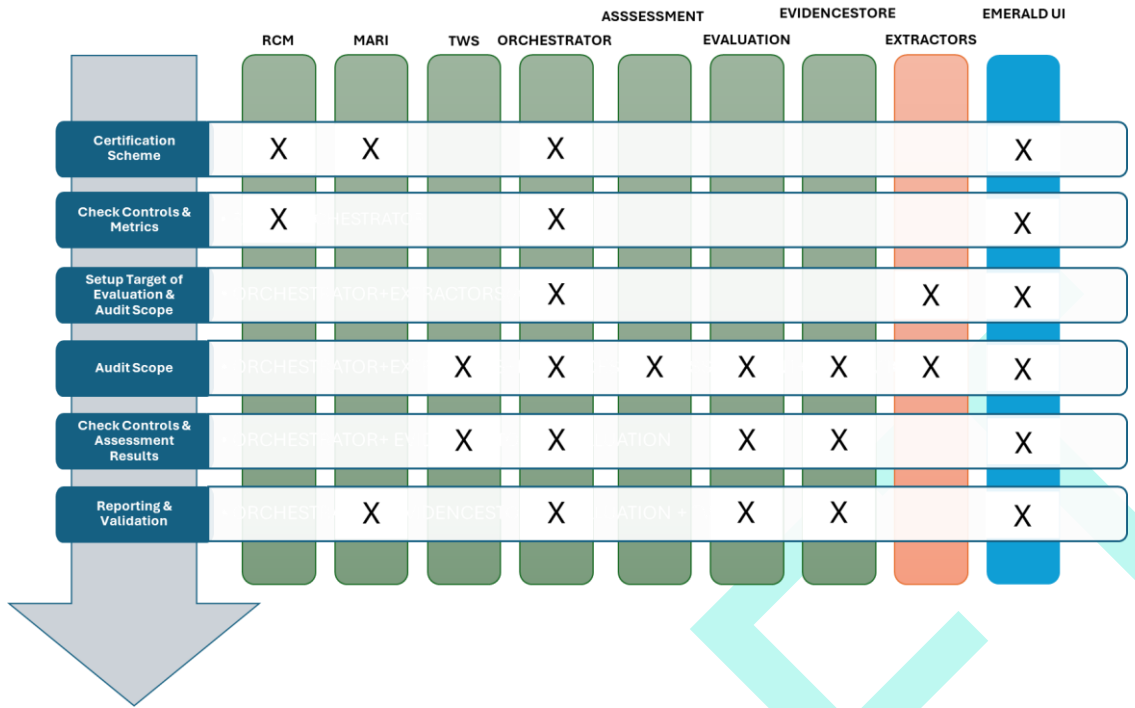


Figure 2. Involvement of the framework’s components in the EMERALD blueprint for audit preparation

2.1.1 Design of the CI/CD Solution

The initial CI/CD strategy for the EMERALD framework, which was outlined in D1.5 [4] and completed in D1.6 [5], involves CI/CD practices designed for EMERALD that are implemented in the build, deploy and security pipelines (see Figure 3). The project has adopted a Continuous Integration and Continuous Deployment (CI/CD) strategy that, based in the source code of the EMERALD components, detects every change and starts a chain of automated activities that ends with the deployment of the component in the integration environment. This is done with the help of the *GitLab* CI tool.

A version control system (VCS) manages software source code so that several people can work on the implementation, and the history of changes can be tracked. For this purpose, all EMERALD components are available in *GitLab* repositories.

Each **Merge Request** (MR) of the code in the *Gitlab* project triggers a set of actions (code compilation, container image build, unit tests) whose results determine whether the MR can be finally merged (see Figure 3). Later, integration tests of the EMERALD framework as a whole will ensure that the components work in tandem; in other words, that they can be installed, run and used together in accordance with the user-defined workflows set out in section 2.1.1.

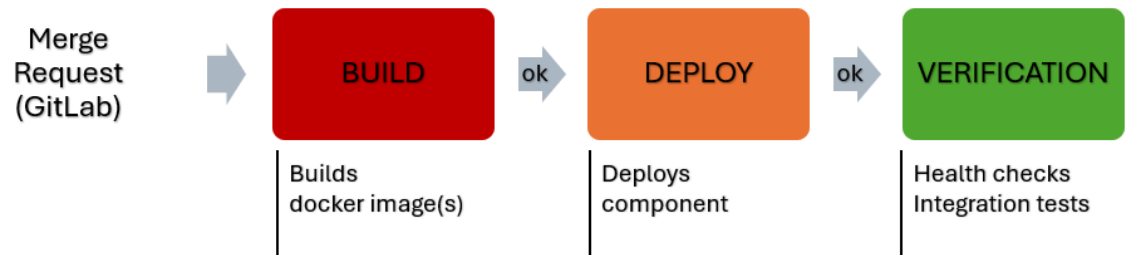


Figure 3. Merge Request and generic CI/CD pipelines

In the EMERALD project we have defined two main pipelines: **Component build**, and **CaaS framework deployment**. The “Component build” pipeline automates building a component, creating the *Docker* image and pushing it to the *Artifactory*. It can also include different functional and non-functional validation activities. The “CaaS framework deployment” pipeline will automatically deploy each component to the integration environment and then verify it to determine if it is stable enough to be promoted into the production environment.

2.2 Components Integrated into the EMERALD framework v3

All the components, 13 in total, have been integrated in this third version of the EMERALD framework. They are listed in Table 2, together with their respective Key Result (KR) and the deliverable in which they are described.

Table 2. Components in the EMERALD framework v3

Component name	Key Result	Deliverable
<i>AI-SEC - AI Security Evidence Collector</i>	KR5 AIPOC	D2.7 [6]
<i>AMOE - Assessment and Management of Organisational Evidence</i>	KR2 CERTGRAPH	D2.5 [7]
<i>Clouditor-Discovery</i>	KR1 EXTRACT	D2.9 [8]
<i>Codyze</i>	KR1 EXTRACT	D2.3 [9]
<i>eknows-e3</i>	KR1 EXTRACT	D2.3 [9]
<i>TWS - Trustworthiness System</i>	KR7 INTEROP	D3.4 [10]
<i>MARI - Mapping Assistant for Regulations with Intelligence</i>	KR3 OPTIMA	D3.4 [10]
<i>RCM - Repository of Controls and Metrics</i>	KR7 INTEROP	D3.4 [10]
<i>Orchestrator</i>	KR4 MULTICERT	D3.4 [10]
<i>Evidence Store</i>	KR2 CERTGRAPH	D3.4 [10]
<i>Assessment</i>	KR4 MULTICERT	D3.4 [10]
<i>Evaluation</i>	KR4 MULTICERT	D3.4 [10]
<i>EMERALD UI</i>	KR6 EMERALD UI/UX	D4.6 [11]

All components expose and consume interfaces for integration. REST⁵-based interfaces follow standard REST API practices and are described in the OpenAPI⁶ format, while gRPC/Connect⁷-based interfaces are defined using Protocol Buffers. Where applicable, the published interfaces and the interactions among components are detailed in Section 3.

Apart from the components developed in the project, the EMERALD framework also requires an Identity and Access Management (IAM) solution to manage users and roles, and to secure the communication among components. So, we have implemented a Role Based Access Control (RBAC) along the components. We have chosen *Keycloak*⁸ as open-source IAM solution for the project.

⁵ Representational State Transfer (REST) is an architectural style for distributed hypermedia systems. More information is available at <https://restfulapi.net/>

⁶ The OpenAPI Specifications provide a formal standard for describing HTTP API. More information is available at <https://www.openapis.org/>

⁷ gRPC is a modern open-source high-performance Remote Procedure Call (RPC) framework that can run in any environment. Connect is a modern implementation of it. More information is available at <https://connectrpc.com>

⁸ <https://www.keycloak.org/>

2.3 Test Bed Environments

The EMERALD CaaS framework is supported by two different environments: **Integration** and **Production** (see Figure 4). The components are first deployed in the Integration environment in containerized form. Once the integration tests have been passed, components are promoted to the Production environment.

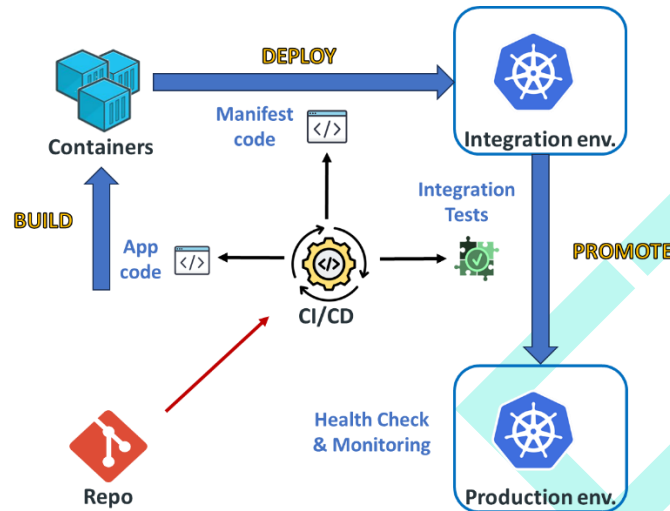


Figure 4. Integration and Production environments in the EMERALD CaaS framework

Each test bed consists of several Virtual Machines (VMs) located within TECNALIA's server infrastructure. The environments are built in a six-node **Kubernetes**⁹ cluster over an **OpenStack**¹⁰ platform.

The Virtual Machines (VMs) of the Kubernetes nodes share the same specifications:

RAM: 16GB Cores: 8 HD Disk: 200GB + 200GB OS: Ubuntu 24.04

These specifications can be scaled up as needed. Since the first version of the CaaS framework, comprising three nodes, other three VMs have been added to the Kubernetes cluster, in line with the project's computational needs. Across the five environments (the integration environment plus pilots 1, 2, 3 and 4), there are a total of around 150 pods running.

The access to the virtual machines is provided via SSH¹¹ (Secure Shell) protocol, using digital certificates through a bastion host¹². The REST API exposed by each component is reachable from the Internet using this URL naming convention:

`<component>.<environment (dev or pilotN)>.emerald.digital.tecnalia.dev`

where dev/pilotN refer to integration/production environment, respectively. This convention is depicted in Figure 5. For example, if the user needs to refer to the API exposed by the *RCM*

⁹ <https://kubernetes.io/docs/home/>

¹⁰ <https://www.openstack.org>

¹¹ <https://www.ssh.com/academy/ssh/protocol>

¹² https://en.wikipedia.org/wiki/Bastion_host

component running in the Kubernetes pilot 1 environment, it will be addressed as `rcm.pilot1.emerald.digital.tecnalia.dev`.

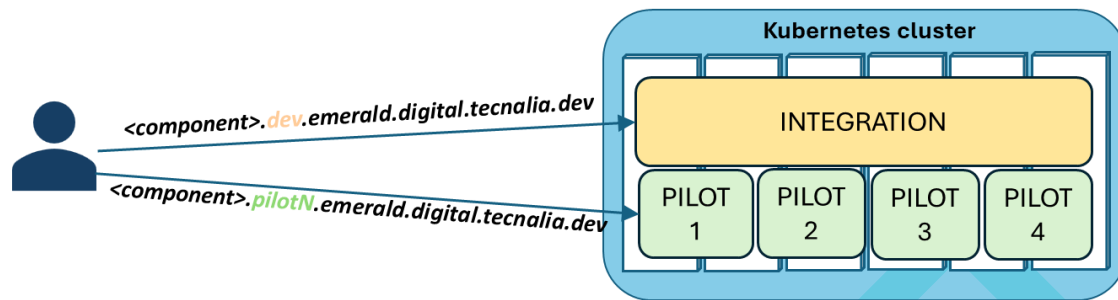


Figure 5. URL naming convention for integration/production environments

When selecting technologies for the testbed, we prioritised those close to the production state of practices. To simplify the installation and operation of Kubernetes, we used **Rancher Kubernetes Engine (RKE2¹³)**, an open-source, enterprise-ready Kubernetes distribution. Using RKE2, we have deployed a high availability configuration where all nodes are configured as master and worker and share a virtual IP (VIP). This VIP is associated with the project testbed host domain `*.emerald.digital.tecnalia.dev`.

We have followed an Infrastructure as Code (IaC) approach for the deployment. For the creation and configuration of the cluster we have used **OpenTofu¹⁴** and **Ansible¹⁵** technologies. **OpenTofu** is used to create the nodes, networks, network interfaces, and security groups among other infrastructural elements. **Ansible** is used to configure the nodes with the software packages required to implement the Kubernetes cluster. The IaC files are also subject to a configuration management process, in the project's *Gitlab* repository. The usage of IaC provides several advantages for project management:

- It allows the redeployment of the cluster from scratch, should a migration be necessary.
- It simplifies the horizontal scalation of Kubernetes if more capacity is required (as has been the case).
- It can be reused by pilots, should they have a similar infrastructure.

2.3.1 Container orchestration

The EMERALD framework's functionality is based on the collaboration of micro-services, which communicate with one another via APIs, are packaged into Docker images and run in containers. *Kubernetes* orchestrates all these containers within a virtual environment running on a high-availability cluster.

We also use an IaC approach based on **Kustomize¹⁶** to describe the deployment and collaboration of all components of the EMERALD project. The container orchestration is stored in a separate *GitLab* repository of the project named "CaaS Framework". This repository contains

¹³ <https://docs.rke2.io/>

¹⁴ <https://opentofu.org>

¹⁵ <https://www.ansible.com>

¹⁶ *Kustomize* traverses a *Kubernetes* manifest to add, remove or update configuration options without forking. More information is available at <https://kustomize.io/>

a folder named “components” with the deployment details for each of the components, as well as other folders describing the environments, named “integration” and “production”.

2.3.2 Storage

The micro-services can store their data in an easy and secure way thanks to the configuration of a distributed filesystem provided by **Longhorn**¹⁷. Indeed, each node of the cluster provides 200 GB of storage, managed by **Longhorn** and is exposed as a single, unified cluster filesystem. Thus, the data is replicated across the six nodes, and a total of 1,23 TB fault-tolerant and high availability of storage are assured, as shown in Figure 6.

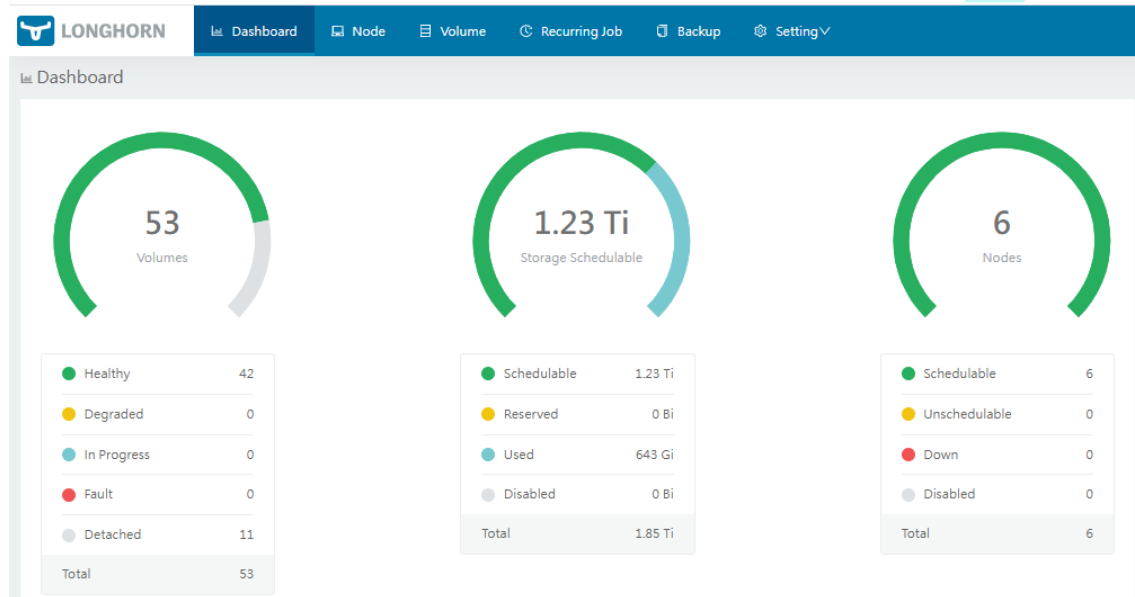


Figure 6. Longhorn dashboard in Rancher

2.3.3 Docker registry

The micro-services running on the **Kubernetes** cluster are packaged in **Docker** images and stored in a private **Docker** registry running in the TECNALIA infrastructure’s **Artifactory**¹⁸. To access the **Docker** registry, a **Kubernetes** secret has been created with the credentials. This allows **Kubernetes** to pull the micro-service images and then run them on the cluster. The **GitLab** CI/CD pipelines build and push the images to the **Docker** registry, following the folder structure agreed for the project (see Figure 7). The images are stored under the following URLs:

`https://artifact.tecnalia.dev/ui/native/emerald-docker-dev-local/<component>/`

¹⁷ **Longhorn** is an open-source, cloud-native distributed storage solution for delivering block storage persistent with low requirements and overhead. For more details see <https://rook.io/docs/rook/v1.8/>

¹⁸ <https://ifrog.com/artifactory/>

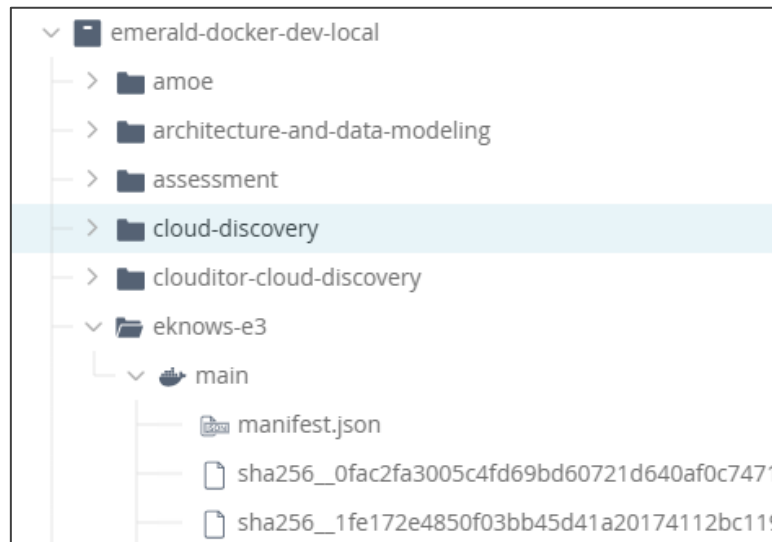


Figure 7. EMERALD Docker registry

2.3.4 Network

On the *Kubernetes* cluster, a *nginx*¹⁹ service is configured as a proxy to redirect all the requests to the correct micro-service component. The binding between the *nginx* service and the public IP is setup with *KubeVip*²⁰, a network load-balancer that associates the public IP to the *nginx* service and uses standard routing protocols to make available (part of) the network behind the *Kubernetes* cluster. *KubeVip* is essential for the EMERALD cluster because, unlike managed public cloud environments, neither *nginx* nor *Kubernetes* natively provide an external load balancer on bare-metal or private cloud infrastructure such as *OpenStack*.

2.3.5 Dashboard

The *Kubernetes* cluster includes two user accounts with distinct permission levels: an “admin” account with full access to all cluster resources and administrative functions; and an “emerald_developer” account with restricted access, limited to the integration and production namespaces.

Rancher provides a web-based dashboard for the *Kubernetes* cluster (see Figure 8), which helps to deploy containerised applications to a *Kubernetes* cluster, troubleshoot them, and manage cluster resources. As can be seen, there are currently around 190 pods (equivalent to containers) being managed in the cluster (this includes some extra resources, such as Keycloak and pods that are not currently running and are in the process of being evicted). The dashboard is accessible over HTTPS in the dev environment²¹.

¹⁹ <https://www.nginx.com/>

²⁰ <https://kube-vip.io/>

²¹ <https://k8so.emerald.digital.tecnalia.dev/dashboard/>

State	Name	Namespace	Image	Ready	Restarts	IP	Node	Age
Running	amoe-66b7d65644-4ngtk	emerald-pilot4	emerald-docker-dev-local.artifact.tecnalia.dev/amoe.1.6.0	1/1	0	10.42.2.24	k8so03-emerald	1.2 days
Running	amoe-6d6488049d-bgwex	emerald-pilot2	emerald-docker-dev-local.artifact.tecnalia.dev/amoe.1.6.0	1/1	11 (26h ago)	10.42.0.8	k8so01-emerald	3.1 days
Running	amoe-7d5d3d0d6d-pchq	emerald-pilot1	emerald-docker-dev-local.artifact.tecnalia.dev/amoe.1.6.0	1/1	13 (26h ago)	10.42.0.239	k8so01-emerald	21 days
Running	amoe-578d597796-zg8tk	emerald-pilot3	emerald-docker-dev-local.artifact.tecnalia.dev/amoe.1.6.0	1/1	6 (3d1h ago)	10.42.0.250	k8so01-emerald	3.1 days
Running	amoe-d5d6568f-sq92s	emerald-dev	emerald-docker-dev-local.artifact.tecnalia.dev/amoe/snapshot-manual-evidencefield	1/1	0	10.42.1.229	k8so02-emerald	2.1 hours
Running	amoe-mongodb-0	emerald-dev	mongo:7.0.37-jammy	1/1	0	10.42.5.71	k8so06-emerald	1.2 days
Running	amoe-mongodb-0	emerald-pilot1	mongo:7.0.37-jammy	1/1	0	10.42.2.249	k8so03-emerald	29 days
Running	amoe-mongodb-0	emerald-pilot2	mongo:7.0.37-jammy	1/1	0	10.42.5.70	k8so06-emerald	1.2 days
Running	amoe-mongodb-0	emerald-pilot3	mongo:7.0.37-jammy	1/1	0	10.42.0.211	k8so01-emerald	29 days
Running	amoe-mongodb-0	emerald-pilot4	mongo:7.0.37-jammy	1/1	0	10.42.2.180	k8so03-emerald	3.2 days
Running	amoe-realm-configuration-gwqj	emerald-dev	adorsys/keycloak-config-clf:6.5.1-25.0.1	1/1	0	10.42.4.61	k8so05-emerald	3.7 days
Running	amoe-realm-configuration-zrsh	emerald-pilot4	adorsys/keycloak-config-clf:6.5.1-25.0.1	1/1	0	10.42.4.92	k8so05-emerald	3.1 days
Running	amoe-redis-0	emerald-dev	redis:alpine	1/1	0	10.42.2.183	k8so03-emerald	51 days
Running	amoe-redis-0	emerald-pilot1	redis:alpine	1/1	0	10.42.1.174	k8so02-emerald	3.1 days
Running	amoe-redis-0	emerald-pilot2	redis:alpine	1/1	0	10.42.1.173	k8so02-emerald	3.1 days
Running	amoe-redis-0	emerald-pilot3	redis:alpine	1/1	0	10.42.2.184	k8so03-emerald	51 days
Running	amoe-redis-0	emerald-pilot4	redis:alpine	1/1	0	10.42.5.209	k8so06-emerald	38 days
Running	architecture-and-data-modeling-b49f5d47f-2d19	emerald-ext	emerald-docker-dev-local.artifact.tecnalia.dev/architecture-and-data-modeling:1.5.3	1/1	0	10.42.2.176	k8so03-emerald	51 days
Running	assessment-54f9cd8db-dg92j	emerald-pilot4	emerald-docker-dev-local.artifact.tecnalia.dev/assessment:v0.2.17	1/1	0	10.42.1.131	k8so02-emerald	1.8 days
Running	assessment-594894d555-7p525	emerald-dev	emerald-docker-dev-local.artifact.tecnalia.dev/assessment:v0.2.18	1/1	0	10.42.1.203	k8so02-emerald	3.1 days
Running	assessment-d8c6595bc-2p2tq	emerald-pilot2	emerald-docker-dev-local.artifact.tecnalia.dev/assessment:v0.2.18	1/1	0	10.42.4.197	k8so05-emerald	2.2 days

Figure 8. Rancher Dashboard

2.3.6 Certificates

Access to the Dashboard is secure via HTTPS. The certificates are installed using **Cert-Manager**²². **Cert-Manager** automates the provisioning of certificates and provides a set of custom resources to issue certificates and attach them to services. EMERALD secures web apps and APIs with SSL certificates from **Let's Encrypt**²³. We have installed **Cert-Manager** using the manifest file, created an issuer that uses the **Let's Encrypt** API for the Dashboard domain and exposed it over HTTPS.

2.3.7 Deployment view

The EMERALD CaaS framework is deployed on the **Kubernetes** cluster. As we explained in Section 2.3, the cluster is currently composed of six nodes. Figure 9 shows a deployment diagram of the solution, showing all components deployed (in blue). Each component is composed by one or more containers, represented by artifacts (white boxes). The figure also includes additional artifacts: those auxiliary tools needed in the nodes (in brown) and the components integrated in the **GitLab** CI/CD (in green).

²² <https://cert-manager.io/docs/>

²³ <https://letsencrypt.org/>

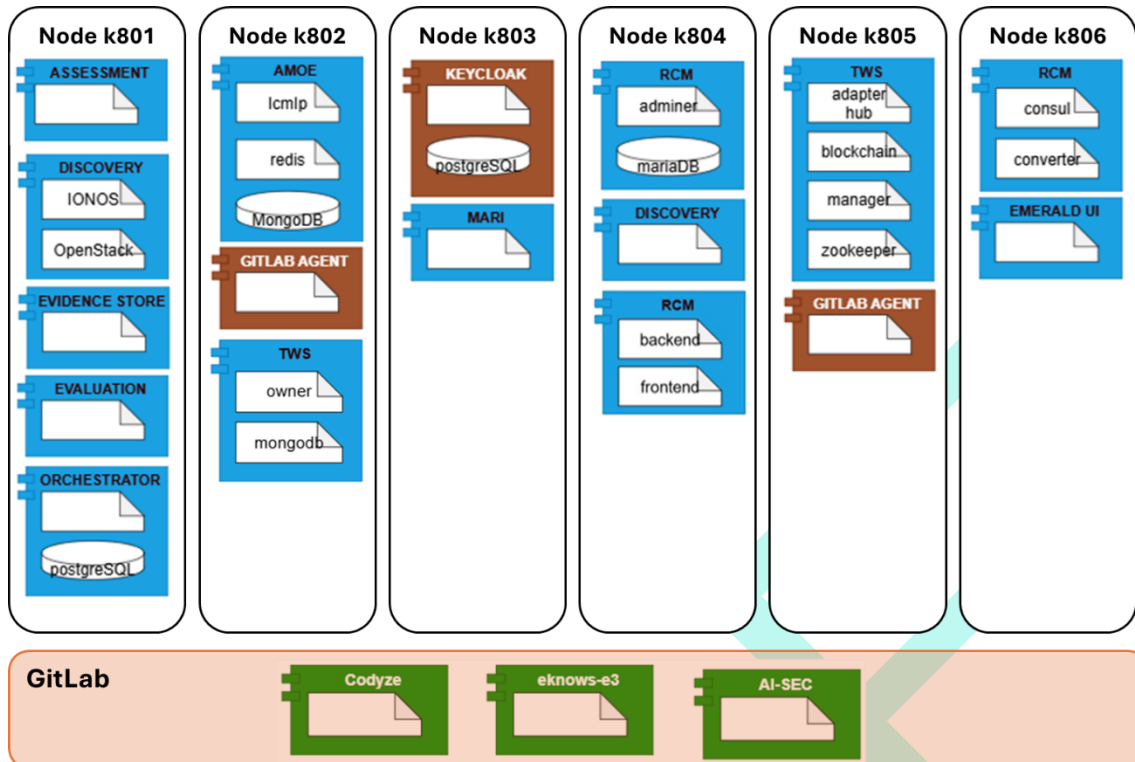


Figure 9. Deployment diagram of EMERALD CaaS framework V3

Figure 9 represents the deployment of one of the environments (Pilot1) at a given time. The distribution of the artifacts among the nodes is managed and automatically modified by Kubernetes attending to its own performance and resources management criteria, and it is possible that a single component has its artifacts distributed on different nodes (as *TWS* and *RCM* in the figure).

2.4 Steps to Integrate a Component

Once the Test Bed environment has been installed and properly configured, the next step is the deployment of all components in the cluster.

To better organize the integration, we have adopted the following methodology, which presents the actions to be taken until the complete release of the EMERALD framework. Figure 10 shows the main steps in the integration and deployment of a component²⁴:

1. The source code of each component must be uploaded to the private *GitLab* repository.
2. Once the source code is uploaded, the build process must complete successfully, and all relevant tests must pass.
3. The component must be containerised into a *Docker* image, so it must provide *dockerfile(s)* in the *GitLab* repository, which help automate the building of the images after any changes in the code.
4. The *Docker* image must be made available on the private *Docker* registry *Artifactory*. The configuration and side services for the component should be specified in the CaaS Framework repository.
5. The configuration must be manually tested to perform standalone, point-to-point, and workflow tests, to verify that each component has been deployed correctly,

²⁴ The integration of non-open-source components skips steps 1 and 2.

communicates with its peers, and the workflows (described in Section 2.1.1) are correctly implemented.

6. If the tests are passed, the release can be merged in the integration environment.
7. Automated integration tests are performed in the integration environment.
8. If the tests are passed, the version is promoted to the production environment, and a new release is created.

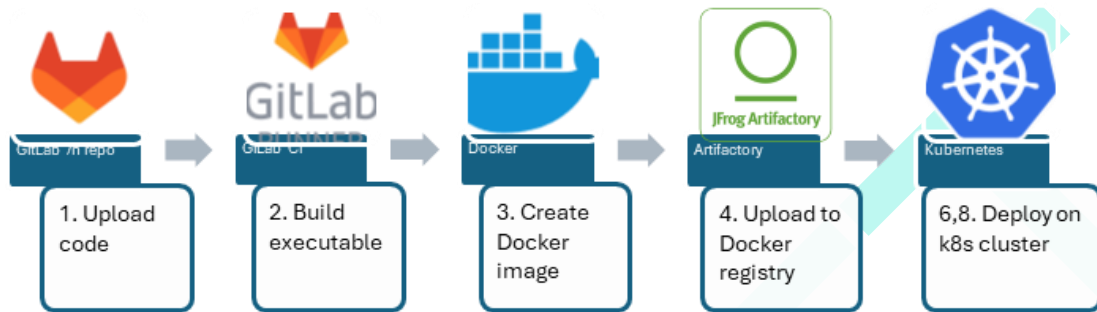


Figure 10. Main steps of the component integration

The integration plan includes three phases to be completed in months M18, M30, and M34, respectively. Currently, we have performed the first, second and third rounds.

To facilitate deployment and integration of components, during the first round, the WP1 lead delivered workshops to the rest of the consortium to introduce the main concepts of *Docker* and *Kubernetes* and explain the defined integration steps. During the second round, WP1 lead also delivered a workshop focusing on the setup of the framework in the pilots.

All components were developed concurrently, and their code history was tracked using the *GitLab* version control system. Besides, a semantic release numbering was implemented to track the progress of the project. All components were containerised and were deployed on the project-internal integration server. *Dockerfile* recipes were made available to easy recreate the integration environment. All REST API endpoints were exposed on a common network to enable communication between components. The EMERALD framework was installed entirely using *Kubernetes* manifests.

In the first round, the components' deployment in the integration environment was carried out manually by each partner. In the second round, we introduced the automated update of new versions of the components. Besides, we also established the pilot environments with a manual deployment. In the third (and last) round, we will implement the promotion of new versions to the pilots' environments based on the fixings performed in the integration environment.

2.5 Overall Status of the Integration

This section provides an overview of the integration status of the components in the EMERALD framework, that is further detailed in Section 3. Table 3 shows the steps to be carried out for the integration of each component, as well as their degree of completion.

Table 3. Integration status

Component	License	Gitlab Repo	Public Repo	README	Docker Images	OpenAPI spec	K8s file	Deployed (integr.)
AMOE	✓ (Open Core, Apache)	✓	✓	✓	✓	✓	✓	✓
MARI	✓ (Open Core)	✓	✓	✓	✓	✓	✓	✓
RCM	✓ (Open Core, Apache)	✓	✓	✓	✓	✓	✓	✓
TWS *	✓ (Proprietary)	N/A	N/A	✓	✓	✓	✓	✓
Assessment	✓ (Open Core, Apache)	✓	✓	✓	✓	✓	✓	✓
Clouditor-Discovery	✓ (Open Core, Apache)	✓	✓	✓	✓	N/A	✓	✓
Evaluation	✓ (Open Core, Apache)	✓	✓	✓	✓	✓	✓	✓
Evidence Store	✓ (Open Core, Apache)	✓	✓	✓	✓	✓	✓	✓
Orchestrator	✓ (Open Core, Apache)	✓	✓	✓	✓	✓	✓	✓
Codyze	✓ (Open Core, Apache)	✓	✓	✓	✓	✓ (CLI)	N/A	N/A
eKnows-e3 *	✓ (Open Core, Apache)	✓	✓ & N/A	✓	✓	✓ (CLI)	N/A	N/A
AI-SEC	✓ (Apache)	✓	✓	✓	✓	N/A	N/A	N/A
Emerald UI	✓ (Open Core, Apache)	✓	✓	✓	✓	✓	✓	✓

The starting point of the integration process is the source code of the components, which has been uploaded to the public *GitLab* repository²⁵ (except for two of them -*TWS* and *eknows-e3*—which are not open source licensed or require some component that is not open-source). The repository of each component contains a *dockerfile*. The respective images created have been uploaded to the *Docker* repository in *Artifactory*.

The next step is to deploy the images on the *Kubernetes* cluster. For this, several manifest files have been developed for each component, depending on their nature (pods, services, volumes, etc.). In the early stages of the development, the integration process allowed a manual deployment using *kustomize* and *kubectl*²⁶, which allows for the identification of bugs/adjustments in an agile way, without waiting for an automated deployment process to be completed. This process has been further automated in the second phase through the *GitLab* CI pipelines, which are detailed in D1.6 [5].

The last column of Table 3 indicates if the component is available in the integration environment. All of them are deployed, excepting some evidence extractors that are not intended to be integrated in the CaaS framework but in the pipeline to analyse files (i.e., *Codyze*, *eknows-e3*, and *AI-SEC*). These extractors are executables that are designed to be run over the assets to be evaluated in order to extract evidence. They are executed in CI/CD runners, where they access the code directly and upload the evidence to the CaaS framework. To facilitate their usage in CI/CD workflows, some of them provide container images that can be instantiated in the CI/CD pipelines.

The final and fundamental part of the integration is the communication among components, which is done through the APIs defined and developed in EMERALD. Besides the details provided in the Section 3 of this document, Table 4 presents a consolidated view of the status of the interaction of each component with the others. The first column of the table (“Component A”) refers to the component that *implements* the API, while the second column (“Component B”) is the component that *invokes* it.

²⁵ <https://git.code.tecnalia.dev/emerald/public>

²⁶ A command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API. See <https://kubernetes.io/docs/reference/kubectl/> for details.

The status has been categorized into several stages²⁷, reflecting the progress made in integrating each component into the EMERALD Framework:

- **Not Started:** The integration process has not yet begun.
- **Developing API:** The component is currently in the process of developing its API. This stage involves defining how the component will interact and its implementation.
- **API Finished:** The API development has been completed and is ready for testing.
- **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:** The component has successfully established connections with other components. Data exchange can occur, but further testing is needed to ensure full compatibility.
- **Testing:** The integration of the component with others is currently being tested. This phase involves checking the data flow between the components to identify and fix any issues that may arise.
- **Integrated:** The component has been fully integrated into the framework. It has passed the necessary tests and is functioning as intended, interacting seamlessly with other components in the EMERALD system.

Table 4. Components' point-to-point integration status

Component A (implementer)	Component B (invoker)	Status	Comment
AI-SEC	EMERALD UI	Integrated	AI-SEC is currently in the tool testing phase. It starts to integrate by connecting to the <i>Evidence Store</i> .
	Evidence Store	Integrated	AI-SEC is currently connected to the <i>Evidence Store</i> and can send evidences to it.
AMOE	Evidence Store	Integrated	-
	RCM	Integrated	-
	Orchestrator	Integrated	-
	EMERALD UI	Integrated	-
Clouditor-Discovery	Evidence Store	Integrated	-
Codyze	[CI/CD] *	Integrated	Codyze is integrated as a CI/CD component.
	Evidence Store	Integrated	-
	TWS	Integrated	-
eknows-e3	[CI/CD] *	Integrated	The CI/CD uses <i>eknows-e3</i> to extract and save evidence in the <i>Evidence Store</i> . A demo showcases how the <i>eknows-e3</i> can be integrated. Integration tests are currently being developed.
	Evidence Store	Integrated	Communication is implemented; integration tests are completed. A demo showcases how the <i>eknows-e3</i> component can be integrated into a <i>GitLab</i> pipeline. Integration tests are completed.
TWS	EMERALD UI	Testing	Testing on-going in pilots.
	Evidence Store	Integrated	-

²⁷ This categorization was first used in deliverable D3.5 [19]

Component A (implementer)	Component B (invoker)	Status	Comment
	<i>Evidence extractors</i>	Integrated	-
<i>MARI</i>	<i>RCM</i>	Integrated	Mapping API operational; mapping results are returned to and stored by the RCM.
<i>RCM</i>	<i>EMERALD UI</i>	Integrated	API has been updated and extended to cover UI requirements
	<i>Orchestrator</i>	Integrated	Input calls to RCM and output calls to the <i>Orchestrator</i> have been tested and integrated.
	<i>MARI</i>	Integrated	Mapping API is finished unless bugs are detected in further tests
	<i>AMOE</i>	Integrated	Metrics information is correctly sent.
<i>Orchestrator</i>	<i>EMERALD UI</i>	Testing	The <i>EMERALD UI</i> is integrated and can use the main <i>Orchestrator</i> endpoints. This integration will be further exercised and refined during the pilot phase, where feedback from real pilot usage may lead to adjustments.
	<i>RCM</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>Assessment</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>Evaluation</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>Evidence Store</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>Evidence Store</i>	<i>Assessment</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>Orchestrator</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>AMOE</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
	<i>Codyze</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
	<i>eknows-e3</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
	<i>AI-SEC</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
	<i>Clouditor-Discovery</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>EMERALD UI</i>	Testing	Additional testing is required; existing approach looks promising. Requires metrics to be integrated in the public repository and updates to the component for final validation.
<i>Assessment</i>	<i>Evidence Store</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>Orchestrator</i>	Integrated	Minor changes may occur in the pilot testing phase.
	<i>TWS</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>Evaluation</i>	<i>Orchestrator</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>EMERALD UI</i>	<i>AMOE</i>	Integrated	-
	<i>Evidence Store</i>	Testing	Additional testing is required

Component A (implementer)	Component B (invoker)	Status	Comment
	<i>Orchestrator</i>	Testing	Additional testing is required
	<i>RCM</i>	Integrated	-
	<i>TWS</i>	Testing	Additional testing is required

(*) CI/CD refers not to a component, but to the GitLab pipelines that are the invokers in these cases.

The information reflected in Table 4 can be taken as a basis for a consolidated status of the point-to-point integration. If “Not started” is considered as 0% progress, and “Integrated” is considered as 100% progress (with “Locally Tested” being 50%), an approximation to the global value can be calculated, resulting in 99% integration completed at M34. The majority of the elements in the table are in the more advanced statuses, i.e. “Testing” and “Integrated” status.

2.5.1 Requirements Summary

This section tracks the status of the technical requirements of the EMERALD framework components, building on the completion of all architecture-related and component-related deliverables foreseen in the project.

Table 5 lists all the technical requirements, along with their priority, timeline²⁸ and completion status. WP1 has managed a total of 61 requirements, covering both functional requirements (associated with individual components) and non-functional ones (more general, associated with WP1). 59 of these requirements fall within the M34 deadline –the submission date of this deliverable–, and 2 of them have the deadline at M36. Completed requirements are written in green, while discarded ones are highlighted in grey. Please note that the requirements relating to the *EMERALD_UI* component have been tracked in WP4 [11].

Table 5. Requirements prioritization matrix and status at M34

Req. ID ²⁹	Title	Priority	Timeline	Status
AI-SEC.01	The extractor tool includes selected criteria	MUST	M24 (C-V2)	100%
AMOE.01	Upload PDF document	MUST	M24 (C-V2)	100%
AMOE.02	Provision of extracted evidence to Evidence Store	MUST	M24 (C-V2)	100%
AMOE.03	Refine evidence extraction approach	MUST	M24 (C-V2)	100%
AMOE.04	Compare results from multiple documents	SHOULD	M12 (C-V1)	100%
AMOE.05	Select metrics per document	SHOULD	M24 (C-V2)	100%
AMOE.06*	Classify document, select respective metrics	COULD	Discarded	-
AMOE.07	Metric states	SHOULD	M24 (C-V2)	100%
CLDISC.01	Discovery of security properties of infrastructure components	MUST	M30 (I-V2)	100%
CODYZE.01	Extraction of security features from source code	MUST	M30 (I-V2)	100%
CODYZE.02	Generate provenance reports on Codyze execution	SHOULD	M30 (I-V2)	100%
CODYZE.03	Extraction of security features from source code	COULD	M30 (I-V2)	100%
EKNOWS.01	Integration into existing systems	MUST	M18 (I-V1)	100%
EKNOWS.02	Resilience while analysing erroneous code	SHOULD	M24 (C-V2)	100%
EKNOWS.03	Support languages required by pilots	MUST	M24 (C-V2)	100%

²⁸ Timeline indicates the due date month (e.g. M24) and, in parentheses, the associated project milestone (e.g. C-V1, I-V1), where C indicates Components, I means Integration, and Vn refers to the corresponding version number.

²⁹ The requirements have been defined in the EMERALD GitLab private repository [Authenticated is required].

Req. ID ²⁹	Title	Priority	Timeline	Status
EKNOWS.04	Support EMERALD evidence format	MUST	M18 (I-V1)	100%
EKNOWS.05	Static code analysis	MUST	M24 (C-V2)	100%
TWS.01	Provide integrity proof of evidence	MUST	M24 (C-V2)	100%
TWS.02	Provide integrity proof of assessment results	MUST	M24 (C-V2)	100%
TWS.03	Provide access through REST API or graphical interface	MUST	M24 (C-V2)	100%
TWS.04	Use a general-purpose public-private Blockchain network	MUST	M24 (C-V2)	100%
TWS.05*	Include suitable documentation	MUST	Discarded	-
TWS.06	Allow evidence integrity proofs from the evidence sources	SHOULD	M24 (C-V2)	100%
TWS.07	Allow confidential recording in the Blockchain	COULD	M30 (I-V2)	100%
TWS.08	Allow access to confidential information in the Blockchain based on trustworthy attributes	COULD	M30 (I-V2)	100%
MARI.01	AI-based	MUST	M30 (I-V2)	100%
MARI.02	Automatic association	MUST	M30 (I-V2)	100%
MARI.03	Performance evaluation	MUST	M30 (I-V2)	100%
MARI.04	Visualization	MUST	M30 (I-V2)	100%
MARI.05*	Strategies	COULD	Discarded	-
RCM.01	Multi-scheme support	MUST	M12 (C-V1)	100%
RCM.02	Accessible by the rest of components	MUST	M18 (I-V1)	100%
RCM.03	Include metrics for all schemes supported	MUST	M24 (C-V2)	100%
RCM.04	Mapping of schemes	SHOULD	M24 (C-V2)	100%
RCM.05	Import/export of security schemes in OSCAL	MUST	M30 (I-V2)	100%
RCM.06	Import/export of security schemes in CSV format	COULD	M24 (C-V2)	100%
RCM.07	Support for personalized catalogues	MUST	M30 (I-V2)	100%
RCM.08	Support updating/versioning of schemes	SHOULD	M30 (I-V2)	100%
RCM.09	Self-assessment questionnaires (EUCS)	COULD	M24 (C-V2)	100%
RCM.10	Mapping of metrics to controls	SHOULD	M24 (C-V2)	100%
RCM.11	Metrics downloading from common repository	COULD	M30 (I-v2)	100%
ORCH.01	Final compliance decision	MUST	M24 (C-V2)	80%
ORCH.02	REST API Gateway for UI	MUST	M30 (I-V2)	100%
ORCH.03	Role Based Access Control	MUST	M30 (I-V2)	100%
ORCH.04*	Manage Tools via API	COULD	Discarded	-
ORCH.05	Provide an API for audit workflow	MUST	M30 (I-V2)	100%
ESTORE.01	Storage of ontology entities in graph database	MUST	M30 (I-V2)	100%
ESTORE.02	Allow Interaction with Third-Party Evidence Collectors	SHOULD	M34 (I-v3)	100%
ASSESS.01	Assessment based on evidence	MUST	M30 (I-V2)	100%
ASSESS.02	Assessment rules for 80% of the defined metrics	MUST	M30 (I-V2)	100%
ASSESS.03	Display cause of assessment result	COULD	M30 (I-V2)	100%
EVAL.01	Display cause of failing evaluation result	COULD	M30 (I-V2)	100%
EVAL.02	Evaluation based on assessment results	MUST	M30 (I-V2)	100%
WP1.01	Performant network	SHOULD	M34 (I-v3)	95%
WP1.02	Portability	SHOULD	M24 (C-V2)	100%
WP1.03	Scalability	SHOULD	M34 (I-v3)	100%
WP1.04	Installability	SHOULD	M34 (I-v3)	100%
WP1.05	Documentation	MUST	M36 (Final)	95%
WP1.06	Agile development methodology	MUST	M12 (C-v1)	100%
WP1.07	Observability	SHOULD	M34 (I-v3)	100%
WP1.08	Security	SHOULD	M36 (Final)	100%

(*) Rationale for discarded requirements:

- **AMOE.06** has been considered out of scope due to limited resources.
- **TWS.05** has been absorbed by the more general requirement **WP1.05-Documentation**.
- **MARI.05** did not address a concrete user need and was therefore considered unnecessary for the intended operational scenarios.
- **ORCH.04** has been considered out of scope due to limited resources.

Excluding the four requirements that were discarded for various reasons and will not be implemented, there are 54 fully implemented requirements in M34. The remaining requirements, as reflected in Figure 11, are “in progress”, with the majority already over 95% complete, and only one remaining below 90%.

The average project completion rate at M34 is 99%.

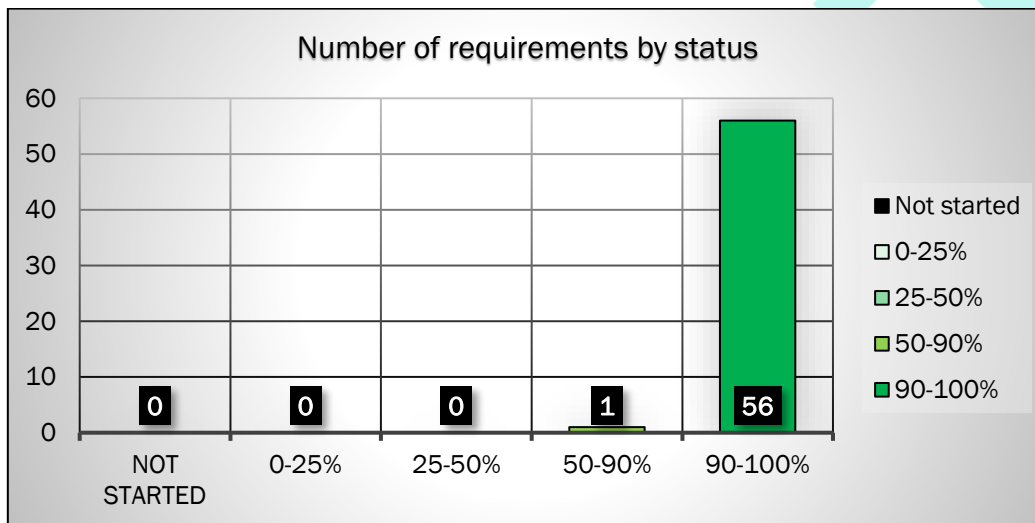


Figure 11. Number of requirements by status at M34

Figure 12 shows the distribution of the progress in the implementation of the requirements, where 2% are between 50% and 90% completion; and **98% of the requirements are above 90% completion.**

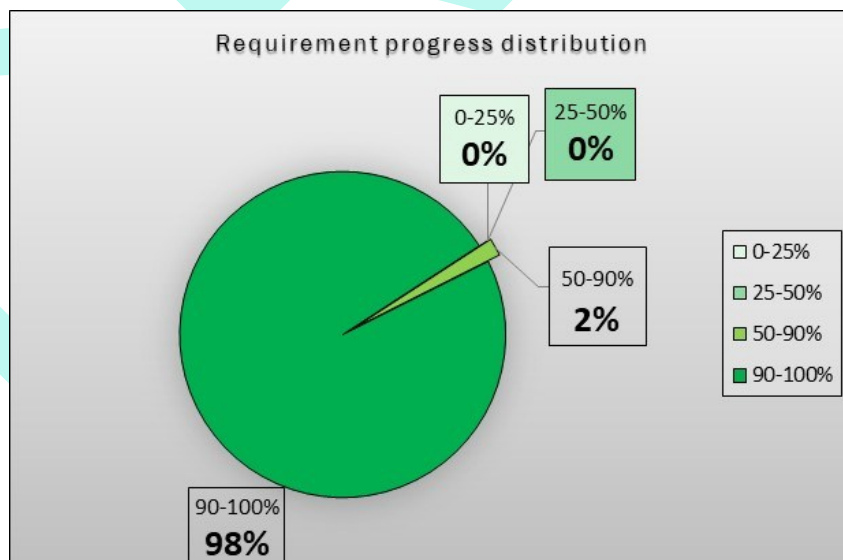


Figure 12. Requirement progress distribution at M34

Looking at the timeline of the requirements, most target dates fall within the final year of the project. This reflects the coarse granularity at which the requirements were defined: since each requirement covers broad functionality, full implementation and the corresponding testing are expected to conclude only towards the project's end. As a result, the few requirements that are still work in progress are very near of completion.

Table 6 summarises the requirement status by component. Each component's completion status reflects the average progress of its individual requirements at the time of writing this deliverable. The table also breaks down requirements by category: 'Discarded', 'Not started', 'Partially implemented' and 'Fully implemented'.

Table 6. Summary of requirements status at M34 (by component)

Component	Discarded	Not started	Partially implemented	Fully implemented	TOTAL	Average Status
AI-SEC	-	-	-	1	1	100%
AMOE	1	-	-	6	7	100%
Discovery	-	-	-	1	1	100%
Codyze	-	-	-	3	3	100%
eknows-e3	-	-	-	5	5	100%
TWS	1	-	-	7	8	100%
MARI	1	-	-	4	5	100%
RCM	-	-	-	11	11	100%
Evidence Store	-	-	-	2	2	100%
Orchestrator	1	-	1	3	5	95%
Assessment	-	-	-	3	3	100%
Evaluation	-	-	-	2	2	100%
NFR (WP1)	-	-	2	6	8	99%
TOTAL	4	0	3	54	61	99%

Note: Requirements marked as 'Discarded' are excluded from the average calculation.

To better grasp the data in the table, Figure 13 provides a graphical representation of it. The lighter parts of the bars reflect the advancement since the previous version v2 (M30 milestone).

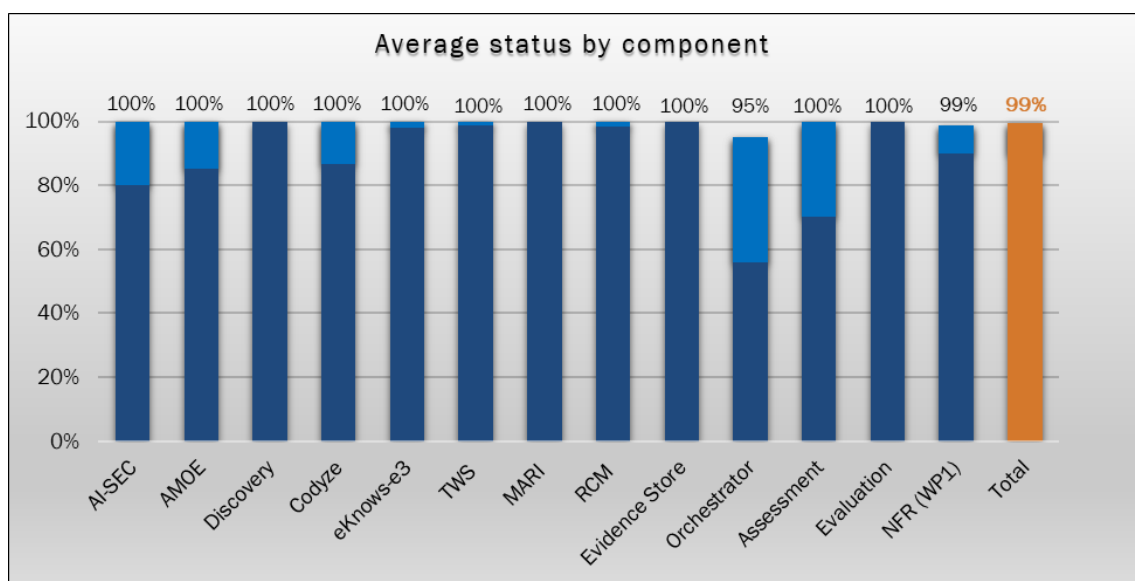


Figure 13. Requirements completion per component

Summarizing, we can state that **eleven components out of thirteen have reached 100% implementation, and the rest three are above 95% completion, being the average implementation of EMERALD components of 99%.**

3 Integration of Components

This section provides more details on the integration status of each EMERALD component. The components are grouped into three groups: (i) Evidence extractors; (ii) Evidence Assessment and Certification; and (iii) User Interface.

Each component is introduced by a short description, followed by the expected behaviour concerning inputs and outputs. Next, the APIs published by the component, or the Command Line Interfaces (CLI), if applicable, are listed. Finally, an update on the status of the integration with other components is provided.

3.1 Evidence Extractors

Evidence extractors –highlighted in the Figure 14 below– are the components in charge of extracting different types of data from the targets of evaluation and providing them as evidence, which is then processed within the EMERALD framework to determine compliance.

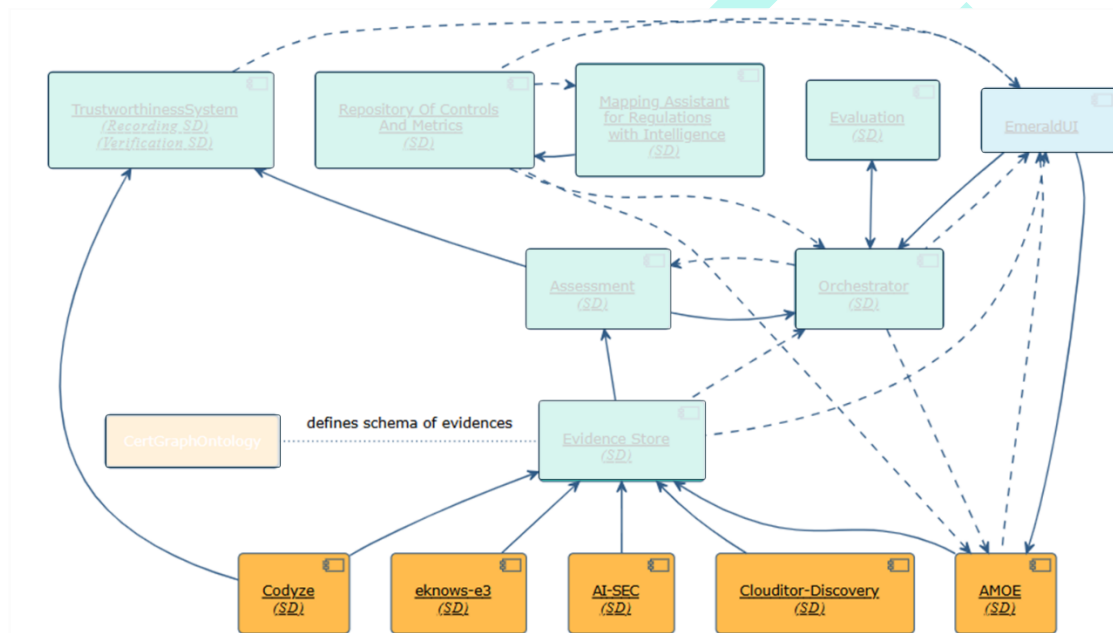


Figure 14. Evidence extractors in the EMERALD framework architecture

3.1.1 AI-SEC

AI Security Evidence Collector (AI-SEC) is an evidence extractor designed to extract relevant information from machine learning models, as stated in D2.7 [6]. Based on the Criteria Catalogue for AI Cloud Services (AIC4) [12], *AI-SEC* extracts various characteristics of machine learning models, e.g. robustness, privacy levels and explainability. *AI-SEC* establishes a process that contains methods for extracting these features. These methods are typically applicable to both image and language models.

3.1.1.1 Expected behaviour (inputs/outputs)

The expected input should be the machine learning model and its (partial) training data, while the output will be the computed evidence.

AI-SEC is connected to the *EMERALD UI* and the *Evidence Store*:

- **EMERALD UI:** It is indirectly connected to *AI-SEC* via the *Evidence Store*. Interaction takes place via the *Evidence Store* API, rather than through a direct integration with the UI.
- **Evidence Store:** The evidence collected by *AI-SEC* will be forwarded to the *Evidence Store*.

3.1.1.2 Published APIs

AI-SEC does not need any published APIs. It only uses the APIs of the components listed above.

3.1.1.3 Integration Status

Table 8 provides the status of the connection to the *EMERALD UI* and the *Evidence Store*.

Table 7. Integration status of *AI-SEC* with other EMERALD components

Component	Status	Comment
<i>EMERALD UI</i>	Integrated	The <i>EMERALD UI</i> is connected indirectly with <i>AI-SEC</i> via the <i>Evidence Store</i> .
<i>Evidence Store</i>	Integrated	<i>AI-SEC</i> is currently connected to the <i>Evidence Store</i> and can send evidence to it.

3.1.2 AMOE

Assessment and Management of Organizational Evidence (AMOE) is an evidence extractor that extracts relevant parts of policy documents, as described in D2.5 [7]. Based on the meta data provided by the security metrics stored in the *RCM*, *AMOE* uses natural language processing techniques and question answering to process the documents. *AMOE* offers an API to upload documents and collect the processed outputs. Furthermore, the extracted results can be reviewed and forwarded to the EMERALD system. *AMOE* will be controlled via user actions in the *EMERALD UI*, which connects to the *AMOE* API.

3.1.2.1 Expected behaviour (inputs/outputs)

AMOE is connected to the *EMERALD UI*, the *Evidence Store*, the *Orchestrator* and the *RCM*.

- **RCM:** *AMOE* retrieves data as input from the *RCM* (e.g. security metrics, security controls).
- **Orchestrator:** *AMOE* also retrieves input from the metric Implementation (such as custom target values) and information about Cloud Services (audit scope, target of evaluation) from the *Orchestrator*.
- **Evidence Store:** After a human review, the confirmed evidence results are forwarded to the *Evidence Store*.
- **EMERALD UI:** The *EMERALD UI* connects to *AMOE* to upload, download, view, review, or delete policy documents and extracted evidence. *AMOE* will be controlled by a user via the *EMERALD UI*, which connects to the API.

3.1.2.2 Published APIs

The API endpoints of *AMOE* are listed below.

evidence ^

DELETE	/api/v2/evidence/	AMOE Delete Evidence	▼
GET	/api/v2/evidence/	AMOE Get Evidence	▼
PUT	/api/v2/evidence/assessment	AMOE Set Assessment Result	▼
GET	/api/v2/evidence/file/	AMOE Get HTML File	▼
GET	/api/v2/evidence/list/	AMOE Get List Evidence For File	▼
POST	/api/v2/evidence/list_per_metric_id	AMOE Get List Evidence Per Metric	▼
PUT	/api/v2/evidence/send_to_orchestrator/	AMOE Send Assessment Result	▼

file ^

GET	/api/v2/file/	AMOE Get File	▼
PUT	/api/v2/file/	AMOE Upload PDF File	▼
DELETE	/api/v2/file/delete/	AMOE Delete File And Evidence	▼
GET	/api/v2/file/get_metric_ids_for_file		▼
GET	/api/v2/file/html		▼
GET	/api/v2/file/htmlonly		▼
GET	/api/v2/file/is_evidence_extraction_stopped		▼
GET	/api/v2/file/last/	AMOE Get last file	▼
GET	/api/v2/file/pdf/	AMOE Get PDF File	▼
PUT	/api/v2/file/start_evidence_extraction	AMOE Start evidence extraction for list of metric_ids	▼
PUT	/api/v2/file/stop_evidence_extraction		▼
PUT	/api/v2/file/upload_files		▼

files ^

GET	/api/v2/files/	AMOE List Files Cloud Service	▼
POST	/api/v2/files/	AMOE List Files Cloud Services	▼

Listing 1. AMOE API overview

3.1.2.3 Integration Status

AMOE has been deployed on the *Kubernetes* cluster. Table 8 provides the current status of the connection to the *EMERALD UI*, the *Evidence Store*, the *Orchestrator* and the *RCM*. AMOE is ready to be adapted to the new APIs of the different components once they are implemented.

Table 8. Integration status of AMOE with other EMERALD components

Component	Status	Comment
<i>Evidence Store</i>	Integrated	-
<i>RCM</i>	Integrated	-
<i>Orchestrator</i>	Integrated	-
<i>EMERALD UI</i>	Integrated	-

3.1.3 Clouditor-Discovery

The *Clouditor-Discovery* is one of the evidence extractors in the EMERALD CaaS framework. As described in D2.9 [8], the *Clouditor-Discovery* is responsible for discovering security-related configurations from cloud resources, including Virtual Machines, Object Storage, and Network interfaces. It is implemented for multiple CSPs like *Azure*, *AWS* and *Kubernetes*. An implementation for *OpenStack* and *IONOS* is now available as well. The collected runtime information is mapped to the EMERALD evidence format and stored in the *Evidence Store*.

3.1.3.1 Expected behaviour (inputs/outputs)

The *Clouditor-Discovery* has only one connection, to the *Evidence Store*:

- **Evidence Store:** It receives the evidence with the security properties from the *Clouditor-Discovery*.

3.1.3.2 Published APIs

The *Clouditor-Discovery* does not have any published API, nor any CLI command defined, as it starts directly upon deployment.

3.1.3.3 Integration Status

The *Clouditor-Discovery* has already been deployed on the *Kubernetes* cluster for the discovery of *OpenStack (IaaS)*, *OpenStack (PaaS)* and *IONOS*. Table 9 provides the current status of the connection to the *Evidence Store*.

Table 9. Integration status of Clouditor-Discovery with other EMERALD components

Component	Status	Comment
<i>Evidence Store</i>	Integrated	-

3.1.4 Codyze

Codyze is a tool suite consisting of *Codyze-Compliance* and *Codyze-Provenance*. *Codyze-Compliance* is a static software analysis tool that uses a code property graph (CPG) to represent code properties as a language agnostic graph. Based on this graph representation, presence or absence of specified code properties can be tested [9]. *Codyze-Provenance* verifies through user-defined queries if the source code complies to security metrics [8].

Codyze-Provenance supports *Codyze-Compliance* by generating provenance and attestation reports. These reports link inputs, such as source code files, to outputs, such as build artefacts,

in a forgery-proof manner. As a result, they enforce traceability from source to artefact with a strong link to evidence.

3.1.4.1 Expected behaviour (inputs/outputs)

Codyze integrates with a CI/CD system responsible for deploying new and updated cloud services. Therefore, *Codyze* uses the source code repository of a cloud service as its input. Through this repository, *Codyze-Compliance* has access to the source code that needs to be evaluated for compliance. *Codyze-Provenance* integrates with the CI/CD definition through a configuration-as-code approach, where CI/CD pipelines are configured through files hosted together with the source code.

Codyze connects to:

- the **Evidence Store**: The generated pieces of evidence are sent to the *Evidence Store*. They are annotated with the terms of the ontology enabling assessment by the *Assessment*, and
- the **TWS**: The generated pieces of evidence are sent to the *TWS* to facilitate additional, independent validation of evidence.

3.1.4.2 Published CLIs

Codyze uses CLI based tools. These tools are executed as part of a CI/CD pipeline execution. Common CLI parameters and options are the following:

```
--tool-id <string>           ID of the tool
--evidence-store-url <url>    URL of the Evidence Store
--trustworthiness-system-url <url> URL of the TWS
--oauth-url <url>            URL to an OAuth endpoint for
                             authentication
--oauth-client-id <string>    OAuth client ID
--oauth-client-secret <string> OAuth client secret

<targetid>                  ID of the ToE
```

3.1.4.3 Integration Status

Codyze is integrated as a CI/CD component in *GitLab* and has connections to the *Evidence Store* and *Trustworthiness System*.

Table 10. Integration status of *Codyze* with other EMERALD components

Component	Status	Comment
<i>Evidence Store</i>	Integrated	
<i>TWS</i>	Integrated	

3.1.5 eknows-e3

The *eknows-e3* evidence extractor is based on the software analysis platform *eknows*³⁰ and delivers required evidence to verify if an application source code complies to security metrics. As described in D2.3 [9], *eknows-e3*, in contrast to *Codyze*, reuses prefabricated parsing, analysis, and generation modules of the *eknows* platform and supports multi-language static codes analysis.

³⁰ <https://www.scch.at/software-science/projekte/detail/eknows>

The cornerstone of its implementation is a generic programming language-independent representation of source code that can be reused across analysis and generation modules to prepare suitable security-related evidence. Thereby, generic modules for the model-guided symbolic execution of use case-specific conformity checks and fact extraction are extended. New analyses have been added to break down high-level security controls from catalogues, such as EUCS or BSI C5, into checkable source code properties, and new generation functions to create evidence based on these source code properties and to integrate them into the ontology [13] have been provided.

3.1.5.1 Expected behaviour (inputs/outputs)

The static analyser component sends and receives information to and from different sources:

- **Source code repositories:** *eknows-e3* obtains technical evidence from the analysis of the source code and code-related artifacts (e.g. build system configuration and metadata from version control systems) of Cloud applications. It is integrated in a CI/CD pipeline at the customer side, which establishes the connection to relevant source code repositories.
- **Evidence Store:** Extracted evidence is mapped to the EMERALD evidence format using the terms described in the Ontology. This evidence information represents raw evidence and is delivered to the *Evidence Store*, where it is stored according to the defined schema.

3.1.5.2 Published CLIs

The *eknows-e3* component supports a CLI to start and configure the analysis process (e.g., endpoint for evidence, authentication, file(s) to analyse, etc.):

```
usage: eknows-evidence-extractor
      --clouditor.clientId <arg>
      --clouditor.clientSecret <arg>
      --clouditor.storeUrl <arg>
      --clouditor.tokenUrl <arg>
      --evidence.targetOfEvaluationId <arg>
      --evidence.toolId <arg>
      --evidence.toolName <arg>
      -sd,--srcDir <arg>
      -jv,--javaVersion <arg>
      -bt,--buildTool <arg>
      -r,--remote <arg>
      -b,--branch <arg>
```

In addition to command-line execution, the *eknows-e3* component can also be integrated and triggered directly within a *GitLab* CI/CD pipeline. The component can be included and configured as follows:

```
include:
  - component:
      "git.code.tecnalia.dev/emerald/public/components/eknows/eknows-3/eknows-e3@main"
  - inputs:
      oauth-url: "http://clouditor:8080/v1/auth/token"
      oauth-client-id: "confirmate"
      oauth-client-secret: "confirmate"
      evidence-collector-url: "http://confirmate:8080/"
      target-id: "00000000-0000-0000-0000-000000000000"
      src-dir-path: "testdir"
```

```
java-version: "17"
```

3.1.5.3 Integration Status

The CI/CD component uses *eknows-e3* to extract and store evidence in the *Evidence Store*. *eknows-e3* uses almost the same set of input parameters as the *Codyze-Provenance* component. It runs in a *Docker* container using the image which was built and pushed to *Artifactory*. Since the component pulls out this image from the private EMERALD *Artifactory*, authentication is needed. To do this, we generate JSON Base64 Authentication and set it as a variable `DOCKER_AUTH_CONFIG` in the repository we want to test. Table 11 provides the current status of the connection to the *Evidence Store*.

Table 11. Integration status of *eknows-e3* with other EMERALD components

Component	Status	Comment
<i>Evidence Store</i>	Integrated	Communication is implemented; integration tests are completed. A demo showcases how the <i>eknows-e3</i> component can be integrated into a <i>GitLab</i> pipeline. Integration tests are completed.

3.2 Evidence Assessment and Certification

Evidence Assessment and Certification components –highlighted in Figure 15– are used for storing, assessing and evaluating evidence. This group comprises the *Orchestrator*, which controls the overall workflow, as well as complementary tools like the *Repository of Control and Metrics (RCM)*, the *Mapping Assistant for Regulations with Intelligence (MARI)*, the *Evidence Store*, the *Assessment*, the *Evaluation* and the *Trustworthiness System (TWS)*.

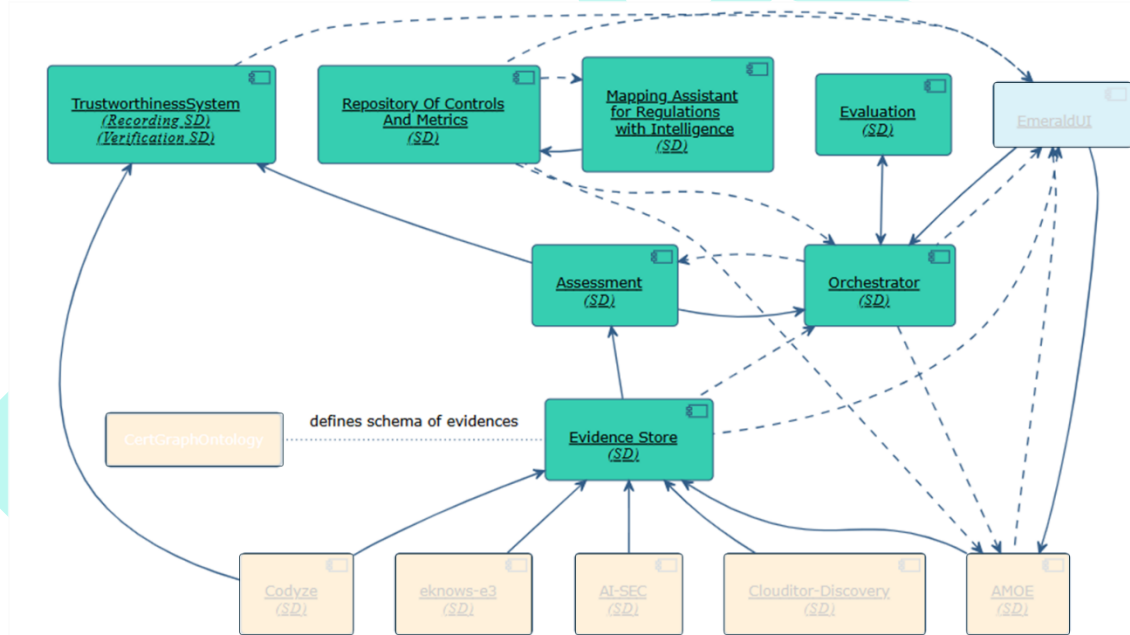


Figure 15. Evidence assessment and Certification tools in the EMERALD architecture

3.2.1 TWS

The *Trustworthiness System (TWS)* ensures the trustworthiness, fairness, and transparency of the evidence and assessment results stored in EMERALD, safeguarding their integrity and authenticity. As described in D3.4 [10], Its primary function is to facilitate secure registration and

verification of proofs of integrity related to evidence and assessment results, both from the *Evidence Store* as well as directly from the evidence sources.

The *TWS* is supported by a blockchain network to ensure information security features like integrity, trustworthiness, and transparency. To enhance usability, a *Blockchain Viewer* is also included, making the system accessible to non-technicians. Additionally, an automatic evidence verification service has been integrated to improve automation and facilitate seamless integration within the EMERALD solution, eliminating the need for manual interaction.

3.2.1.1 Expected behaviour (inputs/outputs)

The *TWS* sends and receives information from different sources:

- **Assessment:** The interaction with this component takes place in two ways:
 - The *Assessment* component provides information (proofs of integrity) related to evidence and assessment results to be recorded on the Blockchain.
 - The automatic verification service requests the current values of evidence and assessment results stored in EMERALD's internal evidence storage to validate their integrity against the information previously recorded on the Blockchain.
- **Evidence extractors:** Proofs of integrity for evidence can be directly provided from the evidence extractors. In particular, *Codyze* has been considered as a proof of concept.
- **EMERALD UI:** The graphical interface of the *TWS* automatic verification service is integrated into the *EMERALD UI*, allowing auditors to easily verify the trustworthiness of evidence and assessment results, and determine their reliability.

3.2.1.2 Published APIs

The API endpoints of the *TWS* are listed below.

MANAGER#ADMIN			^
GET	/manager/admin/count	Get total number of admins	🔒 ▼
GET	/manager/admin/{address}/isadmin	Check if an address is admin	🔒 ▼
POST	/manager/admin	Add a new admin	🔒 ▼
DELETE	/manager/admin/{address}	Remove an admin by address	🔒 ▼
MANAGER#OWNER			^
GET	/manager/owner/count	Returns the number of all owners	🔒 ▼
GET	/manager/owner/{address}/isowner	Returns if given address is owner or not	🔒 ▼
POST	/manager/owner	add a owner	🔒 ▼
DELETE	/manager/owner/{address}	Remove owner	🔒 ▼
MANAGER#ORCHESTRATOR			^
GET	/manager/orchestrator	Retrieve the orchestrator address associated with the authenticated user	🔒 ▼
GET	/manager/orchestrator/{owner}	Get orchestrator address by Ethereum owner address	🔒 ▼

Listing 2. TWS API Endpoints for Account Management

OWNER#ORCHESTRATOR			^
GET	/orchestrator	Get complete orchestrator information	🔒 ✓
GET	/orchestrator/isRegistered	Check if orchestrator is registered	🔒 ✓
GET	/orchestrator/owner	Get orchestrator owner address	🔒 ✓
GET	/orchestrator/creationtime	Get orchestrator contract creation time	🔒 ✓
GET	/orchestrator/address	Get orchestrator contract address	🔒 ✓

Listing 3. TWS API Endpoints for evidence sources management

OWNER#EVIDENCE			^
POST	/evidence	Register a new evidence	🔒 ✓
GET	/evidence	List all evidences	🔒 ✓
GET	/evidence/{id}	Get evidence by ID	🔒 ✓
PUT	/evidence/{id}/verifybyhash	Verify evidence hash	🔒 ✓
PUT	/evidence/{id}/verify	Verify evidence using stored data	🔒 ✓

Listing 4. TWS API Endpoints Evidence Management

OWNER#ASSESSMENT			^
POST	/assessment	Register a new assessment	🔒 ✓
GET	/assessment	List all registered assessments	🔒 ✓
GET	/assessment/{id}	Get assessment by ID	🔒 ✓
PUT	/assessment/{id}/verify	Verify assessment against a provided hash	🔒 ✓
PUT	/assessment/{id}/complianceverify	Verify assessment compliance using the provided hash	🔒 ✓

Listing 5. TWS API Endpoints for Assessments Results Management

3.2.1.3 Integration Status

Currently, the TWS has been successfully deployed on the *Kubernetes* cluster. It uses the *Alastria* Blockchain network. Table 12 provides the current state of the connections of the TWS with other components.

Table 12. Integration status of TWS with other EMERALD components

Component	Status	Comment
EMERALD UI	Testing	Testing is on-going. Changes in the API might be necessary.
Evidence Store	Integrated	

Component	Status	Comment
Evidence extractors (Codyze)	Integrated	

3.2.2 MARI

The *Mapping Assistant for Regulations with Intelligence (MARI)* is an AI-based component for compliance management. As described in D3.4 [10], its main functionality is to automatically associate relevant technical metrics with security controls and to identify semantically related controls across different certification schemes. The current version extends the initial zero-shot implementation through domain adaptation and fine-tuning of Sentence Transformer models. The training process uses a domain-specific corpus containing control-to-control and control-to-metric semantic pairs. Controls, candidate controls, and metrics are represented in a shared embedding space and ranked according to their semantic relevance. The approach was evaluated on independent control-to-metric and cross-standard control-to-control test sets using nDCG@10, with the fine-tuned models outperforming their corresponding zero-shot baselines.

3.2.2.1 Expected behaviour (inputs/outputs)

MARI exchanges information exclusively with the *RCM*.

- **RCM:** The *RCM* sends mapping requests to *MARI*, including information about the source and target certification schemes and the available technical metrics. *MARI* computes ranked control-to-control or control-metric associations and returns the results to the *RCM*, where they are stored and made available for further use.

3.2.2.2 Published APIs

MARI provides two POST endpoints for mapping controls and metrics: the `/mapControls` endpoint, which maps controls from one certification scheme to semantically related controls in another scheme; and the `/mapMetrics2Controls` endpoint, which associates technical metrics with the corresponding controls. Both endpoints use a domain-adapted Sentence Transformer model to score and rank candidate associations. The returned results can be filtered according to the configured similarity threshold.

Mapping

POST	<code>/mapControls</code>	Map controls between two schemas	^
POST	<code>/mapMetrics2Controls</code>	Map metrics to controls	^

Listing 6. MARI API Endpoints for mapping

3.2.2.3 Integration Status

MARI has been deployed on the Kubernetes cluster. It interacts exclusively with the *RCM* through a predefined API. Table 13 provides the current state of connections of *MARI* with the *RCM*.

Table 13. Integration status of MARI with other EMERALD components

Component	Status	Comment
RCM	Integrated	Mapping API operational; mapping results are returned to and stored by the <i>RCM</i> .

3.2.3 RCM

The *Repository of Control and Metrics (RCM)* is a smart catalogue of controls and metrics. As described in D3.4 [10], the *RCM* supports multi-scheme and multi-level compliance and incorporates the definition of the metrics used in EMERALD to obtain and assess evidence.

The *RCM* also provides mechanisms to update the catalogues and allow OSCAL-based [14] import/export to facilitate the reuse and composition of the catalogue elements; stores the mapping of controls and metrics provided by the *MARI* component; and includes a self-assessment questionnaire to assess EUCS [15] compliance.

3.2.3.1 Expected behaviour (inputs/outputs)

The *RCM* sends and receives information from different sources.

- **Orchestrator:** The *Orchestrator* collects information from the *RCM* about schemes and metrics, which is then used to configure extractors and organize evidence. The *RCM* also obtains the assessment results required for the questionnaire utility from the *Orchestrator*.
- **MARI:** *MARI* receives the mapping requests, which include information about schemes and metrics, from the *RCM*. The responses (mappings) are then sent back to the *RCM*, which stores them for further use.
- **EMERALD UI:** When the user navigates through the content of the repository, the *EMERALD UI* calls the *RCM* API. The information required is packed in JSON format in the REST call and sent to the *EMERALD UI* for displaying. The same happens, when the user creates new security schemes or fills in the self-assessment questionnaire.
- **AMOE:** *AMOE* receives the definition of the security metrics used to extract and evaluate evidence from policy documents from the *RCM*.

3.2.3.2 Published APIs

The API endpoints of the *RCM* component are listed below. The API has been extended to include more specific endpoints, primarily in response to user requests as the design and development of the graphical user interface progressed.

framework-resource

PUT	/api/frameworks/{uuid}	Update (full) a Framework.	▼
PATCH	/api/frameworks/{uuid}	Update (partial) a Framework.	▼
GET	/api/frameworks	Retrieve all the existing Frameworks.	▼
POST	/api/frameworks	Create a new Framework.	▼
GET	/api/frameworks/{id}	Retrieve a single Framework.	▼
DELETE	/api/frameworks/{id}	Remove an existing Framework.	▼
GET	/api/frameworks/full	Retrieve the complete information of all the existing Frameworks.	▼

Listing 7. RCM API Endpoints for schema information

control-resource

PUT	/api/controls/{uuid}	Update (full) a Control.	▼
PATCH	/api/controls/{uuid}	Update (partial) a Control.	▼
GET	/api/controls	Retrieve all the existing Controls.	▼
POST	/api/controls	Create a new Control.	▼
GET	/api/controls/{id}	Retrieve a single Control.	▼
DELETE	/api/controls/{id}	Remove an existing Control.	▼
GET	/api/controls/map-controls-to-metrics	Perform the mapping of Metrics (for all the Frameworks).	▼
GET	/api/controls/map-controls-to-metrics-test	Perform the mapping of Metrics (Test Mode).	▼
GET	/api/controls/map-controls-to-metrics-restricted	Perform the mapping of Metrics (Restricted Mode).	▼
GET	/api/controls/map-controls-to-controls	Perform the mapping of Similar Controls (for all the Frameworks).	▼

*Listing 8. RCM API Endpoints for control information***metric-resource**

PUT	/api/metrics/{uuid}	Update (full) a Metric.	▼
PATCH	/api/metrics/{uuid}	Update (partial) a Metric.	▼
GET	/api/metrics	Retrieve all the existing Metrics.	▼
POST	/api/metrics	Create a new Metric.	▼
GET	/api/metrics/{id}	Retrieve a single Metric.	▼
DELETE	/api/metrics/{id}	Remove an existing Metric.	▼
GET	/api/metrics/related-controls/{id-metric}	Retrieve the related Controls mapped to a Metric.	▼

Listing 9. RCM API Endpoints for Metric information

similar-control-resource			^
PUT	/api/similar-controls/{uuid}	Update (full) a Similar Control.	▼
PATCH	/api/similar-controls/{uuid}	Update (partial) a Similar Control.	▼
GET	/api/similar-controls	Retrieve all the existing Similar Controls.	▼
POST	/api/similar-controls	Create a new Similar Control.	▼
POST	/api/similar-controls/multiple	Create multiple Similar Controls.	▼
GET	/api/similar-controls/{id}	Retrieve a single Similar Control.	▼
DELETE	/api/similar-controls/{id}	Remove an existing Similar Control.	▼
GET	/api/similar-controls/formatted	Retrieve all the existing Similar Controls (formatted).	▼

Listing 10. RCM API Endpoints for similar controls

related-metric-resource			^
PUT	/api/related-metrics/{uuid}	Update (full) a Related Metric.	▼
PATCH	/api/related-metrics/{uuid}	Update (partial) a Related Metric.	▼
GET	/api/related-metrics	Retrieve all the existing Related Metrics.	▼
POST	/api/related-metrics	Create a new Related Metric.	▼
GET	/api/related-metrics/{id-metric}	Retrieve all mapped Controls for a given Metric.	▼
DELETE	/api/related-metrics/{id-metric}	Remove an existing Related Metric.	▼
GET	/api/related-metrics/map/{id-control}	Retrieve all mapped Metrics for a given Control.	▼

Listing 11. RCM API Endpoints for related metrics

questionnaire-resource			^
GET	/api/questionnaires	Retrieve all the existing Questionnaires.	▼
POST	/api/questionnaires	Create a new Questionnaire.	▼
PATCH	/api/questionnaires/{uuid}	Update (partial) a Questionnaire.	▼
GET	/api/questionnaires/{id}	Retrieve a single Questionnaire.	▼
DELETE	/api/questionnaires/{id}	Remove an existing Questionnaire.	▼
GET	/api/questionnaires/report/{id}	Generate the report for a given Questionnaire.	▼
GET	/api/questionnaires/compliancefull/{id}	Retrieve the compliance level (by Categories) of a given Questionnaire.	▼
GET	/api/questionnaires/compliance/{id}	Retrieve the compliance level of a given Questionnaire.	▼

Listing 12. RCM API Endpoints for Questionnaire resources

3.2.3.3 Integration Status

The RCM has been deployed on the *Kubernetes* cluster. Table 14 provides the current state of connections of the RCM with other components.

Table 14. Integration Status of the RCM with other EMERALD components

Component	Status	Comment
EMERALD UI	Integrated	API has been updated and extended to cover UI requirements.
Orchestrator	Integrated	Input calls to RCM are already integrated. Output call to the <i>Orchestrator</i> has been tested and assessment results are received.
MARI	Integrated	Mapping API is finished unless bugs are detected in further tests.
AMOE	Integrated	Metrics information is correctly sent.

3.2.4 Orchestrator

The *Orchestrator* is the central orchestration point in the EMERALD framework. As described in D3.4 [10], the *Orchestrator* serves as a key element that manages the compliance process within the EMERALD framework, linking various components together. This component is also responsible for making the final compliance decision, assessing whether a target of evaluation adheres to a specified security standard.

3.2.4.1 Expected behaviour (inputs/outputs)

The *Orchestrator* sends and receives information from different sources.

- **EMERALD UI:** It receives information from the *Orchestrator* to be displayed to the user, e.g. audit scope or evidence.
- **RCM:** It provides the *Orchestrator* with catalogue and metric information and receives assessment results.

- **Assessment:** Assessment results, as well as evidence, are sent by the *Assessment* to the *Orchestrator*.
- **Evaluation:** For evaluating the compliance of controls of a security catalogue, the *Orchestrator* sends assessment results to the *Evaluation* and gets the compliance status of each control (i.e. the evaluation result).
- **Evidence Store:** The *Orchestrator* pulls evidence from it.

3.2.4.2 Published APIs

The API endpoints of the *Orchestrator* for handling assessment results and tools are listed below.

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}	▼

Listing 13. Orchestrator API endpoints for assessment results

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

Listing 14. Orchestrator API endpoints for audit scopes

GET	/v1/orchestrator/audit_trail_events	▼
-----	-------------------------------------	---

Listing 15. Orchestrator API endpoint for audit trail events

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}/category/{categoryName}	▼

Listing 16. Orchestrator API endpoints for catalogues

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

Listing 17. Orchestrator API endpoints for handling certificates

GET	/v1/orchestrator/public/certificates	▼
-----	--------------------------------------	---

Listing 18. Orchestrator API endpoints for certificates (publicly available)

GET	/v1/orchestrator/controls	▼
GET	/v1/orchestrator/controls/{controlId}	▼
GET	/v1/orchestrator/controls_in_scope	▼
POST	/v1/orchestrator/controls_in_scope	▼
GET	/v1/orchestrator/controls_in_scope/{id}	▼
PUT	/v1/orchestrator/controls_in_scope/{id}	▼
DELETE	/v1/orchestrator/controls_in_scope/{id}	▼
POST	/v1/orchestrator/controls_in_scope/{id}/transition	▼

Listing 19. Orchestrator API endpoints for controls

GET	/v1/orchestrator/evaluation_results	▼
POST	/v1/orchestrator/evaluation_results	▼

Listing 20. Orchestrator API endpoints for evaluation results

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	▼

Listing 21. Orchestrator API endpoints for metrics

GET	/v1/orchestrator/runtime_info	▼
-----	-------------------------------	---

Listing 22. Orchestrator API endpoint for runtime info

GET	/v1/orchestrator/targets_of_evaluation	▼
POST	/v1/orchestrator/targets_of_evaluation	▼
GET	/v1/orchestrator/targets_of_evaluation/statistics	▼
PUT	/v1/orchestrator/targets_of_evaluation /{audit_scope.target_of_evaluation_id}/audit_scopes /{audit_scope.catalog_id}	▼
PUT	/v1/orchestrator/targets_of_evaluation /{configuration.target_of_evaluation_id}/metric_configurations /{configuration.metric_id}	▼
GET	/v1/orchestrator/targets_of_evaluation/{targetOfEvaluationId}	▼
DELETE	/v1/orchestrator/targets_of_evaluation/{targetOfEvaluationId}	▼
GET	/v1/orchestrator/targets_of_evaluation/{targetOfEvaluationId} /metric_configurations	▼
GET	/v1/orchestrator/targets_of_evaluation/{targetOfEvaluationId} /metric_configurations/{metricId}	▼
PUT	/v1/orchestrator/targets_of_evaluation/{target_of_evaluation.id}	▼

Listing 23. Orchestrator API endpoints for targets of evaluation

GET	/v1/users	✓
GET	/v1/users/me	✓
GET	/v1/users/permissions	✓
GET	/v1/users/permissions/{filter.object_type}/{filter.object_id}	✓
GET	/v1/users/permissions/{filter.object_type}/{filter.object_id}/users/{filter.user_id}	✓
DELETE	/v1/users/permissions/{objectType}/{objectId}/users/{userId}	✓
PUT	/v1/users/permissions/{user_permission.object_type} /{user_permission.object_id}/users/{user_permission.user_id} /{user_permission.permission}	✓
GET	/v1/users/roles	✓
GET	/v1/users/{userId}	✓
DELETE	/v1/users/{userId}	✓

Listing 24. Orchestrator API endpoints for users

3.2.4.3 Integration Status

The *Orchestrator* has already been deployed on the *Kubernetes* cluster and is interacting with the other EMERALD components it is supposed to work with. Table 15 provides the current state of connections of the *Orchestrator* with other components.

Table 15. Integration status of Orchestrator with other EMERALD components

Component	Status	Comment
EMERALD UI	Testing	The EMERALD UI is connected 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.
RCM	Integrated	Minor changes may occur in the pilot testing phase.
Assessment	Integrated	Minor changes may occur in the pilot testing phase.
Evaluation	Integrated	Minor changes may occur in the pilot testing phase.
Evidence Store	Integrated	Minor changes may occur in the pilot testing phase.

3.2.5 Evidence Store

The *Evidence Store* serves as a central repository for evidence collected from various evidence extractors. It retrieves evidence from the evidence extractors, saves them in a *Postgres* database, and forwards evidence to the *Assessment* and the *TWS* to improve the integrity of the evidence. A detailed description can be found in the deliverables D3.4 [10].

3.2.5.1 Expected behaviour (inputs/outputs)

The *Evidence Store* sends and receives information from different sources:

- **Assessment:** It receives evidence from the *Evidence Store* for the assessment.
- **Orchestrator:** It receives evidence from the *Evidence Store* (and forwards it to the *EMERALD UI*)
- **AMOE:** It sends evidence to the *Evidence Store* for storage.
- **Codyze:** It sends evidence to the *Evidence Store* for storage.
- **eknows-e3:** It sends evidence to the *Evidence Store* for storage.
- **AI-SEC:** It sends evidence to the *Evidence Store* for storage.
- **Clouditor-Discovery:** It sends evidence to the *Evidence Store* for storage.
- **EMERALD UI:** It receives evidence from the *Evidence Store* to display.

3.2.5.2 Published APIs

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

EvidenceStore Manages the storage of evidences.

POST	/v1/evidence_store/evidence	▼
GET	/v1/evidence_store/evidences	▼
GET	/v1/evidence_store/evidences/{evidenceId}	▼
GET	/v1/evidence_store/resources	▼
GET	/v1/evidence_store/supported_resource_types	▼
GET	/v1/evidence_store/tools	▼

Resources Maps cloud resources and its properties to the format of the ontology

GET	/v1/evidence/graph/edges	▼
POST	/v1/evidence_store/resources/{resource.id}	▼

Listing 25. Evidence Store API endpoints

3.2.5.3 Integration Status

The *Evidence Store* has already been deployed on the *Kubernetes* cluster. Table 16 provides the status of the individual connections.

Table 16. Integration status of Evidence Store with other EMERALD components

Component	Status	Comment
<i>Assessment</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>Orchestrator</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>AMOE</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
<i>Codyze</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
<i>eknows-e3</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
<i>AI-SEC</i>	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
<i>Cloudfitor-Discovery</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>EMERALD UI</i>	Testing	Minor changes may occur in the pilot testing phase.

3.2.6 Assessment

As described in D3.4 [10], the *Assessment* is tasked with assessing evidence according to specific metrics established within the EMERALD framework.

3.2.6.1 Expected behaviour (inputs/outputs)

The *Assessment* sends and receives information from different sources:

- **Evidence Store:** The *Assessment* receives evidence from the *Evidence Store* and creates assessment results based on these as well as metrics.
- **Orchestrator:** Assessment results are sent to the *Orchestrator*.
- **TWS:** Evidence and assessment results are forwarded to the *TWS* for improving the integrity of the EMERALD framework.

3.2.6.2 Published APIs

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

Assessment

POST /v1/assessment/evidences

Listing 26. Assessment API endpoint for evidence

3.2.6.3 Integration Status

Table 17 provides the current status of connections of the *Assessment* with other components.

Table 17. Integration status of Assessment with other EMERALD components

Component	Status	Comment
<i>Evidence Store</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>Orchestrator</i>	Integrated	Minor changes may occur in the pilot testing phase.
<i>TWS</i>	Integrated	Minor changes may occur in the pilot testing phase.

3.2.7 Evaluation

As described in D3.4 [10], the *Evaluation* component is responsible for analysing one or more assessment results to demonstrate compliance with particular controls outlined in a security catalogue.

3.2.7.1 Expected behaviour (inputs/outputs)

The *Evaluation* only communicates directly with the *Orchestrator*, which is, e.g., triggering the evaluation of certain security catalogue controls.

3.2.7.2 Published APIs

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

GET	/v1/evaluation/evaluate	✓
POST	/v1/evaluation/evaluate/{auditScopeId}/start	✓
POST	/v1/evaluation/evaluate/{auditScopeId}/stop	✓

Listing 27. Evaluation API endpoints

3.2.7.3 Integration Status

Table 18 provides the current status of the connections of the *Evaluation* with other EMERALD components.

Table 18. Integration Status of Evaluation with other EMERALD components

Component	Status	Comment
<i>Orchestrator</i>	Integrated	Minor changes may occur in the pilot testing phase.

3.3 EMERALD UI

The *EMERALD UI* –highlighted in the Figure 16 below– enables users to perform the different user tasks needed to interact with the EMERALD framework. As describe in D4.6 [11], it provides an overview of the data managed by the EMERALD components and combines the information into different workflows to improve the user experience.

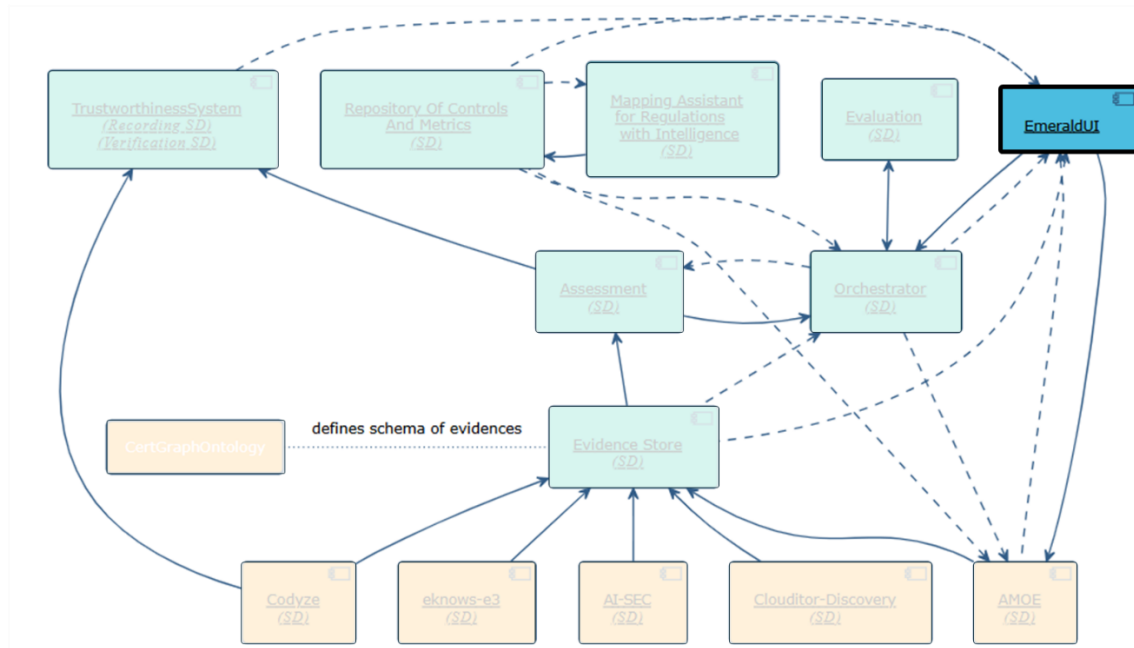


Figure 16. EMERALD UI in the EMERALD architecture

3.3.1 Expected behaviour (inputs/outputs)

The *EMERALD UI* communicates with *AMOE*, the *Evidence Store*, the *Orchestrator*, the *RCM* and the *TWS*.

Some of the other components' data is collected indirectly via the APIs of the components listed in Section 3.1 and 3.2. Depending on the functionality provided by the different APIs, the *EMERALD UI* provides different views and forms to adjust the content of the respective components.

Apart from the code/backend functionality, the *EMERALD UI* component implements the user interface itself, i.e., the part of *EMERALD* with which the user interacts. The user interface itself is being developed in WP4. The UI pages have been implemented based on the mock-ups defined in Figma³¹. The mock-ups were prioritized according to their occurrence in the User Journeys (both are listed in D4.4 [3]) and their presumed readiness for implementation.

The user interface is described in detail in the deliverable D4.6 [11] (M27); for a more detailed description, please refer to that document³².

3.3.2 Published APIs

There are no APIs published by the *EMERALD UI* – it only uses the APIs of the listed components.

3.3.3 Integration Status

The *EMERALD UI* has been deployed to the *EMERALD Kubernetes* development environment. As development proceeds, local tests and integration tests are being conducted with the different components. Once the component APIs are updated, they are integrated into the UI. At present, data can be retrieved from *AMOE*, *Evidence Store*, *Orchestrator*, *RCM* and *TWS*, but in some

³¹ <https://www.figma.com/>

³² D4.6 is not a public deliverable, so it is only available to project partners.

cases further testing and implementation work is required on by all parties involved. The point-to-point integration status is shown in Table 19.

Table 19. Integration status of EMERALD UI with other EMERALD components

Component	Status	Comment
AMOE	Integrated	-
Evidence Store	Testing	-
Orchestrator	Testing	-
RCM	Integrated	-
TWS	Testing	-

4 Conclusions

The deliverable D1.9 reflects the third version of the integrated EMERALD framework. The document demonstrates the integration of various components developed across different technical work packages, providing a cohesive prototype of the Compliance-as-a-Service (CaaS) framework.

The integration of the thirteen components –*AI-SEC*, *AMOE*, *Clouditor-Discovery*, *Codyze*, *eknows-e3*, *TWS*, *MARI*, *RCM*, *Orchestrator*, *Evidence Store*, *Assessment*, *Evaluation*, and *EMERALD UI*– is practically finished. In the second version at M30 all components were deployed, at different completion states, in the development environment. Since then, the completion of the components has advanced significantly –passing from a 90% to a 99% of global requirements completion–, and the components have also been deployed in the four pilots environments. Most of the integrations between components have now been completed, and the user interface provides a comprehensive view of the EMERALD framework, enabling users to accomplish a large proportion of the defined workflows. Once the components had been connected, a period of exhaustive testing has been carried out to detect bugs and deficiencies.

The component that is most affected by testing, changes and last-minute corrections is the *EMERALD-UI*. This is the price to be paid for being the component that integrates the APIs of the other components, helps to detect bugs and inconsistencies, and has to wait for the other components to function correctly before it can test its own processes.

The framework is supported by technologies as *Kubernetes* and *Docker*, which have proven to be robust and easy to scale. During last round of integration, main tasks have been centred in enhancing and expanding the CI/CD pipelines to extend the automate deployment to every component.

This is the third and final EMERALD prototype, which fulfils all the functionalities and requirements defined for the framework, and has been deployed in the four pilots as a representation of real-world scenarios. User's feedback on the previous version from the validation work package (WP5), and the identified bugs have been tackled to have this final (v3) integrated version of the framework as reliable as possible. It is being tested and evaluated by the final users, which will provide their comments in WP5 deliverables.

Although this is the final deliverable in WP1, there is still work to be done to complete the development of those few components that are not yet 100 per cent finished and to integrate them. Hence, tasks like testing, bugs fixing and the integration of the whole EMERALD framework are expected to continue until the end of the project.

5 References

- [1] EMERALD Consortium, “D1.7 EMERALD Integrated Solution - v1,” 2025.
- [2] EMERALD consortium, “D1.2 Data modelling and interaction mechanisms - v2,” 2025.
- [3] EMERALD Consortium, “D4.4 User interaction and user experience concept - v2,” 2025.
- [4] EMERALD Consortium, “D1.5 DevOps Methodology and CICD strategy - v1,” 2024.
- [5] EMERALD Consortium, “D1.6 DevOps methodology and CI/CD strategy for EMERALD-v2,” 2025.
- [6] EMERALD Consortium, “D2.7 ML model certification – v2,” 2026.
- [7] EMERALD Consortium, “D2.5 AMOE – v2,” 2026.
- [8] EMERALD Consortium, “D2.9 Runtime evidence extractor - v2,” 2026.
- [9] EMERALD Consortium, “D2.3 Source Evidence Extractor – v2,” 2026.
- [10] EMERALD Consortium, “D3.4 Evidence assessment and Certification–Implementation-v2,” 2025.
- [11] EMERALD Consortium, “D4.6 - EMERALD UI - v2,” 2026.
- [12] BSI - Bundesamt für Sicherheit in der Informationstechnik, “Secure, robust and transparent application of AI,” [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/Secure_robust_and_transparent_application_of_AI.html. [Accessed August 2026].
- [13] EMERALD Consortium, “D2.11 Certification Graph-v2,” 2026.
- [14] NIST - National Institute of Standards and Technology, “OSCAL: the Open Security Controls Assessment Language,” [Online]. Available: <https://pages.nist.gov/OSCAL>. [Accessed August 2026].
- [15] ENISA, “EUCS - Cloud Services Scheme,” [Online]. Available: <https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme>. [Accessed August 2026].
- [16] EMERALD Consortium, “D3.5 - Evidence assessment and Certification – Integration - v1,” 2025.
- [17] EMERALD Consortium, “D1.8 EMERALD Integrated Solution - v2,” 2026.