EMERALD

Deliverable D1.8

EMERALD Integrated solution – v2

Editor(s):	Iñaki Etxaniz, Gorka Benguría
Responsible Partner:	TECNALIA Research & Innovation
Status-Version:	Final-v1.0
Date:	30.04.2026
Type:	Other (SW)
Distribution level (SEN, PU):	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	EMERALD Integrated solution – v2
Due Date of Delivery to the EC	30.04.2026

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):	Nico Haas (Fraunhofer) Cristina Martínez, Juncal Alonso (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP1, WP2, WP3, WP4, WP5

Abstract:	Second 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	03.02.2026	ToC defined	TECNALIA
v0.2	13.02.2026	First draft version	TECNALIA
v0.3	18.03.2026	Contributions by consortium partners to Section 3	FABAR, TECNALIA, Fraunhofer, CNR, SCCH
v0.4	27.03.2026	Updated Sections 1 and 2	TECNALIA
v0.5	01.04.2026	Conclusions and Executive Summary. Sent to QA review	TECNALIA
v0.6	13.04.2026	Reviewed with comments	Fraunhofer
v0.7	24.04.2026	Addressed recommendations from QA review. Sent to final review.	TECNALIA
v1.0	30.04.2026	Final version submitted to the European Commission	TECNALIA

Table of contents

List of tables

List of figures

List of listings

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
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
RAM	Random Access Memory
RBAC	Role-Based Access Control
RCM	Repository of Controls and Metrics
REST	Representational State Transfer
RKE	Rancher Kubernetes Engine
SSL	Secure Sockets Layer
TWS	Trustworthiness System
UI/UX	User Interface / User Experience
URL	Uniform Resource Locator
VCS	Version Control System
VM	Virtual Machine
WP	Work Package

[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.

Executive Summary

The deliverable D1.8 consists of the integrated EMERALD CaaS framework v2, 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 integration, and the components involved.

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 enables collaboration and communication between the different components developed in other technical work packages.

The integration approach was already described in the first version of this deliverable [1]. The integration of the components 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 a repository based in *Artifactory*. The second one deploys the components to the test bed environment and verifies them. The test bed is based on *OpenStack* Virtual Machines. On top of all this, a three-node *Kubernetes* cluster has been mounted, with two environments: integration and production.

For the integration of a component in the EMERALD framework, an eight-step procedure was defined. The integration plan includes three phases that will be completed in months M18, M30, and M34, respectively. This deliverable corresponds to the second phase. 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 steps.

The document details the level of integration of each component in the framework, including the list of published APIs, and the status of interconnection with the other components. Four components have reached 100% implementation, and most of the components (11 out of 13) are above 80% completion. This version of the deliverable includes a summary of the status of the requirements for each component, and based on this, we can state that, at time of writing this report, the average project completion rate stands at 90%.

The key outcomes of this deliverable include the development of a second prototype of the EMERALD CaaS framework, which consolidates the first version; the documentation of the integration status of each component; and the deployment of components in the integration and production environments, providing a solid foundation for the deployment of the framework in the pilots.

Future related work in the task T1.3 will focus on the continuous integration of the EMERALD framework, including a third release of the framework featuring all the developed functionalities and incorporating user feedback. This work will be reflected in a third version of this deliverable, D1.9, scheduled for month 34.

1 Introduction

1.1 About this deliverable

This is the companion document of the software deliverable D1.8, which aims to have a second prototype of the EMERALD Compliance-as-a-Service[2] (CaaS) Framework that integrates the components developed by the other technical work packages. This second version of the integrated solution corresponds to the Milestone *MS6 – Integrated Audit Suite v2* and is mainly based on the version v2 of the EMERALD components (M24), although some additional development made until M30 has also been included in some cases. All the referred software is available in the project's public Gitlab (https://git.code.tecnalia.dev/emerald/public).

The document includes first an overview of the integration approach, to provide the reader an overview of what components are integrated, where and how, and the status of the overall integration task. It also describes the hardware equipment used to setup the test bed[3], the resources needed for the installation, and the configuration. The methodology through which a component is integrated in the framework is introduced as well. The document also includes the description of the main workflow in several scenarios and briefly describes 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.

A third version of this deliverable is planned for month 34 of the project. This third –and last– version will incorporate advancements and improvements made between the two last releases. It is expected that some of the improvements will come from user feedback, testing, and discussions on the current version.

1.2 Document structure

The remainder of the document is organized as follows:

The rest of this 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 integration 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 provides the integration status of each EMERALD component in more detail, describing the implemented APIs and the connection with other components.

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 project[4]: *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 approach of the architecture and the integration in EMERALD 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 robustness, we have selected different tools in EMERALD to replace equivalent ones (e.g. *nginx*, *Rancher* or *Longhorn*). The integration workflow is similar to the one used in MEDINA, relying on the same GitLab CI pipelines for build and deployment automation. The main infrastructure difference 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 D1.7

The deliverable evolves from D1.7 [1], and with the goal of making the document self-contained and easier to follow, parts of the content remain unchanged, while other parts are new. To simplify tracking progress from the previous version, Table 1 shows a summary of changes and additions in the document.

Table 1. Summary of updates with respect to D1.7

Section	Changes
1. Introduction	Included two new sections to summarize the technological advances from MEDINA and the changes with respect to deliverable D1.7.
2. Integration Overview	Minor changes have been made to reflect updates to the test bed environment. Section 2.5 - *Overall Status of the Integration* has been updated to reflect the status in M30. A subsection has been added to summarize the status of requirements.
3. Integration of Components	The status of the integration task has been updated.
4. Conclusions	Updated to reflect changes in the document.

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

2 Integration Overview

The integration strategy that has been defined 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 a component behaviour, either by accessing to the logs or login into the container that runs the component.
- Have rights to destroy their components and deploy again with changes or fixes.
- Automatically deploy new versions of their components as they are 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 indicates the component function in the framework, while the dashed/full lines denote pull/push of data.

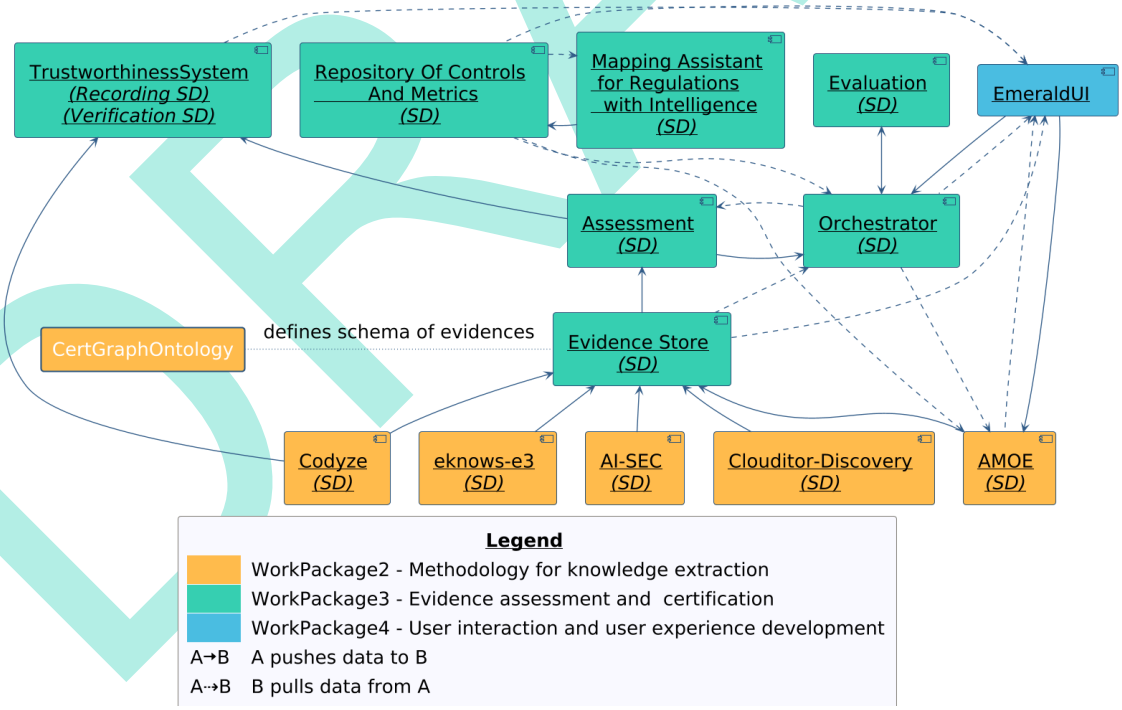


Figure 1. EMERALD Components

The following evidence collectors (in orange) collect different forms of data and extract evidence that is then shared in 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 relevant metadata related 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 following evidence assessment and compliance components (in green) are the next step in the EMERALD workflow:

- 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 about the current audit process, such as the Target of Evaluation, 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), that is key to implement the business cases for the different user roles in EMERALD:

- The ***EmeraldUI*** wraps all the components functionality in a unique User Interface. It calls the APIs provided by the components to interchange information and present it to the 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 (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 participate in the early phases of the workflow. The *Evidence extractors* and the *Assessment* participate in the intermediate phases, and the *TWS*, *Evaluation* and *Evidence Store* participate towards the end of the process. The *EMERALD UI*, for its part, takes part in the entire workflow.

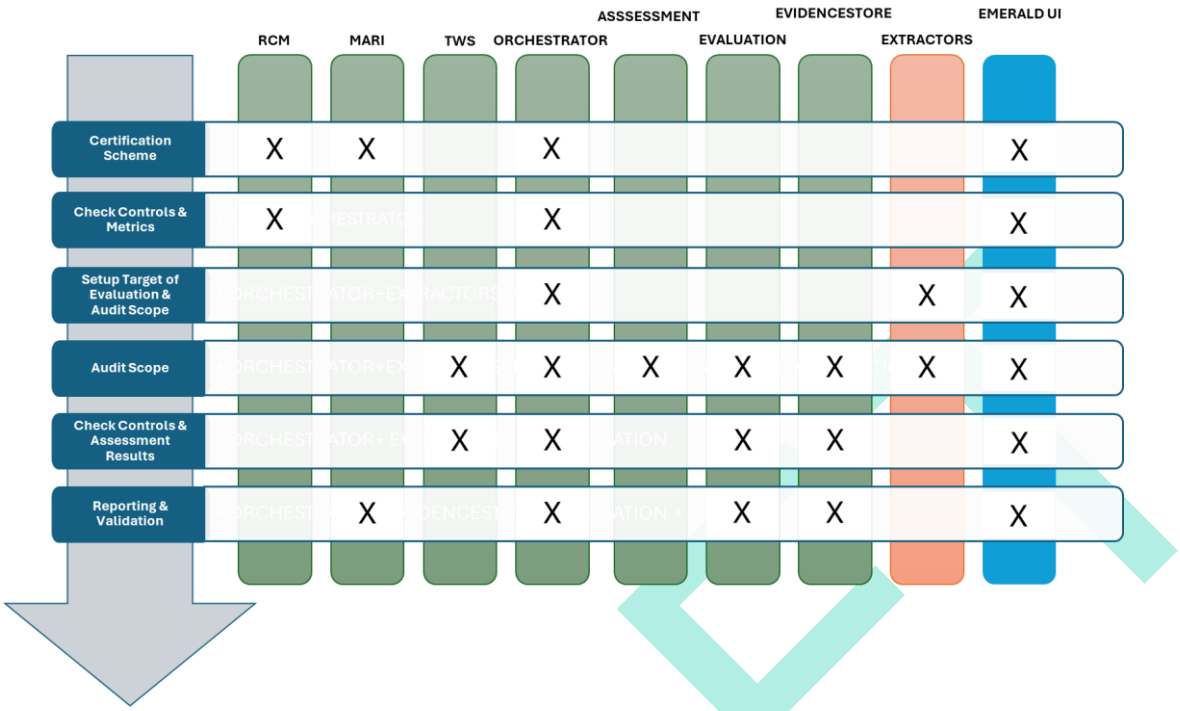


Figure 2. Participation of the components in the EMERALD blueprint for audit preparation

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 the source code of the software so that different 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 will be finally merged (see Figure 3). Later, integration test on the entire EMERALD framework will ensure that the components work in tandem, i.e., they can be installed, run and used together following the user-defined workflows.

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 the component to the integration environment and then verify it to determine if it is stable enough to be promoted into the production environment.

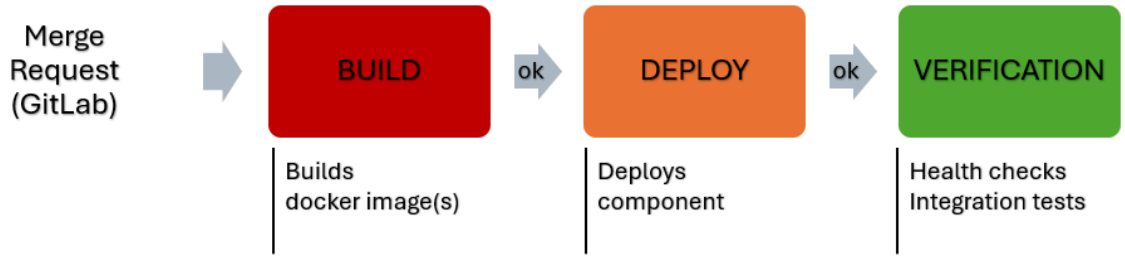


Figure 3. Merge Request and generic CI/CD pipelines

2.2 Components Integrated into the EMERALD framework v2

The components involved in the second version of the EMERALD framework are those that were available at month 24 of the project. These components are listed in Table 2, together with their respective Key Results (KR) and the deliverables in which they are described.

Table 2. Components in the EMERALD framework v2

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

All components expose and consume interfaces for integration. REST[5]-based interfaces follow standard REST API practices and are described in the OpenAPI[6] format, while gRPC[7]-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. In EMERALD, we implement a Role Based Access Control (RBAC) along the components, where the *Orchestrator* is a central piece assigning access to the

[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. More information is available at https://grpc.io/

Targets of Evaluation to the right users. We have chosen *Keycloak*[8] as open-source IAM solution for the project.

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.

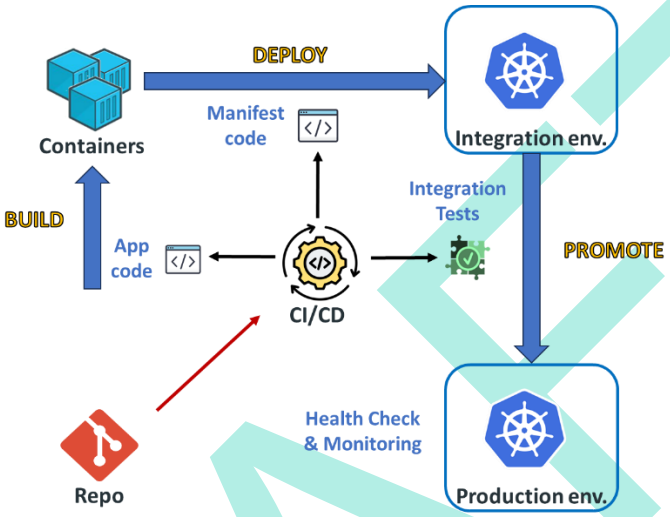


Figure 4. Integration and Production environments in the EMERALD CaaS framework

Each test bed is composed by several Virtual Machines (VM) located in the server infrastructure of TECNALIA. The environments are built in a three-node ***Kubernetes***[9] cluster over an ***OpenStack***[10] platform.

The Virtual Machines (VMs) of the Kubernetes nodes (named *k8so01-emerald, k8so02-emerald,* and *k8so03-emerald*) share the same specifications:

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

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

The access to the virtual machines is provided via SSH[11] (Secure Shell) protocol, using digital certificates through a bastion host[12]. The REST API exposed by each component is reachable from the Internet using this URL naming convention:

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

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

where dev/prod 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* component running in the Kubernetes production environment, it will be addressed as `rcm.prod.emerald.digital.tecnalia.dev`.

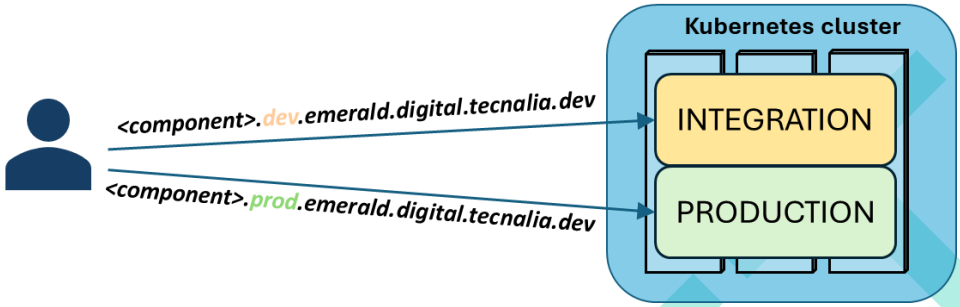


Figure 5. URL naming convention for integration/production environments

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*** (**RKE2**[13]), an open-source, enterprise-ready Kubernetes distribution. Using RKE2, we have deployed a high availability configuration (see Figure 6), 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`.

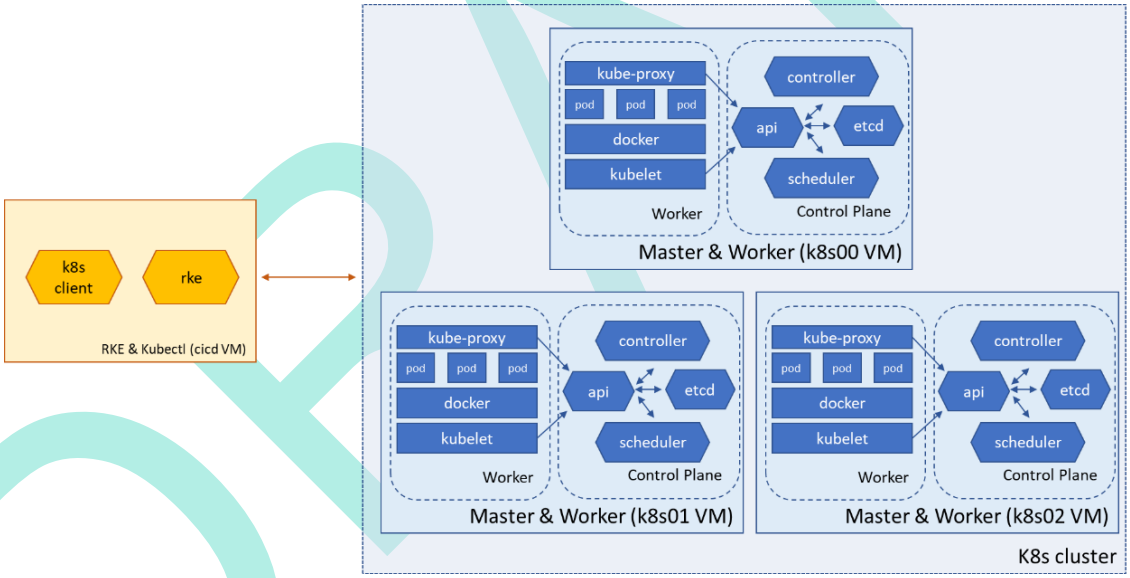


Figure 6. Kubernetes RKE2 cluster

We have followed an Infrastructure as Code (IaC) approach for the deployment. For the creation and configuration of the cluster we have used ***OpenTofu***[14] and ***Ansible***[15] 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 under a configuration

[13] https://docs.rke2.io/
[14] https://opentofu.org
[15] https://www.ansible.com

management process, in the Gitlab repository of the project. The usage of IaC provides several advantages for project management:

- Allows the redeployment of the cluster from scratch, if we need to migrate.
- Simplifies the horizontal scalation of the Kubernetes , if more capacity is required.
- Is reusable by pilots, in case they have similar infrastructure.

2.3.1 Container orchestration

The EMERALD framework functionalities are made up by the collaboration of micro-services, which communicate each other through APIs, are packaged in docker images and run in containers. *Kubernetes* orchestrates all these containers in a virtual environment running in a highly available cluster.

We also use an IaC approach based on ***Kustomize***[16] 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 a folder named "components" with the details of the deployment of the individual components, and other folders that describe 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 three nodes, and a total of 989 GB fault-tolerant and high availability of storage are assured, as shown in Figure 7.

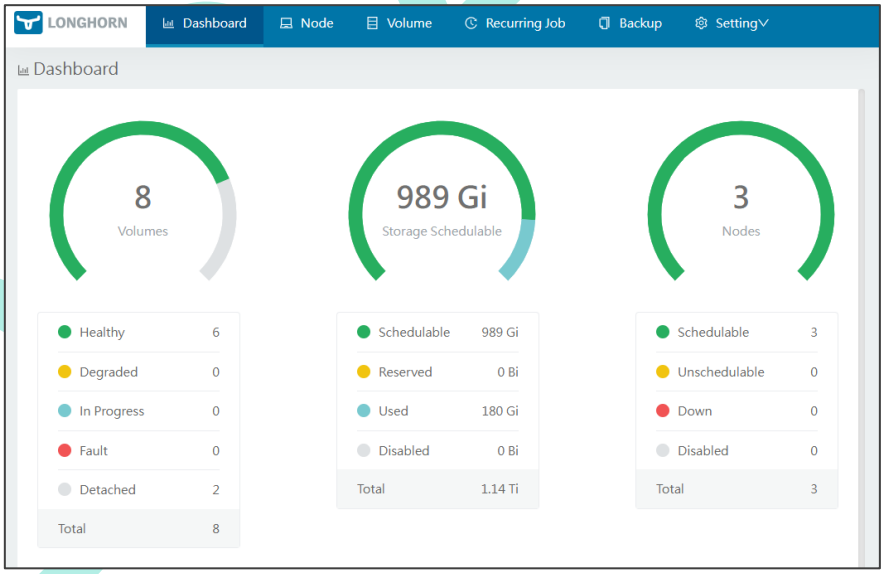


Figure 7. Longhorn dashboard in Rancher

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

[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/

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 8). The images are stored under the following URLs:

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

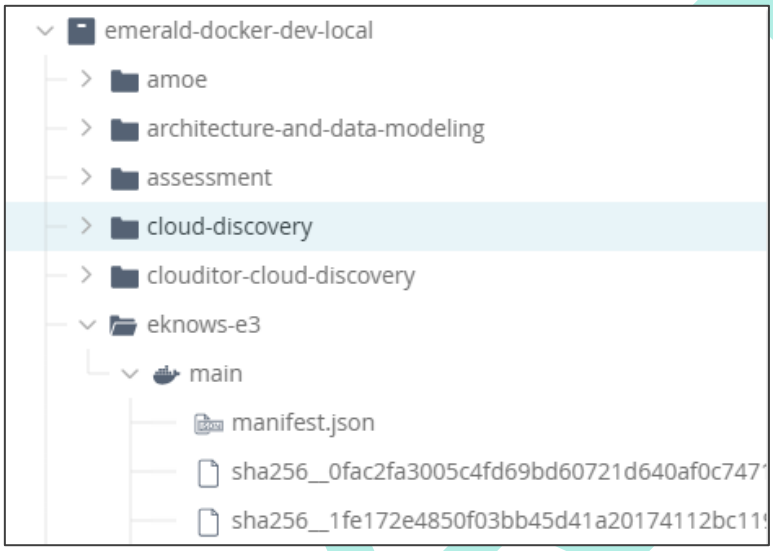


Figure 8. EMERALD Docker registry

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 runs 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 9). It helps to deploy containerised applications to a Kubernetes cluster, troubleshoot them, and manage cluster resources. The dashboard is accessible over HTTPS in the dev environment[21].

[18] https://jfrog.com/artifactory/
[19] https://www.nginx.com/
[20] https://kube-vip.io/
[21] https://k8so.emerald.digital.tecnalia.dev/dashboard/

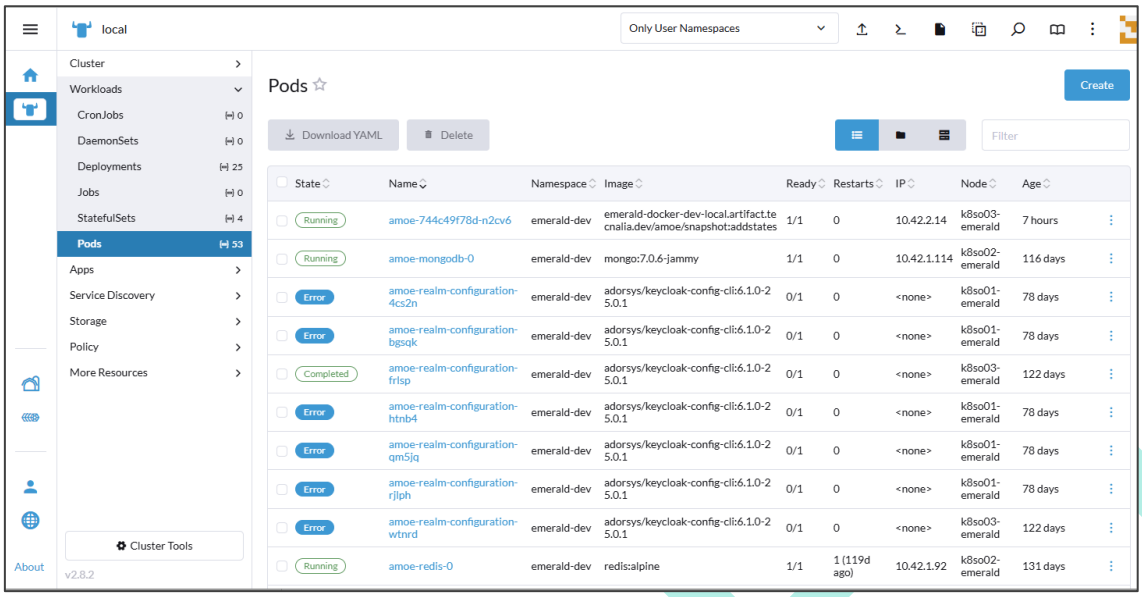


Figure 9. Rancher Dashboard

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 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 three nodes. Figure 10 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/

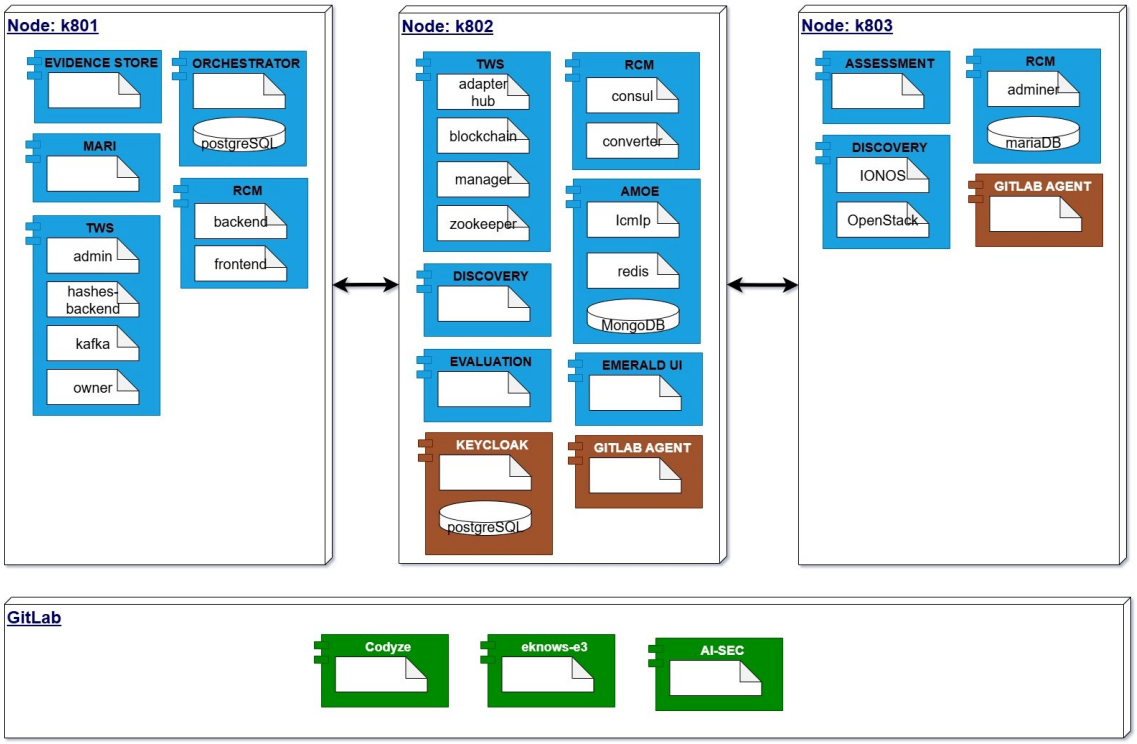


Figure 10. Deployment diagram in V2

Figure 10 represents the deployment 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 11 shows the main steps in the integration and deployment of a component[24]:

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 is deployed correctly, 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.

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

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.

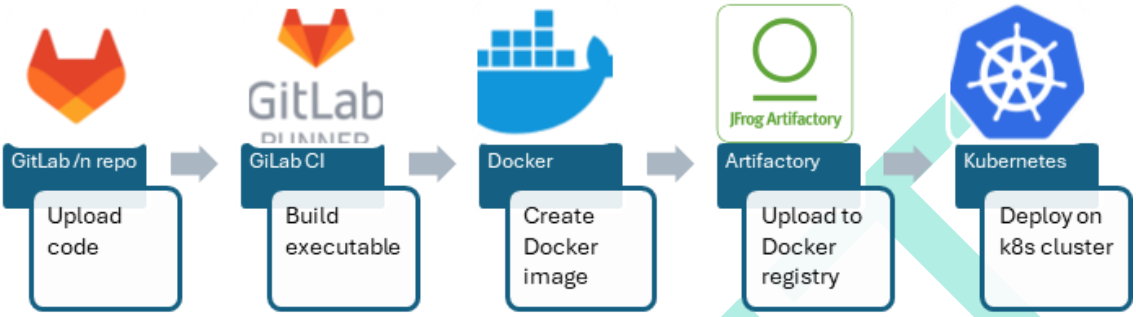


Figure 11. Main steps of the component integration

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

During the first round, WP1 delivered workshops to the rest of the consortium to introduce the main concepts of *Docker* and *Kubernetes* and explain the integration steps. During the second round, WP1 performed a workshop centred in 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 phase, the components' deployment in the integration environment was carried out manually by each partner. In the second phase, 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) phase 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	√ (Apache)	√	√	√	√	√	√	√
MARI	√ (Apache)	√	√	√	√	√	√	√
RCM	√ (Apache)	√	√	√	√	√	√	√
TWS *	√ (Propietary)	N/A	N/A	√	√	√	√	√
Assessment	√ (Apache)	√	√	√	√	√	√	√
Clouditor-Discovery	√ (Apache)	√	√	√	√	N/A	√	√
Evaluation	√ (Apache)	√	√	√	√	√	√	√
Evidence Store	√ (Apache)	√	√	√	√	√	√	√
Orchestrator	√ (Apache)	√	√	√	√	√	√	√
Codyze	√ (Apache)	√	√	√	√	√ (CLI)	N/A	N/A
eKnows-e3 *	√ (Apache, others)	√	√ & N/A	√	√	√ (CLI)	N/A	N/A
AI-SEC	√ (Apache)	√	√	√	√	N/A	N/A	N/A
Emerald UI	√ (Apache)	√	√	√	√	√	√	√

The starting point of the integration process is the source code of the components, which has been uploaded to the public GitLab repository[25] (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 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 stages[27], 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*	Connected	*AI-SEC* is currently in the tool testing phase. It starts to integrate by connecting to the *Evidence Store*.
	Evidence Store	Locally Tested	*AI-SEC* is currently connected to the *Evidence Store* and testing the extracted evidence.
AMOE	*Evidence Store*	Testing	Waiting for updated Ontology and full integration of (new) metrics based on the updated data model.
	RCM	Locally Tested / Connected	Tested with initial API. Tests will continue with new metrics once they are imported.
	Orchestrator	Locally Tested / Testing	Full testing of the *AMOE* workflow via the UI is to be done after the next integration step has been finished (waiting for metric integration).
	EMERALD UI	Testing / Connected	Collected files and extracted results. Full implementation remains to be tested by pilots. The *Keycloak* configuration needs to be updated for a stable deployment and test setup (permissions / alignment of user to target of evaluations) – this requires additional implementations or decisions in *Orchestrator*.
Clouditor-Discovery	*Evidence Store*	Connected	Testing pending
Codyze	*[CI/CD] **	Locally Tested	*Codyze* is integrated as a CI/CD component. Currently, a proof-of-concept integration for *Codyze-Provenance* exists for GitLab. For *Codyze-Compliance*, a similar integration is planned.

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

Component A (implementer)	**Component B** (invoker)	**Status**	**Comment**
	Evidence Store	Testing against remote endpoint	We can send pieces of evidence. However, they are not fully filled, i.e., some ontology terms are missing for the assessment.
	TWS	Testing	Initial testing of evidence submission to the *TWS.*
eknows-e3	*[CI/CD] **	Locally Tested	The CI/CD uses *eknows-e3* to extract and save evidence in the *Evidence Store*. A demo showcases how the *eknows-e3* can be integrated. Integration tests are currently being developed.
	Evidence Store	Integrated	Communication is implemented; integration tests are completed. A demo showcases how the *eknows-e3* component can be integrated into a GitLab pipeline. Integration tests are completed
TWS	*EMERALD UI*	Testing	Testing on-going
	Evidence Store	Integrated	
	Evidence extractors	Integrated	
MARI	*RCM*	Integrated	-
RCM	*EMERALD UI*	Testing	API has been updated and extended to cover UI requirements
	Orchestrator	Locally Tested	*RCM* output call still to be connected
	MARI	Integrated	Mapping API is finished unless bugs are detected in further tests
	AMOE	Testing	
Orchestrator	*EMERALD UI*	Integrated	The *EMERALD UI* is integrated and can use the main *Orchestrator* endpoints. This integration will be further exercised and refined during the pilot phase, where feedback from real pilot usage may lead to adjustments.
	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.
Evidence Store	*Assessment*	Integrated	Minor changes may occur in the pilot testing phase.
	Orchestrator	Integrated	Minor changes may occur in the pilot testing phase.
	AMOE	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
	Codyze	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
	eknows-e3	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
	AI-SEC	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.

Component A (implementer)	**Component B** (invoker)	**Status**	**Comment**
	Clouditor-Discovery	Integrated	Minor changes may occur in the pilot testing phase.
	EMERALD UI	Connected	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.
Assessment	*Evidence Store*	Integrated	Minor changes may occur in the pilot testing phase.
	Orchestrator	Integrated	Minor changes may occur in the pilot testing phase.
	TWS	Integrated	Minor changes may occur in the pilot testing phase.
Evaluation	*Orchestrator*	Integrated	Minor changes may occur in the pilot testing phase.
EMERALD UI	*AMOE*	Connected / Testing	Additional testing is required; existing approach looks promising. Requires metrics to be integrated in the public repository[28] and updates to the component for final validation.
	Evidence Store	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.
	Orchestrator	Testing	Further testing required. Waiting for updates to integrate and implement features such as role-based access, control assignment and control filtering based on the workflow or compliance status. Basic features are implemented and tested (e.g. creating ToE or audit scope).
	RCM	Testing	Waiting for updates to the API. Existing endpoints integrated and successfully tested.
	TWS	Testing	Testing is ongoing; requires more examples / additional test data, for final validation.

(*) 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 86% integration completed at M30. The majority of the elements in the table are in the more advanced statuses, i.e. "Connected", "Testing" and "Integrated" status.

[28] The public metric repository is an initiative of the European Cluster for Cybersecurity Certification (https://cybersecuritycertcluster.eu), of which EMERALD is a member. It collects security metrics that can be used for continuous certification and is available at https://github.com/Cybersecurity-Certification-Hub/security-metrics.

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 the technical requirements due by M30 –the submission date of this deliverable– along with their priority, timeline[29] and completion status. WP1 manages a total of 63 requirements, covering both functional requirements (associated with individual components) and non-functional ones; 50 of these fall within the M30 deadline. The table highlights completed requirements in green, and marks discarded ones in grey.

Table 5. Requirements prioritization matrix and status at M30

Req. ID	Title	Priority	Timeline	Status
AI-SEC.01	The extractor tool includes selected criteria	MUST	M24 (C-V2)	80%
AMOE.01	Upload PDF document	MUST	M24 (C-V2)	100%
AMOE.02	Provision of extracted evidence to Evidence Store	MUST	M24 (C-V2)	50%
AMOE.03	Refine evidence extraction approach	MUST	M24 (C-V2)	70%
AMOE.04	Compare results from multiple documents	SHOULD	M12 (C-V1)	90%
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)	80%
CODYZE.02	Generate provenance reports on Codyze execution	SHOULD	M30 (I-V2)	90%
CODYZE.03	Extraction of security features from source code	COULD	M30 (I-V2)	90%
EKNOWS.01	Integration into existing systems	MUST	M18 (I-V1)	100%
EKNOWS.02	Resilience while analysing erroneous code	SHOULD	M24 (C-V2)	90%
EKNOWS.03	Support languages required by pilots	MUST	M24 (C-V2)	100%
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)	99%
TWS.02	Provide integrity proof of assessment results	MUST	M24 (C-V2)	99%
TWS.03	Provide access through REST API or graphical interface	MUST	M24 (C-V2)	95%
TWS.04	Use a general-purpose public-private Blockchain network	MUST	M24 (C-V2)	100%
TWS.05	Include suitable documentation	MUST	Discarded	-

[29] 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.

Req. ID	Title	Priority	Timeline	Status
TWS.06	Allow evidence integrity proofs from the evidence sources	SHOULD	M24 (C-V2)	99%
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)	90%
RCM.08	Support updating/versioning of schemes	SHOULD	M30 (I-V2)	100%
RCM.09	Self-assessment questionnaires (EUCS)	COULD	M24 (C-V2)	90%
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.02	REST API Gateway for UI	MUST	M30 (I-V2)	75%
ORCH.03	Role Based Access Control	MUST	M30 (I-V2)	75%
ORCH.04	Manage Tools via API	COULD	Discarded	-
ORCH.05	Provide an API for audit workflow	MUST	M30 (I-V2)	50%
ESTORE.01	Storage of ontology entities in graph database	MUST	M30 (I-V2)	100%
ASSESS.01	Assessment based on evidence	MUST	M30 (I-V2)	80%
ASSESS.02	Assessment rules for 80% of the defined metrics	MUST	M30 (I-V2)	30%
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%
WP1.02	Portability	SHOULD	M24 (C-V2)	100%
WP1.06	Agile development methodology	MUST	M12 (C-v1)	100%

At M30, 22 requirements are fully implemented, and four requirements have been discarded for different reasons and are not to be implemented. The rest of the requirements (as reflected

in Figure 12) are “work in progress”, with the majority already well advanced, none falls below 25% and only a few remain below 50%.

The average project completion rate at M30 is 90%.

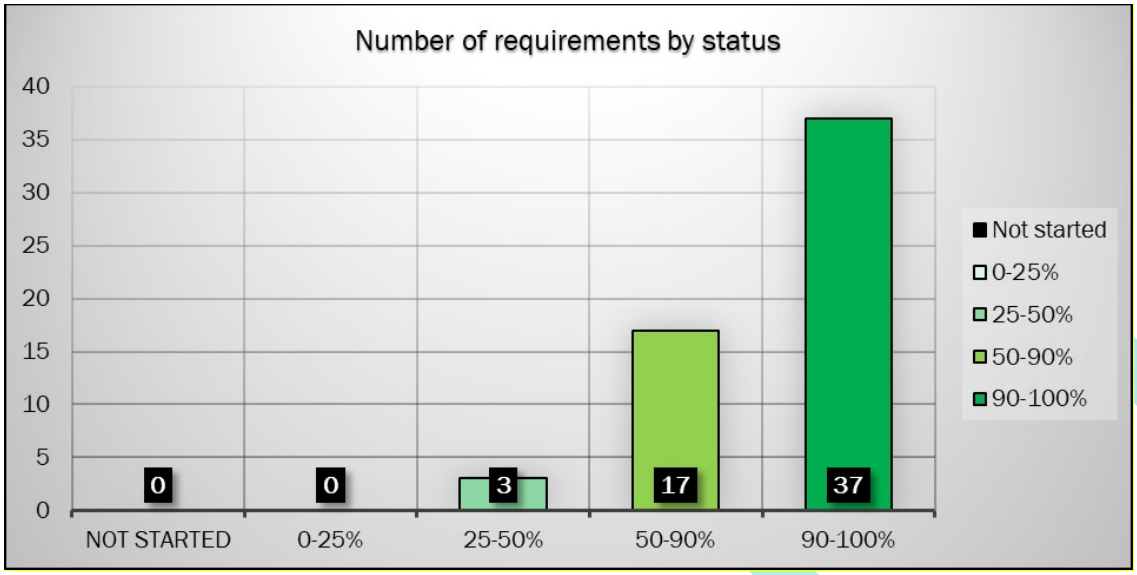


Figure 12. Number of requirements by status at M30

Figure 13 shows the distribution of the progress in the implementation of the requirements, where only 5% of them are below 50% completion; 30% are between 50% and 90% completion; and **65% of the requirements are above 90% completion**.

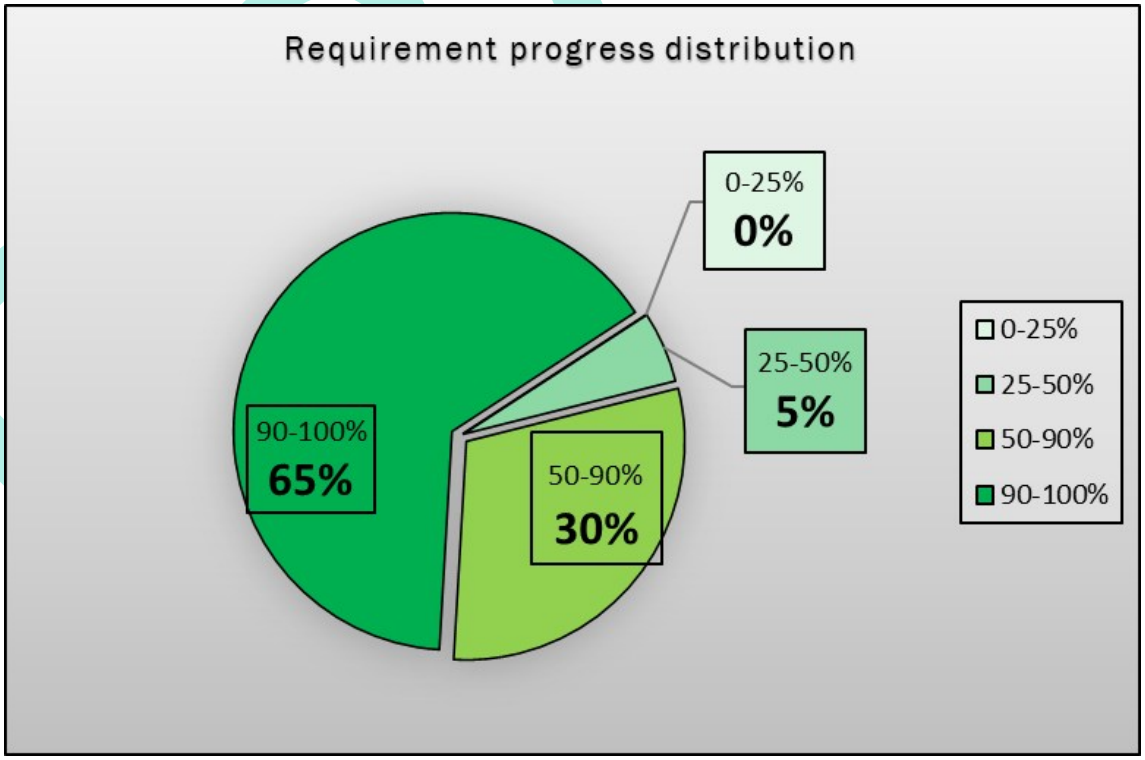


Figure 13. Requirement progress distribution at M30

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, most requirements are near completion but still classified as work in progress.

Table 6 summarises the requirements grouped 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 M30 (by component)

Component	Discarded	Not started	Partially implemented	Fully implemented	TOTAL	Average Status
AI-SEC	-	-	1	-	1	**80%**
AMOE	1	-	3	3	7	**85%**
Discovery	-	-	-	1	1	**100%**
Codyze	-	-	3	-	3	**87%**
eknows-e3	-	-	1	4	5	**98%**
TWS	1	-	4	3	8	**99%**
MARI	1	-	-	4	5	**100%**
RCM	-	-	2	9	11	**98%**
Evidence Store	-	-	-	2	2	**100%**
Orchestrator	1	-	2	3	6	**56%**
Assessment	-	-	2	1	3	**70%**
Evaluation	-	-	-	2	2	**100%**
NFR (WP1)	-	-	4	4	8	**90%**
TOTAL	**4**	**0**	**22**	**36**	**62**	**90%**

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

To better grasp the data in the table, Figure 14 provides a graphical representation of it. The white section of the bars reflects the advancement since the M24 milestone.

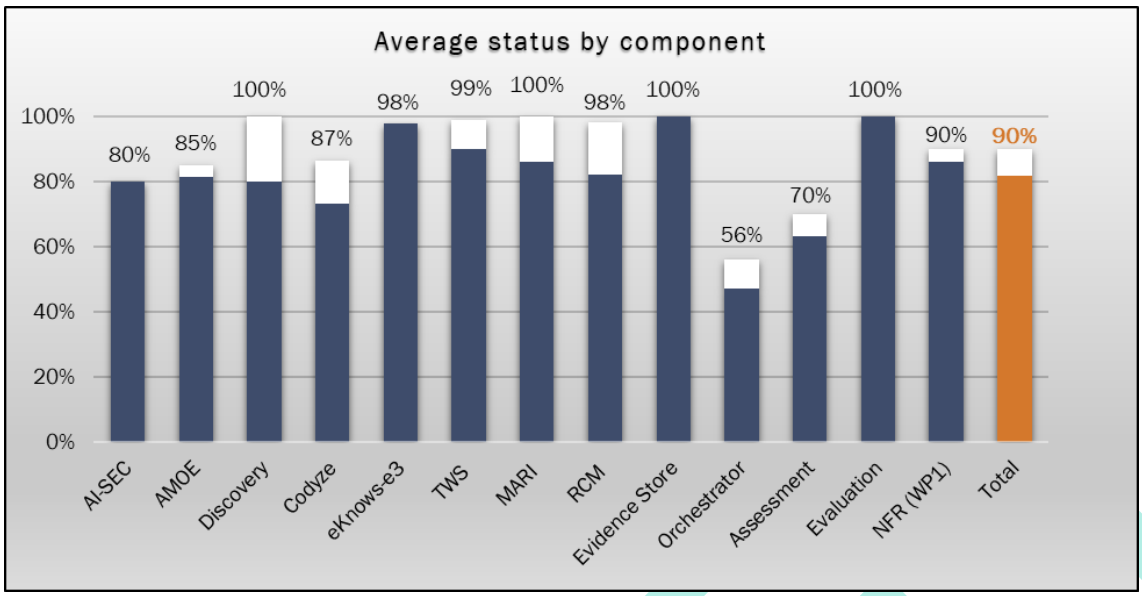


Figure 14. Requirements completion status per component

Four components (*Discovery*, *MARI*, *Evidence Store* and *Evaluation*) have reached 100% implementation, and **most of the components (11 out of 13) are above 80% completion**, being the average implementation of **90%**.

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, a status of the integration with other components is provided.

3.1 Evidence Extractors

Evidence extractors –highlighted in the Figure 15 below– are the components in charge of extracting different forms of data from the targets of evaluation and providing them as evidence that is then processed in the EMERALD framework to decide on compliance.

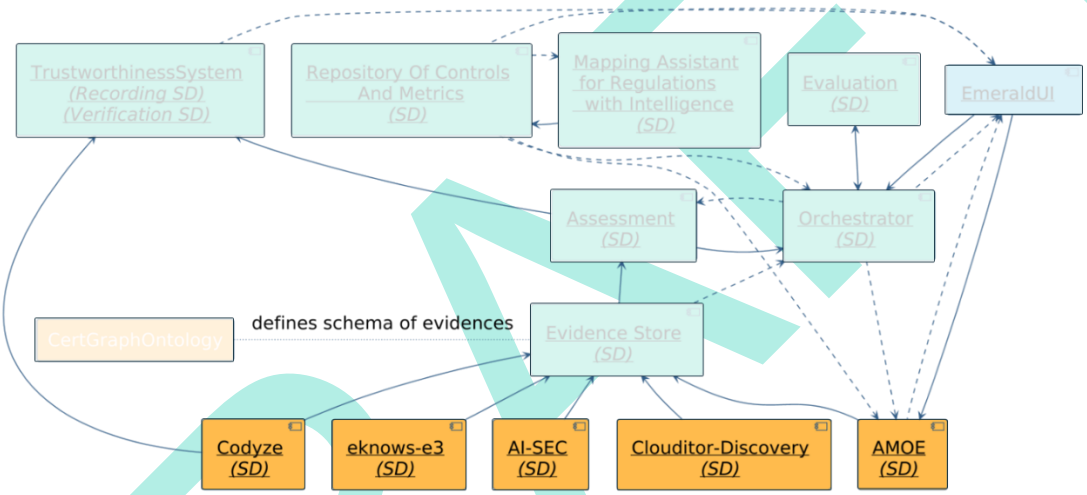


Figure 15. Evidence extractors in the EMERALD architecture

3.1.1 AI-SEC

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*. The interaction is performed via the *Evidence Store* API rather than a direct UI integration.
- ***Evidence Store***: Evidence 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 listed components.

3.1.1.3 Integration Status

A proof-of-concept integration for *AI-SEC* is under testing. 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
EMERALD UI	Connected	The *EMERALD UI* is connected indirectly with *AI-SEC* via the *Evidence Store*.
Evidence Store	Locally Tested	*AI-SEC* is currently connected to the *Evidence Store* and testing the extracted evidence.

3.1.2 AMOE

AMOE is an evidence extractor that extracts relevant parts of policy documents [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.

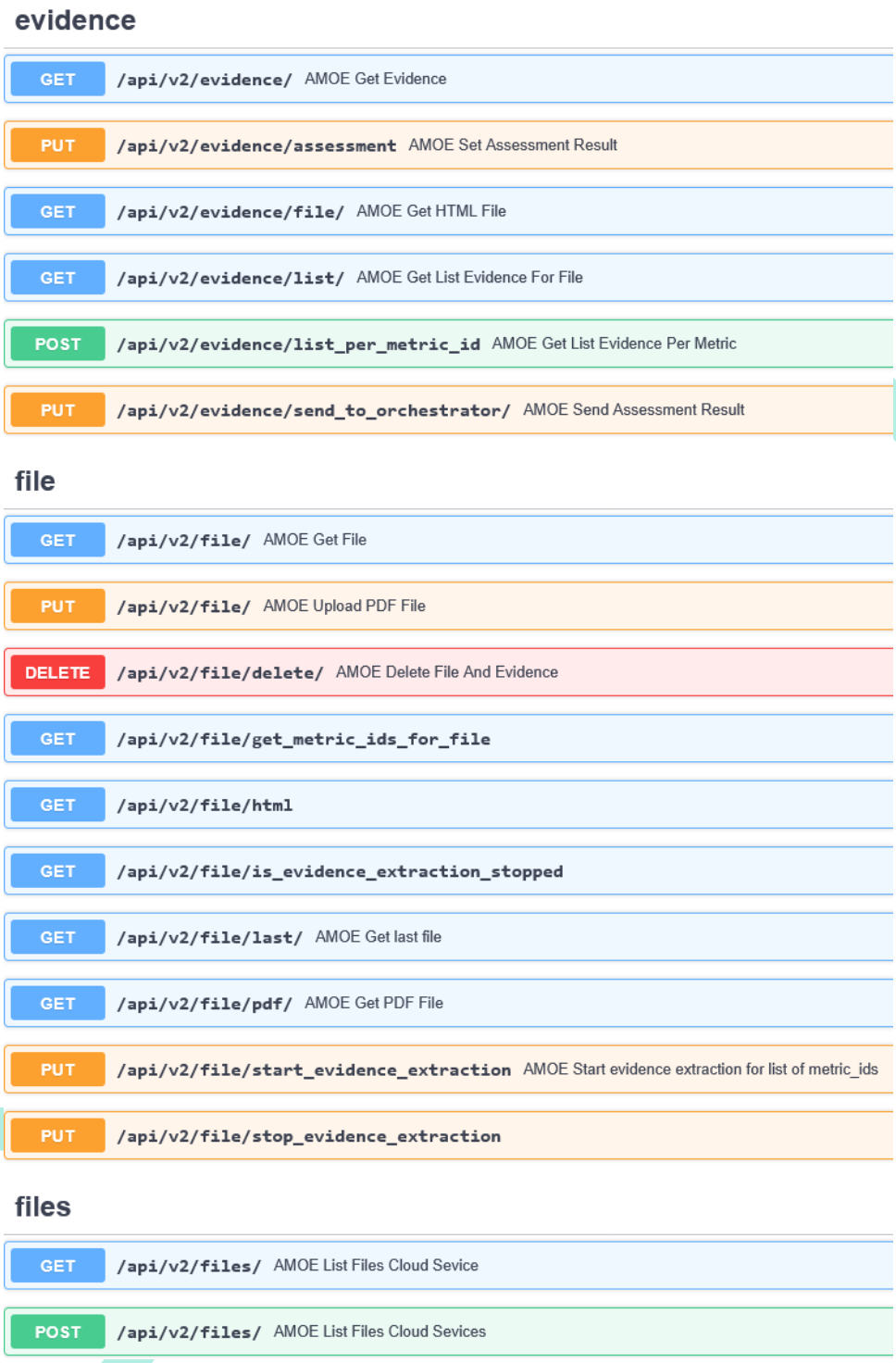
3.1.2.1 Expected behaviour (inputs/outputs)

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.



Listing 1. AMOE API overview

3.1.2.3 Integration Status

AMOE has already 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
Evidence Store	Testing	Waiting for updated Ontology and full integration of (new) metrics based on the updated data model.
RCM	Locally Tested / Connected	Tested with initial API. Continue tests with new metrics once they are imported.
Orchestrator	Locally Tested / Testing	Full testing of the AMOE workflow via the UI is to be done after next integrations step has been finished (waiting for metric integration).
EMERALD UI	Connected / Testing	Collected files and extracted results. Full implementation remains to be tested by pilots. The *Keycloak* configuration needs to be updated for a stable deployment and test setup (permissions / alignment of user to target of evaluations) – this is requiring additional implementations or decisions on the *Orchestrator* side.

3.1.3 Clouditor-Discovery

The *Clouditor-Discovery* is one of the evidence extractors in the EMERALD CaaS framework. As described in D2.8 [13], 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 die 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
Evidence Store	Connected	-

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-Compliance* 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 *Asessment, 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:

```
--id <string>              ID of the cloud service
--rules <path>             Path to rule set to be checked for compliance
--endpoint <url>           URL to the Evidence Store
--oauth-endpoint <url>     URL to an OAuth endpoint for authentication
--username <string>        Username for authentication
--password <string>        Password for authentication
--config <path>            Path to a configuration file for Codyze
```

In addition to CLI parameters and options, a configuration file can be used to store the corresponding values as part of the source code in the source code repository.

3.1.4.3 Integration Status

Codyze is integrated as a CI/CD component. Currently, a proof-of-concept integration for *Codyze-Provenance* exists for GitLab (see Table 10). For *Codyze-Compliance*, a similar integration is planned.

Table 10. Integration status of Codyze with other EMERALD components

Component	Status	Comment
Evidence Store	Testing against remote endpoint	We can send pieces of evidence. However, they are not fully filled, i.e., some ontology terms are missing for the assessment.

TWS	Testing against remote endpoint	Initial testing of evidence submission to the *TWS.*

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.

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 [14] will be 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,--srcDirToAnalyze <arg>
    -jv,--javaVersion <arg>
```

[30] https://www.scch.at/software-science/projekte/detail/eknows

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: "clouditor"
         oauth-client-secret: "clouditor"
         evidence-collector-url: "http://clouditor: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 ekows-e3 with other EMERALD components

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

3.2 Evidence Assessment and Certification

Evidence Assessment and Certification components –highlighted in Figure 16– are used for storing, assessing and evaluating evidence. This group comprises the *Orchestrator*, that controls the overall workflow, as well as complementary tools like the *Repository of Control and Metrics (RCM)*, the *Mapping Assistant (MARI),* the *Evidence Store*, the *Assessment*, the *Evaluation* and the *Trustworthiness System* (TWS).

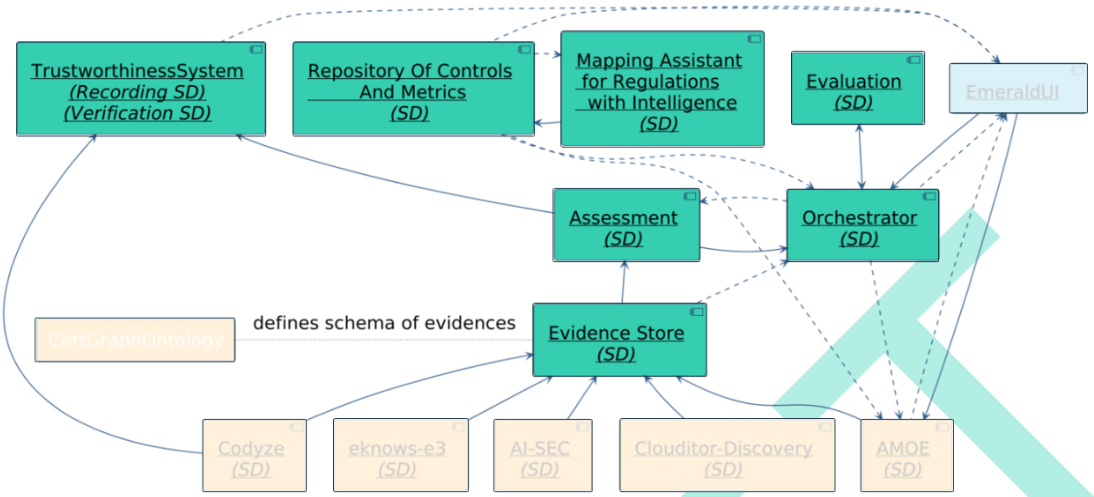


Figure 16. Evidence assessment and Certification tools in the EMERALD architecture

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.

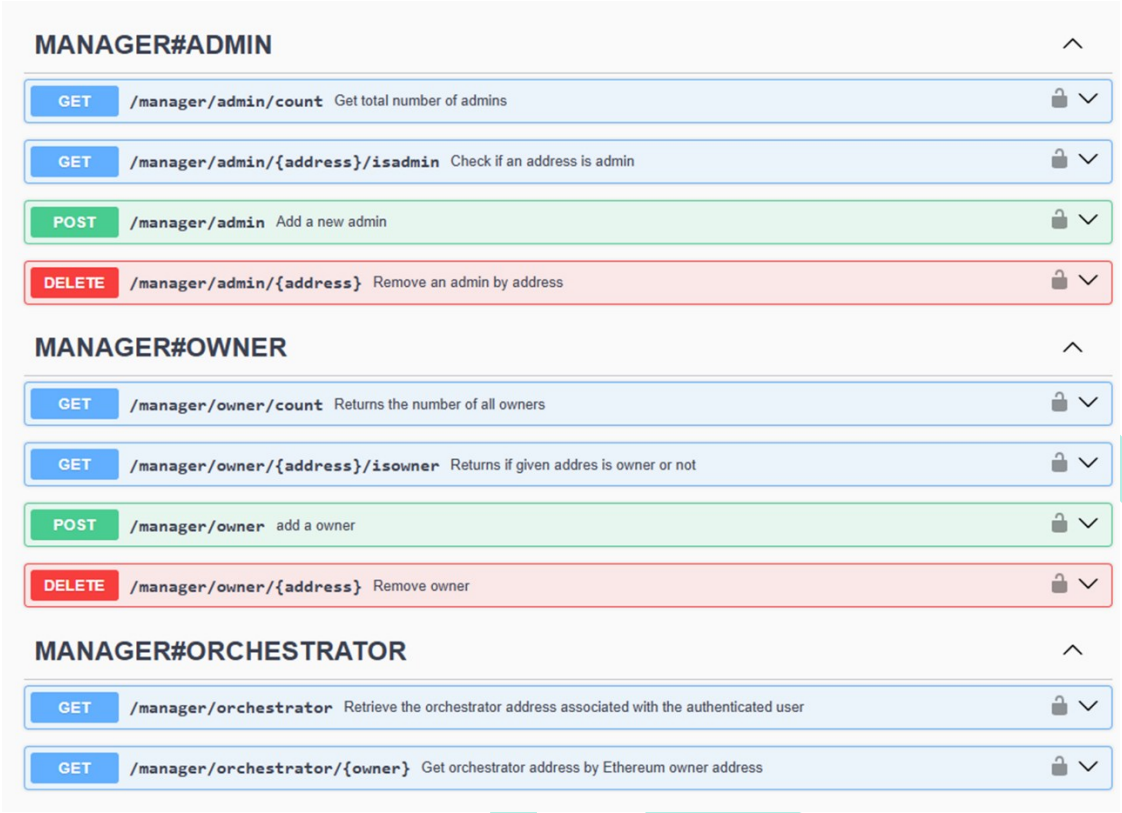
3.2.1.1 Expected behaviour (inputs/outputs)

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

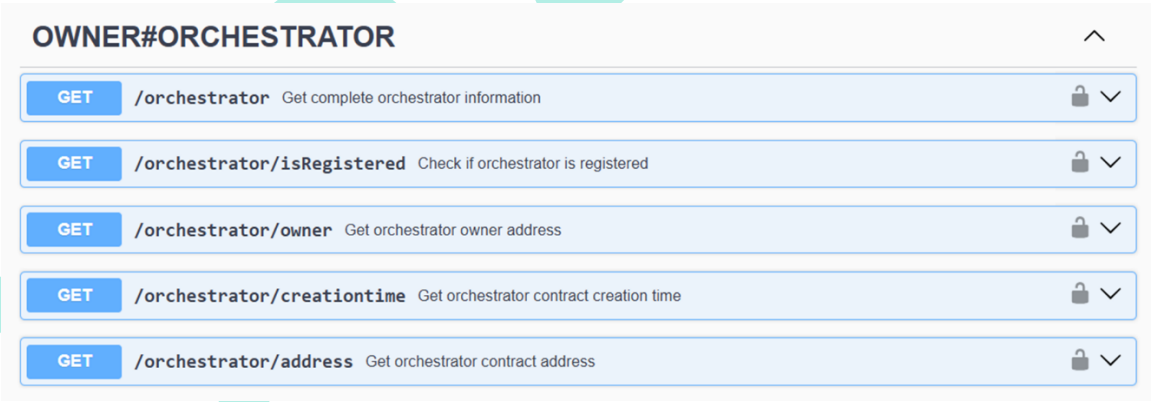
- ***Assessment*:** The interaction with this component occurs in two ways:
  - The *Assessment* component provides information (proofs of integrity) related to evidence and assessment results to be recorded on the Blockchain.
  - The automatic verification service requests the current values of evidence 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* will be 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.

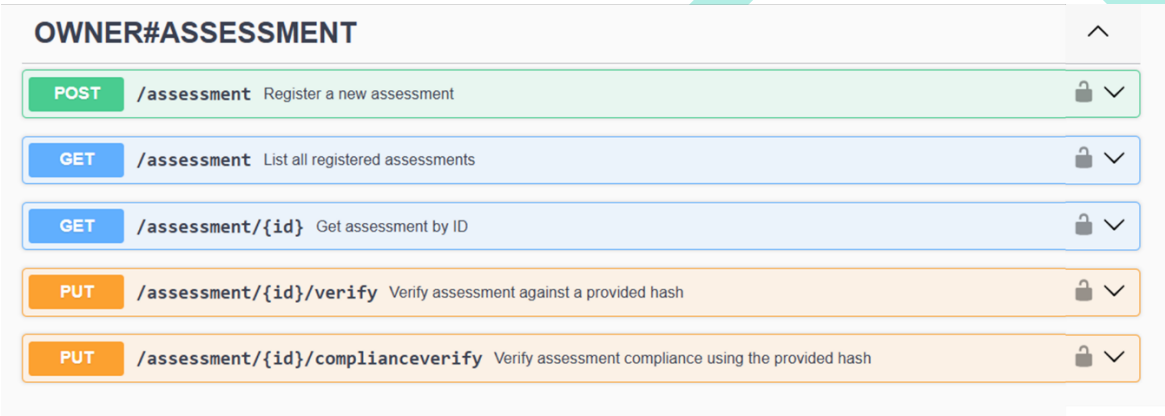


Listing 2. TWS API Endpoints for Account Management



Listing 3. TWS API Endpoints for evidence sources management

Listing 4. TWS API Endpoints Evidence Management



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 on-going. Changes in the API might be necessary.
Evidence Store	Integrated	
Evidence extractors (Codyze)	Integrated	

3.2.2 MARI

The *Mapping Assistant for Regulations with Intelligence* (*MARI*) is an intelligent system for compliance management. As described in D3.4 [10], its main functionality is to automatically associate relevant metrics with controls and facilitate the mapping of controls across multiple certification schemes. This automation significantly reduces manual effort and improves performance in compliance management processes. The *MARI* is built as an NLP-based tool, leveraging a sentence transformer model to generate vector embeddings that capture the semantic meaning of controls and metrics. The associations between controls and metrics, as

well as between controls across different certification schemes, are then performed by measuring the similarity between these embeddings in the vector space.

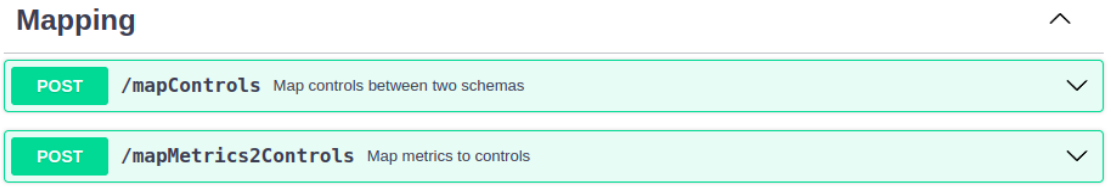
3.2.2.1 Expected behaviour (inputs/outputs)

The *MARI* exchanges information exclusively with the *RCM*.

- ***RCM***: It sends the mapping requests to the *MARI*, including information about the certification schemes and the metrics. Once the *MARI* performs the mappings, the results are sent back to the *RCM*, which receives and stores them for further use.

3.2.2.2 Published APIs

The *MARI* provides two endpoints for mapping controls and metrics: the *mapControls* endpoint that maps controls from a schema to another by evaluating a similarity threshold, actively matching controls that meet the required standard; and the *mapMetrics2Controls* endpoint that links metrics to the corresponding controls based on the same similarity principle.



Listing 6. MARI API Endpoints for mapping

3.2.2.3 Integration Status

The *MARI* has already 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 the *MARI* with the *RCM*.

Table 13. Integration status of MARI with other EMERALD components

Component	Status	Comment
RCM	Integrated	-

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 [15] 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 [16] compliance.

3.2.3.1 Expected behaviour (inputs/outputs)

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

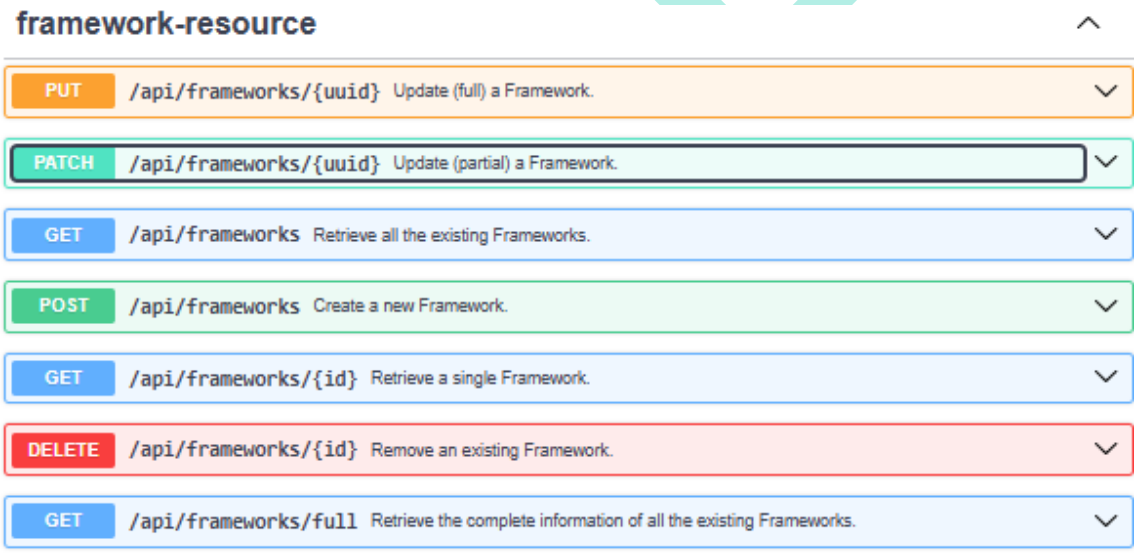
- ***Clouditor-Orchestrator***: It retrieves information about schemes and metrics from the *RCM*, which is then used to configure extractors and organize evidence. The *RCM* also

obtains the assessment results required for the questionnaire utility from the *Orchestrator*.

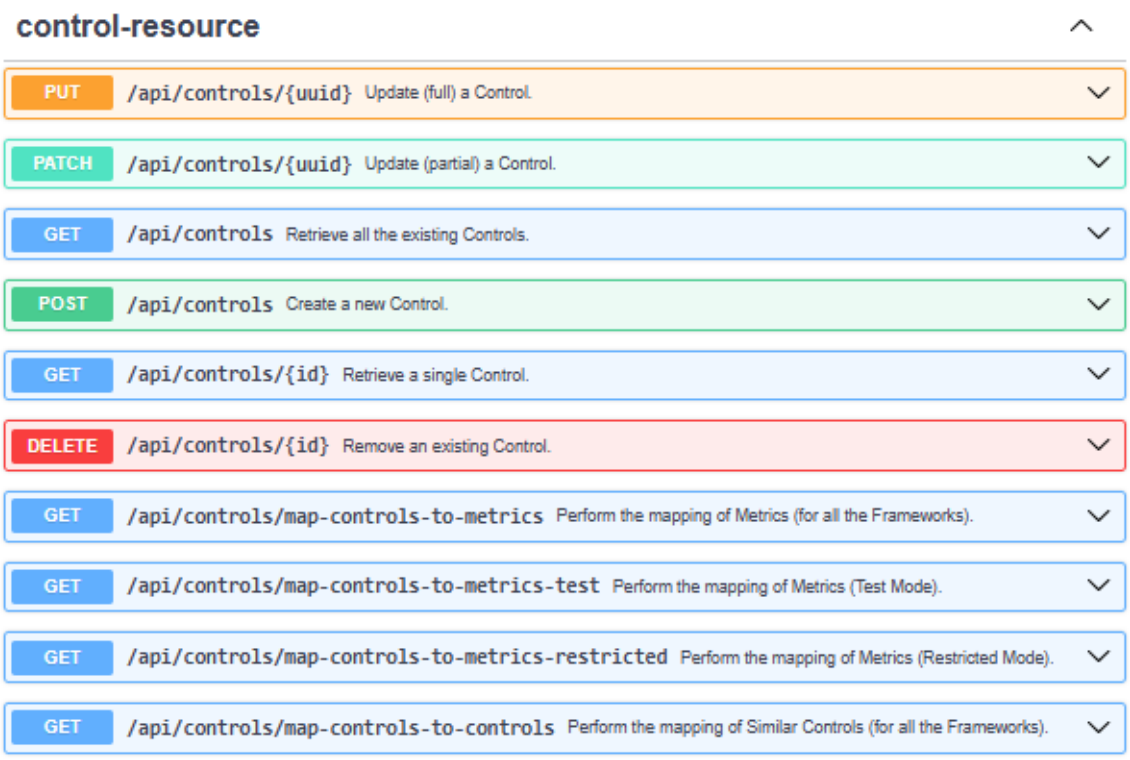
- ***MARI*:** It receives the mapping requests, which include information about schemes and metrics, from the *RCM.* The responses (mappings) are then sent back to the *RCM*, which stores them for further use.
- ***EMERALD UI*:** When the user navigates through the content of the repository, the *EMERALD 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***: It receives the definition of the security metrics that are used to extract and evaluate evidence from policy documents from the *RCM*.

3.2.3.2 Published APIs

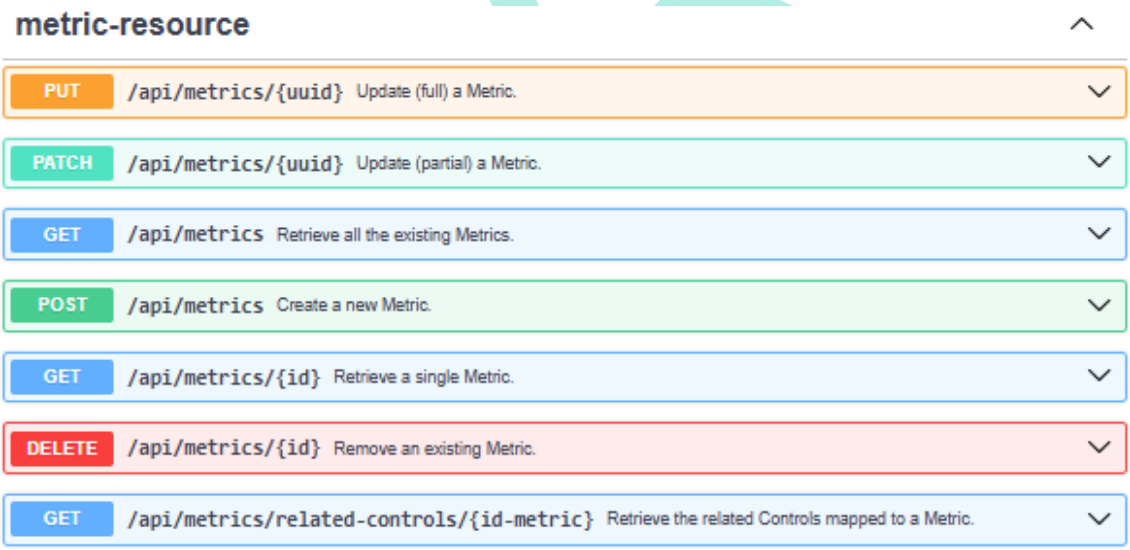
The API endpoints of the *RCM* component are listed below. The API has been extended to include more specific endpoints, primarily in response to user requests as the design and development of the graphical user interface progressed.



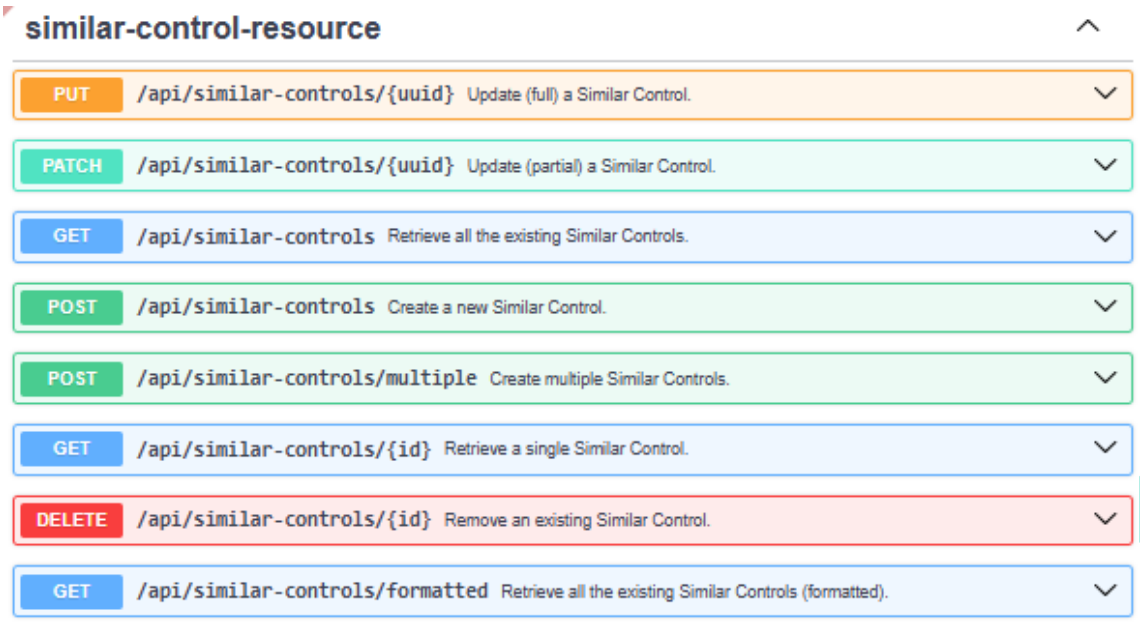
Listing 7. RCM API Endpoints for schema information



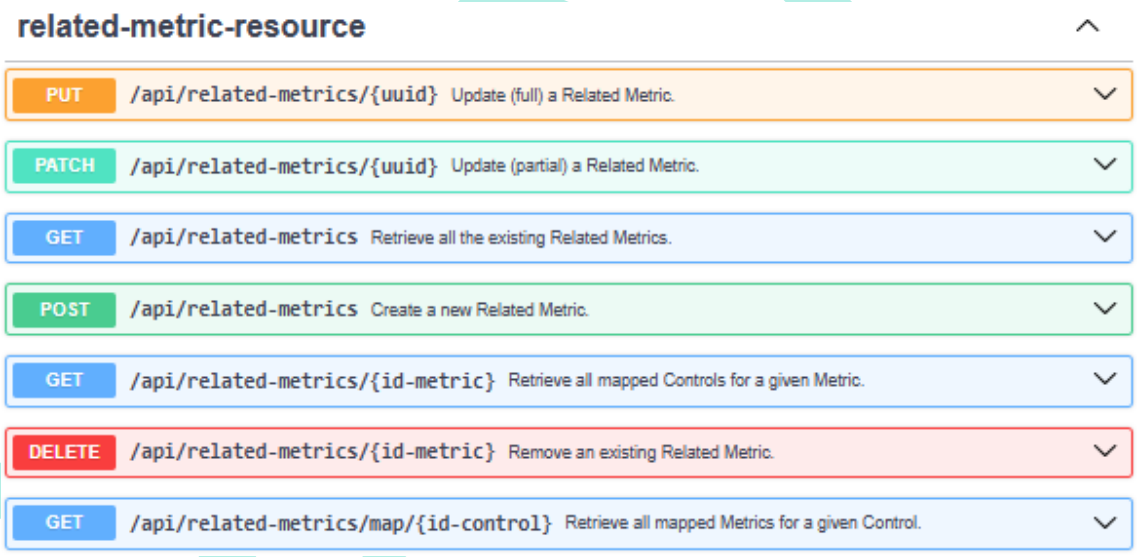
Listing 8. RCM API Endpoints for control information



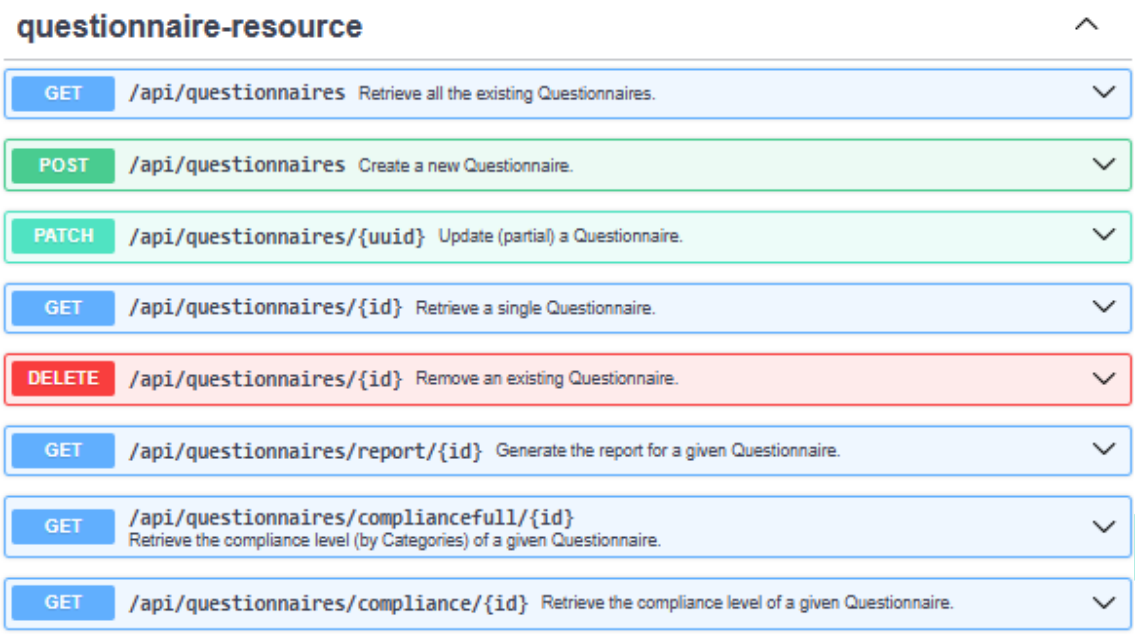
Listing 9. RCM API Endpoints for Metric information



Listing 10. RCM API Endpoints for similar controls



Listing 11. RCM API Endpoints for related metrics



Listing 12. RCM API Endpoints for Questionnaire resources

3.2.3.3 Integration Status

The *RCM* has already 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	Testing	API has been updated and extended to cover UI requirements
Orchestrator	Locally Tested	Input calls to RCM are already integrated. Output call to the *Orchestrator* has been tested locally.
MARI	Integrated	Mapping API is finished unless bugs are detected in further tests
AMOE	Testing	

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)

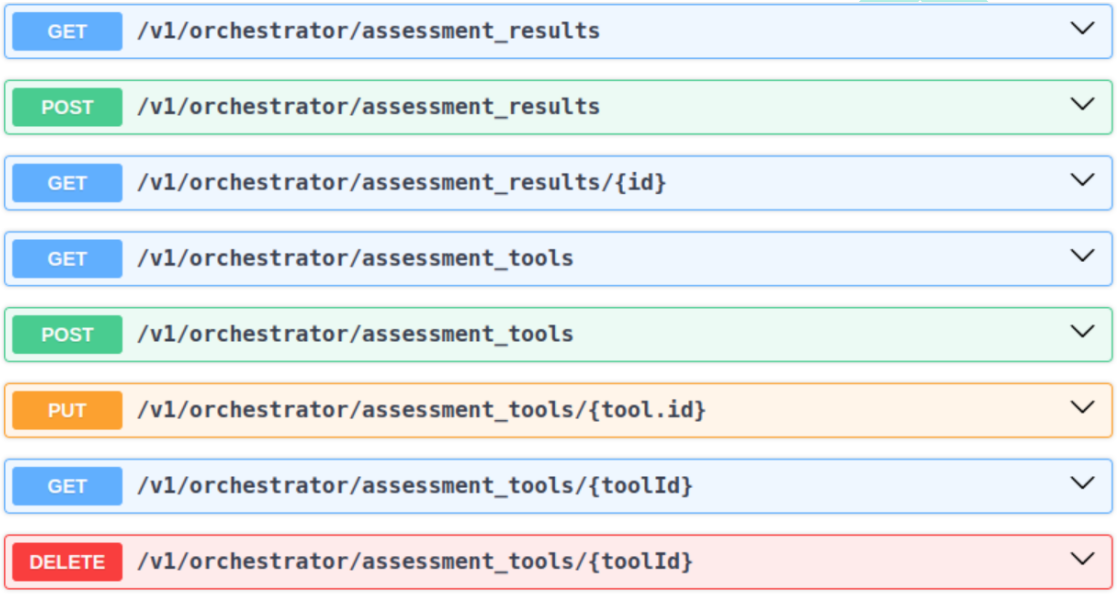
The *Orchestrator* sends and receives information from different sources.

- ***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.

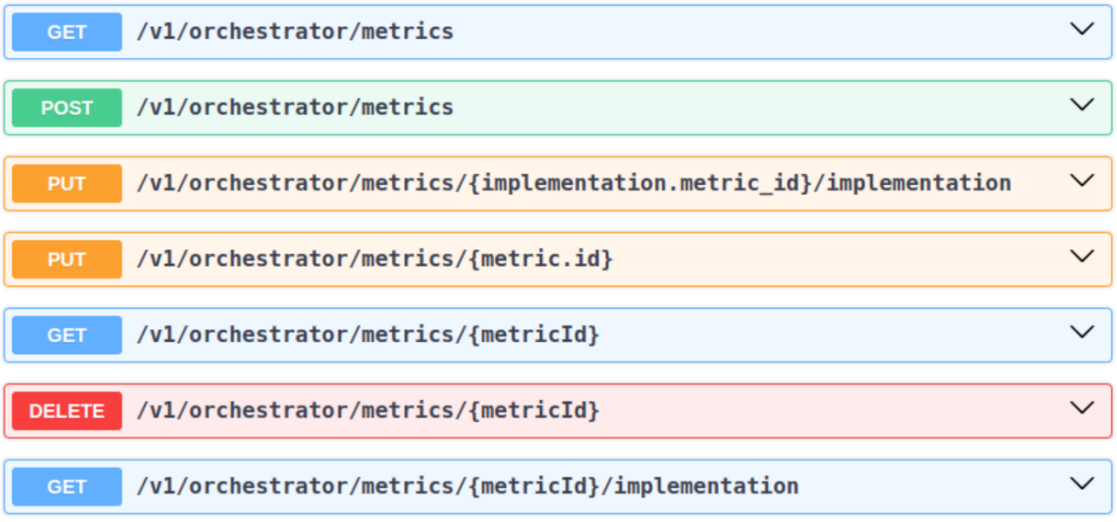
- ***Assessment*:** Assessment results, as well as evidence, are sent by the *Assessment* to the *Orchestrator*.
- ***Evaluation*:** For evaluating the compliance of controls of a security catalogue, the *Orchestrator* sends assessment results to the *Evaluation* and gets the compliance status of each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

3.2.4.2 Published APIs

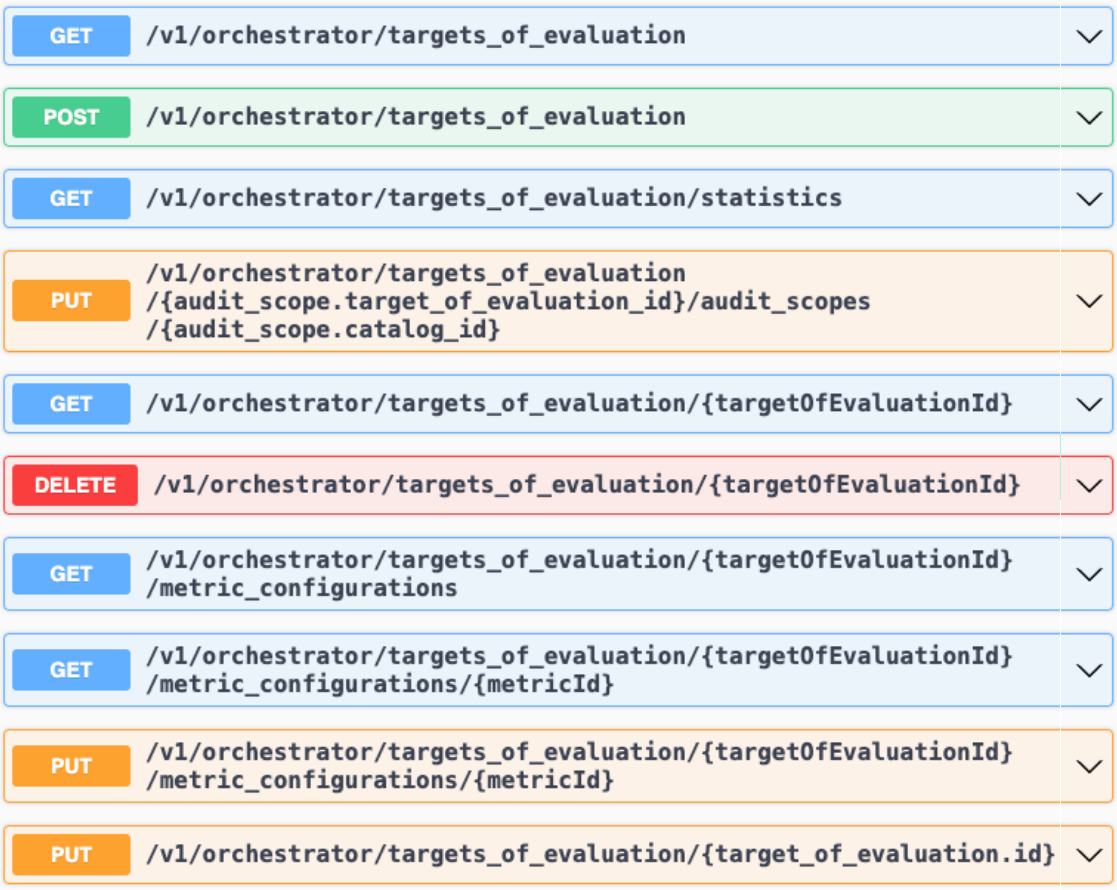
The API endpoints of the *Orchestrator* for handling assessment results and tools are listed below.



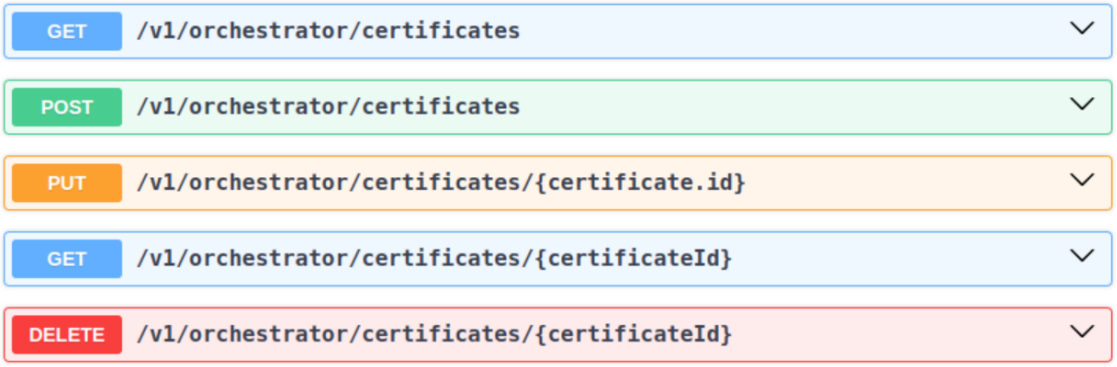
Listing 13. Orchestrator API endpoints for assessment results



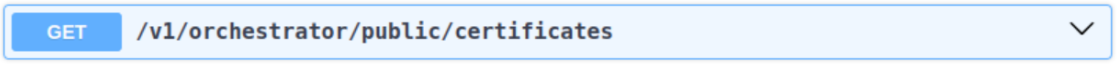
Listing 14. Orchestrator API endpoints for metrics



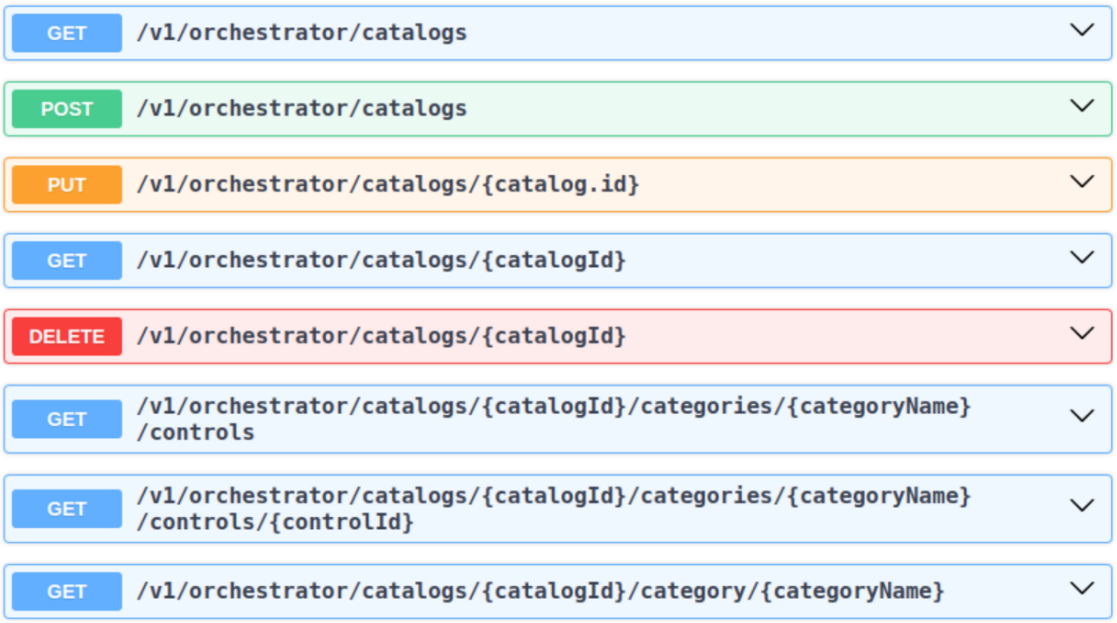
Listing 15. Orchestrator API endpoints for targets of evaluation



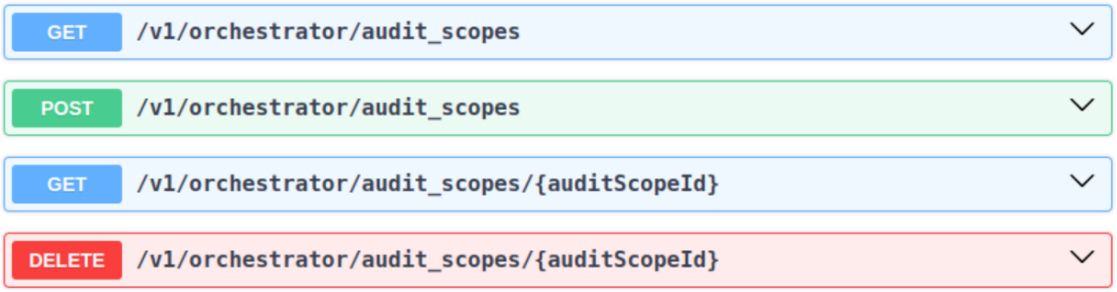
Listing 16. Orchestrator API endpoints for handling certificates



Listing 17. Orchestrator API endpoints for certificates (publicly available)



Listing 18. Orchestrator API endpoints for catalogues



Listing 19. Orchestrator API endpoints for audit scopes

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].

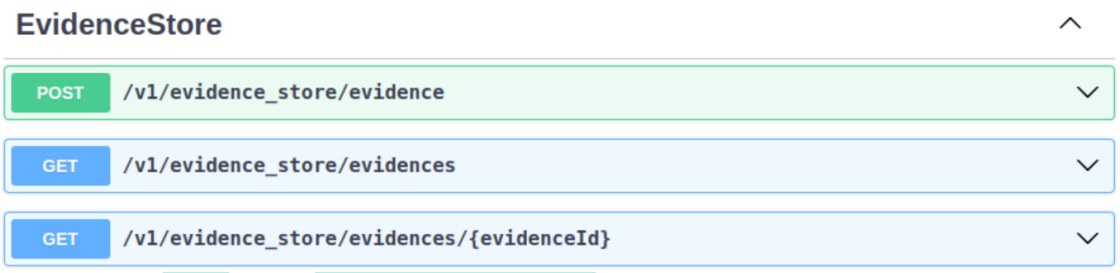
3.2.5.1 Expected behaviour (inputs/outputs)

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.



Listing 20. 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. The Postgres database is likely to be replaced by a graph database in the future.

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

Component	Status	Comment
Assessment	Integrated	Minor changes may occur in the pilot testing phase.
Orchestrator	Integrated	Minor changes may occur in the pilot testing phase.
AMOE	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
Codyze	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may

		occur.
eknows-e3	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
AI-SEC	Integrated	Further metrics and, thus, new evidence are still being developed. Minor changes may occur.
Clouditor-Discovery	Integrated	Minor changes may occur in the pilot testing phase.
EMERALD UI	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.

Listing 21. 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
Evidence Store	Integrated	Minor changes may occur in the pilot testing phase.
Orchestrator	Integrated	Minor changes may occur in the pilot testing phase.
TWS	Integrated	Minor changes may occur in the pilot testing phase.

3.2.7 Evaluation

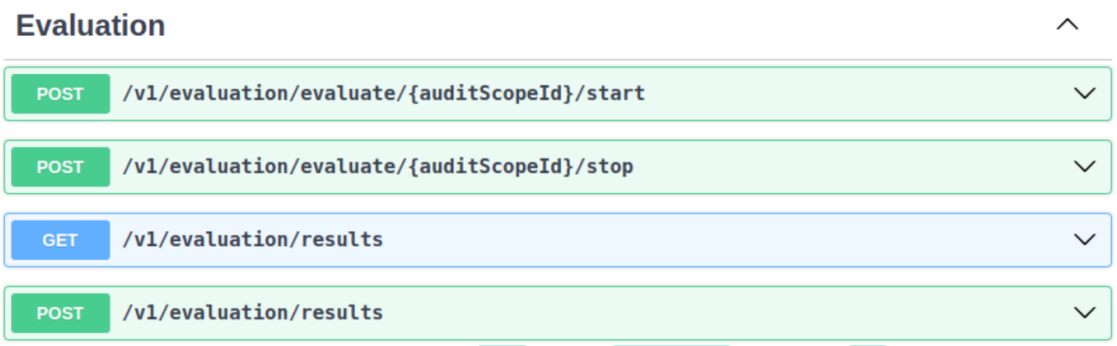
As described in D3.4 [10], the *Evaluation* component is responsible for analysing one or more assessment results to demonstrate compliance with particular controls outlined in a security catalogue.

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.



Listing 22. 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
Orchestrator	Integrated	Minor changes may occur in the pilot testing phase.

3.3 EMERALD UI

The *EMERALD UI* –highlighted in the Figure 17 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.

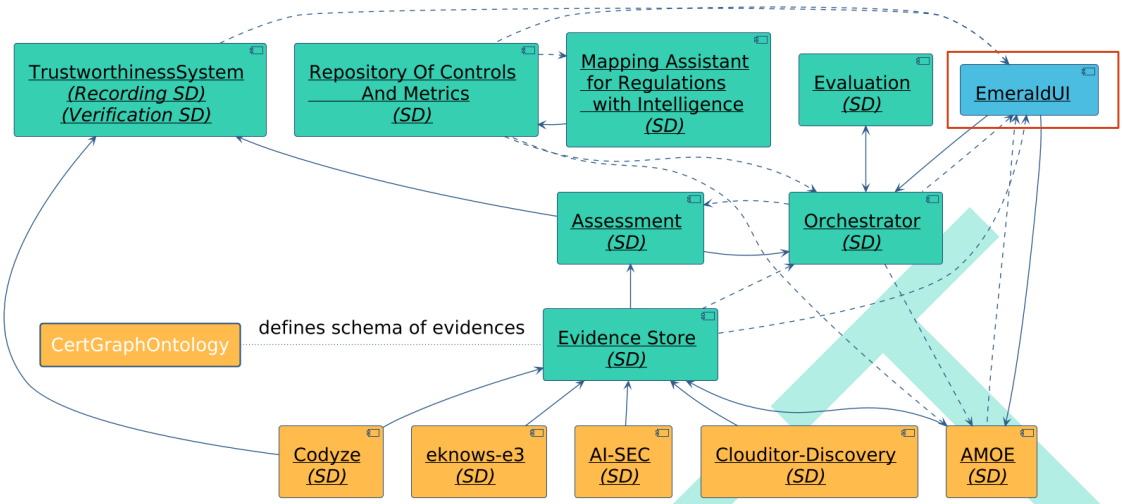


Figure 17. EMERALD UI in the architecture

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 offers 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 Figma[31]. 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 has been presented in the deliverable D4.6 [11] (M27) in detail, please refer there for a more detailed description[32].

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 will be conducted with the different components. Once the component APIs have been updated, they will be integrated into the UI. Currently, data from *AMOE, Evidence Store, Orchestrator, RCM* and TWS can be retrieved, but further testing and implementation is required on all sides. The point-to-point integration status is shown in Table 19.

[31] https://www.figma.com/

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

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

Component	Status	Comment
AMOE	Connected / 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.
Evidence Store	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.
Orchestrator	Testing	Further testing required. Waiting for updates to integrate and implement features such as role-based access, control assignment and control filtering based on the workflow or compliance status. Basic features are implemented and tested (e.g. creating TOE or audit scope).
RCM	Testing	Waiting for updates to the API. Existing endpoints integrated and successfully tested.
TWS	Testing	Testing is ongoing; requires more examples / additional test data, for final validation.

4 Conclusions

The deliverable D1.8 presents the second 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 components –*AI-SEC*, *AMOE*, *Clouditor-Discovery*, *Codyze*, *eknows-e3*, *TWS*, *MARI*, *RCM*, *Orchestrator*, *Evidence Store*, *Assessment*, *Evaluation*, and *EMERALD UI*– since the first version at M18 has progressed significantly. The advancement in the APIs completion of each component has facilitated the integration tasks, and most of the tests have passed from a local environment to the development environment. Many inter-component integrations are already completed, allowing the implementation of a major part of the intended workflow within the EMERALD framework.

The framework is supported by technologies as *Kubernetes* and *Docker*, which has proven to be robust and scalable. During this period, some fine tuning has been implemented, in occasions to solve issues arose in the development environment. To troubleshoot them, the implemented *Rancher* dashboard has been very useful. The CI/CD pipelines have also been extended and refined, to cover the efficient deployment of all components.

This second prototype of the EMERALD framework consolidates the first version and provides a good basis for the deployment of the framework in the pilots as a representation of real-world scenarios.

The testing tasks will continue until the final release, to incorporate the features still being integrated into the framework. User feedback, coming from the validation work package (WP5), and extensive testing of features will be tackled with this version, and all the identified bugs will be tackled to have a final (v3) integrated version of the framework as reliable as possible. The next and final version of the deliverable (D1.9) will contain the results of these tasks and will provide the final status of the integration of the different components.

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,” 2025.

[7] EMERALD Consortium, “D2.5 AMOE – v2,” 2025.

[8] EMERALD Consortium, “D2.9 Runtime evidence extractor - v2,” 2025.

[9] EMERALD Consortium, “D2.3 Source Evidence Extractor – v2,” 2025.

[10] EMERALD Consortium, “D3.4 Evidence assessment and Certification–Implementation-v2,” 2025.

[11] EMERALD Consortium, “D4.6 - EMERALD UI - v2,” 2026.

[12] BSI - Bundesamtes 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 April 2026].

[13] EMERALD Consortium, “D2.8 Runtime evidence extractor - v1,” 2024.

[14] EMERALD Consortium, “D2.11 Certification Graph-v2,” 2026.

[15] NIST - National Institute of Standards and Technology, «OSCAL: the Open Security Controls Assessment Language,» [En línea]. Available: https://pages.nist.gov/OSCAL. [Último acceso: April 2026].

[16] ENISA, “EUCS - Cloud Services Scheme,” [Online]. Available: https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme. [Accessed April 2026].

[17] EMERALD Consortium, “D3.5 - Evidence assessment and Certification – Integration - v1,” 2025.