Deliverable D1.7

EMERALD Integrated solution – v1

Editor(s):	Iñaki Etxaniz
Responsible Partner:	TECNALIA Research & Innovation
Status-Version:	Final-v1.0
Date:	30.04.2025
Type:	Other (SW)
Distribution level (SEN, PU):	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	EMERALD Integrated solution – v1
Due Date of Delivery to the EC	30.04.2025

Workpackage responsible for the Deliverable:	WP1 - Concept and methodology of EMERALD
Editor(s):	Iñaki Etxaniz (TECNALIA)
Contributor(s):	FABA, 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:	Initial 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	11.02.2025	ToC defined	TECNALIA
v0.2	25.02.2025	First draft version	TECNALIA
v0.3	15.03.2025	Updated Sections 1 and 2	TECNALIA
v0.4	18.03.2025	Contributions by consortium partners to Section 3	FABA, TECNALIA, Fraunhofer, CNR, SCCH
v0.5	03.04.2025	Conclusions and Executive Summary. Sent to QA review	TECNALIA
v0.6	14.04.2025	Addressed recommendations from QA review. Sent to final review.	Fraunhofer, TECNALIA
v0.7	19.04.2025	Addressed recommendations from final review	TECNALIA
v1.0	30.04.2025	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 D1.7 presents the initial integrated solution of the EMERALD framework, which includes the integration of 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 related to Work Package 1 (WP1), which focuses on the project's concept and methodology. The integrated solution is a crucial part of the project as it enables collaboration and communication between the different components developed in other work packages.

The integration approach is described first. The integration of the components is based on two main pipelines that automate it: the first one builds the project, creating the *Docker* images and pushing them to the *Artifactory*. The second one deploys the components to the test bed environment and verifies it. The test bed is composed by two environments: integration and production and is based on *OpenStack* Virtual Machines. A three-node *Kubernetes* cluster is mounted on top of them.

An eight-step procedure is defined for the integration of a component in the EMERALD framework. The status of the integration of each component is provided, including the APIs published, and the interconnection with the rest of components.

The main results of this deliverable include the development of an initial prototype of the EMERALD framework; the implementation of an integration strategy that allows collaboration between components; the deployment of components in the integration and production environments; and the documentation of the integration status of each component, providing a basis for future improvements and developments.

Future related work in the project will focus on the continuous integration of the EMERALD framework, including new releases with more functionalities and feedback from the users of the first version of the framework. This work will be reflected in a second version of the deliverable, D1.8, scheduled for month 30 of the project, which will include the updated status of the integration of the EMERALD components.

1 Introduction

1.1 About this deliverable

This is the companion document of the software deliverable D1.7, which aims to have an initial prototype of the EMERALD Compliance-as-a-Service[2] (CaaS) Framework that integrates the components developed by the other technical work packages. This first version of the integrated solution corresponds to the Milestone *MS3 – Integrated Audit Suite v1* and is mainly based on the version v1 of the EMERALD components (M12), although some additional development made until M15 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 second version of the deliverable is planned for month 30 of the project. This second version will incorporate advancements and improvements made in the time between the two 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:

Section 2 presents a general description of the integration strategy and tools. 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 integration status is also provided.

Section 3 provides in more detail the integration status of each EMERALD component, in terms of 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)

2 Integration Overview

The integration strategy that has been defined 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 providing:

- An easy-to-deploy CaaS framework that facilitates on premise installation.
- A stable release that provides some guaranties of supporting 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 shows a general view of the EMERALD components and the data flow, as defined in D1.2 [1]. The colour indicates the component function in the framework, while the dashed/full lines denote pull/push of data.

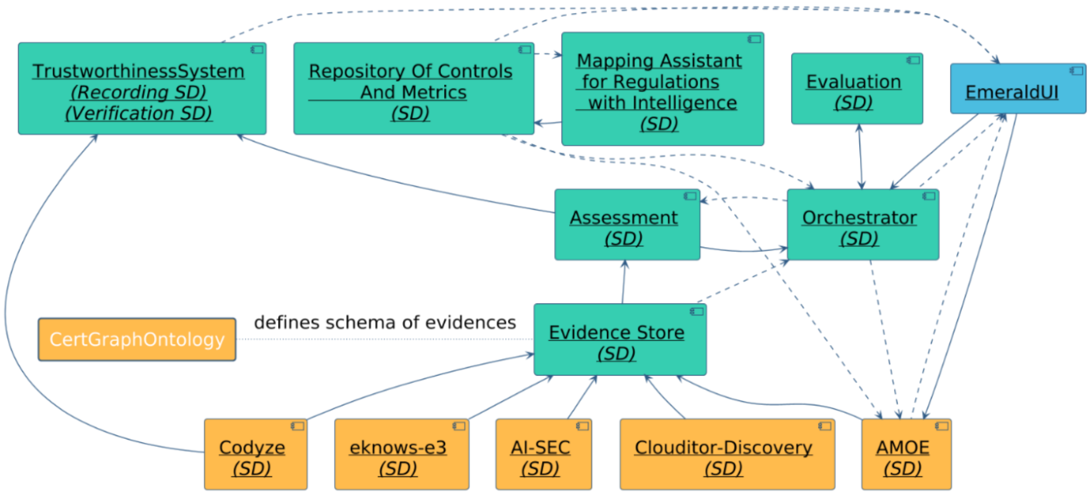


Figure 1. EMERALD Components

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 collected from the cloud service

environment. It offers language-independent static codes analysis as well as business rule extraction.

- ***AI-SEC*** is an evidence collection tool that extracts various security and robustness information from AI models.
- ***Clouditor-Discovery*** is an evidence gathering 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 collector components during the compliance process. It utilizes a graph-based database to organize and manage evidence in an efficient and accessible manner.
- 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 ***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 ***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.

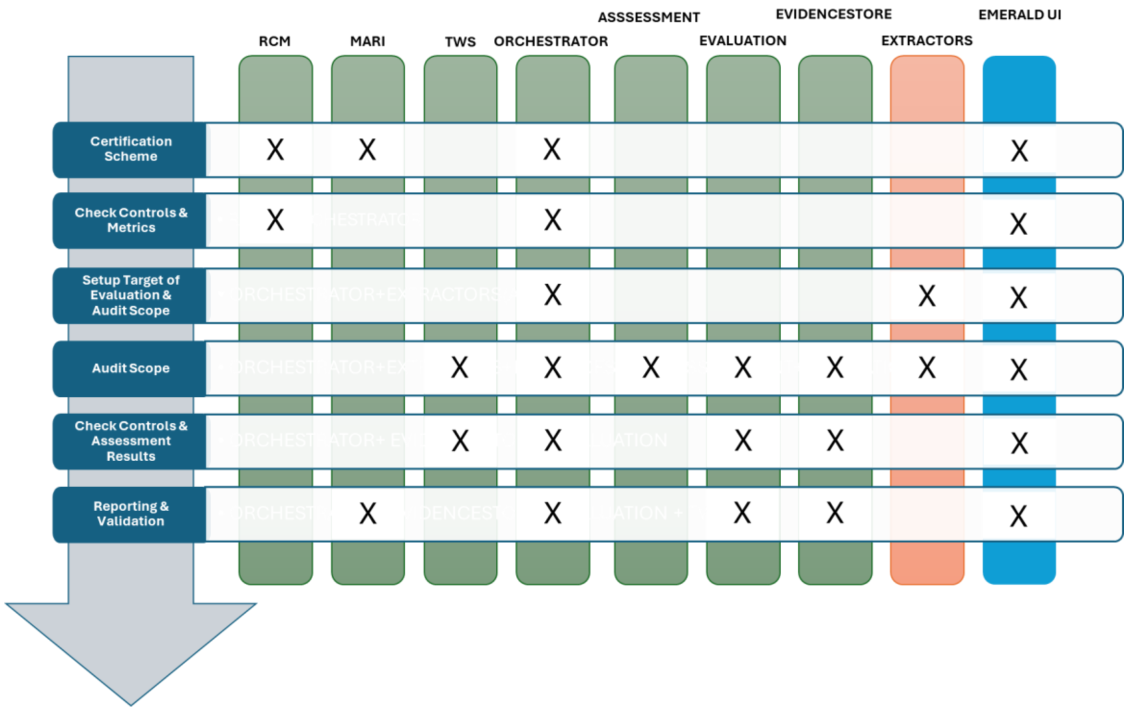
2.1.1 Workflows

The work processes for the first version of the EMERALD framework have been reported for the different pilots and user roles in the WP4 deliverables, where a number of process steps are detailed and based on them, a blueprint for a universal application to implement EMERALD in audit preparation workflows has been elicited.

A condensed version of these steps is provided below, which is then used to provide an overview of the involvement of the components in the processes (for more detailed information, see D4.2 [2]):

- **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 automatically assigned metrics 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 person can add evidence manually and set the control/metric to 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 presented in the Figure 2. There, we can see how the *RCM* and *MARI* components participate in the early phases of the workflow. The *Orchestrator* governs all steps. 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.



	RCM	MARI	TWS	ORCHESTRATOR	ASSSESSMENT	EVALUATION	EVIDENCESTORE	EXTRACTORS	EMERALD UI
Certification Scheme	X	X		X					X
Check Controls & Metrics	X			X					X
Setup Target of Evaluation & Audit Scope				X				X	X
Audit Scope			X	X	X	X	X	X	X
Check Controls & Assessment Results			X	X		X	X		X
Reporting & Validation		X		X		X	X		X

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 [3] and completed in D1.6 [4], 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 each 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 the project, 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.

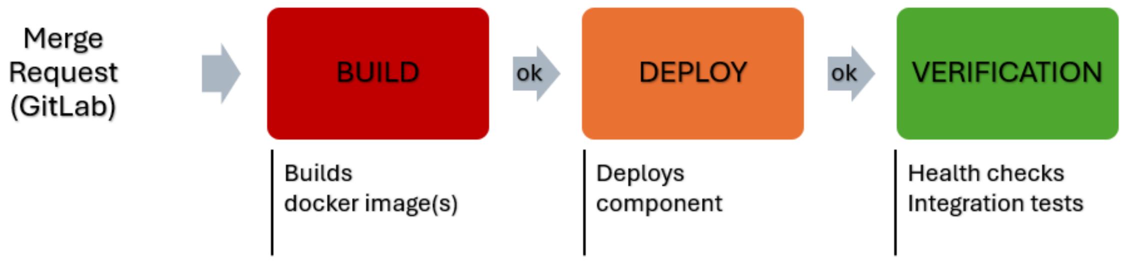


Figure 3. Merge Request and generic CI/CD pipelines

2.2 Components Integrated in the EMERALD Framework v1

The components involved in the integration of the first version of the EMERALD framework are those that were available at month 12 of the project. These components are listed in Table 1, together with their respective Key Results (KR) and the deliverables in which they are described.

Table 1. Components in the EMERALD framework v1

Component name	Key Result	Deliverable
AI-SEC	KR5	D2.6 [5]
AMOE - Assessment and Management of Organisational Evidence	KR2	D2.4 [6]
Clouditor-Discovery	KR1	D2.8 [7]
Codyze	KR1	D2.2 [8]
eknows-e3	KR1	D2.2 [8]
TWS - Trustworthiness System	KR7	D3.3 [9]
MARI - Mapping Assistant for Regulations with Intelligence	KR3	D3.3 [9]

Component name	Key Result	Deliverable
RCM - Repository of Controls and Metrics	KR7	D3.3 [9]
Orchestrator	KR4	D3.3 [9]
Evidence Store	KR2	D3.3 [9]
Assessment	KR4	D3.3 [9]
Evaluation	KR4	D3.3 [9]
EMERALD UI	KR6	D4.5 [10]

All components described in this deliverable (except *EMERALD UI*) use interfaces for integration that follow the well-known gRPR[4] or REST[5] API practices and are described in the OpenAPI[6] format. The interfaces themselves and the interaction with them are described in Section 3. The *EMERALD UI*, in turn, is a web user interface component that uses the components' APIs to interact with them.

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 Targets of Evaluation to the right users. We have chosen *Keycloak*[7] as open-source IAM solution for the project.

2.3 Test Bed Environment

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, the components are promoted to the Production environment.

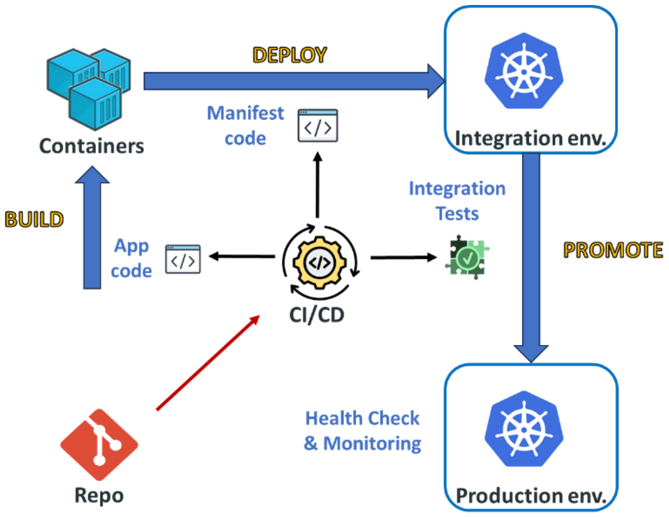


Figure 4. Integration and Production environments in the EMERALD CaaS framework

[4] https://grpc.io/

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

The 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***[8] cluster over an ***OpenStack***[9] 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[10] (Secure Shell) protocol, using digital certificates through a bastion host[11]. 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
```

where dev/prod refer to integration/production environment, respectively. The 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`.

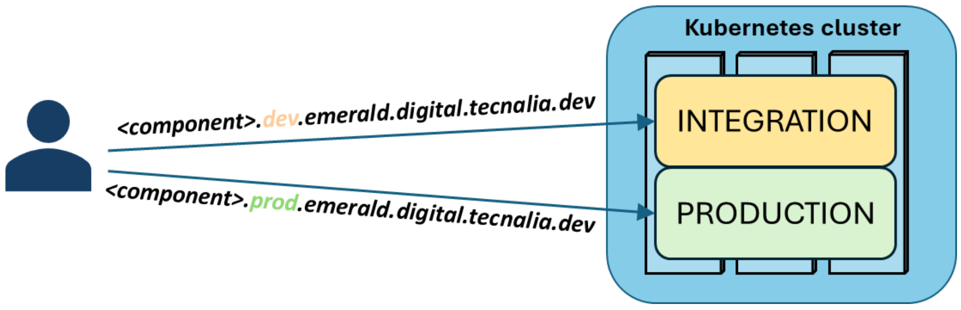

Kubernetes cluster
<component>.dev.emerald.digital.tecnalia.dev
<component>.prod.emerald.digital.tecnalia.dev
INTEGRATION
PRODUCTION

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**[12]), an open-source, enterprise-ready Kubernetes distribution, focused on the security and compliance within the U.S. Federal Government sector. 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`.

[8] https://kubernetes.io/docs/home/
[9] https://www.openstack.org
[10] https://www.ssh.com/academy/ssh/protocol
[11] https://en.wikipedia.org/wiki/Bastion_host
[12] https://docs.rke2.io/

We have followed an Infrastructure as Code (IaC) approach for the deployment. For the creation and configuration of the cluster we have used ***OpenTofu***[13] and ***Ansible***[14] 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 management process, in the Gitlab repository of the project. The usage of IaC provides several advantages to the 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.
- 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***[15] 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". The repository contains a folder with the details of the deployment of the individual components (called *components*), and other folders that describe the environments: integration and production.

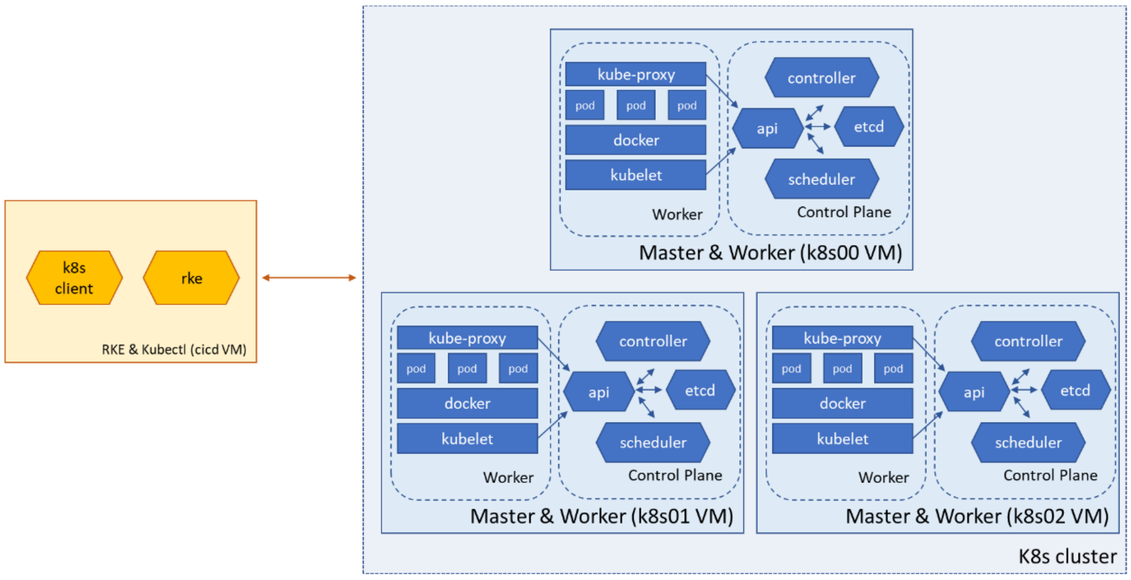
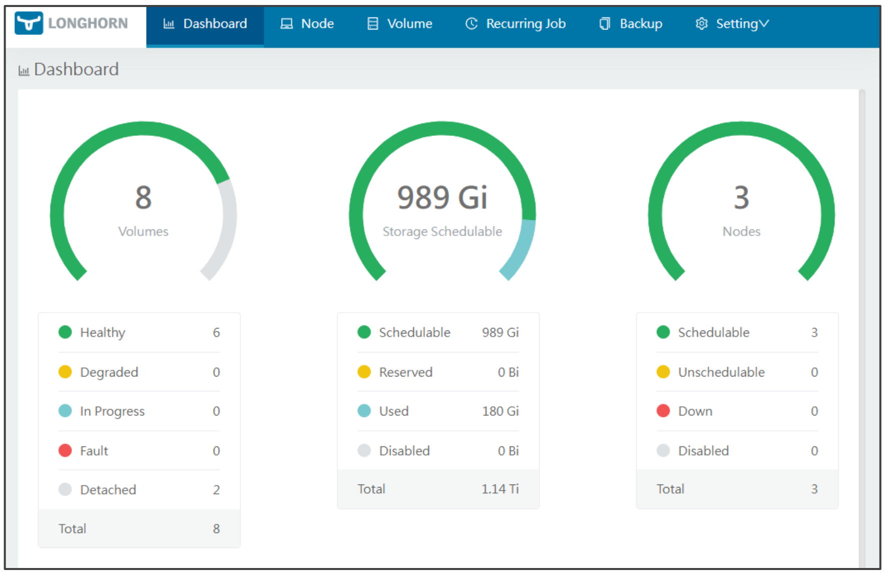


Figure 6. Kubernetes cluster installation with RKE2

[13] https://opentofu.org
[14] https://www.ansible.com
[15] *Kustomize* traverses a *Kubernetes* manifest to add, remove or update configuration options without forking. More information is available at https://kustomize.io/

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***[16]. 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.



8 Volumes

● Healthy	6
● Degraded	0
● In Progress	0
● Fault	0
● Detached	2
Total	8

989 Gi Storage Schedulable

● Schedulable	989 Gi
● Reserved	0 Bi
● Used	180 Gi
● Disabled	0 Bi
Total	1.14 Ti

3 Nodes

● Schedulable	3
● Unschedulable	0
● Down	0
● Disabled	0
Total	3

Figure 7. Longhorn dashboard in Rancher

2.3.3 Docker registry

The micro-services running on the Kubernetes cluster are packaged in *Docker* images and stored in a private *Docker* Registry running in the TECNALIA infrastructure's ***Artifactory***[17]. 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 images are pushed to the *Docker* registry by the GitLab CI/CD pipelines in the following URL according to the structure agreed for the project, as shown in Figure 8.

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

[16] *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/

[17] https://jfrog.com/artifactory/

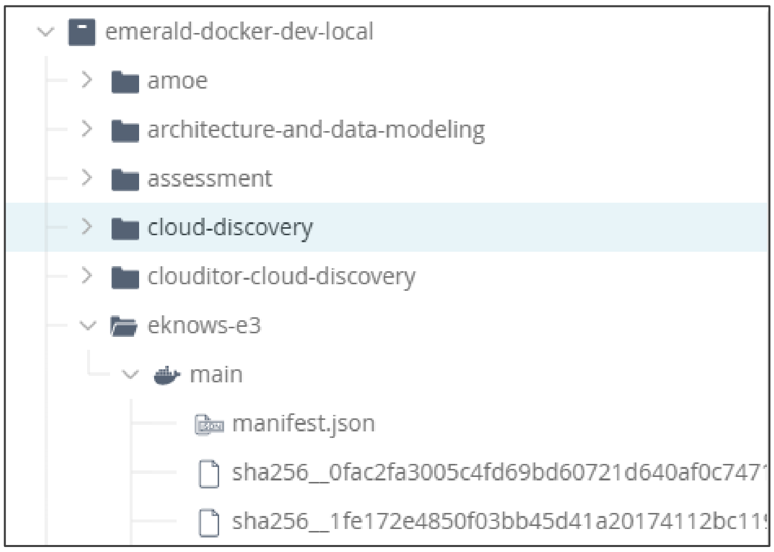

```
emerald-docker-dev-local
  amoe
  architecture-and-data-modeling
  assessment
  cloud-discovery
  clouditor-cloud-discovery
  eknows-e3
    main
      manifest.json
      sha256__0fac2fa3005c4fd69bd60721d640af0c747
      sha256__1fe172e4850f03bb45d41a20174112bc11
```

Figure 8. EMERALD Docker registry

2.3.4 Network

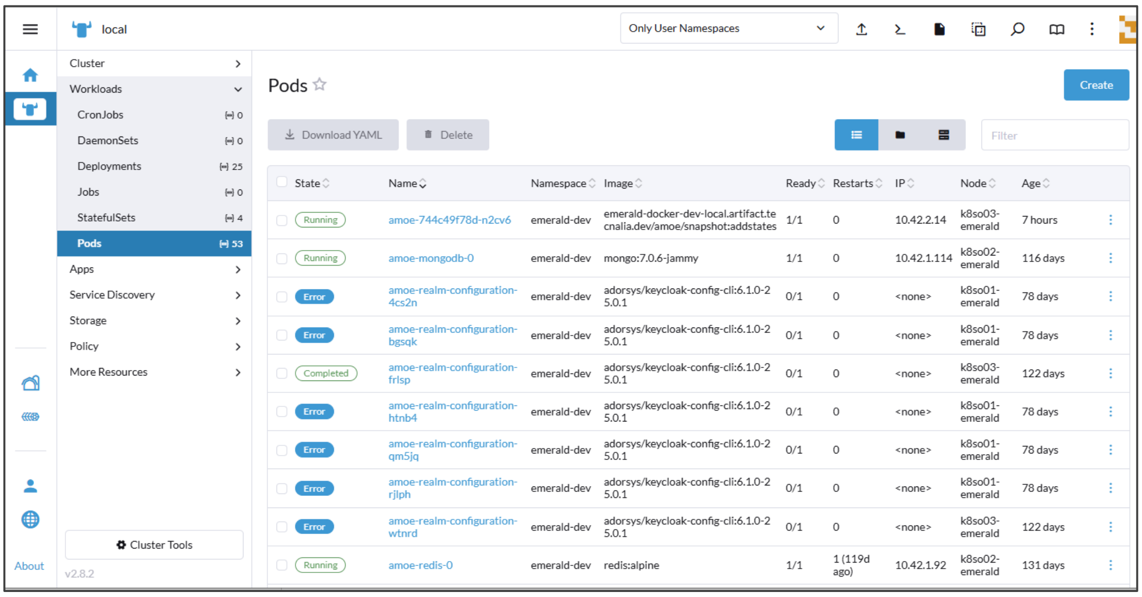
On the Kubernetes cluster, a ***nginx***[18] service is configured as a proxy to redirect all the requests to the correct micro-service component. The binding between the *nginx* service and the public IP is setup with ***KubeVip***[19], a network load-balancer that associates the public IP to the *nginx* service and uses standard routing protocols to make available (part of) the network behind the Kubernetes cluster. It is essential for the EMERALD cluster because, unlike a public cloud provider cluster, *nginx* has no load balancer, and Kubernetes does not provide it by itself.

2.3.5 Dashboard

We have two accounts in Kubernetes, with different permissions: one that has access to all cluster resources and to the administration options ("admin"), and one that has the permissions restricted to the integration and production namespaces ("emerald_developer").

Rancher provides a web-based dashboard for the Kubernetes cluster (see Figure 9). It is helpful to deploy containerised applications to a Kubernetes cluster, troubleshoot them, and manage cluster resources.

[18] https://www.nginx.com/
[19] https://kube-vip.io/



State	Name	Namespace	Image	Ready	Restarts	IP	Node	Age
Running	amoe-744c49f78d-n2cv6	emerald-dev	emerald-docker-dev-local.artifact.tecnalia.dev/amoe/snapshot:addstates	1/1	0	10.42.2.14	k8so03-emerald	7 hours
Running	amoe-mongodb-0	emerald-dev	mongo:7.0.6-jammy	1/1	0	10.42.1.114	k8so02-emerald	116 days
Error	amoe-realm-configuration-4cs2n	emerald-dev	adorsys/keycloak-config-cli:6.1.0-25.0.1	0/1	0	<none>	k8so01-emerald	78 days
Error	amoe-realm-configuration-bgsqk	emerald-dev	adorsys/keycloak-config-cli:6.1.0-25.0.1	0/1	0	<none>	k8so01-emerald	78 days
Completed	amoe-realm-configuration-frlsp	emerald-dev	adorsys/keycloak-config-cli:6.1.0-25.0.1	0/1	0	<none>	k8so03-emerald	122 days
Error	amoe-realm-configuration-htnb4	emerald-dev	adorsys/keycloak-config-cli:6.1.0-25.0.1	0/1	0	<none>	k8so01-emerald	78 days
Error	amoe-realm-configuration-qm5jq	emerald-dev	adorsys/keycloak-config-cli:6.1.0-25.0.1	0/1	0	<none>	k8so01-emerald	78 days
Error	amoe-realm-configuration-rjlph	emerald-dev	adorsys/keycloak-config-cli:6.1.0-25.0.1	0/1	0	<none>	k8so01-emerald	78 days
Error	amoe-realm-configuration-wtnrd	emerald-dev	adorsys/keycloak-config-cli:6.1.0-25.0.1	0/1	0	<none>	k8so03-emerald	122 days
Running	amoe-redis-0	emerald-dev	redis:alpine	1/1	1 (119d ago)	10.42.1.92	k8so02-emerald	131 days

Figure 9. Rancher Dashboard

2.3.6 Certificates

Access to the Dashboard is secure via HTTPS. The certificates are installed using ***Cert-Manager***[20]. *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***[21]. 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. The Dashboard is exposed over HTTPS at the address: https://k8so.emerald.digital.tecnalia.dev/dashboard/.

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).

[20] https://cert-manager.io/docs/
[21] https://letsencrypt.org/

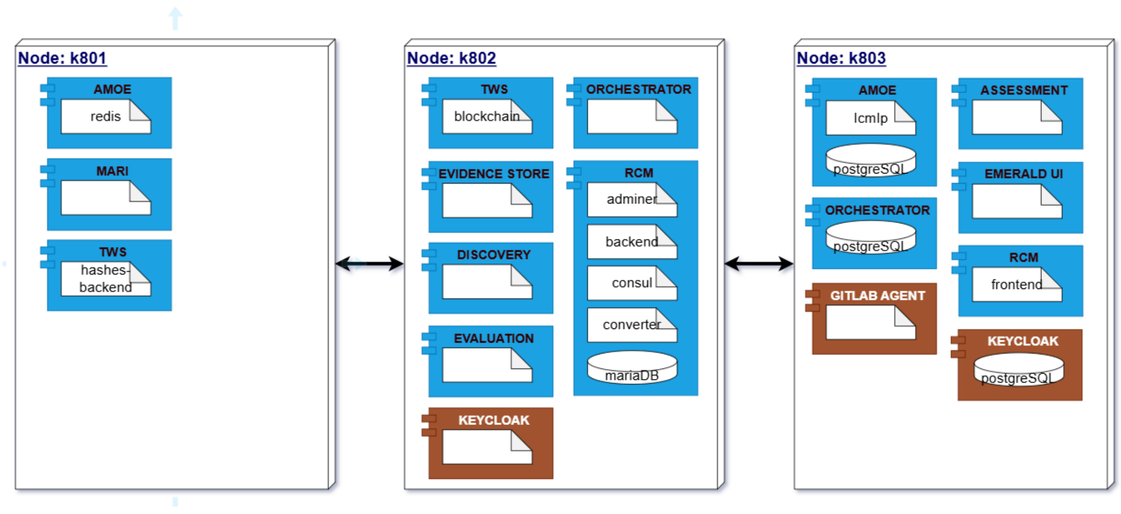


Figure 10. Deployment diagram

Figure 10 represents the deployment at a given time. The distribution of the artifacts among the nodes is managed by Kubernetes, and it is possible that a single component has its artifacts distributed on different nodes (e.g., *AMOE* or *RCM*). This distribution is automatically modified by Kubernetes attending to its own performance and resources management criteria.

The *Codyze*, *eknows-e3* and *AI-SEC* components are not present in Figure 10, as they are evidence extractors that are intended to run in a separate environment, alongside a GitLab runner that gives them access to the files to be analysed.

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[22]:

1. The source code of each component must be uploaded to the private GitLab repository.
2. Once finalised and tested, each 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.
3. The *Docker* image must be made available on the private docker registry *Artifactory*.
4. The configuration and side services for each 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 with the integration environment.
7. Automated integration tests are performed in the integration environment.
8. If the tests are passed, the version is promoted to the production environment and a new release is created.

[22] The integration of non-open source component skips steps 1 and 2.

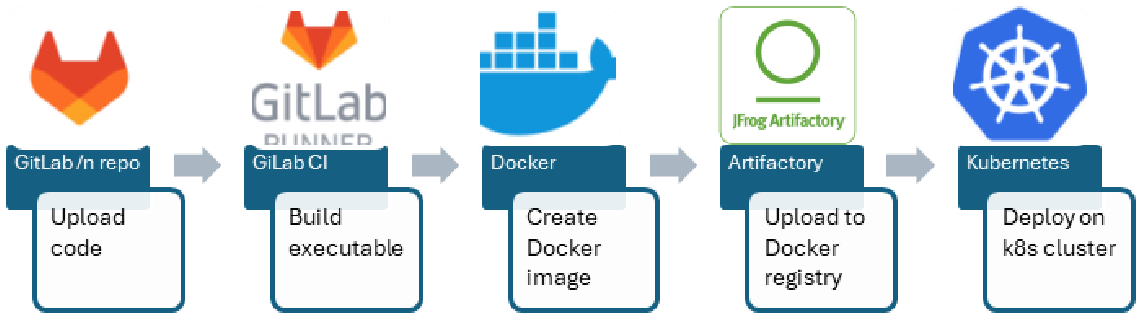


Figure 11. Component integration main steps

The integration plan includes three phases that will be completed in months M18, M30, and M34, respectively. Currently, we have performed the first round, where the integration of components has been carried out manually by each partner.

During this first round, WP1 delivered a workshop to the rest of the consortium to introduce the main concepts of *Docker* and *Kubernetes* and explained the integration steps. 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 are containerised and have been deployed on the project-internal integration server. Dockerfile recipes are available to easy recreate the integration environment. All REST API endpoints are exposed on a common network to enable communication between components. The EMERALD Framework is installed entirely using Kubernetes manifests.

In phases 2 and 3 we expect to have the deployment fully automated, governed by the CI/CD pipelines. We will also set-up the production environment, with a stable version always available in it. And finally, the installation will be implemented in the pilots.

2.5 Overall status of the integration

This section provides an overview of the integration status of the components in the EMERALD framework. Table 2 shows the steps to be carried out for the integration of each component, as well as their degree of completion. Section 3 provides more details on the level of integration of each component.

Table 2. Integration status

Component	License	Gitlab Repo	Public Repo	README	Docker Images	OpenAPI spec	K8s file	Deploy pipeline	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)	√	√	√	X	√ (CLI)	N/A	X	X
eKnows-e3 *	√ (Apache, others)	√	√ & N/A	√	√	√ (CLI)	N/A	X	X
AI-SEC	√ (Apache)	√	√	√	X	X	N/A	X	X
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[23] (except for two of them -*TWS* and *eknows-e3*– which are not open source licensed). 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*[24]. This 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 through the GitLab CI pipelines, which are detailed in D1.6 [4].

The last column of Table 2 indicates if the component is available in the integration environment. All of them are deployed, excepting those 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*).

The final and fundamental part of the integration has to do with the communication among components, which is done through the APIs defined and developed in EMERALD. Besides the details provided in the third section for each component, Table 3 presents a consolidated view of the current status of the interaction of each component with the others. The status has been categorized into several stages[25], reflecting the progress made in integrating the 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.
- **Tested Locally:** The component has undergone local testing, verifying its functionality in isolation. While it works as intended on its own, it has not yet been tested in conjunction with other components.
- **Connected**: 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.

Please note that in Table 3, the column “Component A” refers to the component that implements the API and “Component B” is the component that invokes it.

[23] https://git.code.tecnalia.dev/emerald/public
[24] 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.
[25] This categorization was first used in deliverable D3.5 [16]

Table 3. Point-to-point integration status

Component A	Component B	Status	Comment
AI-SEC	*EMERALD UI*	Not started	AI-SEC is currently in the tool testing phase and has not yet started its integration
AMOE	*Evidence Store*	Testing	Waiting for Ontology and full integration of new metrics based on updated data model
AMOE	*RCM*	Tested locally / Connected	Tested with initial API. Waiting for adjustment to new metric data model and inclusion of extended set of metrics
AMOE	*Orchestrator*	Not started	Waiting for updates to the *Orchestrator* API
AMOE	*EMERALD UI*	Testing, Connected	Collected files and extracted results. Full implementation remains to be tested. Also, extensive testing in EMERALD integration setup remains to be done. The *Keycloak* configuration needs to be updated for a stable deployment and test setup.
Clouditor-Discovery	*Evidence Store*	Connected	Testing pending
Codyze	*[CI/CD] **	Tested locally	*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.
Codyze	*Evidence Store*	Tested locally	We can send pieces of evidence. However, they are not fully filled.
Codyze	*TWS*	Not started	Currently postponed in favour of the integration with *Evidence Store* and awaiting final API specification of *TWS*.
eknows-e3	*[CI/CD] **	Tested locally	The CI/CD component uses *eknows-e3* to extract and save evidence in the *Evidence Store*. A demo showcases how the *eknows-e3* component can be integrated. Integration tests are currently being developed.
eknows-e3	*Evidence Store*	Testing	Communication is implemented, integration tests are in development.
TWS	*EMERALD UI*	Developing API	Currently updating the API details due to the migration process.
TWS	*Evidence Store*	Developing API	Currently updating the API details due to the migration process.
TWS	*Evidence collectors*	Developing API	Currently updating the API details due to the migration process.
MARI	*RCM*	Developing API	New API defined .
RCM	*EMERALD UI*	Developing API	Updating / extending the API.
RCM	*Clouditor-Orchestrator*	Developing API	Changes are needed due to data model updates.
RCM	*MARI*	Developing API	New mapping API already defined.

Component A	Component B	Status	Comment
RCM	*AMOE*	Connected	Changes are needed due to data model updates.
Orchestrator	*Assessment*	API Finished	-
Orchestrator	*EMERALD UI*	API finished	Requires coordination with WP4 and testing.
Orchestrator	*RCM*	Developing API	-
Orchestrator	*Assessment*	Connected	Testing pending.
Orchestrator	*Evaluation*	Connected	Testing pending.
Evidence Store	*Assessment*	Connected	Testing pending.
Evidence Store	*Orchestrator*	Connected	Testing pending.
Evidence Store	*AMOE*	Testing	-
Evidence Store	*Codyze*	Tested locally	-
Evidence Store	*eknows-e3*	Testing	-
Evidence Store	*AI-SEC*	Testing	-
Evidence Store	*Clouditor-Discovery*	Connected	Testing pending.
Assessment	*Evidence Store*	Connected	Testing pending.
Assessment	*Orchestrator*	Connected	Testing pending.
Assessment	*TWS*	Developing API	To be tested.
Evaluation	*Orchestrator*	Connected	Testing pending.
EMERALD UI	*AMOE*	Connected	Testing remains to be done. Only a subset of the endpoints has been fully integrated yet.
EMERALD UI	*Orchestrator*	Developing API	Waiting for *Orchestrator* results.
EMERALD UI	*RCM*	Tested locally	Waiting for updates to the API.
EMERALD UI	*TWS*	Not started	Discussion for endpoints and integration needed.

The information reflected in Table 3 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 "Tested locally" being 50%), an approximation to the global value can be calculated, resulting in 40% at the end of this point-to-point integration at M18. The majority of the 38 elements in the table are in the "Connected" status (13), followed by the "Developing API" status (11).

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 Collectors; (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 Collectors

Evidence collectors –highlighted in the Figure 12 below– are the components in charge of collecting 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.

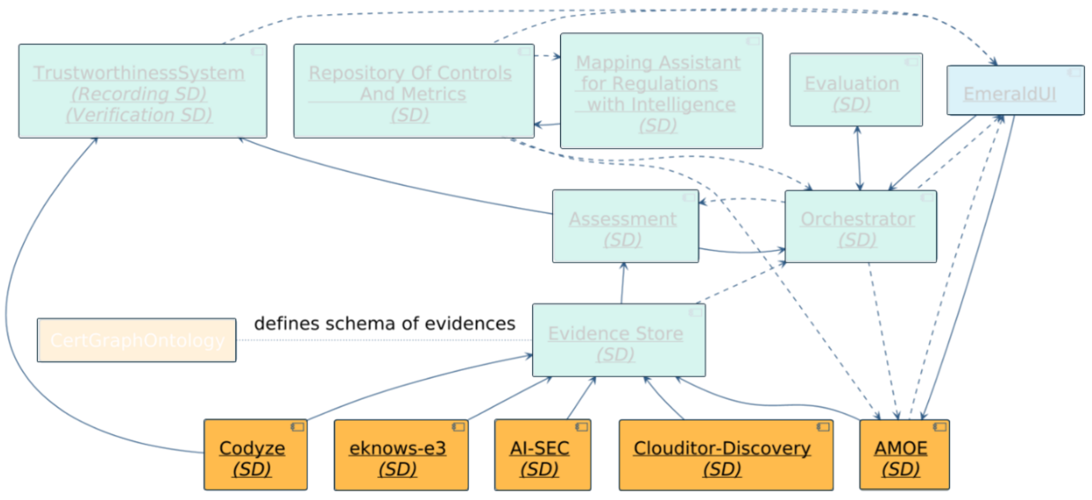

TrustworthinessSystem (Recording SD) (Verification SD)
Repository Of Controls And Metrics (SD)
Mapping Assistant for Regulations with Intelligence (SD)
Evaluation (SD)
EmeraldUI
Assessment (SD)
Orchestrator (SD)
CertGraphOntology
defines schema of evidences
Evidence Store (SD)
Codyze (SD)
eknows-e3 (SD)
AI-SEC (SD)
Clouditor-Discovery (SD)
AMOE (SD)

Figure 12. Evidence collectors in the EMERALD architecture

3.1.1 AI-SEC

AI-SEC is an evidence collector designed to extract relevant information from machine learning models. Based on the Criteria Catalogue for AI Cloud Services (AIC4) [11], *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 connects to *AI-SEC* for uploading, downloading, viewing, and deleting policy documents. *AI-SEC* will be controlled by the user via the *EMERALD UI*, which connects to the API.
- ***Evidence Store***: Evidence will be forwarded to the *Evidence Store*.

3.1.1.2 Published APIs

AI-SEC does not have any published APIs.

3.1.1.3 Integration Status

Table 5 provides the status of the connection to the *EMERALD UI* and the *Evidence Store*.

Table 4. Integration status of AI-SEC with other EMERALD components

Component	**Status**	**Comment**
EMERALD UI	Not started	*AI-SEC* is currently in the tool testing phase and has not yet started its integration tasks.
Evidence Store	Not started	*AI-SEC* is currently in the tool testing phase and has not yet started its integration tasks.

3.1.2 AMOE

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

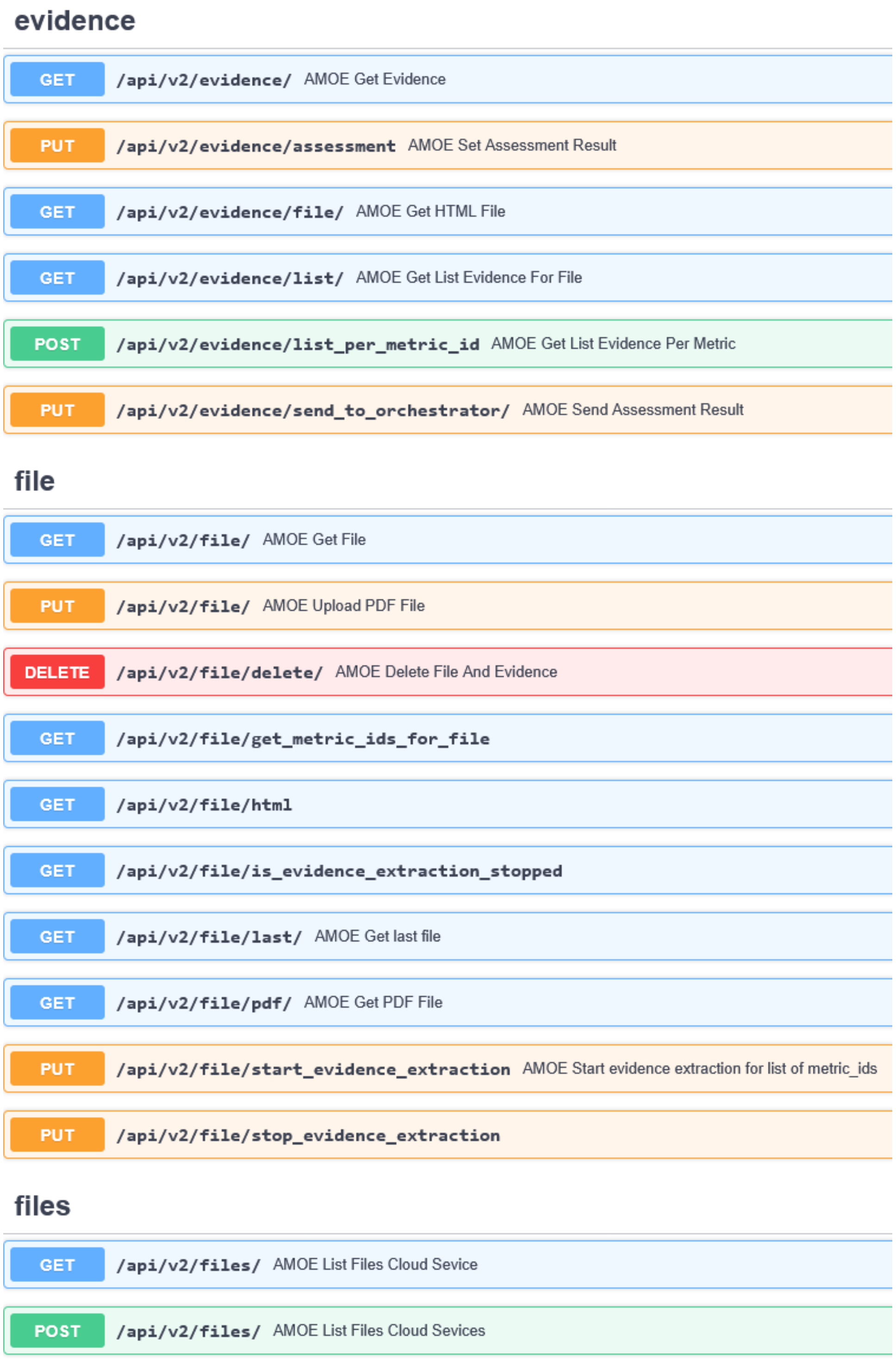
3.1.2.1 Expected behaviour (inputs/outputs)

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 the *AMOE* are listed below.



evidence

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

file

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

files

```
GET     /api/v2/files/  AMOE List Files Cloud Sevice
POST    /api/v2/files/  AMOE List Files Cloud Sevices
```

Listing 1. AMOE API overview

3.1.2.3 Integration Status

AMOE has already been deployed on the Kubernetes cluster. Table 5 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 5. Integration status of AMOE with other EMERALD components

Component	Status	Comment
Evidence Store	Testing	Waiting for Ontology and full integration of new metrics based on the updated data model.
RCM	Tested locally / Connected	Tested with initial API. Waiting for adjustment to new metric data model and inclusion of extended set of metrics.
Orchestrator	Not started	Ready to implement, integrate and test parts of the existing *Orchestrator* endpoints, while waiting for updates from the *Orchestrator* API.
EMERALD UI	Testing, Connected	Collected files and extracted results. Full implementation remains to be tested. Also, extensive testing in EMERALD integration setup remains to be done. The *Keycloak* configuration needs to be updated for a stable deployment and test setup.

3.1.3 Clouditor-Discovery

The *Clouditor-Discovery* is one of the evidence collectors in the EMERALD framework. As described in D2.8 [7], 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* 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. Table 6 provides the current status of the connection to the *Evidence Store*.

Table 6. 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. *Codyze-Compliance* verifies through user-defined queries if the source code complies to security metrics. *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 has only one connection, to the *Evidence Store*:

- ***Evidence Store***: The generated pieces of evidence are sent to the *Evidence Store*. They are annotated with the terms of the ontology enabling assessment by the *Assessment*.

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:

Parameter / Option	Description
`--id <string>`	ID of the cloud service
`--rules <path>`	Path to the 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 7). For *Codyze-Compliance*, a similar integration is planned.

Table 7. Integration status of Codyze with other EMERALD components

Component	Status	Comment
Evidence Store	Tested locally	We can send pieces of evidence. However, they are not fully filled, i.e., the ontology terms are missing for the assessment.
TWS	Not started	Currently postponed in favour of integration with the *Evidence Store* and awaiting final API specification of *TWS*.

3.1.5 eknows-e3

The *eknows-e3* evidence extractor is based on the software analysis platform *eknows*[26] and delivers required evidence to verify if an application source code complies to security metrics. As described in D2.2. [8], *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 will be 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 [12] 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 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 from source code files 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 is integrated into a CI/CD pipeline and 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.certificationTargetId <arg>
    --evidence.toolId <arg>
    --evidence.toolName <arg>
-f,--file <arg> file to analyze
```

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

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

DOCKER_AUTH_CONFIG in the repository we want to test. Currently, only one file can be specified for extracting the evidence. However, it is planned to extend it to directory specification in the future. Table 8 provides the current status of the connection to the *Evidence Store*.

Table 8. Integration status of ekows-e3 with other EMERALD components

Component	Status	Comment
Evidence Store	Testing	Communication is implemented, integration tests are in development.

A demo showcases how the *eknows-e3* component can be integrated. Integration tests are currently being developed.

3.2 Evidence Assessment and Certification

Evidence assessment and certification components –highlighted in Figure 13– 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*, the *Mapping Assistant* or the *Trustworthiness System*.

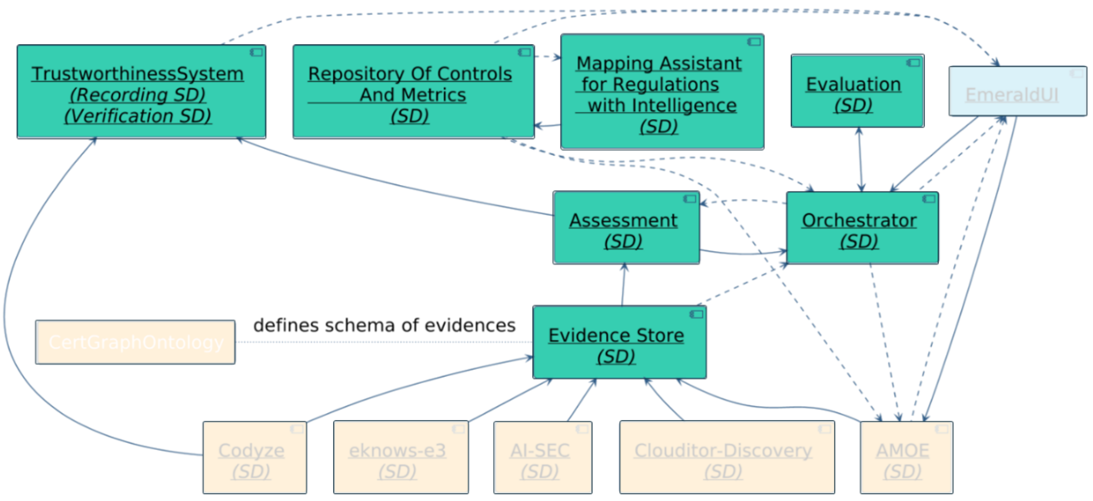

TrustworthinessSystem (Recording SD) (Verification SD)
Repository Of Controls And Metrics (SD)
Mapping Assistant for Regulations with Intelligence (SD)
Evaluation (SD)
EmeraldUI
Assessment (SD)
Orchestrator (SD)
CertGraphOntology
defines schema of evidences
Evidence Store (SD)
Codyze (SD)
eknows-e3 (SD)
AI-SEC (SD)
Clouditor-Discovery (SD)
AMOE (SD)

Figure 13. Assessment tools in the EMERALD architecture

3.2.1 TWS

The *TWS* ensures the trustworthiness, fairness, and transparency of the evidence and assessment results stored in EMERALD, safeguarding their integrity and authenticity. 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.

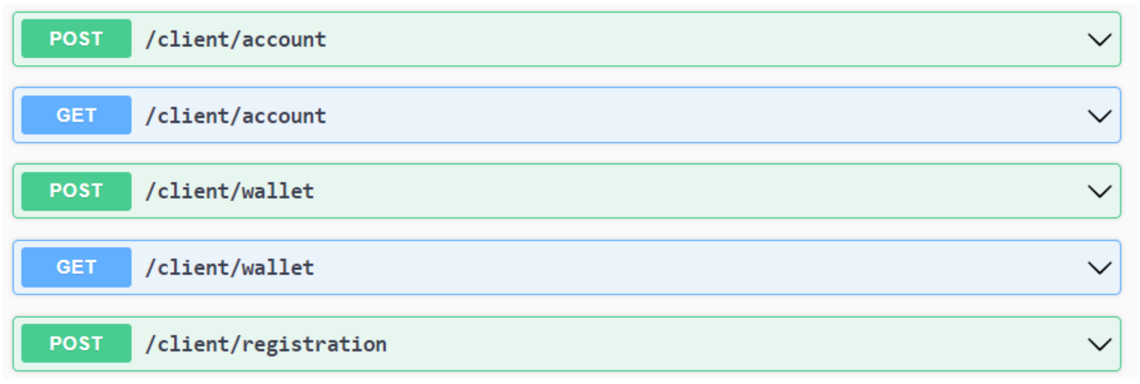
3.2.1.1 Expected behaviour (inputs/outputs)

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 collectors:** Proofs of integrity for evidence can be directly provided from the evidence collectors. 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.

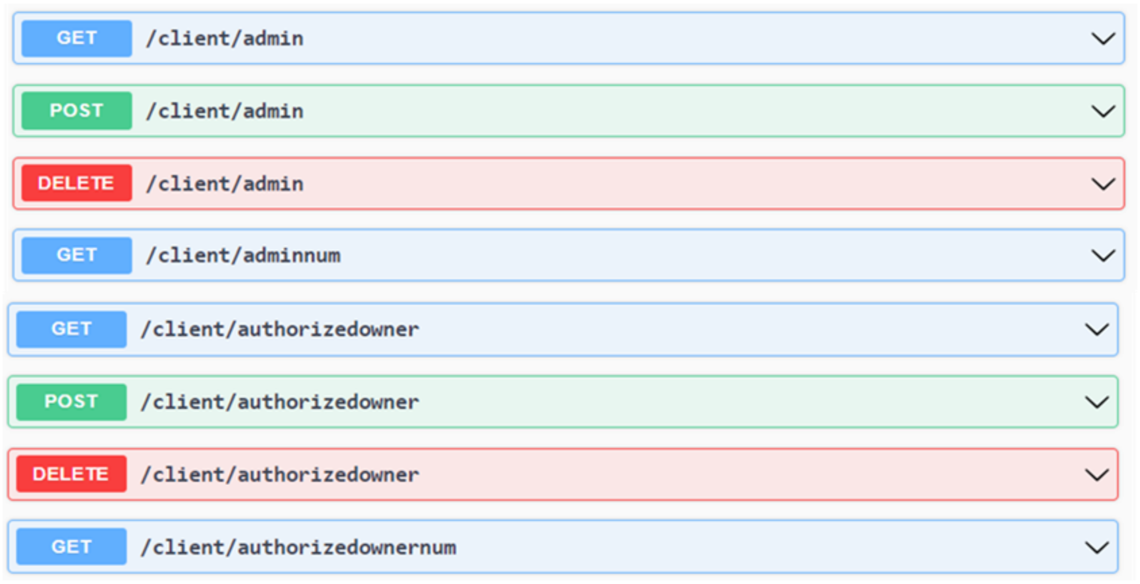
3.2.1.2 Published APIs

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



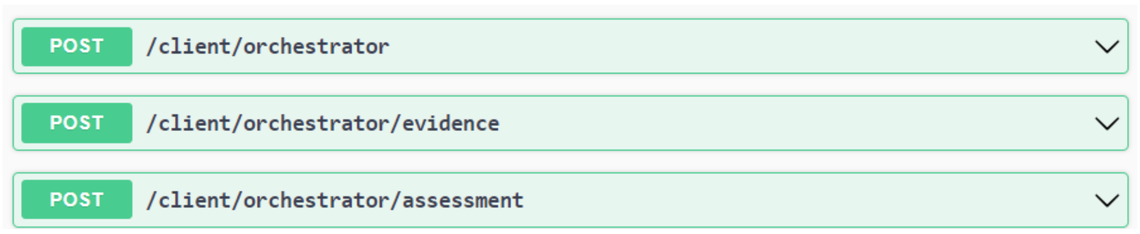
POST /client/account
GET /client/account
POST /client/wallet
GET /client/wallet
POST /client/registration

Listing 2. TWS API Endpoints for Account Management



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

Listing 3. TWS API Endpoints for Users Management

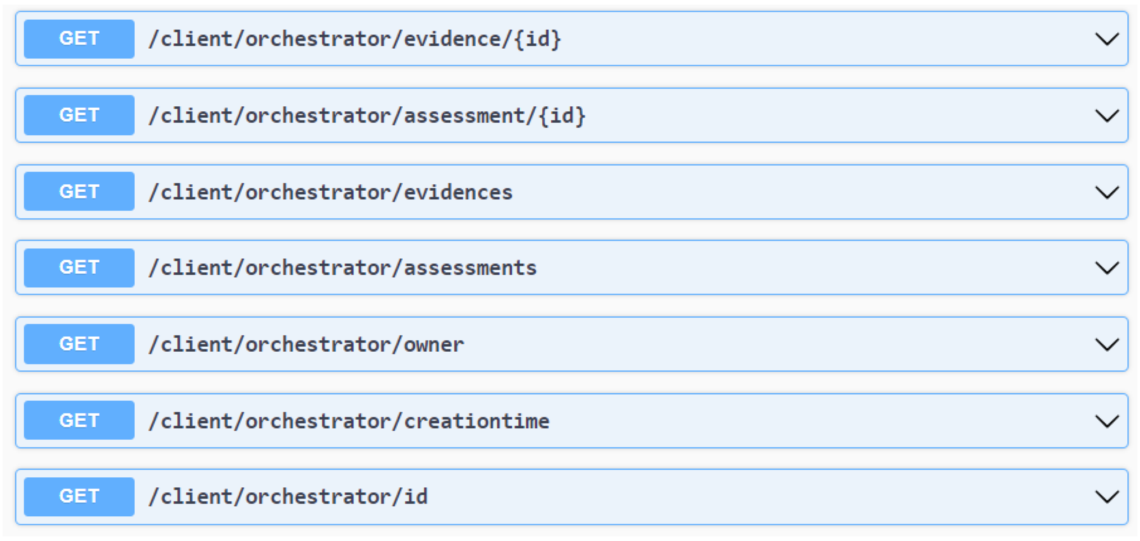


POST /client/orchestrator

POST /client/orchestrator/evidence

POST /client/orchestrator/assessment

Listing 4. TWS API Endpoints for Information Registration



GET /client/orchestrator/evidence/{id}

GET /client/orchestrator/assessment/{id}

GET /client/orchestrator/evidences

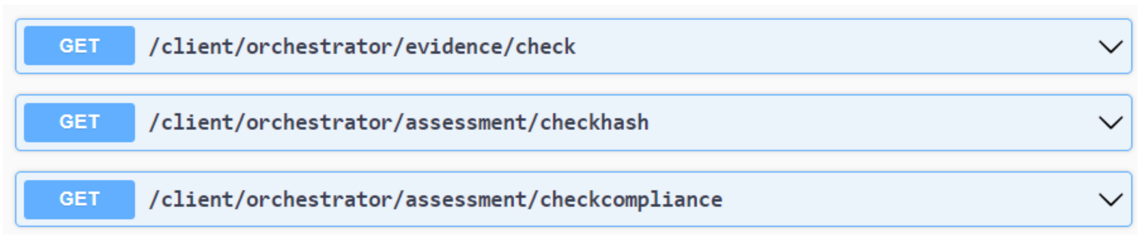
GET /client/orchestrator/assessments

GET /client/orchestrator/owner

GET /client/orchestrator/creationtime

GET /client/orchestrator/id

Listing 5. TWS API Endpoints for Information Access



GET /client/orchestrator/evidence/check

GET /client/orchestrator/assessment/checkhash

GET /client/orchestrator/assessment/checkcompliance

Listing 6. TWS API Endpoints for Integrity Verification

3.2.1.3 Integration Status

Currently, the *TWS* has been successfully deployed on the Kubernetes cluster. However, it has been recently migrated from *Quorum* to the *Alastria* Blockchain network. This migration has slightly delayed the integration process with other EMERALD components. Table 9 provides the current state of the connections of the *TWS* with other components.

Table 9. Integration status of TWS with other EMERALD components

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

3.2.2 MARI

The *Mapping Assistant for Regulations with Intelligence* (*MARI*) is an intelligent system for compliance management. As described in D3.3 [9], 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.

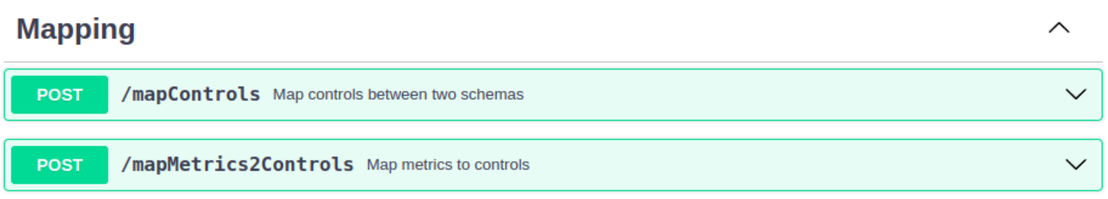
3.2.2.1 Expected behaviour (inputs/outputs)

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.


Mapping

POST /mapControls Map controls between two schemas

POST /mapMetrics2Controls Map metrics to controls

Listing 7. MARI API Endpoints for mapping

3.2.2.3 Integration Status

The *MARI* has already been deployed on the Kubernetes cluster. *MARI* interacts exclusively with the *RCM* through a predefined API. Table 10 provides the current state of connections of the *MARI* with the *RCM*.

Table 10. Integration status of MARI with other EMERALD components

Component	Status	Comment
RCM	Developing API	New API defined

3.2.3 RCM

The *Repository of Control and Metrics* (*RCM*) is a smart catalogue of controls and metrics. As described in D3.3 [9], 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 [13] 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 [14] compliance.

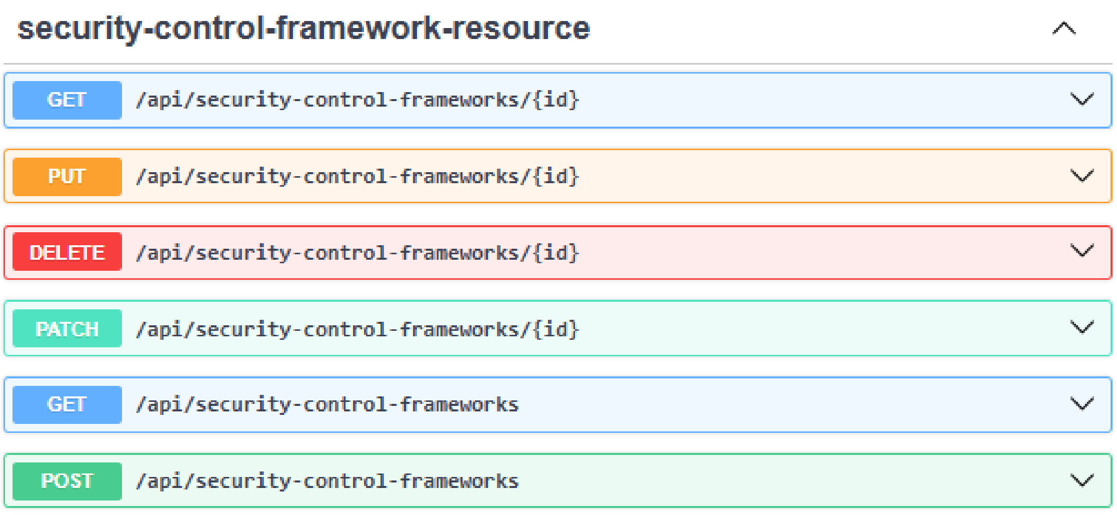
3.2.3.1 Expected behaviour (inputs/outputs)

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.
- ***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 fill 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*.

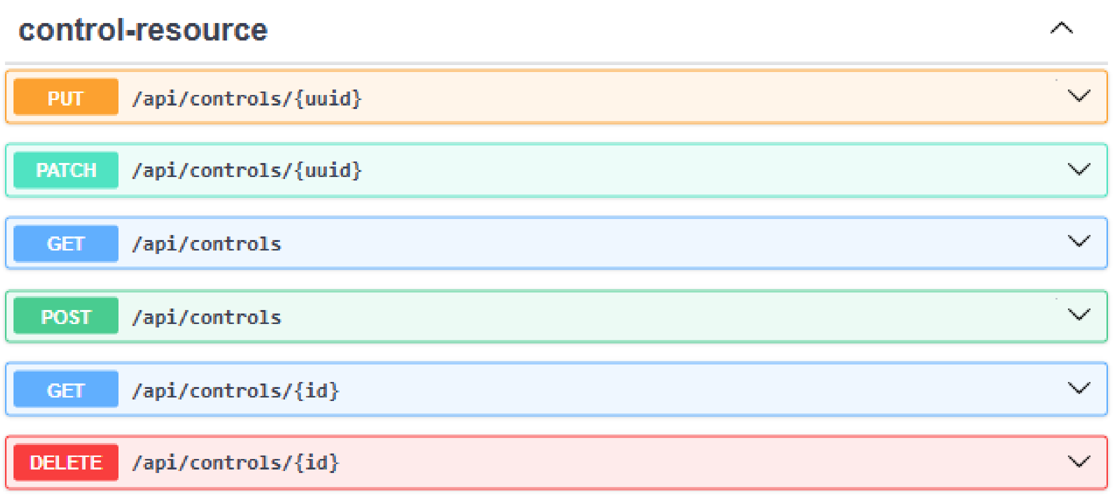
3.2.3.2 Published APIs

The API endpoints of the *RCM* component are listed below.


security-control-framework-resource

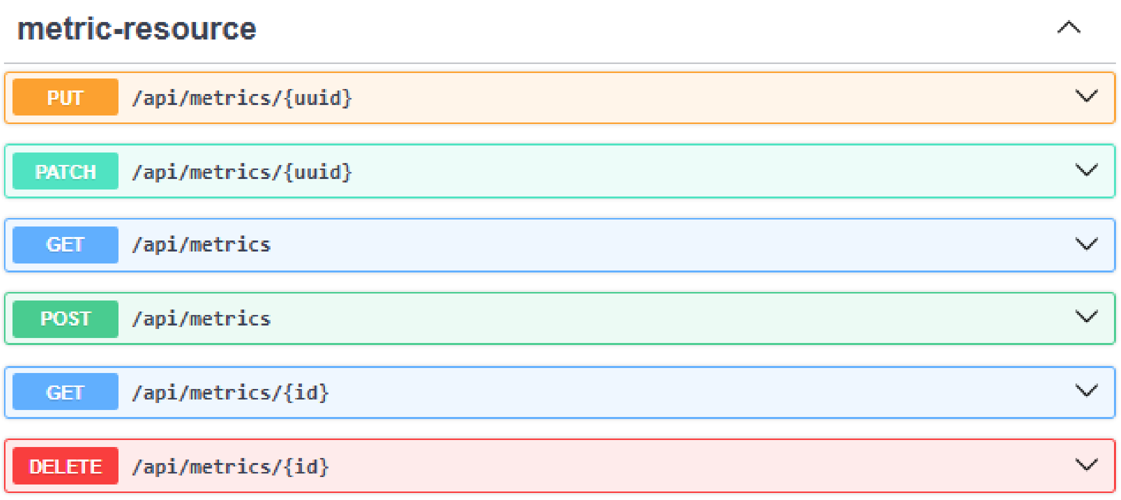
GET	/api/security-control-frameworks/{id}
PUT	/api/security-control-frameworks/{id}
DELETE	/api/security-control-frameworks/{id}
PATCH	/api/security-control-frameworks/{id}
GET	/api/security-control-frameworks
POST	/api/security-control-frameworks

Listing 8. RCM API Endpoints for schema information


control-resource

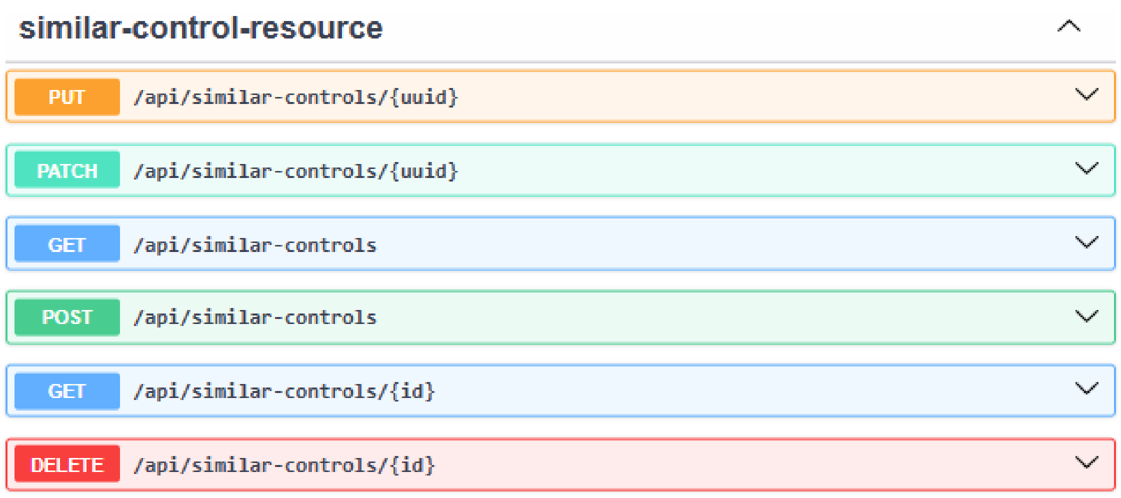
PUT /api/controls/{uuid}
PATCH /api/controls/{uuid}
GET /api/controls
POST /api/controls
GET /api/controls/{id}
DELETE /api/controls/{id}

Listing 9. RCM API Endpoints for control information


metric-resource

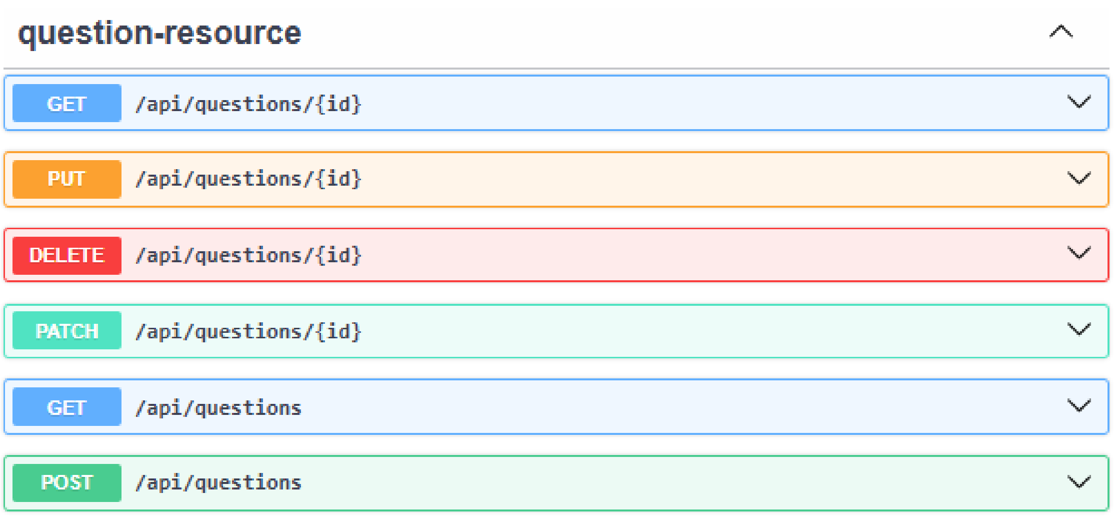
PUT /api/metrics/{uuid}
PATCH /api/metrics/{uuid}
GET /api/metrics
POST /api/metrics
GET /api/metrics/{id}
DELETE /api/metrics/{id}

Listing 10. RCM API Endpoints for Metric information


similar-control-resource

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

Listing 11. RCM API Endpoints for similar control resource



```
question-resource
GET     /api/questions/{id}
PUT     /api/questions/{id}
DELETE  /api/questions/{id}
PATCH   /api/questions/{id}
GET     /api/questions
POST    /api/questions
```

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 11 provides the current state of connections of the *RCM* with other components.

Table 11. Integration Status of the RCM with other EMERALD components

Component	Status	Comment
EMERALD UI	Developing API	Updating / extending the API
Orchestrator	Developing API	-
MARI	Developing API	New API already defined
AMOE	Connected	Changes needed

3.2.4 Orchestrator

The *Orchestrator* is the central orchestration point in the EMERALD framework. As described in D3.3 [9], 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.

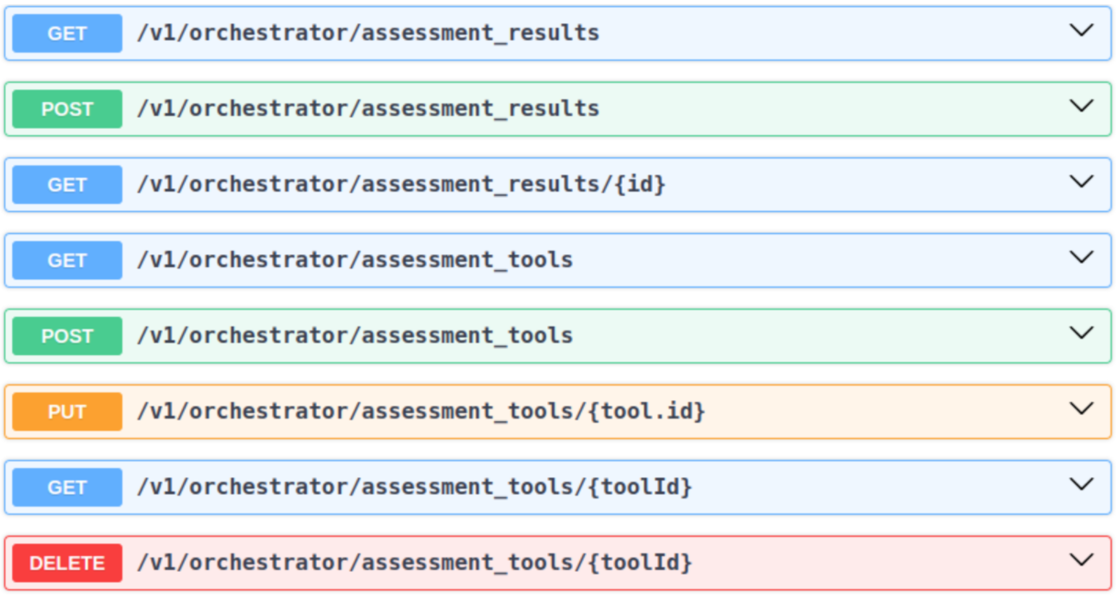
3.2.4.1 Expected behaviour (inputs/outputs)

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

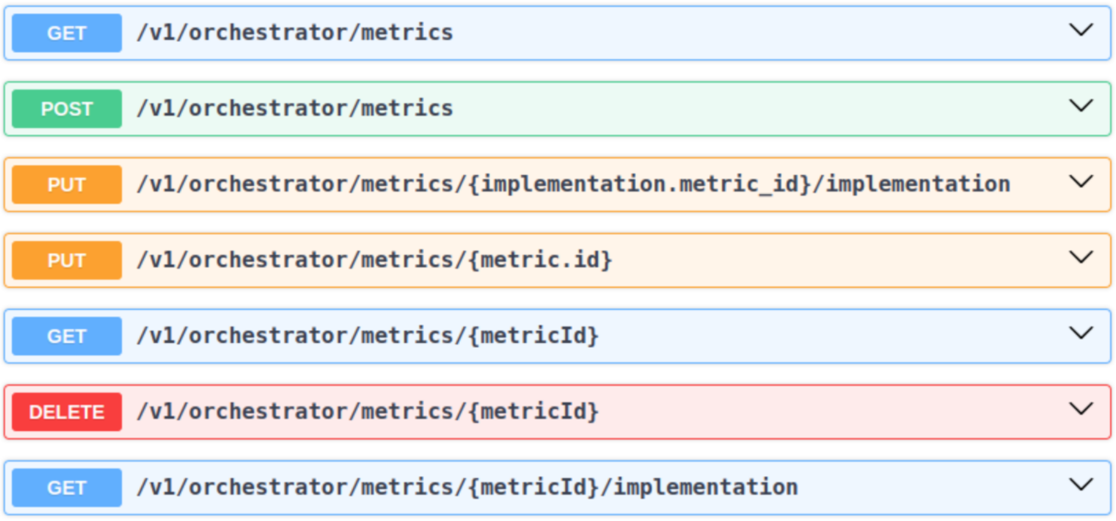
3.2.4.2 Published APIs

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



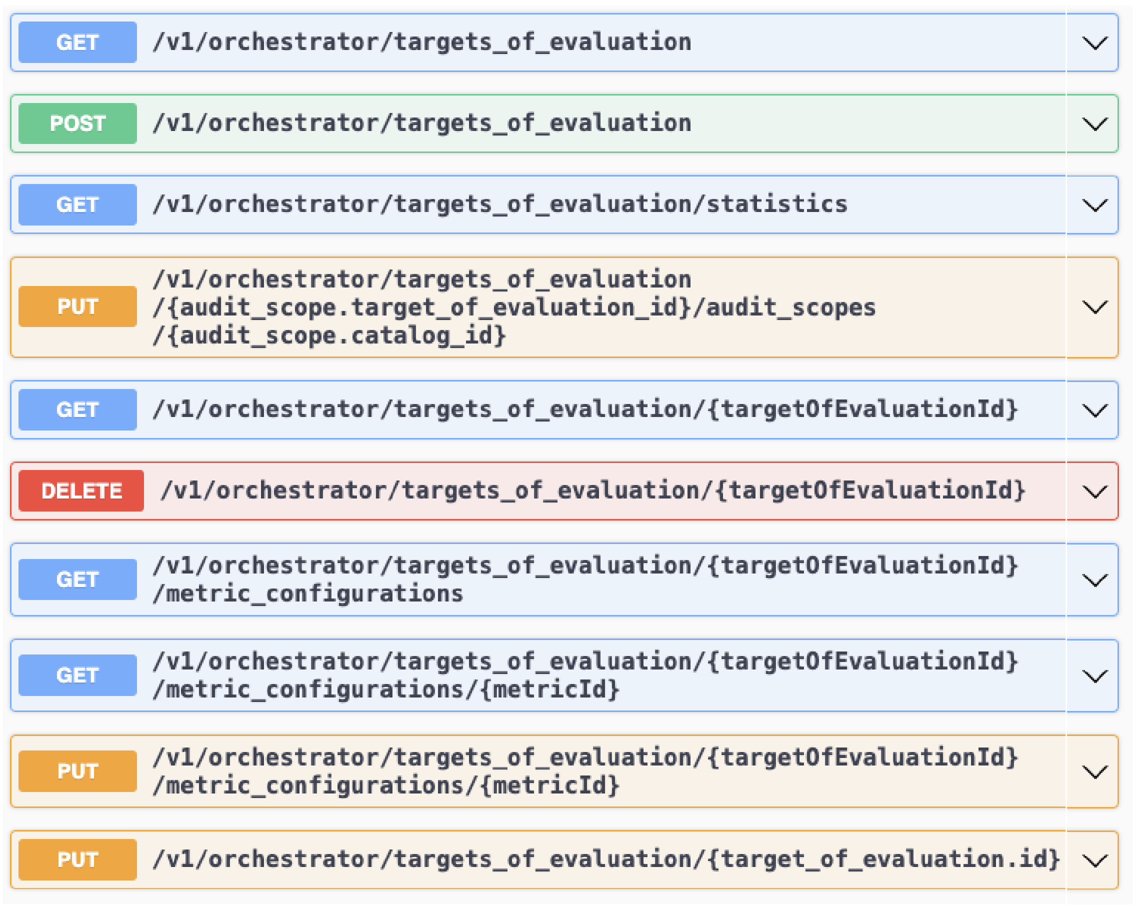
```
GET     /v1/orchestrator/assessment_results
POST    /v1/orchestrator/assessment_results
GET     /v1/orchestrator/assessment_results/{id}
GET     /v1/orchestrator/assessment_tools
POST    /v1/orchestrator/assessment_tools
PUT     /v1/orchestrator/assessment_tools/{tool.id}
GET     /v1/orchestrator/assessment_tools/{toolId}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

Listing 13. Orchestrator API endpoints for assessment results



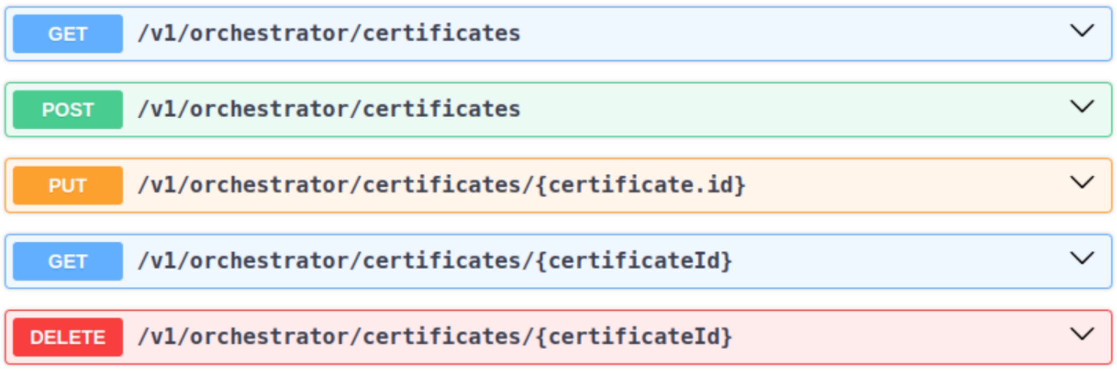
```
GET     /v1/orchestrator/metrics
POST    /v1/orchestrator/metrics
PUT     /v1/orchestrator/metrics/{implementation.metric_id}/implementation
PUT     /v1/orchestrator/metrics/{metric.id}
GET     /v1/orchestrator/metrics/{metricId}
DELETE  /v1/orchestrator/metrics/{metricId}
GET     /v1/orchestrator/metrics/{metricId}/implementation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

Listing 14. Orchestrator API endpoints for metrics



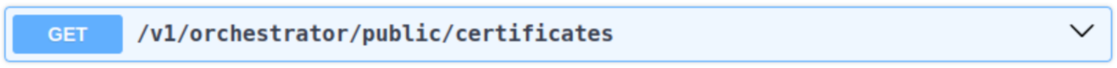
```
GET     /v1/orchestrator/targets_of_evaluation
POST    /v1/orchestrator/targets_of_evaluation
GET     /v1/orchestrator/targets_of_evaluation/statistics
PUT     /v1/orchestrator/targets_of_evaluation
        /{audit_scope.target_of_evaluation_id}/audit_scopes
        /{audit_scope.catalog_id}
GET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

Listing 15. Orchestrator API endpoints for targets of evaluation



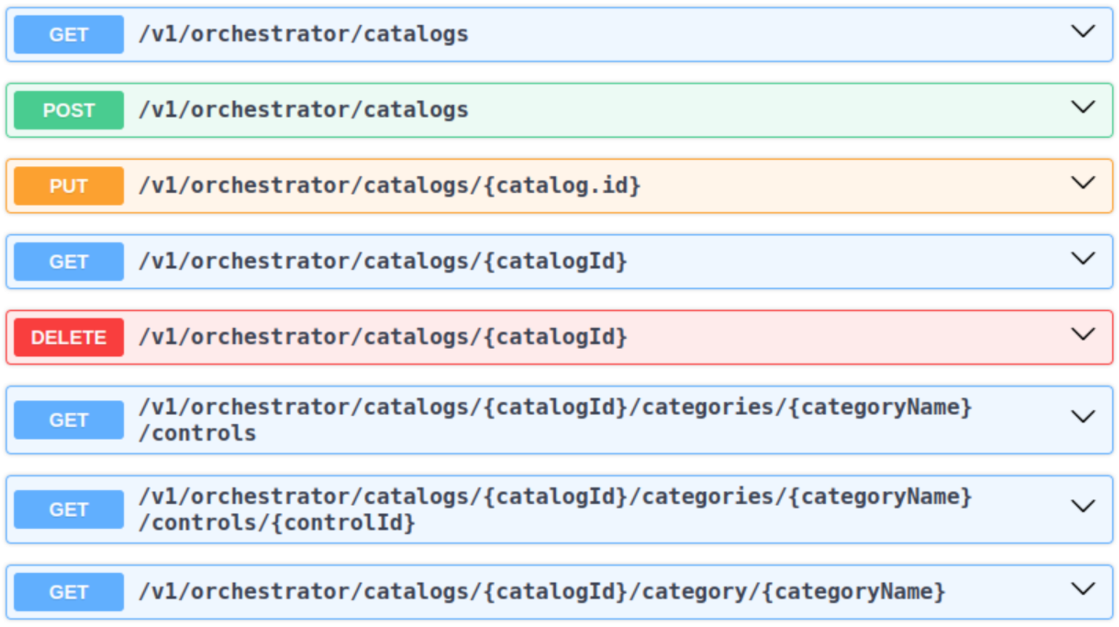
```
GET     /v1/orchestrator/certificates
POST    /v1/orchestrator/certificates
PUT     /v1/orchestrator/certificates/{certificate.id}
GET     /v1/orchestrator/certificates/{certificateId}
DELETE  /v1/orchestrator/certificates/{certificateId}
```

Listing 16. Orchestrator API endpoints for handling certificates



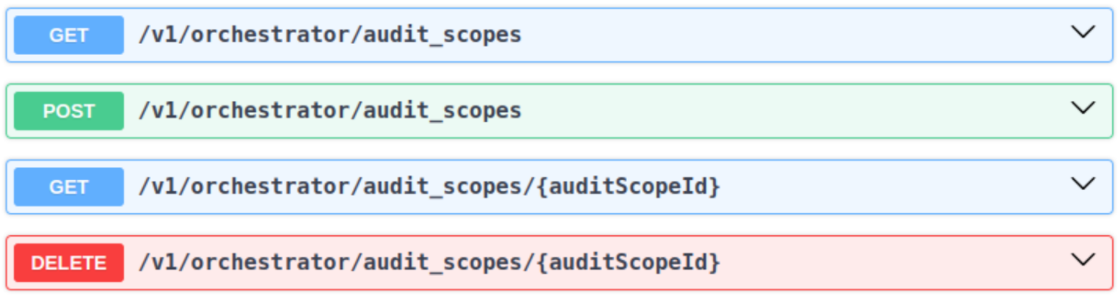
```
GET     /v1/orchestrator/public/certificates
```

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



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

Listing 18. Orchestrator API endpoints for catalogues



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

Listing 19. Orchestrator API endpoints for audit scopes

3.2.4.3 Integration Status

The *Orchestrator* has already been deployed on the Kubernetes cluster. Table 12 provides the current state of connections of the *Orchestrator* with other components.

Table 12. Integration status of Orchestrator with other EMERALD components

Component	Status	Comment
EMERALD UI	API finished	Requires coordination with WP4 and testing.
RCM	Developing API	-
Assessment	Connected	-
Evaluation	Connected	-
Evidence Store	Connected	-

3.2.5 Evidence Store

The *Evidence Store* serves as a central repository for evidence collected from various evidence collectors. It retrieves evidence from the evidence collectors, 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 D3.3 [9] .

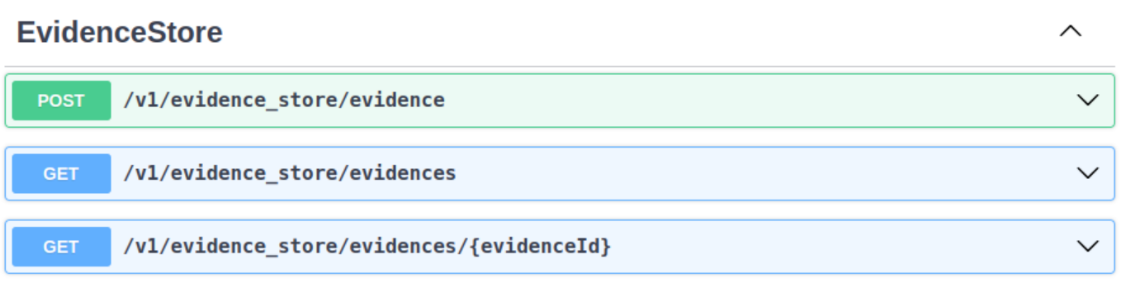
3.2.5.1 Expected behaviour (inputs/outputs)

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.

3.2.5.2 Published APIs

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


EvidenceStore

POST /v1/evidence_store/evidence

GET /v1/evidence_store/evidences

GET /v1/evidence_store/evidences/{evidenceId}

Listing 20. Evidence Store API endpoints

3.2.5.3 Integration Status

The *Evidence Store* has already been deployed on the Kubernetes cluster. Table 13 provides the status of the individual connections. The Postgres database is likely to be replaced by a graph database in the future.

Table 13. Integration status of Evidence Store with other EMERALD components

Component	Status	Comment
Assessment	Connected	-
Orchestrator	Connected	-
AMOE	Testing	-
Codyze	Tested locally	Simple pieces of evidence are received.
eknows-e3	Testing	-
AI-SEC	Testing	-
Clouditor-Discovery	Connected	-

3.2.6 Assessment

As described in D3.3 [9], the *Assessment* is tasked with assessing evidence according to specific metrics established within the EMERALD framework.

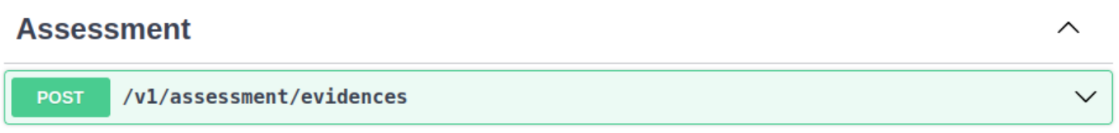
3.2.6.1 Expected behaviour (inputs/outputs)

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

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

3.2.6.2 Published APIs

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


Assessment

POST /v1/assessment/evidences

Listing 21. Assessment API endpoint for evidence

3.2.6.3 Integration Status

Table 14 provides the current status of connections of the *Assessment* with other components.

Table 14. Integration status of Assessment with other EMERALD components

Component	Status	Comment
Evidence Store	Connected	-
Orchestrator	Connected	-
TWS	Developing API (*TWS*)	To be tested

3.2.7 Evaluation

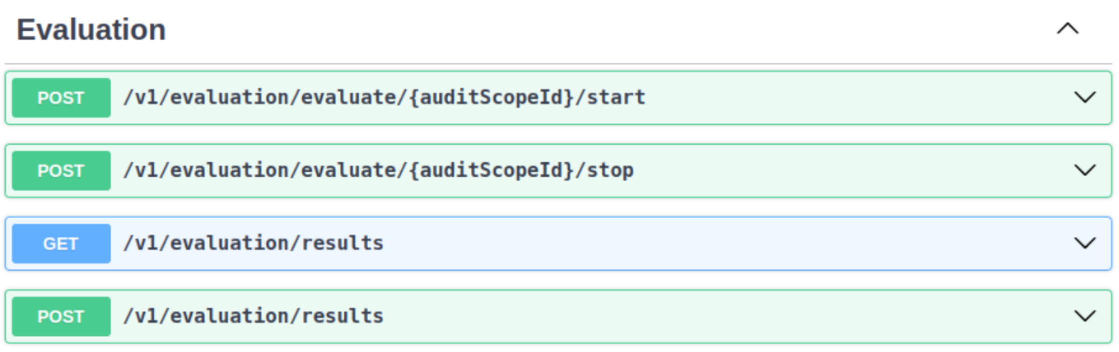
As described in D3.3 [9], the *Evaluation* component is responsible for analysing one or more assessment results to demonstrate compliance with particular controls outlined in a security catalogue.

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.


Evaluation

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

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

GET /v1/evaluation/results

POST /v1/evaluation/results

Listing 22. Evaluation API endpoints

3.2.7.3 Integration Status

Table 14 provides the current status of the connections of the *Evaluation* with other EMERALD components.

Table 15. Integration Status of Evaluation with other EMERALD components

Component	Status	Comment
Orchestrator	Connected	-

3.3 EMERALD UI

The *EMERALD UI* –highlighted in the Figure 14 below– enables users to perform the different user tasks needed to interact with the EMERALD framework. It provides an overview of the data managed by the EMERALD components and combines the information into different workflows to improve the user experience.

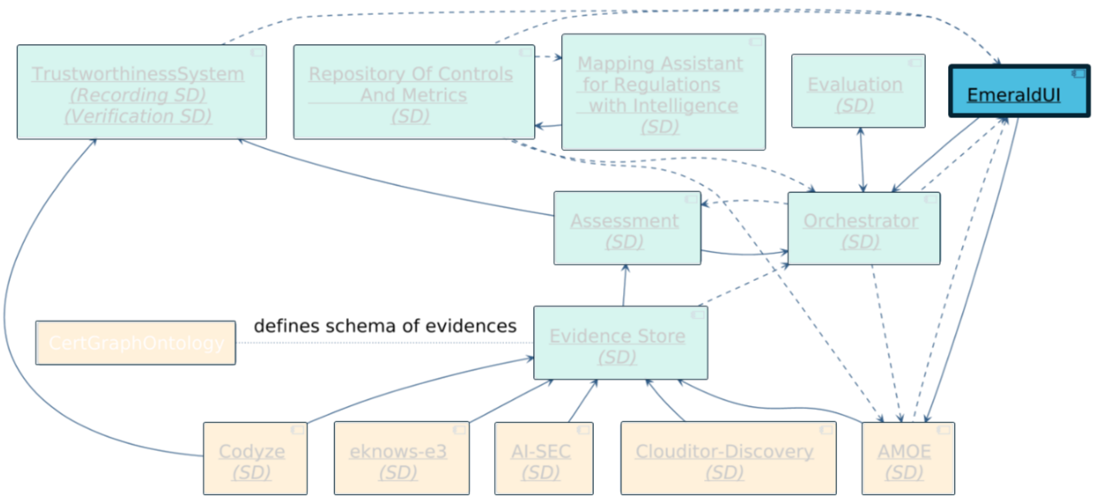

TrustworthinessSystem (Recording SD) (Verification SD)
Repository Of Controls And Metrics (SD)
Mapping Assistant for Regulations with Intelligence (SD)
Evaluation (SD)
EmeraldUI
Assessment (SD)
Orchestrator (SD)
CertGraphOntology
defines schema of evidences
Evidence Store (SD)
Codyze (SD)
eknows-e3 (SD)
AI-SEC (SD)
Clouditor-Discovery (SD)
AMOE (SD)

Figure 14. EMERALD UI in the architecture

3.3.1.1 Expected behaviour (inputs/outputs)

The *EMERALD UI* communicates with *AMOE*, 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[27]. The mock-ups were prioritized according to their occurrence in the User Journeys (both are listed in D4.3 [15]) and their presumed readiness for implementation.

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

The user interface has been presented in the deliverable D4.5 [10] (M15) in detail, please refer there for a more detailed description[28].

3.3.1.2 Published APIs

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

3.3.1.3 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* and *RCM* can be retrieved, but further testing and implementation is required on all sides. The point-to-point integration status is shown in Table 16.

Table 16. Integration status of EMERALD UI with other EMERALD components

Component	**Status**	**Comment**
AMOE	Connected	Testing remains to be done. Only a subset of the endpoints has been fully integrated yet.
Orchestrator	Developing API	Ready to implement, integrate and test parts of the existing *Orchestrator* endpoints, while waiting for updates from the *Orchestrator* API.
RCM	Tested locally	Waiting for updates to the API.
TWS	Not started	Discussion for endpoints and integration needed.

[28] D4.5 is not a public deliverable, so it is only available to project partners.

4 Conclusions

The deliverable D1.7 presents the initial integrated solution of the 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 components such as *AI-SEC*, *AMOE*, *Clouditor-Discovery*, *Codyze*, *eknows-e3*, *TWS*, *MARI*, *RCM*, *Orchestrator*, *Evidence Store*, *Assessment*, *Evaluation*, and the *EMERALD UI* has been achieved in varying degrees. This initial integration ensures seamless communication and collaboration among the components in the future, which is a must for the overall functionality of the EMERALD framework.

The underlaying construction pieces of the EMERALD framework, supported by *Kubernetes* and *Docker* technologies, has proven to be robust and scalable, while the use of the continuous integration and continuous deployment (CI/CD) practices defined will facilitate the efficient deployment and testing of components.

The initial prototype of the EMERALD framework provides a solid foundation for further development and refinement and is the basis for the deployment of the framework in the pilots as a representation of real-world scenarios.

The integration process will continue in next releases, with additional features being integrated into the framework. This will ensure that the EMERALD framework remains comprehensive and up-to-date. User feedback, coming from the validation work package (WP5), and rigorous testing will be essential in identifying –apart from functional features for the components– areas for improving the reliability and effectiveness of the framework. Future versions of the deliverable will reflect the results of these tasks, updating the status of the integration of the different components.

5 References

[1] EMERALD consortium, "D1.2 Data modelling and interaction mechanisms - v2," 2025.

[2] EMERALD Consortium, "D4.2 Results of the UI-UX requirements analysis and the worl processes – v2," 2025.

[3] EMERALD Consortium, "D1.5 DevOps Methodology and CICD strategy - v1," 2024.

[4] EMERALD Consortium, "D1.6 DevOps methodology and CI/CD strategy for EMERALD-v2,' 2025.

[5] EMERALD Consortium, "D2.6 ML model certification – v1," 2024.

[6] EMERALD Consortium, "D2.4 AMOE – v1," 2024.

[7] EMERALD Consortium, "D2.8 Runtime evidence extractor - v1," 2024.

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

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

[10] EMERALD Consortium, "D4.5 EMERALD UI v1," 2025.

[11] BSI - Bundesamtes für Sicherheit in der Informationstechnik, "Secure, robust anc 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 2025].

[12] EMERALD Consortium, "D2.10 Certification Graph-v1," 2025.

[13] NIST - National Institute of Standards and Technology, «OSCAL: the Open Security Control: Assessment Language,» [En línea]. Available: https://pages.nist.gov/OSCAL. [Último acceso April 2025].

[14] ENISA, "EUCS - Cloud Services Scheme," [Online]. Available https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme. [Accessed Apri 2025].

[15] EMERALD Consortium, "D4.3 User interaction and user experience concept - v1," 2024.

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