Deliverable D3.2

Evidence assessment and Certification - Concepts - v2

Editor(s):	Nico Haas (FHG)
Responsible Partner:	Fraunhofer AISEC (FHG)
Status-Version:	Final – v1.0
Date:	30.04.2025
Type:	R
Distribution level (SEN, PU):	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	D3.2 Evidence assessment and Certification–Concepts-v2
Due Date of Delivery to the EC	30.04.2025

Workpackage responsible for the Deliverable:	WP3 – Evidence assessment and Certification
Editor(s):	Nico Haas (FHG)
Contributor(s):	Angelika Schneider (FHG) Marinella Petrocchi (CNR) Cristina Regueiro, Iñaki Etxaniz (TECNALIA)
Reviewer(s):	Verena Geist (SCCH) Cristina Martínez. Juncal Alonso (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP3

Abstract:	Final version of the report on the requirements, design, and integration of WP3 components
Keyword List:	Concept, requirement, design, integration
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	31.03.2025	Update structure and TOC	Nico Haas (FHG)
v0.2	16.04.2025	First draft version	Nico Haas (FHG) Cristina Regueiro, Inaki Etxaniz (TECNALIA) Marinella Petrocchi, Shuya Dong (CNR)
v0.3	22.04.2025	QA Review	Verena Geist (SCCH)
v0.4	27.04.2025	Comments from QA review addressed and sent to final review.	Nico Haas (FHG)
v0.5	29.04.2025	Final review	Cristina Martinez, Juncal Alonso (TECNALIA)
v1.0	30.04.2025	Final version ready to be sent to the EC	Cristina Martinez, Juncal Alonso (TECNALIA)

Table of contents

List of figures

Terms and abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CaaS	Compliance-as-a-Service[1]
CSV	Comma-Separated Values
DL	Deep Learning
DoA	Description of the Action
EBSI	European Blockchain Services Infrastructure
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
gRPC	Google Remote Procedure Call
JPA	Java Persistence API
KPI	Key Performance Indicator
KR	Key Result
MARI	Mapping Assistant for Regulations with Intelligence
MVC	Model, View, Controller
NLP	Natural Language Processing
OSCAL	Open Security Controls Assessment Language
OSS	Open-Source Software
Protobuf	Protocol Buffers
RBAC	Role-Based Access Control
RCM	Repository of Controls and Metrics
REST	Representational State Transfer
SDLC	Software Development Life Cycle
SSI	Self-Sovereign Identity System
TRL	Technology Readiness Level
TWS	Trustworthiness System
UI	User Interface

[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

This deliverable, the final version of evidence assessment and certification concepts, provides the final report on the requirements, design, and integration of the WP3 components within the EMERALD framework. The goal of WP3 is to serve as the central integration point for evidence collection and knowledge extraction tools, contributing to the development of a Compliance-as-a-Service (CaaS)[2] framework for continuous certification of harmonized cybersecurity schemes by assessing the provided evidence to make appropriate compliance attestation. In particular, WP3 and its deliverables address the key results CERTGRAPH (KR2) by implementing the evidence store as a storage that allows graph database queries, OPTIMA (KR3) by providing the optimal set of metrics for a given control of a security scheme, MULTICERT (KR4) by providing certification decision for multiple schemes, and INTEROP (KR7) by providing an interoperability layer for trustworthy systems, assessment results, and catalogue data. These key results are measured using the key performance indicators (KPIs) defined in the DoA [1], which are outlined below.

D3.1 [2] and this deliverable, D3.2, are integral to the EMERALD project, aligning with its overarching objective of enabling multi-scheme auditing of cloud services comprising AI systems. While D3.1 laid the foundation for the concepts, this deliverable provides information on the progress and changes made during the course of the project. These deliverables inform about the development of WP3 components, including the *Clouditor-Orchestrator*, *Clouditor-Assessment*, *Clouditor-Evidence Store*, *Clouditor-Evaluation*, *Mapping Assistant for Regulations with Intelligence* (*MARI*), *Repository of Controls and Metrics* (*RCM*), and *Trustworthiness System* (*TWS*).

At the beginning, the purpose of the deliverable, its context, and the document structure are shown. We then aim to demonstrate a clear understanding of the components being discussed and to illustrate how they are placed in the project. For this purpose, we first provide a short description of each component. Secondly, a high-level WP3 architecture as well as the integration in the EMERALD framework is shown. Finally, the main part of this document delves into each component's requirements, design, integration, planned implementation, and advancements within EMERALD.

Providing certificate decisions by meeting the ambitious objectives set in EMERALD requires various tools to work cohesively together: assessing evidence coming from the WP2 evidence collection tools (KPI 4.1); storing evidence in a database allowing database queries to enable sophisticated assessment of evidence distributed across various layers of a target of evaluation (KPI 2.1); the *RCM* component to store catalogues and metrics in an interoperable way (KPIs 7.1 and 7.2), the *MARI* component to provide metrics that are suitable for a given (set of) security schemes (KPIs 3.1 and 3.2), and the *TWS* component to improve the auditor's trust in the evidence (KPIs 7.1 and 7.2). To implement these components in a manner that ensures cohesive operation, they must be carefully designed. The main contributions of this deliverable to the project are therefore to demonstrate the purpose and roles of each WP3 component in the project, the definition of the requirements, and the proposed design to ensure seamless implementation and integration in the whole framework.

The structure of the WP3 deliverables closely resembles the software development life cycle (SDLC) approach: After publishing the first versions of concepts, implementation and integration, this deliverable describes the final concepts (requirements and design). The next

[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.

steps include the final version of the actual implementation (D3.4 “Evidence assessment and Certification–Implementation-v2” M24) as well as the final integration (D3.6 “Evidence assessment and Certification–Integration-v2” M27). Following the SDLC approach, we ensure continuous improvement and refinement of the components (also considering changes occurring in other work packages).

1 Introduction

This document is the successor of D3.1 [2], which presented the first version of the WP3 concepts that was applied as a baseline during the first stage of the project. D3.2 follows the same structure as well as keeps a part of content from the previous deliverable version to keep the document self-contained and easier to follow. Section 1.3 presents the modifications of this document compared to its first version.

1.1 About this deliverable

The EMERALD project aims to pave the way towards Compliance-as-a-Service[2] (CaaS) for continuous certification of harmonized cybersecurity schemes, such as the EUCS [3]. It addresses the critical need for enhanced transparency, accountability, and trustworthiness in European cloud services. The project focuses on developing robust evidence management components and providing a proof of concept for AI certification schemes.

Within this context, WP3 plays a pivotal role by serving as the central integration point for evidence collection and knowledge extraction tools developed in WP2, while also acting as the interface for auditors and pilots utilizing the WP3 components via the EMERALD UI. The main goal of WP3 is to contribute to the CaaS framework by assessing the provided evidence to make appropriate attestation decisions.

This deliverable serves as the final report on the requirements, design, and integration of the WP3 components within the EMERALD project. The main goal is to lay the foundational framework for understanding and developing the WP3 components. This deliverable is also crucial for providing a clear understanding of the role and interaction of each WP3 component and how they collectively contribute to the project's goals.

In summary, this document aims to provide a thorough understanding of the final concepts, requirements, and design of the WP3 components, setting the stage for their seamless implementation and integration within the EMERALD framework.

1.2 Document structure

This document is structured to provide a comprehensive overview of the WP3 components and their roles within the EMERALD project. The rest of this document is structured as follows:

Section 2 presents an overview of the WP3 architecture. It offers a concise look at the various components of WP3, their high-level architecture, and their integration within the EMERALD framework. This section aims to give readers a clear understanding of how the different components work together to achieve the project's objectives.

Section 3 contains the main contribution of the document: it delves into each component's requirements, design, integration, planned implementation, and advancements within the EMERALD project.

Finally, Section 4 reports the conclusions.

1.3 Updates from D3.1

This deliverable evolves from D3.1 [2], and with the ultimate goal of making the document self-contained and easier to follow, part of the content comes from D3.1 since it has not changed,

and other parts are new. To simplify tracking progress and updates from the previous version, Table 1 shows a brief summary of the changes and additions to each section of the document.

Table 1. Overview of deliverable updates with respect to D3.1

Section	Changes
Overall document	The consortium agreed on changing the term of the former "cloud service" and "certification target" to "target of evaluation". These changes have been made throughout the entire document.
2. WP3 Architecture	The following updates have been made in this chapter: • Updated Figure 1. Overview of the EMERALD components • Added content that reflects the tasks of *Orchestrator* to limit access to specific users and allow the delegation of tasks.
3.1 Clouditor-Orchestrator	Requirements: • ORCH.02: Added endpoints for handling users, adapted description, updated progress and adapt milestone • ORCH.03: Updated progress • ORCH.04: Adapted milestone • ORCH.05: Changed status und updated progress Design: • Updated the RBAC key element • Added the "Delegation of tasks" key element
3.2 Clouditor-Assessment	Requirements: • ASSESS.01: Adapted description to incorporate the fact that we plan to use graph query language for assessing evidence but not using a full graph database (evidence format). Also increased progress to a small degree because we already tested the assessment using REGO in the cluster.
3.3. Clouditor-Evidence Store	The main changes include the transition from the planned native graph database to a hybrid approach using graph database queries in a traditional relational database. Requirements: • ESTORE.01: Changed description, increased progress and adapted milestone • ESTORE.02: Increased progress because we deployed and tested different evidence sources Design, Integration, Planned Implementation (Functionalities), Advancements: • Adapted and added new content to make clear why we follow the new approach.

Section	Changes
3.4 Mapping Assistant for Regulation with Intelligence (MARI)	Updated the design part, the integration part (including the change to Figure 1 in Section 2.2), and the planned implementation part about *MARI*.
3.5 Clouditor-Evaluation	Nothing to report since the actual further implementation (enhancement) is just starting after all components have been successfully deployed. With respect to the concepts, there won't be major changes in this component.
3.6 Repository of Controls and Metrics (RCM)	Added two new requirements that deal with the EUCS questionnaire and control-metrics mapping. For the rest of requirements, we have mainly updated the status and adapted some description. Advancement within EMERALD section has been updated and refined.
3.7 Trustworthiness System	A new requirement has been added because the *TWS* will be also considered from the evidence sources. The status of the other already existing requirements has been updated according to the current situation. The design and integration sections have been updated to cover the new requirement as well as to make some corrections due to internal improvements. The Blockchain network information has been updated to Alastria. Finally, the planned implementation and the advancement sections have been refined.
4. Conclusions	Future steps have been updated.

2 WP3 Architecture Overview

This section offers a foundational overview of the WP3 components within the EMERALD project; setting the context for the detailed analysis of each component's requirements, design, integration, planned implementation, and advancements within EMERALD, which will be covered in Section 3. The objective is to provide an initial understanding of the individual components, their interactions, and their integration within the broader EMERALD framework. Rather than an in-depth exploration, this chapter aims to give a concise and clear snapshot of each component.

Section 2.1 provides an overall overview of each WP3 component, offering short explanations to clarify their roles. Section 2.2 presents a high-level architecture of these components, illustrating how they interact and function together – also with other components within the EMERALD framework.

2.1 Overall Overview of the Components

WP3 comprises several key components, each playing a crucial role in the evidence assessment and attestation process within the EMERALD project (see Figure 1). Below is a brief overview of each component:

The ***Clouditor-Evidence Store*** functions as a centralized repository for storing evidence collected during the attestation process. It utilizes a graph query language to organize and manage evidence in an efficient and accessible manner. The evidence is sourced from the collector components developed in WP2, ensuring that all relevant data is systematically stored and readily available for assessment.

The ***Clouditor-Assessment*** component is responsible for assessing the collected evidence (stored in the *Clouditor-Evidence Store*) and providing the *Clouditor-Orchestrator* with assessment results. It calculates the assessment results using the metrics provided by the *Repository of Controls and Metrics* (*RCM*). Note that assessment results are independent from specific certification schemes. It is the *Clouditor-Evaluation* component that considers specific certification schemes, see below.

The ***Clouditor-Orchestrator*** is the central component responsible for orchestrating the compliance attestation process. It includes a state machine, providing a snapshot of the target of evaluation's state regarding its attestation. Among others, it also offers an interface for compliance managers to select certification schemes (via the EMERALD UI) and coordinates the assessment tools.

The ***Clouditor-Evaluation*** component is responsible for combining assessment results relevant to a specific control of a certification scheme to create an evaluation result for this control. It uses these assessment results to determine the compliance state of a control, which is either compliant or non-compliant. Which assessment results must be considered is based on the metrics selected by *MARI* (an assessment result has a direct relationship to a metric since one assessment result presents one metric calculated at a specific time).

The ***Repository of Controls and Metrics* (*RCM*)** serves as a smart catalogue of controls and metrics. Controls exist mostly in natural language within various security frameworks and standards like the EUCS. In this project, a *control* refers to a specific countermeasure designed to protect target of evaluations. We follow the definition of OSCAL where a control "*is a requirement or guideline, which when implemented will reduce an aspect of risk related to an*

information system and its information"[3]. Note that the naming of a control can also differ from security standard to security standard, e.g., in the EUCS there are controls and requirements, where a *control* provides a more abstract description and puts multiple requirements together, while a *requirement* gives a concrete definition of a countermeasure. A *metric*, on the other hand, refers to a rule (in fact, a measurable value) used to assess one or more properties of a control. The *RCM* also incorporates other features such as automatic import/export mechanisms to facilitate the reuse and composition of the catalogue elements.

The ***Mapping Assistant for Regulations with Intelligence* (*MARI*)** is an intelligent system designed to select suitable metrics for demonstrating compliance with certification schemes. It leverages advanced AI techniques, including Natural Language Processing (NLP), to analyse security controls and recommend optimal metrics. Therefore, MARI supports the *Clouditor-Orchestrator* to ensure that the selected metrics align with the security catalogues and facilitate accurate assessments.

The **Trustworthiness System (TWS)** enhances the integrity and transparency of the attestation process. It deploys a general-purpose Blockchain network, thereby improving the trustworthiness of the evidence and assessment results. By leveraging blockchain technology, *TWS* ensures that all actions and data within the attestation process are tamper-proof and verifiable, thus boosting the trust of the auditors.

Section 2.2 presents how all these components collectively contribute to achieve WP3 goals.

2.2 Architectural Overview and Integration of WP3 components

Section 3 will present the components of WP3 in detail. We will now consider them within the bigger picture, examining which components communicate with each other and what information they exchange. We also look at the communication to other components within the EMERALD framework. Note that the specific data attributes of the shared information are provided in D1.2 [4].

For the sake of clarity, we will show the typical workflow in EMERALD to demonstrate how the various components work together. Figure 1 shows the current status of all the framework components, as well as their interactions. The goal is to check the compliance of a given target of evaluation with respect to a certification scheme (e.g., EUCS [3]), which is typically divided into several controls (e.g., CKM-02 ENCRYPTION OF DATA IN TRANSIT).

[3] https://pages.nist.gov/OSCAL/learn/concepts/terminology/#control

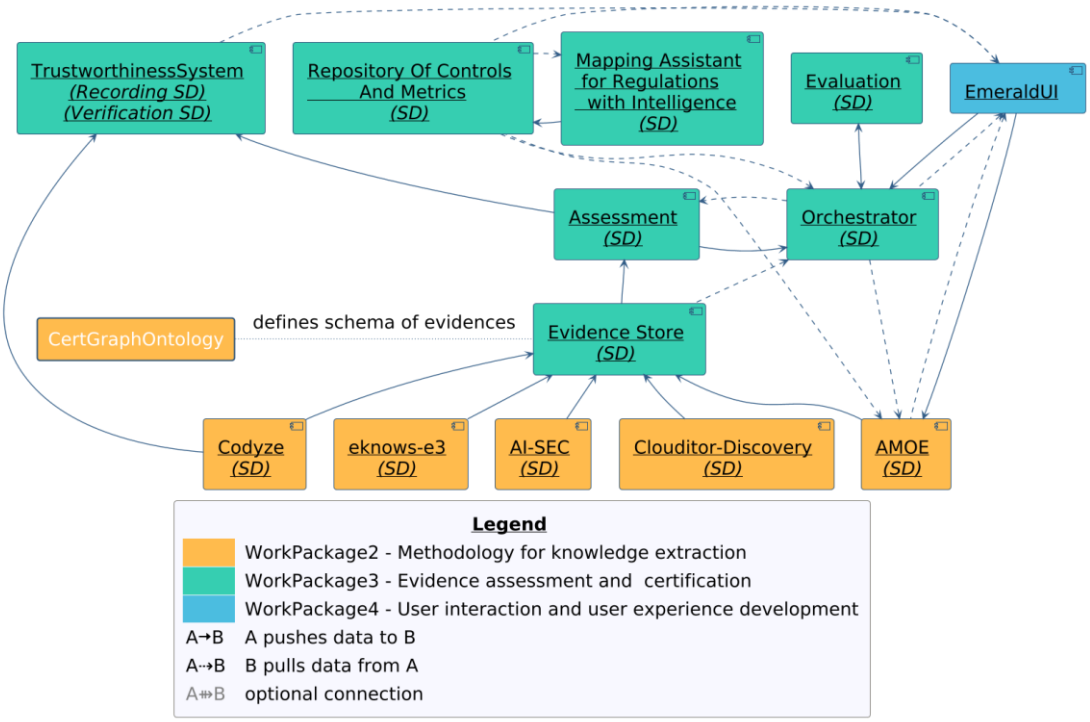

Figure 1. Overview of the EMERALD components [4]

The collector components of WP2 (orange boxes at the bottom in Figure 1) discover information from the targets of evaluation, e.g., the *AI-SEC* component collects information about one or more AI systems deployed in the cloud. This information is transformed according to a uniform format defined in WP2, known as the ontology for the certification graph (*CertGraphOntology*), which is described in D2.1 [5] and D2.10 [6].

Evidence is then sent to the ***Clouditor-Evidence Store***, which stores and forwards the evidence to the *Clouditor-Assessment* component. The ***Clouditor-Assessment*** creates assessment results based on the input evidence and one or more rules (i.e., metrics). The assessment component then sends this information to the ***Clouditor-Orchestrator*** (where it is stored in a database), as well as to the ***TWS***, which stores hashes of both evidence and assessment results to ensure and prove their integrity (details are explained in Section 3.7).

The *Clouditor-Orchestrator* sends the assessment results to the ***Clouditor-Evaluation,*** which decides whether a given target of evaluation is compliant with respect to a control. Therefore, the *Orchestrator* needs to know which assessment results are relevant for which controls. Each assessment result is tied to a metric. Typically, multiple metrics (i.e., multiple assessment results at a specific point in time as well) are aggregated to fulfil one control. To support the decision on which metrics are needed, the ***MARI*** component leverages AI techniques to find the optimal set of metrics for a given control.

All metrics, as well as the various security catalogues, are stored in the ***RCM*** component. Both the *Clouditor-Orchestrator* and *MARI* retrieve the suitable information from there.

Once information on compliance for the individual controls of a security catalogue is available, a decision must be made as to whether the target of evaluation obtains a compliance attestation. This decision is ultimately made in the ***Clouditor-Orchestrator***, which provides a snapshot of the target of evaluation's status.

It is possible to communicate with the WP3 components via API interfaces to retrieve, add, or modify information – depending on the permissions. Different personas, e.g. auditor or compliance manager, can perform the corresponding operations via the UI (see the blue *EMERALD UI* box at the top of Figure 1). In this way and in conjunction with the deployed *Keycloak* instance, the *Orchestrator* limits the access to certain audit scopes (i.e., combination of target of evaluation and certification scheme) to certain users. In addition, the *Orchestrator* allows that a compliance manager can delegate tasks to other users (normally these are other employees), e.g. to being responsible for a specific control and, thus, responsible to set actions that will fulfil this control.

This integrated architecture ensures that all components interact effectively to achieve the project's goals. Each component has a specific role, but they are all interconnected, working together to provide a comprehensive and reliable attestation process within the EMERALD framework. In Section 3, we will take a closer look on how the individual components work.

More detailed information about the architecture is available in the WP1 “Concept and Methodology of EMERALD” deliverables, particularly in D1.3 “EMERALD solution architecture-v1” [7] (M12) and D1.4 “EMERALD solution architecture-v2” (M24).

3 WP3 Components

This section provides an overview of each component of WP3 of the EMERALD project. It describes the requirements, design, integration, planned implementation, and advancements made for each component.

Each requirement is presented along the common EMERALD requirement definition table consisting of the following fields:

- **Requirement id**: Contains the unique identifier for the requirement.
- **Short title**: Contains a short title for the requirement.
- **Description**: Describes the requirement in more detail.
- **Status**: Contains the status of the requirement, consisting of one of the following values: Proposed → Accepted/Discarded → Work in Progress → Implemented (Partial/Full) → Tested → Validated
- **Priority**: Priority values are: Must; Should; Could.
- **Component:** Contains the name of the component the requirement is related to.
- **Source**: Defines where the requirement comes from: Pilot, Component, DoA, or KPI.
- **Type**: Describes the type of the requirement. "Technical" in the case of WP3 requirements.
- **Related KR**: Describes the related key result of the DoA.
- **Related KPI:** Describes the related key performance indicator of the DoA (see below).
- **Validation acceptance criteria:** Describes how to validate the requirement.
- **Progress:** Indicates the extent to which the requirement has already been met.
- **Milestone:** Specifies the milestone the requirement belongs to.

Please note that this section only shows the technical requirements for each component. There are also other types of requirements, namely user interface and pilot requirements, which are detailed in deliverables D4.2 [8] and D5.1 [9], respectively. To bridge the gap between technical and non-technical requirements, pilot requirements and user interface requirements have been linked to the technical requirements, so that in the end all the requirements from the user interface and pilot perspective are satisfied in the components.

Finally, WP3 requirements are related to the following **KPIs** defined in the DoA [1]:

- **KPI 1.1:** Provide support for evidence extraction from different sources (infrastructure, code, processes).
- **KPI 2.1:** Provide a schema for storing and linking heterogeneous evidence information.
- **KPI 2.3:** Provide scalability for storing/processing continuously collected evidence; demonstrated in the pilots.
- **KPI 3.1:** Provide scheme to scheme mapping functionality based on metrics, recommended to the user.
- **KPI 3.2:** Provide metric-to-requirement-mapping functionality by improving MEDINA approaches and incorporating KPI 5.1 results.
- **KPI 3.3:** Provide insights for the mapping decision and how the recommendation process works.
- **KPI4.1:** Provide realizable metrics that demonstrate compliance to at least two security certification schemes.
- **KPI 4.2:** Provide metric assessment for 80 % of the metrics in KPI 4.1 based on the certification graph.
- **KPI 6.2:** Provide concept for the UI of EMERALD and integration of evidence collection components, databases and orchestrating components.

- **KPI 6.3:** Provide a graphical user interface for role-based access to certification information content.
- **KPI 7.1:** Conventionalize import and export functionalities to take or share data with external sources.
- **KPI 7.2:** Incorporate input from standardisation bodies and synchronize data formats and protocols.

3.1 Clouditor-Orchestrator

The *Clouditor-Orchestrator* is the central component orchestrating the attestation process and connecting multiple components together of the EMERALD framework. Finally, this component takes care of the certification decision, i.e., whether a target of evaluation is compliant with a security catalogue or not. The *Clouditor-Orchestrator* component is based on the respective microservice of *Clouditor*[4] and was already used in MEDINA [10]. It will be further developed by leveraging the functionality of the *Life-Cycle Manager* component in MEDINA to provide the final certificate decision.

3.1.1 Requirements

The main technical requirements for the *Clouditor-Orchestrator* are as follows:

Field	Description
Requirement ID	ORCH.01
Short title	Final certificate decision
Description	Since we do not have a dedicated life-cycle manager component in EMERALD, the *Orchestrator* must take care of the final certificate decision. The decision is based on the input of the *Evaluation* component providing the *Orchestrator* with an evaluation result for each control.
Status	Accepted
Priority	Must
Component	*Clouditor-Orchestrator*
Source	KPI
Type	Technical
Related KR	KR4_MULTICERT
Related KPI	KPI 4.1, KPI 4.2
Validation acceptance criteria	If an assessment and evaluation fail, the certificate must go to a suspended state.
Progress	0%
Milestone	MS5: Components V2 (M24)

Field	Description
Requirement ID	ORCH.02
Short title	REST API Gateway for UI
Description	The *Orchestrator* should provide a REST API gateway for the UI that serves a central API endpoint for all information needed from the *Orchestrator*, *Assessment*, *Evaluation* and other components. This includes:

[4] https://github.com/clouditor/clouditor/tree/main/service/orchestrator

	• List of all controls, catalogues, etc. (which are each cached from the *RCM*) • List of all targets of evaluation (+add/edit/remove) • List of all audit scopes (+add/edit/remove) • List of all tools (extractors, assessment) (+register/edit/remove/disable). See [ORCH.04] • List of all assessment results • List of all evidence • List of all evaluation results • List of all certificates (decisions). See [ORCH.01] • List of all audit workflow assignments (+add/edit/delete comment). See [ORCH.05] • Handle users and their permissions (+add/get/update/delete/list)
Status	Work in Progress
Priority	Must
Component	*Clouditor-Orchestrator*
Source	KPI
Type	Technical
Related KR	KR6_EMERALD UI/UX
Related KPI	KPI 6.2
Validation acceptance criteria	The *Orchestrator* has a functioning API endpoint that provides all the required information from the connected components.
Progress	75%
Milestone	MS3: Integrated audit suite v1 (M18)

Field	Description
Requirement ID	ORCH.03
Short title	Role-Based Access Control
Description	Since the UI wants to selectively disclose information to users and/or roles, we need a RBAC mechanism in our API endpoints, mainly in the *Orchestrator*.
Status	Work in Progress
Priority	Must
Component	*Clouditor-Orchestrator*
Source	KPI
Type	Technical
Related KR	KR6_EMERALD UI/UX
Related KPI	KPI 6.3
Validation acceptance criteria	The UI only displays information after a successful login.
Progress	50%
Milestone	MS5: Components V2 (M24)

Field	Description
Requirement ID	ORCH.04
Short title	Manage Tools (such as evidence extractors) via API

Description	We need to manage external tools, such as evidence extractors in the *Orchestrator*. We want to have the following functionality: • Register a new tool (e.g., with a token) for a particular target of evaluation • "Hello" from the tool -> returns the configured target of evaluation ID • List all tools • Get status of tool (whatever the status is). Last evidence sent, etc. • Remove tool -> do not accept forever • Disable or suspend tool -> temporarily do not accept evidence, assessment result, etc. from the tool We need different "types" / categories of tools. This needs to be specified (e.g., either evidence extractor or *Assessment* tool). Note: The list of configured tools is specific for a particular target of evaluation.
Status	Accepted
Priority	Must
Component	*Clouditor-Orchestrator*
Source	KPI
Type	Technical
Related KR	KR6_EMERALD UI/UX
Related KPI	KPI 6.3
Validation acceptance criteria	The UI only displays information about registered tools.
Progress	0%
Milestone	MS5: Components V2 (M24)

Field	Description
Requirement ID	ORCH.05
Short title	Provide an API for audit workflow
Description	We want to assign people to controls within an audit instance that have a particular task. The exact definition of this has to be done. Probably we want to have the possibility to add comments to controls? Maybe also add the manual evidence? Status of the control?
Status	Work in Progress
Priority	Must
Component	*Clouditor-Orchestrator*
Source	KPI
Type	Technical
Related KR	KR6_EMERALD UI/UX
Related KPI	KPI 6.3
Validation acceptance criteria	All the controls of the scheme are displayed with the required information.
Progress	25%
Milestone	MS6: Integrated audit suite V2 (M30)

3.1.2 Design

The *Clouditor-Orchestrator* is based on the respective microservice of *Clouditor*[5], whose modular architectural approach ensures scalability and flexibility. Key design elements of the *Clouditor-Orchestrator* in EMERALD include:

- **Workflow Management**: The *Orchestrator* manages the workflow of the compliance attestation process, coordinating with other components to ensure that evidence is collected, assessed, and evaluated in a timely and efficient manner. It ensures that each step in the attestation process is executed correctly, from the initial collection of evidence to the final attestation. This involves scheduling tasks, handling dependencies between different components, and ensuring that all necessary actions are completed to achieve compliance with the selected security catalogues.
- **Modular Architecture**: The *Orchestrator* component is part of the *Clouditor* tool, which is built using a microservice architecture. This architecture divides the *Clouditor* tool into multiple independent services that can be developed, deployed, and scaled separately. The *Orchestrator* is one of these microservices, responsible for orchestrating the attestation process. Other microservices within *Clouditor* include the *Discovery* and *Assessment* services, among others. This modularity allows for easier maintenance and scalability, as each microservice can be updated or scaled independently.
- **Communication protocols (API endpoints)**: A RESTful API is provided to facilitate interactions with the UI and other components. Additionally, the *Orchestrator*, as well as other *Clouditor* components (e.g., *Discovery* and *Assessment*), are able to communicate via gRPC. This dual API approach ensures flexibility in the communication protocols, allowing for efficient data exchange and integration with various systems. The APIs serve as central endpoints for retrieving and managing information from the *Orchestrator*, *Assessment*, *Evaluation*, and other *Clouditor* components.
- **User Interface**: The *Orchestrator* integrates with the EMERALD UI, allowing compliance managers to interact with the system. The UI provides functionalities for selecting security catalogues, viewing assessment results, and making final attestation.
- **Role-Based Access Control (RBAC)**: To ensure that information is selectively disclosed to users based on their roles, the *Orchestrator* implements an RBAC mechanism. This mechanism controls access to API endpoints and ensures that only authorized users can view or modify specific information. In the beginning, only highly privileged users, e.g. compliance managers, have access to view and modify information of audit scopes. This user then can add other users with limited access to this information. In a further step we plan to enable also to set the access via groups of users.
- **Delegation of Tasks**: A compliance manager wants to delegate tasks, e.g. to make a specific user responsible for fulfilling a specific control that is currently non-compliant. Therefore, specific users like the compliance manager should be able to delegate tasks. We plan to implement this delegation on control level for now, but it could be also broken down to the level of assessment or metrics. But this idea needs to be discussed further.

3.1.3 Integration

Because it plays the central role (see Section 2.2) in orchestrating many components and their data flows, the integration of the *Clouditor-Orchestrator* with other components within the EMERALD framework is crucial for achieving a seamless attestation process. The following components the *Clouditor-Orchestrator* is communicating with:

[5] https://github.com/clouditor/clouditor/tree/main/service/orchestrator

- ***EMERALD UI***: The *Orchestrator* integrates with the *EMERALD UI*, allowing compliance managers and other users to interact with the system. The UI provides functionalities for selecting security catalogues as well as viewing assessment results, evidence, and compliance attestations. The UI communicates with the *Orchestrator* through the REST API.
- ***Clouditor-Assessment***: The *Orchestrator* receives assessment results sent by the *Assessment* component which is then used to create evaluation results. The *Clouditor-Assessment* communicates with the *Orchestrator* via gRPC allowing sending of assessment results with high performance.
- ***Clouditor-Evaluation*:** The *Orchestrator* utilizes the *Evaluation* component to send the respective assessment results to get evaluation results for given controls. These results are the base for making the attestation decision and ensuring compliance with the selected security catalogues.
- ***Clouditor-Evidence Store***: Upon a request from the *EMERALD UI*, the *Orchestrator* receives evidence from the *Evidence Store* and forwards it to the UI.
- ***Repository of Controls and Metrics* (*RCM*):** The *Orchestrator* accesses the *RCM* to retrieve relevant metrics and controls (as well as the respective mapping provided by *MARI*). This information is used to define assessment criteria and ensure that the attestation process aligns with the required standards. The *RCM* communicates with the *Orchestrator* through the REST API.

3.1.4 Planned Implementation

The planned implementation of the *Clouditor-Orchestrator* involves several functionalities and utilizes a specific technology stack to ensure efficient and effective operation. Below is an overview of the planned functionalities and the technology stack.

Functionalities:

- Orchestration Module, i.e., orchestrating components and controlling the data flow
- API Gateway, i.e., providing a REST API allowing other components communicating with the *Orchestrator* that only support RESTful APIs
- Compliance checking, i.e., calculate evaluation and certification decision
- Supporting RBAC, allowing only authorized users to access or modify specific information

Technology Stack:

- Go as programming language, providing simple and efficient implementation, concurrency support, and ease of deployment in microservice architectures
- Communication Protocols: REST API and gRPC (including Protobuf)

3.1.5 Advancements within EMERALD

One goal within the EMERALD project is to adopt the *Clouditor* tool by increasing its Technology Readiness Level (TRL) from 5 to 7. Most of the enhancements will be made in other components of the *Clouditor* tool, e.g., by providing a database implementation in the *Clouditor-Evidence Store* that allows to use graph data queries. However, in addition to improving code quality, performance, and fixing bugs, *Clouditor-Orchestrator* will also include the functionality to make final attestation decisions. Previously, in the MEDINA project, the certification decision was handled by a different tool (see section 4.4.2 of D5.5 [10]). Finally, we also introduce the handling of users for access restriction and task handling in the *Orchestrator*.

3.2 Clouditor-Assessment

The *Clouditor-Assessment* component is responsible for assessing evidence based on predefined metrics. The calculated assessment results are inspired by, but de-coupled from, the actual controls of security catalogues. These results are used by the *Clouditor-Evaluation* component if needed to determine compliance with the relevant controls. The *Clouditor-Assessment* component is based on the respective microservice of *Clouditor*[6] and was already used in MEDINA [10]. It will be further developed to handle multiple pieces of evidence that reflect resources on different layers.

3.2.1 Requirements

The main technical requirements for the *Clouditor-Assessment* component are as follows:

Field	Description
Requirement ID	ASSESS.01
Short title	Assessment based on evidence
Description	The assessment should assess evidence utilizing a graph query language.
Status	Work in Progress
Priority	Must
Component	*Clouditor-Assessment*
Source	KPI
Type	Technical
Related KR	KR4_MULTICERT
Related KPI	KPI 4.1, KPI 4.2
Validation acceptance criteria	Evidence can be retrieved and assessed by the assessment component.
Progress	30%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	Description
Requirement ID	ASSESS.02
Short title	Assessment rules for 80% of the defined metrics
Description	Assessment rules must exist for 80% of the metrics defined in KP4.1.
Status	Work in Progress
Priority	Must
Component	*Clouditor-Assessment*
Source	KPI
Type	Technical
Related KR	KR4_MULTICERT
Related KPI	KPI 4.2
Validation acceptance criteria	Existence of assessment rules for 80% of the defined metrics. Existence of unit tests for all assessment rules.
Progress	15%
Milestone	MS6: Integrated audit suite V2 (M30)

[6] https://github.com/clouditor/clouditor/tree/main/service/assessment

Field	Description
Requirement ID	ASSESS.03
Short title	Display cause of assessment result
Description	We want to know why an assessment result fails or passes.
Status	Work in Progress
Priority	Could
Component	*Clouditor-Assessment*
Source	KPI
Type	Technical
Related KR	KR6_EMERALD UI/UX
Related KPI	KPI 6.3
Validation acceptance criteria	The cause why an assessment results fails or passes is shown in the *EMERALD UI*.
Progress	75%
Milestone	MS6: Integrated audit suite V2 (M30)

3.2.2 Design

The *Clouditor-Assessment* is based on the respective microservice of *Clouditor*[7], whose modular architecture approach ensures scalability and flexibility. Key design elements of the *Clouditor-Assessment* in EMERALD include:

- **Modular Architecture:** The *Clouditor-Assessment* component is one of *Clouditor* microservices, responsible for evaluating evidence based on predefined metrics. This modularity allows for easier maintenance and scalability, as each microservice can be updated or scaled independently.
- **Assessment Engine**: The core of the *Clouditor-Assessment* component is the assessment engine, which processes the collected evidence based on predefined metrics. The engine calculates the evidence and generates assessment results, identifying compliance and non-compliance areas. Because the evidence follows the scheme of the CertGraph ontology, it has a unified format. This unified format allows the definition of metrics to assess evidence independent of the extraction component and actual source the evidence is coming from. For example, if the incoming evidence is of type "*VirtualMachine*", the ontology presets how the evidence extractor has to form the evidence. Among others, it has to set if boot logging is enabled (it does not matter if it was collected by *Clouditor-Discovery* or by some other tool. Also, it doesn't matter which actual target of evaluation the information is collected from). Therefore, we can create a simple metric that checks if boot logging is enabled for such incoming evidence, thus decoupling assessment and evidence extraction. We have used the policy language Rego with its respective Go Client to write and apply metrics, respectively[8]. For now, we stick to use Rego, but we plan to utilize graph query language in the *Evidence Store* in the future. This change would allow to assess evidence on different layers that have to be combined in a more native ("graph based") way.
- **Assessment Reporting**: The *Clouditor-Assessment* component generates comments that include the causes of any failing assessment results. These comments are essential

[7] https://github.com/clouditor/clouditor/tree/main/service/assessment
[8] https://www.openpolicyagent.org/docs/latest/policy-language/

for understanding why a particular assessment status is non-compliant, giving an auditor more information about certain evidence.

- **Communication Protocols (API endpoints)**: The *Clouditor-Assessment* component communicates with other *Clouditor* components, such as the *Orchestrator*, via gRPC. This ensures high-performance interactions and efficient data exchange. Additionally, the component provides a REST endpoint for communication with other components not supporting gRPC.
- **Authorization with OAuth 2.0**: To ensure that information is selectively disclosed to clients based on their roles, the *Clouditor-Assessment* component utilizes OAuth 2.0 for secure authorization by checking the JSON Web Token of the request. This mechanism controls access to API endpoints and ensures that only authorized clients can view or modify specific information.

3.2.3 Integration

The following components communicate with the *Clouditor-Assessment*, as shown in Figure 1:

- ***Clouditor-Orchestrator***: The *Assessment* component coordinates with the *Orchestrator* to receive instructions and send back assessment results. In addition, the metrics are initially imported from the *Orchestrator* (originating from the *RCM*). Communication between the *Assessment* component and the *Orchestrator* is facilitated via gRPC.
- ***Clouditor-Evidence Store***: The *Assessment* component retrieves evidence from the *Clouditor-Evidence Store* to perform assessments. The communication with the *Evidence Store* is also done via gRPC to ensure high-performance data exchange.
- ***Trustworthiness System* (*TWS*)**: The *Assessment* component interacts with the *TWS* to provide assessment results as well as the respective evidence. The *Assessment* component communicates with the *TWS* to enhance the integrity and trust of the assessment process for auditors and other users. The *Assessment* component communicates with the *TWS* through a REST endpoint.

This integration ensures that the *Clouditor-Assessment* component can efficiently gather and process evidence, collaborate with other components, and provide accurate assessment results to support the overall attestation process in the EMERALD framework.

3.2.4 Planned Implementation

The planned implementation of the *Clouditor-Assessment* component involves one main functionality and utilizes a specific technology stack to ensure efficient and effective operation. Below is an overview of the planned functionalities and the technology stack.

Functionalities:

- The main functionality is to process evidence from various sources and assess it based on predefined metrics. This includes handling multiple pieces of evidence that reflect resources on different layers, ensuring a comprehensive assessment process. The processing of this multi-layered evidence is planned to be achieved by querying evidence with a graph query language.

Technology Stack:

- Programming Language: As all *Clouditor* components, the *Clouditor-Assessment* component is implemented in Go, providing a simple and efficient implementation, concurrency support, and ease of deployment in microservice architectures.

- Communication Protocols: The component uses gRPC (including Protobuf) for communication with other *Clouditor* components, enabling the handling of thousands of pieces of evidence per minute, which is essential for managing the high volume of resources in distributed environments such as in a cloud. It also provides a REST endpoint for interaction with the *TWS*, ensuring flexible and efficient communication across the system.

3.2.5 Advancements within EMERALD

Unlike the *Orchestrator*, the improvements in the *Clouditor-Assessment* component with respect to the MEDINA version are substantial, as it will support the assessment of evidence distributed across various layers of a target of evaluation. This enhancement will allow for a more comprehensive and accurate assessment, accommodating the complexity of modern cloud environments. Additionally, improvements in code quality, performance, and bug fixes will further enhance the reliability and efficiency of the *Assessment* component.

3.3 Clouditor-Evidence Store

The *Clouditor-Evidence Store* component is responsible for storing and managing evidence of different resource types and collected from various sources. The primary challenge it addresses is transitioning to a storage system representing a certification graph (KR2 CERTGRAPH). The *Clouditor-Evidence Store* is an implementation of the schema developed and defined in WP2, as described in D2.1 [5] and D2.10 [6]. The *Evidence Store* is designed to allow complex assessments and is based on the respective microservice of *Clouditor*[9], which was already used in MEDINA [10], but now it is planned to support evidence that allows graph database queries on it.

3.3.1 Requirements

The main technical requirements for the *Clouditor-Evidence Store* component are as follows:

Field	Description
Requirement ID	ESTORE.01
Short title	Storage of evidence as ontology entities that allows operations as database queries
Description	The *Evidence Store* must store the evidence according to the schema defined by the CertGraph ontology. The preferred way to store this information is either a native graph database or a datastore that utilizes a graph query language (e.g. openCypher, GraphQL, ...) to process evidence.
Status	Work in Progress
Priority	Must
Component	*Clouditor-EvidenceStore*, *Clouditor-Assessment*
Source	KPI
Type	Technical
Related KR	KR2_CERTGRAPH
Related KPI	KPI 2.1, KPI 2.3
Validation acceptance criteria	Evidence entity can be successfully retrieved by the *Assessment* component.
Progress	30% (Start enhancement of *Evidence Store* using graph database queries)

[9] https://github.com/clouditor/clouditor/tree/main/service/evidence

Milestone	MS5: Components V2 (M24)

Field	Description
Requirement ID	ESTORE.02
Short title	Allow Interaction with Third-Party Tools
Description	The *Evidence Store* should be allowed to accept evidence from third-party tools, e.g., using a REST API. Evidence needs to be in the ontology format. Therefore, information about the ontology and data models must be available.
Status	Work in Progress
Priority	Should
Component	*Clouditor-EvidenceStore*
Source	KPI
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	The *Evidence Store* stores evidence from third-party evidence collectors.
Progress	30% (*Evidence Store* is deployed in cluster and gRPC as well as REST endpoints tested)
Milestone	MS8: Integrated audit suite V3 (M34)

3.3.2 Design

The *Clouditor-Evidence Store* is based on the respective microservice of *Clouditor*[10] whose modular architectural approach ensures scalability and flexibility. Key design elements of the *Clouditor-Evidence Store* in EMERALD include:

- **Modular Architecture**: The *Evidence Store* is one of *Clouditor* microservices, responsible for efficiently storing and managing evidence. This modularity allows for easier maintenance and scalability, as each microservice can be updated or scaled independently.
- **Storage Implementation allowing Graph Database Operations**: The *Evidence Store* will use a regular Postgres database for storing the evidence. However, we will utilize a graph query language that allows for efficient organization, retrieval, and updating of evidence, making it well-suited for managing complex relationships between different types of evidence at possibly different layers (e.g., infrastructure vs. code).
- **Communication Protocols (API endpoints)**: The *Evidence Store* will use gRPC (including Protobuf) for the communication with other *Clouditor* components, ensuring high-performance interaction. Additionally, it will provide REST endpoints for flexible and efficient communication with evidence collectors not supporting gRPC.
- **Authorization with OAuth 2.0**: To ensure that information is selectively disclosed to clients based on their roles, the *Evidence Store* utilizes OAuth 2.0 for secure authorization by checking the JSON Web Token of the request. This mechanism controls access to API endpoints and ensures that only authorized clients can view or modify specific information.

[10] https://github.com/clouditor/clouditor/tree/main/service/evidence

3.3.3 Integration

The integration of the *Clouditor-Evidence Store* component with other components in the EMERALD framework is essential for efficient evidence management and retrieval. The following components communicate with the *Clouditor-Evidence Store*, as shown in Figure 1:

- **Evidence Collectors**: The *Evidence Store* is designed to interact with various evidence collectors, which gather evidence from different sources. While gRPC is the primary protocol for high-performance communication, the *Evidence Store* also provides REST endpoints to accommodate evidence collectors that do not support gRPC.
- ***Clouditor-Assessment***: The *Evidence Store* supplies the *Assessment* component with the required evidence for its assessment calculation. This interaction is also managed via gRPC, enabling the *Assessment* component to send evidence quickly and efficiently.
- ***Clouditor-Orchestrator***: The *Orchestrator* can request evidence from the *Evidence Store* for displaying it in the *EMERALD UI*.

3.3.4 Planned Implementation

As mentioned in the required ESTORE.01, the preferred way to implement an evidence store in the EMERALD framework is to either use a native graph database or a data store that allows to access the data via a graph-based query language. Examples for such languages are openCypher (which is currently standardized as Graph Query Language – GQL) or languages such as GraphQL. In order to be compatible to many existing implementations, the EMERALD framework does not impose a choice of a specific database technology on the user or developer of the frameworks. The reason for that is the myriads of different technologies, which all come with certain benefits and shortcomings. Additionally, other factors, such as already existing data store in the deployment environment or licences of projects need to be considered.

Functionalities:

The initial idea was to use a graph database to store the certification graph in the *Clouditor-Evidence Store*. However, in further discussions and research we concluded to pursue a hybrid approach: Using a relational database but utilizing a graph query language to allow more complex queries.

Therefore, the planned implementation of the *Clouditor-Evidence Store* component involves the use of an existing relational database (e.g., postgres) in *Clouditor* (which is already used to store non-graph related information) in combination with a graph-enabled query interface. Implementers of the EMERALD framework, which might not have particular restrictions could potentially leverage a native graph database directly.

The following considerations have led to the particular implementation decision in *Clouditor*. They might be used to guide other implementors following a similar approach:

- License and technology compatibility:
    - We have been examining over 30 graph database candidates[11]. The strict criteria for these databases are that they must be open source with a suitable licence, i.e., compatible with Apache 2.0[12] and compatible with Go, e.g., by providing a Go client. Out of the several candidates, the most promising candidate was dgraph[13], a high-performance graph database written in Go. However, inclusion of dgraph into the

[11] https://db-engines.com/en/ranking/graph+dbms
[12] https://www.apache.org/licenses/LICENSE-2.0.html
[13] https://github.com/hypermodeinc/dgraph

Clouditor software would have introduced over 150 dependencies. This would have greatly increased the complexity of the software. The second choice would have been Neo4J, however the Neo4J server software is licensed at GPLv3 and only the client adapter is Apache 2.0. Therefore, a direct inclusion of the server software (e.g. for a standalone mode, see below) would not have been possible for license reasons.

- Type of data stored:
  - Depending on the deployment model, the *Evidence Store* is either deployed on a standalone database or is sharing a database with the *Orchestrator*. In the former case, it would be easy to deploy a native graph-database since only graph (evidence) data is stored. In the latter case, a mixture of graph database as well as non-graph data (such as information about the target of evaluation or assessment data) needs to be stored. Graph databases excel at scenarios where either deeply nested relationships exist or where data needs to be stored at the edge/relationship themself. Both scenarios do not apply when data such as assessment results need to be stored along-side graph-data. Assessment results are not highly interconnected, they mostly only relate to one particular metric and evidence, but because of the high interval of assessment, 100.000s or million entries might exist, making this a preferred target of relational databases.
- Standalone mode:
  - Besides the distributed deployment of the *Clouditor* components (*Evidence Store*, *Orchestrator*, etc.) as a distributed application, *Clouditor* also supports a "standalone" mode, with an integrated database. The use case for this is to support small deployments and business, which do not have a complex environment, as well as one-time assessments of cloud services during a security audit. In either case a small SQLite database can currently be used instead of postgres. In the latter case, the database can even be ephemeral.

Technology Stack:

The component is implemented in Go, providing simple and efficient implementation, concurrency support, and ease of deployment in microservice architectures. Communication protocols include gRPC (including Protobuf) for high-performance interaction with other *Clouditor* components and REST endpoints for flexible communication with evidence collectors not supporting gRPC. Additionally, the *Clouditor-Evidence Store* will implement authorization to ensure that information is selectively disclosed to users based on their access rights.

3.3.5 Advancements within EMERALD

Although the *Clouditor-Evidence Store* will not support a native graph database for storing and managing evidence, we use the hybrid approach described above to implement the certification graph. This enhancement allows for efficient organization, retrieval, and updating of evidence, accommodating the complexity of modern cloud environments. Additionally, improvements in code quality, performance, and bug fixes will further enhance the reliability and efficiency of the *Evidence Store* component.

3.4 Mapping Assistant for Regulations with Intelligence (MARI)

MARI is an intelligent system capable of selecting the optimal set of metrics to associate with one or more certification schemes. These metrics can be measured to evaluate the target of evaluation's compliance within the certification schemes. The *MARI* component is based on the

Metric Recommender[14] tool developed in MEDINA [10]. A novelty with respect to the past tool is that, in EMERALD, *MARI* performs also the associations between controls of different schemes.

Thus, the objective of MARI is to experiment with Deep Learning (DL) and state-of-the-art NLP tools to create automatic associations between:

- a security control and one or more security metrics, and
- two security controls coming from two different certification schemes.

3.4.1 Requirements

Field	Description
Requirement ID	MARI.01
Short title	AI-based
Description	*MARI* is a tool based on state-of-the-art artificial intelligence, e.g., uses a transformer-based architecture.
Status	Approved
Priority	Must
Component	*MARI*
Source	Component
Type	Technical
Related KR	KR3_OPTIMA
Related KPI	KPI 3.1, KPI 3.2, KPI 3.3
Validation acceptance criteria	*MARI* uses state-of-the-art tools, such as transformer-based architectures, to produce the control-metric(s) association and the control-control association.
Progress	90%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	Description
Requirement ID	MARI.02
Short title	Automatic association
Description	*MARI* takes as input cloud security controls written in natural language, metrics that validate those controls, again written in natural language, and automatically returns as output the association control/metric(s) and the association control/control.
Status	Approved
Priority	Must
Component	*MARI*
Source	Component
Type	Technical
Related KR	KR3_OPTIMA
Related KPI	KPI 3.1, KPI 3.2, KPI 3.3

[14] https://git.code.tecnalia.com/medina/public/nl2cnl-translator

Validation acceptance criteria	The output consists of a list of control/metric(s) pairs. The output consists of a list of control/control pairs (and the controls comes from diverse certification schemes).
Progress	90%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	Description
Requirement ID	MARI.03
Short title	Performance evaluation
Description	The performance of *MARI* should improve the performance of the *Metric Recommender* of EMERALD's predecessor project, MEDINA. We can assume that we measure the performance of *MARI* with the same metrics used for the *Metric Recommender*, namely precision@k and NDCG (Normalised Discounted Cumulative Gain)[15].
Status	Approved
Priority	Must
Component	*MARI*
Source	Component
Type	Technical
Related KR	KR3_OPTIMA
Related KPI	KPI 3.1, KPI 3.2, KPI 3.3
Validation acceptance criteria	Better performances with respect to those obtained with *MARI*'s predecessor
Progress	80%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	Description
Requirement ID	MARI.04
Short title	Visualization
Description	*MARI*'s results must be visualised via the *EMERALD UI*
Status	Approved
Priority	Must
Component	*MARI*
Source	Component
Type	Technical
Related KR	KR3_OPTIMA
Related KPI	KPI 3.1, KPI 3.2, KPI 3.3
Validation acceptance criteria	It is possible to visualize *MARI* associations via the *EMERALD UI*.
Progress	80%
Milestone	MS6: Integrated audit suite V2 (M30)

[15] https://towardsdatascience.com/evaluation-metrics-for-recommendation-systems-an-overview-71290690ecba

Field	Description
Requirement ID	MARI.05
Short title	Strategies
Description	MARI can act according to specific strategies, such as considering only technical controls, or organizational controls, or controls of a certain category, or controls whose implementation costs less in terms of human resources, etc. The strategies will be defined during the project.
Status	Approved
Priority	Must
Component	*MARI*
Source	Component
Type	Technical
Related KR	KR3_OPTIMA
Related KPI	KPI:3.1, KPI: 3.2, KPI:3.3
Validation acceptance criteria	It is possible to obtain the control/metric(s) (control/control) associations selecting at least two strategies defined during the project.
Progress	15%
Milestone	MS6: Integrated audit suite V2 (M30)

3.4.2 Design

The implementation of MARI starts from the MEDINA tool named *Metric Recommender*[16] [10], which takes the description of an EUCS security requirement in natural language, the description of a list of metrics, again in natural language, and as a result returns the list of metrics in descending order of relevance. MARI's implementation takes inspiration from the *Metric Recommender* and introduces some novelties, as introduced below.

In particular, there are two tasks that MARI performs. The first is to associate a security control with one or more metrics. The second is to associate a control of a certification scheme with a similar control of another certification scheme.

Both types of associations, control-metric association and control-control association, are determined by measuring the similarity between embeddings in a vector space, derived from the natural language descriptions of controls and metrics. A higher similarity score indicates greater relevance between the paired items.

To achieve this, MARI employs advanced AI techniques, such as NLP and DL, specifically transformer-based models, to encode and capture the semantic meaning of controls and metrics. This automation reduces manual effort and streamlines the compliance manager's tasks. Additionally, various optimization strategies will be explored to select the most effective subset of metrics for measurement or to meet specific criteria set by the compliance manager.

3.4.3 Integration

The following component communicates with MARI as shown in Figure 1:

[16] https://git.code.tecnalia.com/medina/public/nl2cnl-translator

- ***Repository of Controls and Metrics* (*RCM*)**: MARI will interact with the *RCM* because MARI inputs (i.e., controls and metrics) are stored therein. It will also return the control/control associations and the control/metric(s) associations to the RCM.

3.4.4 Planned Implementation

The MEDINA *Metric Recommender* was implemented in Python using different Python libraries to apply the various steps needed for the approach. In particular, the textual descriptions of the metrics and controls were transformed into feature vectors by pre-trained models (fastText[17]). A K-d tree was then computed on the feature vectors of the metrics, which were used to select the k closest neighbours of the control vector, based on the shortest Euclidean distance.

For the implementation of *MARI*, we stick to Python as programming language, but we adopted new libraries and NLP models to compute associations. The APIs have been defined in collaboration with the owners of *RCM*.

3.4.5 Advancements within EMERALD

Compared to its predecessor, the MEDINA *Metric Recommender*, MARI allows controls belonging to different certification schemes to be automatically mapped, something that had to be done manually in the predecessor project, MEDINA.

As part of the implementation progress, we have developed *MARI* using sentence transformers. This has led to improved accuracy in control-metric associations compared to the original *Metric Recommender*, as demonstrated in experiments with EUCS controls and the metrics that were defined and associated manually in MEDINA to these controls. In addition, transformer-based architectures have been used to enable automatic control-to-control mapping, which has been successfully tested with EUCS and BSI C5 2020 controls, showing promising results. To further improve *MARI*'s mapping performances, we plan to explore different sentence transformer models and evaluate *MARI* across a variety of datasets.

3.5 Clouditor-Evaluation

The *Clouditor-Evaluation* component is responsible for evaluating the compliance of a target of evaluation against controls of security catalogues. This component addresses the challenge of aggregating and interpreting assessment results to determine overall compliance status for a given control. It is based on the respective microservice in *Clouditor*[18].

3.5.1 Requirements

The main requirements for the *Clouditor-Evaluation* component are as follows:

Field	Description
Requirement ID	EVAL.01
Short title	Display cause of failing evaluation result
Description	We want to know why the evaluation result fails or passes. Therefore, it should contain a list of assessment results that cause the evaluation status to be *non-compliant*.
Status	Accepted
Priority	Could
Component	*Clouditor-Evaluation*

[17] https://fasttext.cc/
[18] https://github.com/clouditor/clouditor/tree/main/service/evaluation

Source	Component
Type	Technical
Related KR	KR6_EMERALD UI/UX
Related KPI	N.A.
Validation acceptance criteria	The cause why an evaluation result fails or passes is shown in the evaluation results.
Progress	0%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	**Description**
Requirement ID	EVAL.02
Short title	Evaluation based on assessment results
Description	The evaluation should assess the result based on all the required assessment results stored in the database.
Status	Work in Progress
Priority	Must
Component	*Clouditor-Evaluation*
Source	Component
Type	Technical
Related KR	KR4_MULTICERT
Related KPI	N.A.
Validation acceptance criteria	Assessment results can be retrieved via the *Orchestrator* and evaluated by the *Evaluation* component.
Progress	15%
Milestone	MS6: Integrated audit suite V2 (M30)

3.5.2 Design

The design of the *Clouditor-Evaluation* component focuses on modularity, scalability, and flexibility to effectively evaluate compliance based on assessment results. Key design elements include:

- **Modular Architecture**: The *Evaluation* component is one of *Clouditor* microservices[19], responsible for evaluating compliance based on assessment results. This modularity allows for easier maintenance and scalability, as each microservice can be updated or scaled independently.
- **Compliance Evaluation**: The core functionality of the *Evaluation* component is to assess compliance by analysing assessment results. It evaluates the results against controls of security catalogues, determining whether one or more metrics meet the required standards for certification.
- **Communication Protocols**: The *Evaluation* component uses gRPC for high-performance communication with the *Orchestrator*.
- **Evaluation Reporting**: The *Evaluation* component generates comments or reports that include the causes of any failing evaluation results. These are essential for understanding why a particular evaluation status is non-compliant and for making informed decisions.

[19] https://github.com/clouditor/clouditor/tree/main/service/evaluation

- **Authorization with OAuth 2.0**: To ensure that evaluation information is selectively disclosed to clients based on their roles (communication with the *Orchestrator*), the *Evaluation* component utilizes OAuth 2.0 for secure authorization by checking JSON Web Token of the request.

3.5.3 Integration

The *Clouditor-Evaluation* component primarily communicates with the *Clouditor-Orchestrator* (see Figure 1). The *Orchestrator* coordinates the flow of assessment results to the *Evaluation* component, enabling it to perform compliance evaluations based on these results. This interaction is facilitated via gRPC, ensuring efficient and high-performance communication.

3.5.4 Planned Implementation

Below is an overview of the planned functionality and the technology stack.

Functionality: The *Evaluation* component is responsible for processing assessment results obtained from the *Clouditor-Orchestrator* and determining the compliance status based on predefined criteria (mapping of metrics to controls of a security catalogue). The criteria are given by the selection of *MARI* (see Section 3.4.2). This process involves analysing the assessment results (of the respective metrics), identifying non-compliance areas, and generating an evaluation report.

Technology Stack: The *Evaluation* component will be implemented by using the programming language Go, providing a simple and efficient implementation with strong concurrency support. Communication between the *Evaluation* component and the *Clouditor-Orchestrator* will be facilitated via gRPC, allowing the transmission of thousands of assessment results per minute.

3.5.5 Advancements within EMERALD

As the *Evaluation* component is a rather small and internal service, there will be no major developments with respect to the current service apart from adaptations of the code to the EMERALD framework and bug fixes. The only significant enhancement foreseen is the examination of different aggregation strategies for evaluating the assessment results of a given control.

3.6 Repository of controls and metrics (RCM)

The *Repository of Controls and Metrics* (*RCM*) provides a central point in the EMERALD framework where the **certification schemes are stored and managed**. It consists of a repository capable of containing different certification schemes, including the information of each scheme categorized by classes (e.g., categories, controls, requirements, assurance levels, etc.) and supporting multi-scheme and multi-level certification. The *RCM* also incorporates the definition of the **metrics** used in EMERALD to assess evidence. The *RCM* implementation is based on the MEDINA component *Catalogue of Controls and Metrics*[20] [10].

The *RCM* will provide mechanisms to update the catalogues and maintain a versioning system and will foster the interoperability using OSCAL[21] as exchange format. This feature will allow importing and exporting catalogues into/from the *RCM*. In addition, the *RCM* will manage other information, such as the mappings provided by the *MARI* component, the guidelines (e.g., guidelines for EUCS requirements are already included) and a self-assessment questionnaire to assess compliance with the EUCS scheme.

[20] https://git.code.tecnalia.com/medina/public/catalogue-of-controls
[21] OSCAL: Open Security Controls Assessment Language, https://pages.nist.gov/OSCAL/

The *RCM* provides this information to the rest of the EMERALD components via APIs, which can also be used by the *EMERALD UI* component to visually present the information of the managed schemes to the user.

3.6.1 Requirements

The requirements gathered for the *RCM* component are listed below:

Field	Description
Requirement ID	RCM.01
Short title	Multi-schema support
Description	The repository should contain at least an additional security scheme, apart from the EUCS that is the scheme implemented in MEDINA Catalogue and is inherited in EMERALD.
Status	Implemented
Priority	Must
Component	*RCM*, *EMERALD UI*
Source	DoA
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 4.1
Validation acceptance criteria	The user enters in the *RCM*, can see that several schemes are supported, and can navigate through them.
Progress	100%
Milestone	MS2: Components V1 (M12)

Field	Description
Requirement ID	RCM.02
Short title	Accessible by the rest of components
Description	The repository content should be made accessible to the rest of EMERALD components via API.
Status	Work in Progress
Priority	Must
Component	*RCM*, *Clouditor-Orchestrator*, *EMERALD UI*, *MARI*
Source	Component
Type	Technical
Related KR	KR7_INTEROP
Related KPI	N.A.
Validation acceptance criteria	The API will be tested using an Open API client. All available services will be tested.
Progress	90%
Milestone	MS6: Integrated audit suite v1 (M18)

Field	Description
Requirement ID	RCM.03
Short title	Include metrics for all schemes supported

Description	The repository should include metrics that could be used to assess the compliance with one or more certification schemes.
Status	Work in Progress
Priority	Must
Component	*RCM*, *MARI*, *Clouditor-Orchestrator*
Source	DoA, KPI
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 4.1
Validation acceptance criteria	The *RCM* contains several metrics for each scheme defined in it. The checking can be done via user interface or via API.
Progress	50%
Milestone	MS5: Components V2 (M24)

Field	**Description**
Requirement ID	RCM.04
Short title	Mapping of schemes
Description	The repository should support the mapping of the certification schemes contained: • The scheme-to-scheme mapping will be provided by the *MARI* tool and stored in the repository. • It is done and defined/refined by the user ONCE. • The rationale for the mapping decision will also be stored. • The adaptations of the mapping by the user (additions/deletions) will also be stored.
Status	Work in Progress
Priority	Should
Component	*RCM*, *MARI*, *EMERALD UI*
Source	DoA, KPI
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 3.1, KPI 3.3
Validation acceptance criteria	The user checks that security controls in a scheme are mapped to the controls in another scheme.
Progress	50%
Milestone	MS5: Components V2 (M24)

Field	**Description**
Requirement ID	RCM.05
Short title	Import/export of security schemes in OSCAL
Description	The repository is able to import a new scheme defined in the OSCAL language (this feature can also be used to update an existing scheme). The repository is able to export any available scheme in OSCAL format.
Status	Work in Progress
Priority	Must

Component	*RCM*, *EMERALD UI*
Source	DoA
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 7.1, KPI 7.2
Validation acceptance criteria	Export: The user checks that the scheme is exported by creating a file in OSCAL format. Import: The user checks that the content of the repository is updated with the imported scheme.
Progress	70%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	**Description**
Requirement ID	RCM.06
Short title	Import/export of security schemes in CSV format
Description	The repository can export a scheme to a CSV file and import a CSV file with the same format as a new scheme.
Status	Work in Progress
Priority	Could
Component	*RCM*, *EMERALD UI*
Source	Component
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 7.1
Validation acceptance criteria	Export: The user checks that the exported scheme is contained in a new file with CSV format. Import: The user checks that the content of the repository is updated with the imported scheme.
Progress	60%
Milestone	MS5: Components V2 (M24)

Field	**Description**
Requirement ID	RCM.07
Short title	Support for personalized catalogues
Description	The repository will offer the user the possibility to create a personalized catalogue of controls. T These controls can be taken from the different security schemes already existing in the *RCM*, or be totally new, defined by the user.
Status	Work in Progress
Priority	Must
Component	*RCM*, *EMERALD UI*
Source	Pilots
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 7.1, KPI 7.2

Validation acceptance criteria	The user should be able to define a new, particular catalogue, based on a set of selected controls.
Progress	30%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	Description
Requirement ID	RCM.08
Short title	Support updating/versioning of schemes
Description	The repository has to maintain a versioning system of the schemes it contains, so that if a new version is uploaded, it is able to detect the change and notify the user that a new version is available.
Status	Work in Progress
Priority	Should
Component	*RCM*
Source	Pilots
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 7.1, KPI 7.2
Validation acceptance criteria	1. A new version of the certification scheme "X" is uploaded to the repository. 2. A user using that scheme logs into the EMERALD system. 3. The user receives a notification of the scheme version change.
Progress	40%
Milestone	MS6: Integrated audit suite V2 (M30)

Field	Description
Requirement ID	RCM.09
Short title	Self-assessment questionnaires (EUCS)
Description	The system provides a self-assessment tool to calculate the fulfilment degree of the EUCS certification scheme. It comprises various levels (Basic, Substantial, and High). The questionnaire presents the user with a series of questions for each control. The answers are used to evaluate its fulfilment degree. It also provides the option to enter comments and textual references to locate the evidence supporting the given answer.
Status	Work in Progress
Priority	Could
Component	*RCM*, *EMERALD UI*
Source	Component
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 7.2,
Validation acceptance criteria	The user is able to create and fulfil a questionnaire and check the results and degree of EUCS fulfilment.
Progress	90%
Milestone	MS5: Components V2 (M24)

Field	Description
Requirement ID	RCM.10
Short title	Mapping of metrics to controls
Description	The repository should support the mapping of metrics to controls: • It will be provided by the *MARI* tool and stored in the *RCM*. • It can be refined by the user, but only the last version is stored. • The rationale for the mapping decision will also be stored. • The adaptations of the mapping by the user (additions/deletions) will also be stored
Status	Work in Progress
Priority	Should
Component	*RCM*, *MARI*, *EMERALD UI*
Source	DoA, KPI
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 3.1, KPI 3.3
Validation acceptance criteria	After the mapping has been performed, the users check that some metrics are mapped to some controls, and that they can manipulate the mapping.
Progress	20%
Milestone	MS5: Components V2 (M24)

3.6.2 Design

The *RCM* can be decomposed in three main sub-components (see Figure 2), which are briefly described as follows:

- **Frontend**: This sub-component is the graphical user interface of the *RCM*. It allows users to filter the view and select the set of information they want to check from the existing schemes (e.g., controls of a certain scheme, requirements of a certain assurance level, metrics related to some controls, etc.). This sub-component will be part of the *EMERALD UI* component. It will communicate with the backend via the API.
- **Backend**: This is the core sub-component of the RCM. It implements the APIs to perform the actual management of the scheme data, considering the filters set by the user through the UI or by calling the API. As a microservices-based component, the *RCM* could be composed of many general applications, each containing a few related entities and business rules. In our case, the *RCM* contains two backends: i) *Backend*, which deals with the management of schemes and metrics, and ii) *Backend converter*, which is dedicated to the scheme conversions to/from OSCAL.

- **Registry**: This is an internal sub-component provided by the framework that is used to create a microservice architecture that ties the other subcomponents together and enables them to communicate with each other.

In addition, data persistence is provided by an SQL database connected to the backend.

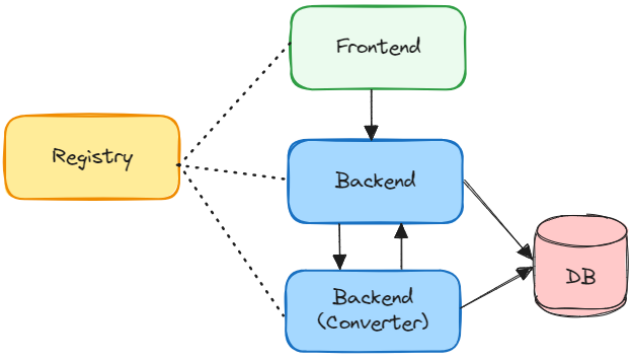

Figure 2. Architecture of the Repository of Controls and Metrics (RCM)

3.6.3 Integration

The RCM interacts with these other EMERALD components, as shown in Figure 1:

- ***Clouditor-Orchestrator*,** which retrieves the information about the schemes and the metrics from the RCM.
- ***Mapping Assistant for Regulations with Intelligence* (*MARI*)**, which provides the results of the mapping functionality to the *RCM* in order to store the results for further uses.
- ***EMERALD UI***, which retrieves the information from the *RCM* to present it in the UI. On the other hand, the user may want to change the contents of the RCM by introducing new schemes, new versions of a scheme, or answering the self-assessment questionnaire.
- ***AMOE***, a knowledge extractor that obtains from the *RCM* the definition of the security metrics needed to evaluate evidence from policy documents.

3.6.4 Planned Implementation

The *RCM* will be developed using Java and the JHipster Framework[22]. The framework provides all the needed mechanisms for a modern web application and a microservices architecture[23]. JHipster uses Spring boot for application configuration.

On the client side, the *Frontend* gateway will use JavaScript, Yeoman, Webpack, Angular, and Bootstrap technologies. On the server side, the *Backend* and the *Registry* will use Maven, Spring MVC REST for the API, Spring Data JPA, Netflix OSS[24], and Python - Flask REST API[25].

3.6.5 Advancements within EMERALD

The main development with respect to the *MEDINA Catalogue* will be the provision of an import/export mechanism for the schemes using standard languages, mainly OSCAL. In a first approach, we implemented such a mechanism for the EUCS scheme using CSV files. Furthermore, OSCAL templates have been defined for EUCS, AIC4, and BSIC5, and the export of the schemes from RCM to OSCAL has been implemented.

Another added feature of the RCM with respect to the *MEDINA Catalogue* is the possibility to create ad-hoc schemes by the user, based on existing controls or brand-new ones. For this purpose, the OSCAL management mechanism will also be used.

[22] https://www.jhipster.tech
[23] https://www.jhipster.tech/tech-stack/
[24] https://www.jhipster.tech/microservices-architecture/
[25] https://flask.palletsprojects.com/en/3.0.x/

3.7 Trustworthiness System (TWS)

As introduced in D3.1 [2], the *Trustworthiness System* (*TWS*) provides a secure mechanism for EMERALD to maintain an audit trail of evidence and assessment results. It is based on **Smart Contracts** backboned by a **Blockchain network**, providing the following functionalities:

- Includes the logic for the evidence sources (i.e., *Codyze*) to **provide evidence proofs of integrity from their origin**.
- Includes the logic for the *Clouditor-Assessment* to **provide the required information to be audited** (about evidence and assessment results).
- Provides **long-term secure information recording**, thanks to the inherent advantages of Blockchain (integrity, decentralization, authenticity, etc.).
- Includes the logic for external users to **access and validate audited information** (about evidence and assessment results) in a **graphical and user-friendly way** through a frontend included in the EMERALD UI.

The *TWS* provides trustworthiness, fairness, and transparency to the evidence and assessment results stored in EMERALD, as the integrity and authenticity of the information is guaranteed.

3.7.1 Requirements

The main requirements for the *TWS* component are:

Field	Description
Requirement ID	TWS.01
Short title	Provide integrity proof of evidence
Description	Provide a tool allowing the verification of evidence integrity without needing to store the evidence itself (for confidentiality reasons).
Status	Implemented
Priority	Must
Component	*TWS*, *EMERALD UI*
Source	DoA
Type	Technical
Related KR	KR7_INTEROP
Related KPI	N.A.
Validation acceptance criteria	This requirement should be validated by accessing the *TWS* component and checking the integrity of a piece of evidence when it has been modified and when it has not been modified.
Progress	95%
Milestone	MS5: Components V2 (M24)

Field	Description
Requirement ID	TWS.02
Short title	Provide integrity proof of assessment results
Description	Provide a tool allowing the verification of assessment results integrity without needing to store the result itself (for confidentiality reasons).
Status	Implemented
Priority	Must
Component	*TWS*, *EMERALD UI*

Source	DoA
Type	Technical
Related KR	KR7_INTEROP
Related KPI	N.A.
Validation acceptance criteria	This requirement should be validated by accessing the *TWS* component and checking the integrity of an assessment result when it has been modified and when it has not been modified.
Progress	95%
Milestone	MS5: Components V2 (M24)

Field	**Description**
Requirement ID	TWS.03
Short title	Provide access through REST API or graphical interface
Description	The integrity validation of evidence and assessment results must be done through REST API or graphical interface (*EMERALD UI*).
Status	Work in Progress
Priority	Must
Component	*TWS*, *EMERALD UI*
Source	DoA
Type	Technical
Related KR	KR7_INTEROP
Related KPI	N.A.
Validation acceptance criteria	This requirement should be validated by making the integrity validation of evidence in both ways.
Progress	75%
Milestone	MS5: Components V2 (M24)

Field	**Description**
Requirement ID	TWS.04
Short title	Use a general-purpose public-private Blockchain network
Description	The *TWS* must be based on a real Blockchain network, with multiple nodes and multiple organizations to guarantee suitable decentralization and governance of the Blockchain network.
Status	Implemented
Priority	Must
Component	*TWS*
Source	DoA
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 7.2
Validation acceptance criteria	It will be validated with the final Blockchain network considered. The Blockchain network will not be locally deployed. An already existing network governed by externals will be considered to avoid security issues as information could not be modified in any way.
Progress	95%

Milestone	MS5: Components V2 (M24)

Field	Description
Requirement ID	TWS.06
Short title	Allow evidence integrity proofs from the evidence sources
Description	The *TWS* should allow the evidence sources to automatically register and verify integrity proofs of their own evidence.
Status	Work in Progress
Priority	Should
Component	*TWS*
Source	DoA
Type	Technical
Related KR	KR7_INTEROP
Related KPI	KPI 7.2
Validation acceptance criteria	It will be validated with at least one of the EMERALD pilots, allowing them to register and verify the integrity of their evidence.
Progress	90%
Milestone	MS5: Components V2 (M24)

3.7.2 Design

Figure 3 shows the updated architecture of the Blockchain-based EMERALD *TWS*. The main update is the inclusion of the evidence extractors as evidence proofs of integrity providers.

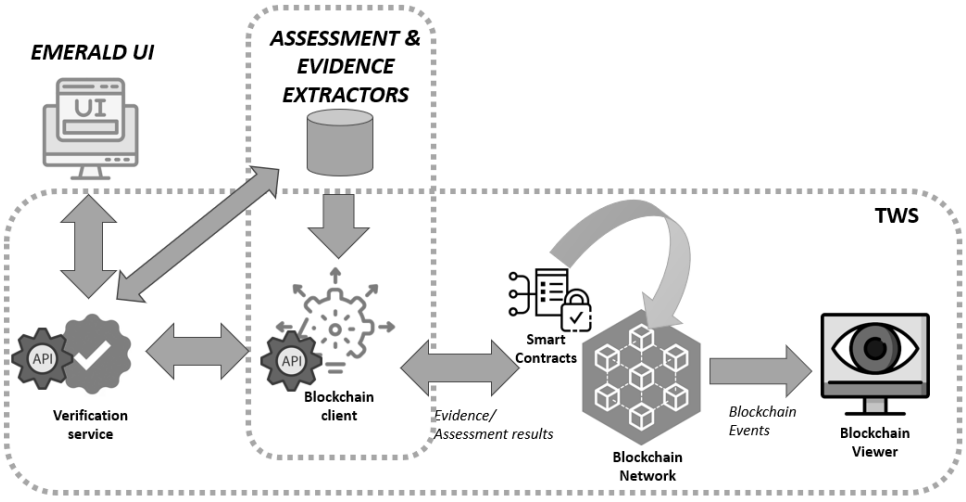

Figure 3. EMERALD Trustworthiness System (TWS) high-level architecture

The *TWS* is composed of five main elements:

- **Blockchain network**. A general-purpose Blockchain network has been considered for the prototyping of the *TWS*. The *European Blockchain Service Infrastructure* (EBSI) was firstly considered. However, they are suffering an internal legal transition process that has stopped the early adopters program. For this reason, *Alastria* Blockchain network has been finally considered. *Alastria* is the first Spanish and one of the world's largest public-permissioned multisector blockchain platforms, bringing together companies, academia, and public administration.

- **Smart Contracts**. The *TWS* auditing functionalities have been implemented through Smart Contracts to be deployed on the Blockchain network (previous element). The Smart Contract functionalities include the registration of data in the Blockchain (evidence and assessment results) to be verified, as well as the use of this previously registered data for integrity verification. In addition, Blockchain-based events are also generated to feed the Blockchain viewer.
- **Blockchain client**. The *Assessment* and the *evidence collector* components have a Blockchain client to interact with the Blockchain and the Smart Contracts (wallet management, transactions generation, etc.).
- **Blockchain viewer**. It listens to Blockchain events from the Smart Contracts and normalises and categorises the details for proper visualization in a dashboard. The Blockchain viewer allows to isolate external users from the need to have a Blockchain client to consume information recorded in the Blockchain.
- **Automatic verification service**: An automatic verification tool for current and recorded evidence and assessment results is included in the *TWS* to provide auditors an automatic way to verify the integrity of evidence and assessment results gathered in EMERALD.

3.7.3 Integration

The *TWS* interacts with three other components of the EMERALD solution (see Figure 1):

- **Evidence extractors** (in particular, *Codyze*): The *Codyze* component provides the information related to extracted evidence to be recorded in the Blockchain.
- ***Clouditor-Assessment*:** The interaction with this component will take place in two different ways:
    1. The *Assessment* component provides the information related to evidence and assessment results to be recorded in the Blockchain.
    2. The automatic verification service requests the current values of evidence and assessment results stored in the EMERALD's internal evidence storage for fair integrity validation against the information previously recorded in the Blockchain.
- ***EMERALD UI***: The automatic verification service will provide the evidence and assessment results integrity status to the *EMERALD UI* so that auditors can easily verify the trustworthiness of evidence and assessment results and determine whether they can trust on them.

3.7.4 Planned Implementation

The *TWS* will be updated to cover the new requirement related to the provision of evidence proofs of integrity directly from the evidence extractors. Besides, the automatic verification service needs to be updated to provide the required information to the EMERALD UI.

Beyond that, the functionality and performance of *TWS* will be improved based on the feedback of the pilot validation.

3.7.5 Advancements within EMERALD

The *TWS'* main functionalities were already developed in the MEDINA project: evidence and assessment results storage in the Blockchain, graphical visualization by external users, and automatic verification against the current values available in the *Assessment* component. However, the *TWS* was just a prototype in MEDINA that needs to be enhanced in EMERALD in three main aspects:

- Deploying the *TWS* functionality on a real general purpose Blockchain network to allow fair and transparent functionality (in MEDINA, it was deployed in a dummy Blockchain network for validation purposes). *Alastria* Blockchain has been considered.
- Analysing the existing implementation for improvements or updates to improve system performance, as the use of Blockchain often degrades performance. Both Smart Contracts as well as the Blockchain client have been internally updated for enhanced modularity, performance, and security.
- Enabling automatic verification of evidence or assessment results in EMERALD (in MEDINA, verification was always performed on demand).

4 Conclusions

This document provides an overview of the overall architecture and key objectives of the WP3 components in the EMERALD framework. Subsequently, we have detailed all the EMERALD components within WP3, including the *Clouditor-Orchestrator*, *Clouditor-Assessment*, *Clouditor-Evidence Store*, *Mapping Assistant for Regulations with Intelligence* (MARI), *Clouditor-Evaluation*, *Repository of Controls and Metrics* (RCM), and *Trustworthiness System* (TWS). The requirements, design, integration, planned implementation, and advancements of each component within the EMERALD project have been thoroughly described.

This deliverable sets the foundation for the subsequent development and integration phases of the project. It outlines the current state of each component and the planned enhancements, providing a clear roadmap for achieving the project's objectives.

Future steps involve the implementation and integration as outlined in deliverables D3.4 and D3.6. Specifically, D3.4 “Evidence assessment and Certification–Implementation-v2” (M24) will focus on the final implementation details of the WP3 components, while D3.6 “Evidence assessment and Certification–Integration-v2” (M27) will address the final integration of these components into the overall EMERALD system.

5 References

[1] EMERALD Consortium, “EMERALD - Annex 1 - Description of Action - GA 101120688,” 2022.

[2] EMERALD, “D3.1 Evidence assessment and Certification–Concepts-v1,” 2024.

[3] ENISA, “EUCS – Cloud Services Scheme,” [Online]. Available: https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme. [Accessed April 2025].

[4] EMERALD Consortium, “D1.2 Data Modelling and Interaction Mechanisms-v2,” 2025.

[5] EMERALD Consortium, “D2.1 Graph Ontology for Evidence Storage,” 2024.

[6] EMERALD Consortium, “D2.10 Certification Graph–v1,” 2025.

[7] EMERALD Consortium, “D1.3 EMERALD solution architecture-v1,” 2024.

[8] EMERALD Consortium, “D4.2 Results of the UI-UX requirements analysis and the work processes–v2,” 2025.

[9] EMERALD Consortium, “D5.1 Pilot definition, set-up & validation plan,” 2024.

[10] MEDINA Consortium, “D5.5 MEDINA integrated solution-v3 (https://medina-project.eu/public-deliverables/),” 2023.