EMERALD

Deliverable D2.8

Runtime Evidence Extractor – v1

Editor(s):	Angelika Schneider, Florian Wendland
Responsible Partner:	Fraunhofer AISEC (FHG)
Status-Version:	Final v1.0
Date:	31.10.2024
Type:	OTHER
Distribution level:	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	D2.8 – Runtime Evidence Extractors – v1
Due Date of Delivery to the EC	31.10.2024

Workpackage responsible for the Deliverable:	WP2 – Methodology for knowledge extraction
Editor(s):	Angelika Schneider (FHG), Florian Wendland (FHG)
Contributor(s):	-
Reviewer(s):	Verena Geist (SCCH) Cristina Martínez, Juncal Alonso (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP1, WP2, WP3, WP4, and WP5

Abstract:	This deliverable presents a tool for evidence extraction from runtime information that can be integrated with the certification graph. It is the result of work performed in Task 2.5. This document is a first/interim version, the final version on runtime evidence extractors will be reported in D2.9.
Keyword List:	Evidence collection, runtime information, cloud, Clouditor-Discovery, technical evidence.
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 4.0 International (**CC BY-SA 4.0 DEED** https://creativecommons.org/licenses/by-sa/4.0/)
Disclaimer	Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	09.09.2024	First draft version, ToC, executive summary, about this deliverable	Angelika Schneider (FHG)
v0.2	23.09.2024	Limitations and future work, prototype architecture, functional description	Angelika Schneider (FHG)
v0.3	24.09.2024	Functional description, document structure, appendices prepared	Angelika Schneider (FHG)
v0.4	25.09.2024	Component card Clouditor-Discovery	Angelika Schneider (FHG)
v0.5	27.09.2024	Renaming in-toto, add discovery component flags image, incorporating comments from Christian Banse.	Angelika Schneider (FHG)
v0.6	07.10.2024	Description of Codyze-Provenance	Florian Wendland (FHG)
v0.7	07.10.2024	Delete appendix, add conclusion	Angelika Schneider (FHG)
v0.8	08.10.2024	Finalizing document for the internal review	Angelika Schneider (FHG)
v0.9	14.10.2024	Internal QA review	Verena Geist (SCCH)
v0.10	16.10.2024	Incorporate QA review comments	Angelika Schneider (FHG)
v0.11	18.10.2024	Incorporate QA review comments	Florian Wendland, Angelika Schneider (FHG)
v0.12	28.10.2024	Final review	Cristina Martínez/ Juncal Alonso (TECNALIA)
v0.13	29.10.2024	Incorporate final review comments	Florian Wendland, Angelika Schneider (FHG)
v1.0	31.10.2024	Submitted to the European Commission	Cristina Martínez/ Juncal Alonso (TECNALIA)

Table of contents

List of tables

List of figures

Terms and abbreviations

AI	Artificial Intelligence
AI-SEC	AI Security Evidence Collector
AMOE	Assessment and Management of Organisational Evidence
API	Application Programming Interface
AWS	Amazon Web Services
CertGraph	Certification Graph
CI/CD	Continuous Integration / Continuous Deployment
CLI	command-line interface
Codyze	Static Code Analyzer from FHG
CSAF	Common Security Advisory Framework
CSP	Cloud Service Provider
eknows	Platform for Software Analysis from SCCH
GA	Grant Agreement to the project
GitLab	Version control and DevOps platform
gRPC	Google Remote Procedure Call
in-toto	Framework defining attestation format for software supply chains
KPI	Key Performance Indicator
KR	Key Result
MEDINA	Predecessor project of EMERALD
MVP	Minimum Viable Product
REST	Representational State Transfer
SLSA	Supply-chain Levels for Software Artifacts
TOM	Technical and Organisational Measure
TRL	Technology Readiness Level
WP	Work Package

Executive Summary

This deliverable presents the initial design, architecture, and implementation state of the runtime evidence extractors *Clouditor-Discovery* and *Codyze-Provenance* of WP2. It contributes to the key result KR1-EXTRACT of EMERALD, a framework to continuously extract runtime information of a cloud service and prepare suitable evidence based on them.

EMERALD follows a knowledge graph-based approach to provide a unified view of the cloud service under certification at different layers of the service, ranging from the infrastructure layer (e.g., virtual resources), to the business layer (e.g., policies and procedures), to the implementation layer (e.g., source code files) and data layer (e.g., increasingly used AI models) in cloud applications. The runtime evidence extractors, developed in Task 2.5 and described in this deliverable, aim on the one hand at identifying critical security-related functionality such as data encryption, transport encryption, or authentication in cloud infrastructure components. On the other hand, they establish software provenance and artefact attestation to completely track software from its inception as source code to deployed build artefacts. This is complementary to the evidence gathered in Task 2.2. Other related deliverables in WP2, all due at project month 12 (October 2024), provide functional and technical details on further evidence extractors from different sources, i.e., D2.2 [1] on source evidence extraction in Task 2.2, D2.4 [2] on evidence extraction from policy documents in Task 2.3, and D2.6 [3] on security and privacy preserving evidence extraction in Task 2.4. All these details contributed to D2.1 [4] on the overall information model of the certification graph in Task 2.1.

This document starts by illustrating how the runtime extractors fit into the overall EMERALD architecture. The main part provides functional and technical descriptions of the evidence extractors *Clouditor-Discovery* and *Codyze-Provenance*, including their purpose and scope, the (current and planned) coverage of the EMERALD requirements, and the components' internal architectures. These descriptions are complemented by information on delivery and usage, as well as on limitations and future work. Finally, the document concludes with a short summary.

The runtime evidence extractors described in this deliverable contribute to KR1-EXTRACT by providing next-generation evidence gathering tools and techniques based on a knowledge graph approach. One extractor – *Clouditor-Discovery* –has currently the initial prototype implemented and is ready to be integrated with other components of the EMERALD architecture. The requirement of the component is already partially satisfied by the presented prototype. The second extractor – *Codyze-Provenance* – is a new component within the EMERALD architecture and is currently in its initial design phase. Based on the work described in this deliverable, the runtime evidence extractors will be further extended and integrated into the EMERALD framework.

This is the first iteration of the deliverable coming from Task 2.5. The second and final version of this deliverable with the updated extractors will be delivered with D2.9 [5] in project month 24 (October 2025). Evidence will be prepared according to the integrated, graph-based model of semantically linked and combined evidence, provided in D2.10 (interim version) [6] in project month 15 (January 2025) and D2.11 (final version) [7] in project month 27 (January 2026). The extracted evidence will be stored and assessed, i.e., to verify the implementation of security metrics, in the scope of WP3.

1 Introduction

EMERALD aims to provide a next generation set of evidence gathering tools and techniques based on a knowledge graph approach. KR1-EXTRACT supports an improved and unified tool-supported approach to continuously extract knowledge from different layers of a cloud service, e.g., infrastructure, platform, runtime information, policy documents, software, and AI models.

The objective of WP2 is to establish a unified view of the cloud service being certified, known as the certification target, by extracting and enriching knowledge of the different layers of the service and providing suitable evidence for security metrics. A major part of this work package is research and design of multiple tools and techniques to extract knowledge out of various sources. A graph-based model, called the certification graph (*CertGraph*), serves as a common structure that is filled by all evidence extraction components.

1.1 About this deliverable

The goal of this deliverable is to present the design and implementation of the EMERALD evidence extractors, that extract runtime information from cloud services. This is a report on the initial prototype reflecting an early stage of implementation and integration of the extractor and is the first of two iterations of deliverables, resulting from Task 2.5.

Evidence on the runtime information level is gathered by the runtime information evidence extractor *Clouditor-Discovery*, which supports the *CertGraph* data model. The *Clouditor-Discovery* component is based on the respective microservice of *Clouditor*[1] and was already used in MEDINA[2]. It focuses on generating evidence for security-related findings, such as encryption in use, at rest encryption or restricted ports. While the component was at TRL 5 in MEDINA, it should be advanced to TRL 7 in EMERALD.

Another source of runtime information are CI/CD pipelines and their jobs. During the build of a cloud service or application from source code, jobs such as application security testing, software composition analysis and secure software development measures augment the confidence in the security of the final build artefact. *Codyze-Provenance* is a new addition to EMERALD and currently under design that intends to gather evidence about CI/CD pipeline executions. This evidence would facilitate to assess security enhancing jobs executed during a build in a CI/CD pipeline. Moreover, *Codyze-Provenance* will provide attestations for executed jobs and link them together to provide provenance. From attestation and provenance reports it becomes possible to track the complete supply chain of software artefacts from source code to deployed artefacts.

Furthermore, application-specific runtime information (e.g., found in log files) might be used to provide additional context regarding the executed functionality. For now, it is just an idea and not implemented.

1.2 Document structure

The document is structured as follows.

In Section 2, we discuss how the runtime evidence extractors fit into the overall EMERALD architecture and their relationship with other components. Section 3 describes the *Clouditor-Discovery* evidence extractor, which provides the extraction of runtime information of cloud services. Section 4 describes the *Codyze-Provenance* evidence extractor, which provides traceability and attestation along CI/CD pipelines. For each extractor, we provide functional and

[1] https://github.com/clouditor/clouditor/tree/main/service/discovery
[2] https://medina-project.eu/

technical descriptions, along with information about delivery, usage, limitations, and future work.

Section 5 ends up with the conclusions of this deliverable.

2 Runtime evidence extractors in the EMERALD architecture

This section describes how the runtime evidence extractors interact with (selected) EMERALD components on a conceptual level. Figure 1 shows the EMERALD high-level architecture as a component diagram [8]. In EMERALD, a component is defined as "any part of the EMERALD ecosystem that has a specific functionality and can be considered a separate entity with respect to other components" [9]. In contrast, a tool is a "software element that has several disparate functions and therefore can be composed by several components" [9]. Therefore, *Clouditor-Discovery* and *Codyze-Provenance* are referred to as components rather than tools in the context of EMERALD.

The components for collecting evidence about technical and organisational measures, i.e., *AMOE*, *eknows*, *AI-SEC, Clouditor-Discovery*, and *Codyze*, are represented at the bottom part of Figure 1. The source evidence extractor components *Codyze* and *eknows* [1] obtain technical evidence from the analysis of the source code of cloud applications. *AMOE* [2], the component for organisational evidence gathering from MEDINA, analyses various documents and policies of the cloud service provider (CSP) and produces evidence about the CSP's compliance to organisational requirements of the certification framework. *Clouditor-Discovery*, also originated from MEDINA, collects evidence about the secure configuration of cloud resources, with a focus on runtime data extraction in EMERALD. *AI-SEC* [3] is a newly developed component in EMERALD and analyses AI models for several key evidence regarding robustness against adversarial attacks, explainability, and fairness. *Codyze-Provenance* is a new addition to *Codyze* to support runtime evidence extraction and is currently under design. In the EMERALD component diagram, it is part of the represented *Codyze* component.

In MEDINA, some extraction components implemented their own assessment, which causes more maintenance effort if requirements change over time. Thus, centralizing assessment is one of the goals in EMERALD. This is done by delivering exclusively (or as far as possible) raw evidence to the *Evidence Store* [10]. All WP2 extraction components, which extract knowledge from the various layers of a cloud service (i.e., policy documents, source code, cloud interfaces, AI models, etc.), provide (part of) evidence (e.g., for transport encryption), which is then mapped to the EMERALD evidence format using the terms described in the *CertGraph Ontology* [4]. This evidence information is stored in the *Evidence Store* following the defined schema and is used to assess the metrics defined in the *Repository of Controls and Metrics* [10].

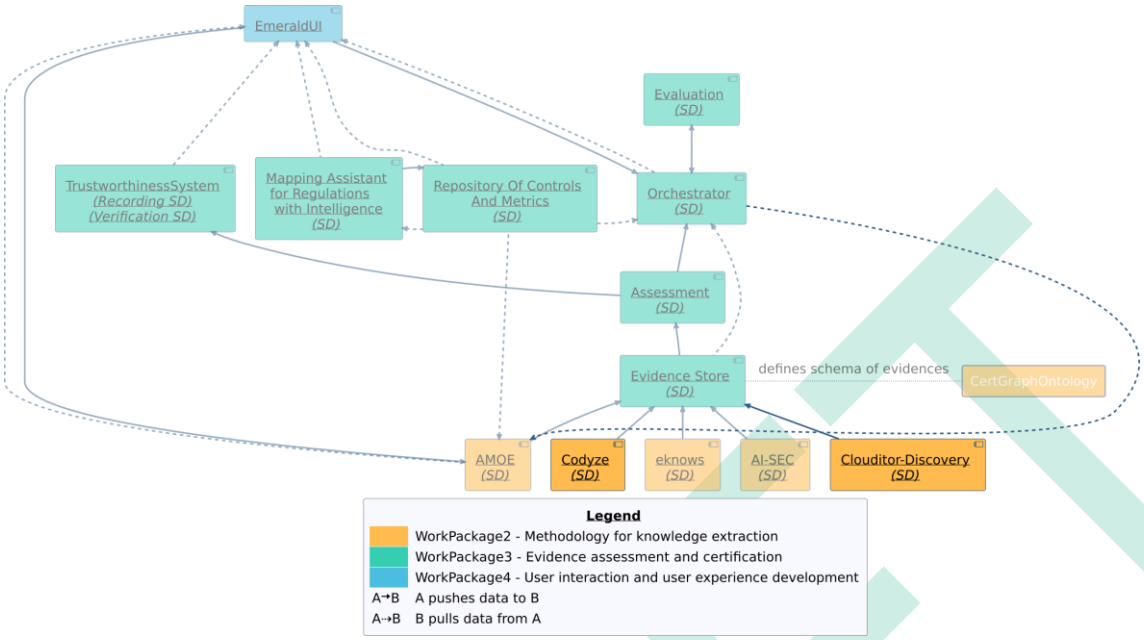

Figure 1. Excerpt of the EMERALD component diagram [8]. The highlighted component Clouditor-Discovery and the component Codyze-Provenance, which is part of the Codyze component, are described in this deliverable.

3 Clouditor-Discovery

The *Clouditor-Discovery* component is responsible for discovering security-relevant configurations from cloud resources. As such, it is one of the evidence collectors that can be integrated into the EMERALD framework. The following sections describe the component in detail regarding the overall purpose, important requirements, architecture, technical aspects, usage and the limitations and future work.

3.1 Functional description

Overall purpose. The component *Clouditor-Discovery* serves the purpose of identifying and discovering security-related configurations across various cloud resources within the EMERALD project. Its primary goal is to enhance security compliance by discovering security-relevant configurations, such as encryption in use, encryption at rest and restricted ports.

Context and scope. *Clouditor-Discovery* is a microservice within the *Clouditor*[3] tool, which has been successfully used in the MEDINA project[4] [11]. In EMERALD, the *Clouditor-Discovery* component differs from MEDINA in the variety of discovered resources (e.g., Azure Key Vaults, Functions, Web Apps) and the use of the *Owl2proto*[5] tool (see D2.1 [4]). Additionally, it will be further developed to support additional discoverers, such as OpenStack[6]. Other components of the Clouditor used in EMERALD, such as the *Orchestrator*, *Assessment* and *Evidence Store*, are described in Deliverable D3.1 [12].

The *Clouditor-Discovery* focuses on cloud resources, including Virtual Machines, Object Storage, and Network Interfaces, from multiple CSPs like Azure. It achieves this by utilizing API calls to collect runtime information, which is mapped to the EMERALD evidence format in accordance with the *CertGraph* Ontology [4], and then stored in the *Evidence Store* component. With the newly developed tool *Owl2proto*, as outlined in D2.1 [4] and the related paper [13], we can seamlessly integrate the EMERALD evidence format through automatically generated proto files from the Ontology directly into the *Clouditor-Discovery* component. This approach enhances efficiency and significantly reduces maintenance efforts compared to the previous manual process for creating and updating Ontology objects.

Motivation. A key goal of the EMERALD project is to assess the cloud service security, making evidence from these services crucial for the assessment. To achieve a comprehensive result across all layers of the cloud services, evidence extractor components are necessary to analyse as many layers as possible. The *Clouditor-Discovery* serves as the key component for extracting runtime configurations from cloud services, and a previous version has already been integrated into the previous MEDINA architecture. In addition to this component, there are other evidence extraction components for various other layers of the cloud services, including source, organisational and AI model evidence extractors.

Requirements. The technical requirement for the *Clouditor-Discovery* component is provided in Table 1. The *Clouditor-Discovery* component is currently implemented for the CSPs Azure, AWS,

[3] https://github.com/clouditor/clouditor
[4] https://medina-project.eu/
[5] https://github.com/oxisto/owl2proto
[6] https://www.openstack.org/

and Kubernetes. Additionally, OpenStack[7] and OpenNebula[8] are planned for implementation in future work.

Table 1. Requirement CLDISC.01 - Discovery of security features of infrastructure components

Field	Description
Requirement ID	CLDISC.01
Short title	Discovery of security features of infrastructure components
Description	The *Clouditor-Discovery* needs to discover security properties of infrastructure components. The evidence with the security properties is sent to the Evidence Store in the ontology format.
Status	Work in Progress
Priority	Must
Component	Clouditor-Discovery
Source	KPI
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	Validated if evidence arrives at the *Evidence Store*.
Progress	Partially implemented- 40%
Milestone	MS6: Integrated audit suite V2 (M30)

3.2 Technical description

In this section, we give the technical description of the runtime evidence collection component *Clouditor-Discovery*.

3.2.1 Prototype architecture

The *Clouditor-Discovery* is a component of the tool Clouditor which employs a microservice architecture allowing individual components to scale or to add new components, e.g., adding several evidence collection components for new cloud services. The components are written in Go[9] and communicate among each other via the gRPC[10] protocol.

For detailed interface specification, refer to the *./api/discovery* folder within the *Clouditor*[11] repository. The specification is written in accordance with the *Protocol Buffer Version 3 Language Specification*[12]. In addition, an auto-generated *.yaml* file that complies with the OpenAPI description for REST APIs is available in the *./openapi/discovery* folder. The implemented API functions are shown in Table 2.

[7] https://www.openstack.org/
[8] https://opennebula.io/
[9] https://go.dev/
[10] https://grpc.io/
[11] https://github.com/clouditor/clouditor/
[12] https://protobuf.dev/programming-guides/proto3/

Table 2. Overview of the important API functions for the Clouditor-Discovery

Function Name	Parameters	Return type	Description
Start	optional resource_group (string) optional csaf_domain (string)	successful (bool)	Triggers the start of the discovery process. Returns true if the component started without errors. The function includes two optional parameters: • With the parameter resource_group only the specified resource group is discovered. • With the parameter csaf_domain only the specified CSAF[13] domain is discovered.

3.2.1.1 Component description

The functionality of the *Clouditor-Discovery* component can be divided into 3 parts:

- Fetching security-relevant configurations of cloud resources
- Creation of evidence based on the EMERALD evidence format (*CertGraph* Ontology)
- Forwarding the evidence to the *Clouditor-Evidence Store*

In *Clouditor-Discovery*, the discovery package is located at the top level. Its primary role is to communicate with the *Evidence Store component* in EMERALD. Initially, this service connects to the *Evidence Store*, then activates different discoverers (e.g., a discoverer for Azure) and continuously sends the gathered evidence through a gRPC channel to the *Evidence Store*.

For each CSP there is a separate sub-package (e.g., AWS, Azure, Kubernetes). Each package contains a CSP-specific file which initializes configurations and credentials that all underlying cloud services share. For the main resource types (e.g., compute, network, storage) there are corresponding Go files that fetch the desired runtime information from the service via API calls. According to the *CertGraph Ontology*, these properties are converted to the EMERALD evidence format which is independent from the used CSP. By using the newly developed tool *Owl2proto* the *Clouditor-Discovery* can directly use the automatically generated proto files without having to manage the creation and updating of the Ontology objects. The properties available for retrieval depend on the range of API calls offered by the respective CSP.

3.2.2 Technical specifications

The *Clouditor-Discovery* component is developed in Go (version 1.22) and consists of four packages for the supported providers. The following list outlines the supported providers along with their corresponding key modules:

- AWS: `github.com/aws/aws-sdk-go-v2`
- Azure: `github.com/Azure/azure-sdk-for-go`
- CSAF: `github.com/csaf-poc/csaf_distribution/v3`
- Kubernetes: `k8s.io/api`

A full list of used libraries can be found in the Clouditor GitHub repository[14].

[13] https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html
[14] https://github.com/clouditor/clouditor/tree/main/service/discovery

3.3 Delivery and usage

The following sections give a short overview of the delivery and usage of the prototype. Further technical details can be found in the Clouditor GitHub Repository[15].

3.3.1 Package information

Table 3 shows the structure of the important folders of *Clouditor-Discovery* and a brief description of them.

Table 3. Overview of the package structure of Clouditor-Discovery.

Folder	Description
`api/discovery`	This folder contains the interface specification file (.proto), the auto-generated *Protobuf,* gRPC files and the code needed for the communication. The .proto file defines the data structure and service methods using Protocol Buffers (Protobuf), a serialization format developed by Google for efficient data exchange.
`api/ontology`	This folder contains the auto-generated *.proto* file based on the *CertGraph* Ontology. The *.proto* file is generated by the *Owl2proto* tool, which converts Ontology files to *Protobuf*. For further information see D2.1 [4].
`cli/commands/service/discovery`	This folder contains the *Clouditor-Discovery* CLI-based source code files.
`cmd/discovery`	This folder contains the main files for the *Clouditor-Discovery* component.
`openapi/discovery`	This folder contains the auto-generated *OpenAPI* files.
`server/commands/discovery`	This folder contains the CLI command to start the *Clouditor-Discovery* server.
`rest/`	This folder contains the REST gateway implementation.
`service/discovery`	This folder contains the source code for the *Clouditor-Discovery* component separated for each CSP: AWS, Azure, Kubernetes and CSAF.

3.3.2 Installation

The installation instructions can be found in the Clouditor README[15]. If only the *Clouditor-Discovery* component should be started, the build and start command must be adapted to

go build –o ./clouditor-discovery cmd/discovery/discovery.go and

./clouditor-discovery

Using the *--help* flag displays all available options as shown in Figure 2.

[15] https://github.com/clouditor/clouditor

```
$ ./discovery  --help
This command starts a Clouditor Discovery service

Usage:
  discovery [flags]

Flags:
      --api-cors-allowed-headers stringArray   Specifies the headers allowed in CORS (default [Content-Type,Accept,Authorization])
      --api-cors-allowed-methods stringArray   Specifies the methods allowed in CORS (default [GET,POST,PUT,DELETE])
      --api-cors-allowed-origins stringArray   Specifies the origins allowed in CORS
      --api-default-password string            Specifies the default API password (default "clouditor")
      --api-default-user string                Specifies the default API username (default "clouditor")
      --api-grpc-port uint16                   Specifies the port used for the Clouditor gRPC API (default 9091)
      --api-http-port uint16                   Specifies the port used for the Clouditor HTTP API (default 8081)
      --api-jwks-url string                    Specifies the JWKS URL used to verify authentication tokens in the gRPC and
                                               HTTP API (default "http://localhost:8080/v1/auth/certs")
      --api-key-password string                Specifies the password used to protect the API private key (default "changeme")
      --api-key-path string                    Specifies the location of the API private key (default "~/.clouditor/api.key")
      --api-key-save-on-create                 Specifies whether the API key should be saved on creation. It will only created if
                                               the default location is used. (default true)
      --assessment-url string                  Specifies the Assessment URL (default "localhost:9093")
      --cloud-service-id string                Specifies the Cloud Service ID (default "00000000-0000-0000-0000-000000000000")
      --dashboard-callback-url string          The callback URL of the Clouditor Dashboard. If the embedded server is used, a public
                                               OAuth 2.0 client based on this URL will be added (default "http://localhost:8080/callback")
      --db-host string                         Provides address of database (default "localhost")
      --db-in-memory                           Uses an in-memory database which is not persisted at all
      --db-name string                         Provides name of database (default "postgres")
      --db-password string                     Provides password of database (default "postgres")
      --db-port uint16                         Provides port for database (default 5432)
      --db-ssl-mode string                     The SSL mode for the database (default "disable")
      --db-user-name string                    Provides user name of database (default "postgres")
      --discovery-auto-start                   Automatically start the discovery when engine starts
      --discovery-csaf-domain string           The domain to look for a CSAF provider, if the CSAF discovery is enabled
  -p, --discovery-provider strings             Providers to discover, separated by comma
      --discovery-resource-group string        Limit the scope of the discovery to a resource group (currently only used in the Azure discoverer
  -h, --help                                   help for discovery
      --log-level string                       The default log level (default "info")
      --service-oauth2-client-id string        Specifies the OAuth 2.0 client ID (default "clouditor")
      --service-oauth2-client-secret string    Specifies the OAuth 2.0 client secret (default "clouditor")
      --service-oauth2-token-endpoint string   Specifies the OAuth 2.0 token endpoint (default "http://localhost:8080/v1/auth/token")
```

Figure 2. Overview of the available options for Clouditor-Discovery

3.3.3 Instructions for use

Within the EMERALD project, the *EMERALD UI* is developed and used to access and manage the workflow within the framework. The *Clouditor-Discovery* is not directly accessible through the *EMERALD UI*; however, it can be attached to the *Certification Target* in the EMERALD UI.

Figure 3 shows the *EMERALD UI* view for selecting an evidence collector. Point 1-3 provide additional information about the evidence extractor, while the button at point 4 allows the user to add a new evidence collector. Point 5 displays the current status of the evidence extractor.

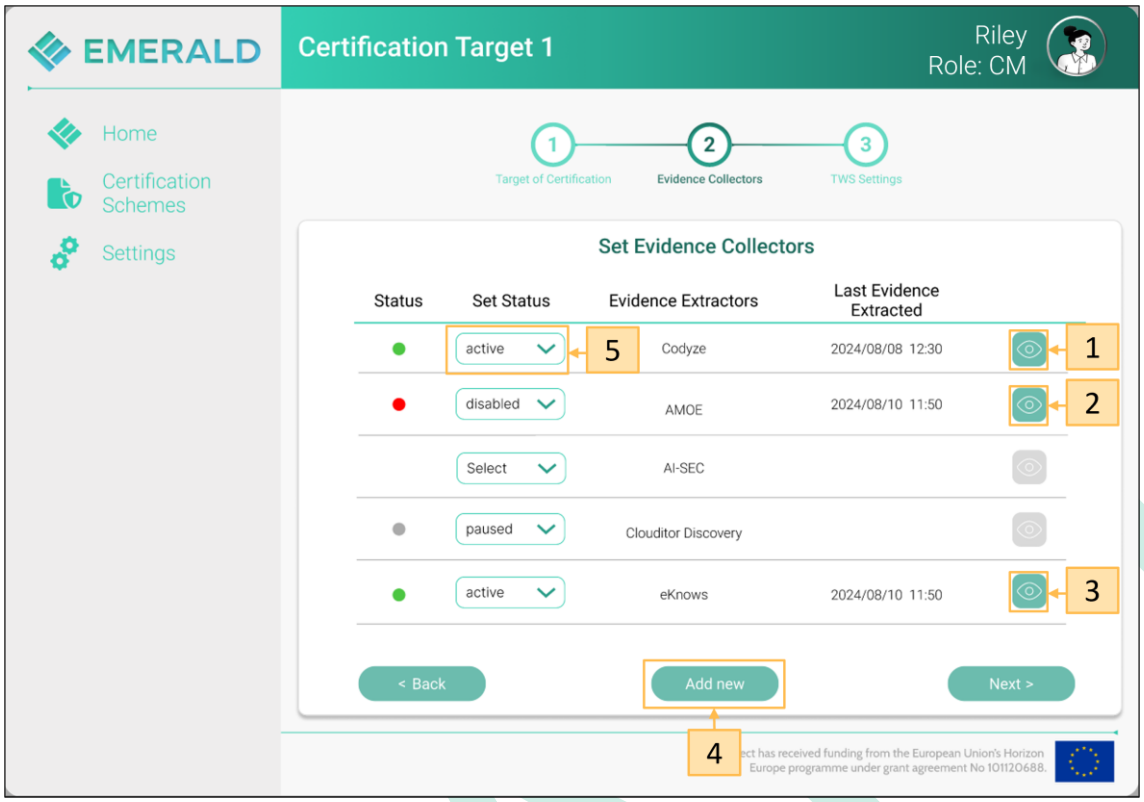

Figure 3. EMERALD UI view for setting the evidence collectors for a Certification Target (D4.3 [14])

Figure 4 presents the subsequent view containing configuration information for the selected evidence collector. The specifics of which information will be displayed and what functionalities will be available for *Clouditor-Discovery* are yet to be defined.

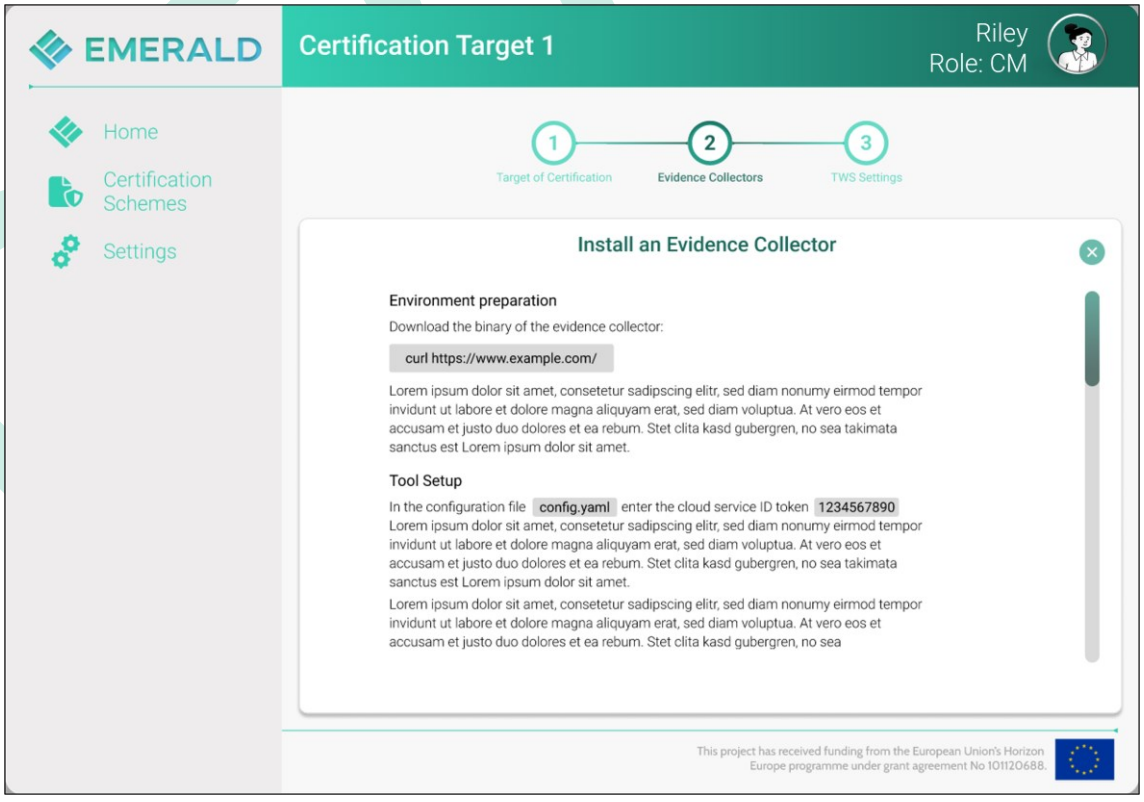

Figure 4. EMERALD UI view for the available functionalities of an evidence collector (D4.3 [14])

Further information regarding the *EMERALD UI* can be found in D4.3 [14].

Note that the installation of the evidence collectors cannot be performed via the *EMERALD UI*; however, users can attach individual evidence collectors to a *Certification Target*.

The available user manual can be found in the Clouditor README[16] and the CLI[17].

3.3.4 Licensing information

The Clouditor is licensed under the open-source Apache-2.0 license including all sub-components such as *Clouditor-Discovery*.

3.3.5 Download

The Clouditor source code can be found in the Clouditor GitHub repository[18]. The adapted EMERALD component *Clouditor-Discovery* can be found in the public EMERALD GitLab repository[19].

3.4 Limitations and future work

This section describes the limitations and future work for the *Clouditor-Discovery* component.

The limitations are as follows:

- *Clouditor-Discovery* currently gathers data from Microsoft Azure, AWS, Kubernetes environments and CSAF, but its functionality is restricted by the access permissions granted to it in user management systems like Azure Active Directory. As a result, it can only access resources that are visible to the assigned user.
- Changes to the cloud provider APIs may require updates to the component. For example, if key security features, such as at rest encryption, are modified, their integration into the EMERALD evidence must be adjusted accordingly.
- The evidence collection is restricted by the capabilities of the cloud provider APIs. If a specific encryption feature is not supported by an API, capturing evidence for that feature will not be possible.
- *Clouditor-Discovery* incorporates ontological terms into the evidence; therefore, the constraints of the ontology must be considered:
    - First, it is crucial that the ontology terms are accurately integrated into the evidence; otherwise, the assessment component may yield incorrect metrics.
    - Second, the ontology requires ongoing updates, and any modifications must be reflected as well in the *Clouditor-Discovery* component.
    - We have already developed a tool called *owl2proto* that converts the modelled ontology into an appropriate Protobuf schema, which can be directly used in various programming languages. However, the code in the *Clouditor-Discovery* component still needs to be adjusted for more significant changes.

Future work will concentrate on implementing additional discoverers for security-related cloud configurations for the CSPs OpenStack[20]and OpenNebula[21], aiming to integrate the pilot partners as well.

[16] https://github.com/clouditor/clouditor
[17] https://github.com/clouditor/clouditor#clouditor-cli
[18] https://github.com/clouditor/clouditor/tree/main/service/discovery
[19] https://git.code.tecnalia.com/emerald/public/components/discovery
[20] https://www.openstack.org/
[21] https://opennebula.io/

4 Codyze-Provenance

Codyze-Provenance is a new addition to the *Codyze* component in EMERALD. It adds runtime evidence extraction capabilities to *Codyze*. It will create a verifiable trail of evidence from source code to running cloud services and applications. This provenance creates a stronger link between static analysis based on source evidence extractors and runtime evidence extractors.

4.1 Functional description

Overall purpose. *Codyze-Provenance* is intended as a realisation of the SLSA provenance framework[22] and the in-toto attestation framework[23]. The SLSA provenance framework specifies measures to harden the security of supply chains for software artefacts. In addition to specifying how each step in a software's supply chain can be security hardened, it also links the different stages to one another to support traceability and provenance. The in-toto attestation framework provides the technical specification on how to create these links by generating verifiable statements about what went into a stage, what happened in a stage and what resulted from a stage. *Codyze-Provenance* creates provenance and attestations of software build processes. This information allows to track and verify what data went into a build process, what tools where executed and what artefacts were created. Thus, it becomes possible to trace the origin of a running cloud service and application back to its specific source code and corresponding build process.

Codyze-Provenance will provide the necessary tool and supporting components to realize SLSA and in-toto in a CI/CD pipeline. Moreover, it will submit provenance and attestations to the *Evidence Store* for further assessment.

Context and scope. *Codyze-Provenance* needs to be integrated into a CI/CD pipeline and executed on every run of the CI/CD pipeline for a cloud service or application. It will adjust the build process to support the generation of provenance and attestations. They are collected and transformed into evidence within the EMERALD framework and submitted to the *Evidence Store*.

Motivation. One challenge during runtime is to verify the origin of a running cloud service or application. Usually, it's still possible to identify the container image or binary that is running. However, it becomes increasingly difficult to provide details on how the service or application was built, which built steps were executed, or what source code version was used in the build. A complete software supply chain would describe all resources and process producing the final deployable service or application. This provenance allows to verify requirements such as mandatory security testing of applications or security checks for dependencies. Moreover, a strong link between sources and build processes on the one hand and the runtime on the other hand is created.

Requirements. Currently, *Codyze-Provenance* is a new addition in EMERALD and corresponding requirements still need to be defined.

Innovation. *Codyze-Provenance* provides a user-friendly approach to integrate SLSA and in-toto into software build processes for cloud services and applications. Collected evidence enhance the ability to assess compliance with respect to compliance schemes.

[22] https://slsa.dev/
[23] https://in-toto.io/

4.2 Technical description

The following subsections outline the technical details of *Codyze-Provenance* as they are envisioned.

4.2.1 Prototype architecture

Codyze-Provenance will consist of an application that collects provenance and attestation reports defined by the SLSA and in-toto framework. These reports are transformed into evidence for EMERALD and submitted to the *Evidence Store*. In addition, *Codyze-Provenance* will provide templates and supporting files to create a CI/CD pipeline that supports the generation of provenance and attestation reports.

At this stage, a more detailed prototype architecture cannot be presented because *Codyze-Provenance* is still in its initial design phase.

4.2.1.1 Sub-components description

At this point of planning, no details on possible sub-components can be provided.

4.2.2 Technical specifications

Codyze-Provenance will be developed in the programming language Kotlin with a Java Virtual Machine as backend. Moreover, it will provide templates and other supporting files to facilitate an integration into CI/CD pipelines. To this end, the focus is on GitLab, which is also used as the repository and CI/CD platform in EMERALD. Further details will be provided in the next iteration of this report on runtime evidence extractors in D2.9 [5].

4.3 Delivery and usage

The following subsections detail the delivery and usage of *Codyze-Provenance*. The provided information is currently work in progress and may change.

4.3.1 Package information

Codyze-Provenance will be delivered as an application bundled in an archive. The structure of this package is summarized in Table 4.

Table 4. Overview of tentative package structure for Codyze-Provenance

Folder	Description
`bin/`	Contains execution scripts for Windows and Linux/macOS
`docs/`	Contains detailed documentation
`etc/`	Contains sample configuration files and supporting files
`lib/`	Contains application and dependent libraries
`LICENSE`	License text (Apache License, Version 2.0)
`README.md`	Short documentation including short summary description, installation and usage instructions, and further information

4.3.2 Installation

Installation instructions will be provided as part of a README and documentation for *Codyze-Provenance* (cf. Table 4).

4.3.3 Instructions for use

Instructions on how to use *Codyze-Provenance* will be provided as part of the released packages (cf. folder '`docs/`' and '`README.md`' in Table 4). This information will also be available in the public GitLab repository[24] of EMERALD.

4.3.4 Licensing information

Codyze-Provenance will be licensed as open source under Apache License, Version 2.0. In addition, it is ensured that third-party dependencies are compatible with the Apache License, Version 2.0.

4.3.5 Download

Codyze-Provenance will be available from the public EMERALD GitLab repository[24] hosted by Tecnalia. The repository is going to host the source code, the documentation, binary artefacts and supporting materials.

4.4 Limitations and future work

Codyze-Providence is a new component added to the EMERALD framework. It is currently in its initial design phase and many of its details are yet to be specified. Hence, the goal is to have an initial MVP as soon as possible.

[24] https://git.code.tecnalia.com/emerald/public/components/codyze/codyze-provenance (WIP)

5 Conclusions

The EMERALD project proposes a holistic approach to evidence collection, encompassing all levels of the cloud service from the infrastructure layer (e.g., virtual resources) to the business layer (e.g., policies and procedure), and the implementation layer (e.g., source code files).

This deliverable contains the technical report on the design, architecture, and current implementation status of the runtime evidence extractor components of Task 2.5 ("Extraction of evidence using runtime information"). The components adhere to the overall EMERALD framework and align with the technical requirements gathered within the scope of WP1. The deliverable outlines the relationship of the presented components with other components of the EMERALD framework and details the internal structure, subcomponents, and technical implementation information of each component -- as far as known at the current time.

The components introduced in this deliverable include the runtime evidence extraction components *Clouditor-Discovery* and *Codyze-Provenance*. The *Clouditor-Discovery* component was already used in MEDINA and has a working prototype that can be integrated into the EMERALD framework. It has been enhanced to incorporate a variety of discovered resources and the use of the newly developed *Owl2proto* tool for the automatic generation of the necessary Ontology objects. *Codyze-Provenance* is a new addition to the *Codyze* component, which will extract evidence from CI/CD pipelines and provide attestations and provenances for software builds and artefacts. It's currently in its initial design phase, with details yet to be specified.

In the upcoming phases of the project, the *Clouditor-Discovery* and *Codyze-Provenance* components will be further developed and integrated into the EMERALD framework. The next and final iteration of this deliverable will provide the updates on these components in project month 24 (D2.9 [5]).

6 References

[1] EMERALD Consortium, "D2.2 Source Evidence Extractor – v1: Evidence extraction from source code that can be integrated with the certification graph," 2024.

[2] EMERALD Consortium, "D2.4 AMOE – v1: Evidence extraction from policy documents that can be integrated with the certification graph," 2024.

[3] EMERALD Consortium, "D2.6 ML model certification – v1: Security and privacy preserving evidence that can be integrated with the certification graph," 2024.

[4] EMERALD Consortium, "D2.1 Graph Ontology for Evidence Storage: Description of a uniform schema for storing and linking heterogenous data," 2024.

[5] EMERALD Consortium, "D2.9 Runtime evidence extractor – v2: Evidence extraction from runtime data that can be integrated with the certification graph," 2025.

[6] EMERALD Consortium, "D2.10 Certification Graph– v1: Integration of the graph with semantically linked and combined evidence," 2025.

[7] EMERALD Consortium, "D2.11 Certification Graph– v2: Integration of the graph with semantically linked and combined evidence," 2026.

[8] EMERALD Consortium, "D1.1 Data modelling and interaction mechanisms - v1," 2024.

[9] EMERALD Consortium, "EMERALD Glossary," 2024.

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

[11] MEDINA Consortium, "D3.6 Tools and techniques for collecting evidence of technical and organisational measures-v3 (https://medina-project.eu/public-deliverables/)," 2023.

[12] EMERALD Consortium, "D3.1 Evidence Assessment and Certification - Concepts - v1," 2024.

[13] C. Banse, A. Schneider and I. Kunz, "owl2proto: Enabling Semantic Processing in Modern Cloud Micro-Services," *To appear in: 16th International Conference on Knowledge Engineering and Ontology Development.*

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