Deliverable D5.2

MEDINA requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy-v2

Editor(s):	Iñaki Etxaniz, Juncal Alonso
Responsible Partner:	Fundación TECNALIA Research & Innovation (TECNALIA)
Status-Version:	Final – v1.0
Date:	04.11.2022
Distribution level (CO, PU):	PU

Project Number:	952633
Project Title:	MEDINA

Title of Deliverable:	MEDINA requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy – v2
Due Date of Delivery to the EC	31.10.2022

Work package responsible for the Deliverable:	WP5 - MEDINA Framework Integration
Editor(s):	Iñaki Etxaniz, Juncal Alonso (TECNALIA)
Contributor(s):	Bosch, FhG, HPE, XLAB, Fabasoft
Reviewer(s):	Jesús Luna (Bosch), Cristina Martínez (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3, WP4, WP6

Abstract:	This deliverable has a threefold goal. Firstly, it contains the requirements of the MEDINA framework in close collaboration with Task 6.1. Secondly, it describes the MEDINA architecture: its components, workflow, and interfaces. Thirdly, it details the DevOps infrastructure, namely the set of tools and services to support the continuous integration and deployment phases, as well as the CI/CD strategy followed for the integration of the MEDINA framework.
Keyword List:	Architecture, Requirements, Use Cases, DevOps, CI/CD
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	01.08.2022	Initial TOC and assignments	TECNALIA
v0.2	14.09.2022	Section 5: added Section 3: completely refurbished Section 4.1: several FR updated	HPE Bosch, Fabasoft TECNALIA, Bosch, FhG, HPE, XLAB, Fabasoft
v0.3	28.09.2022	Section 4: general updates. Section 5.3: component cards updated	TECNALIA, Bosch, FhG, HPE, XLAB, Fabasoft
v0.4	10.10.2022	Architecture (section 4) completed Section 4.4 Requirement Analysis	TECNALIA, Bosch, FhG, HPE, XLAB, Fabasoft
v0.5	17.10.2022	Architecture diagrams updated Introduction & Conclusions added Section 2 updated Section 4.4: summary part added Executive summary	TECNALIA
v0.6	20.10.2022	Internal review	Jesus Luna (Bosch)
v0.7	02.11.2022	Reviewer comments addressed	TECNALIA, Bosch, FhG, HPE
v0.8	02.11.2022	Final quality review	Cristina Martínez (TECNALIA)
v1.0	04.11.2021	Final version submitted to the European Commission	Cristina Martínez (TECNALIA)

Table of contents

List of tables

List of figures

Terms and abbreviations

AMOE	Assessment and management of organizational evidence
API	Application Programming Interface
AWS	Amazon Web Services
BSI	Bundesamt für Sicherheit in der Informationstechnik
CAB	Conformance Assessment Body
CCD	Company Compliance Dashboard
CI/CD	Continuous Integration / Continuous Delivery
CISO	Chief Information Security Officer
CISQ	Consortium for Information & Software Quality
CloudPG	Cloud Property Graph
CNL	Controlled Natural Language
CORS	Cross-Origin Resource Sharing
CS	Cloud Service
CSA or EU CSA	EU Cybersecurity Act
CSP	Cloud Service Provider
CVE	Common Vulnerabilities and Exposures
CVS	Concurrent Versioning System
CWE	Common Weakness Enumeration
DevOps	Development and Operations
DoA	Description of Action
DSL	Domain Specific Language
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
gRPC	Google Remote Procedure Call
ICI	Internal Control Implementer
ICO	Internal Control Owner
IEC	International Electrotechnical Commission
IoT	Internet of Things
ISO	International Organization for Standardization
IUI	Integrated User Interface
KR	Key Result
NCCA	Non-Conformity Certification Authority
NL2CNL	Natural Language To Controlled Natural Language
KPI	Key Performance Indicator
NLP	Natural Language Processing
OCR	Optical Character Recognition
OS	Open Source
OWASP	Open Web Application Security Project
OWL	Web Ontology Language
PII	Personally Identifiable Information
REST	Representational State Transfer

SaaS	Software as a Service
SATRA	Self-Assessment Tool for Risk Analysis
SDLC	Software Development Life Cycle
SSI	Self-Sovereign Identity
ToC	Target Of Certification
TOM	Technical and Organizational Measure (aka Requirement)
TRL	Technology Readiness Level
UC	Use Case
VAT	Vulnerability Assessment Tools
VDE	Virtual Development Environment
VM	Virtual Machine
XML	Extensible Markup Language

Executive Summary

This deliverable, D5.2, is the second release of the detailed definition of the MEDINA framework, being developed in the scope of Task 5.1 and Task 5.2. It updates and enhances the previous version, D5.1 [1], delivered at M12. The goal of this document is to be a comprehensive report containing the general definition of the MEDINA framework, the development methodology to be followed in the project, and the infrastructure used to construct the solution.

This document follows the same structure as D5.1 and includes part of its content to keep the document self-contained and easier to follow, thus avoiding constant references to the first release. In this regard, the chapter dedicated to the requirement management process is mainly unchanged, as the methodology for requirement elicitation was already defined in the previous version and has not changed. The referred process combines requirements coming top-down from the Use Cases and requirements going bottom-up from the component owners.

The two MEDINA use cases are also briefly presented. These are the up-to-date versions that have been extracted from D6.3 [2]. Additionally, the list of UC requirements is included for completeness.

The bulk of the document is devoted to defining the components of the MEDINA framework. To this end, the functional and non-functional requirements of each component have been updated and extended. A total of 100 requirements are presented (88 functional and 12 non-functional). Of them, 21 requirements have been added since the previous version of this report (D5.1), while other 6 have been discarded. An analysis of the relations and dependencies between requirements, Key Results and architecture components is also presented. In this version of the document, we have included a requirement status dashboard to complete the requirements chapter. The conclusions are that 96% of the requirements have been at least partially implemented, and that most of them (53%) are already fully implemented. Only 4% of the requirements are about to be implemented.

The document also outlines the architecture of MEDINA. The structural and behavioural description of the components conforming the MEDINA framework has been detailed. Each component is described by means of a template called "component card", that includes the main information regarding the component: functionalities, sub-components, interfaces, sequence diagrams, etc. As a result of the architecture definition work, two new components have been defined this year, namely the *Automated Self-Sovereign Identity-based certificates management* (SSI) and the *Integrated User Interface*, giving a total of 15 components. The architectural framework is presented as divided in eight "building blocks" or groups, each with distinct functionality.[1]

Finally, the deliverable describes the infrastructure used to build and demonstrate the MEDINA framework and the Continuous Integration (CI) strategy followed, which includes version control functionalities, regular check-ins, and automated testing. The Continuous Delivery (CD), which automates the release of the app into production without manual intervention, is also presented. The containerized deployment model (based on Docker and virtual machines) used in MEDINA is described. The list of tools selected to implement this whole process has been adjusted in the categories of static code analysis and dynamic code analysis.

[1] MEDINA building blocks are: Catalogue, Certification language, Risk assessment and optimisation framework, Continuous Evaluation and Certification Life-Cycle, Organizational Evidence Gathering and Processing, Orchestrator and Databases, Evidence Collection and Security Assessment, and Integrated UI.

The next steps will be dedicated to finish the implementation of components tackling the pending requirements, and integrating and testing them into the MEDINA infrastructure.

1 Introduction

This section explains the goal and purpose of the deliverable, its context, and its structure.

1.1 About this deliverable

This deliverable is the second release of the WP5 report dedicated to describing the general MEDINA framework. It is based on the first version of deliverable (D5.1 [1]), maintaining the same structure.

WP5 "MEDINA framework Integration" has five deliverables that can be divided in two parallel series:

- Those that define the MEDINA framework in detail (D5.1 [1] and D5.2)
- Those that describe the developed solution (D5.3 [3], D5.4, and D5.5)

The main goal of this document is, therefore, to be a comprehensive document containing the general definition of the MEDINA framework, the development methodology followed in the project, and the infrastructure employed to build the solution. It is the result of tasks T5.1 "Requirements, architecture and Infrastructure Specifications" and T5.2 "Framework CI/CD strategy definition" and contributes to the following WP5 purposes:

- To design the overall architecture of the MEDINA framework
- To provide the basis for the integration of the components (key results KR1 to KR6)
- To define and set up the Continuous Integration and Deployment (CI/CD) strategy of MEDINA

The main portion of the document is devoted to defining the components of the MEDINA framework. For this, the functional and non-functional requirements for each component are provided. Then, the description of the architecture of MEDINA forms the second part of the document, i.e., components and interfaces. The structural and behavioural description of the components conforming the MEDINA framework is detailed by means of a template called "component card", which includes the main information related to the component: functionalities, sub-components, interfaces, sequence diagrams, among others. In addition, the MEDINA data model is described, classifying the different entities into eight groups according to the building block to which they belong.

As mentioned above, this deliverable is the second version of D5.1 [1]. Due to the progress of the project in the 12 months since the first version, the content has been updated and expanded. However, as the aim is to provide a self-contained deliverable that facilitates the reader´s understanding, both documents share content that remains unchanged. Although the changes are pointed out throughout the document, a table of the main changes is provided in Section 1.3.

1.2 Document structure

The rest of the document is structured as follows:

- Section 2 describes the two use cases developed in MEDINA to validate and test the framework prototypes. It explains how they are integrated in the actual systems used by Bosch and Fabasoft, and what the main objective of each use case is.
- Section 3 describes the requirements of the MEDINA framework. Requirements are primarily organized as functional vs non-functional requirements. A further division is made attending the component (aka module) owning the requirement.

- Section 4 presents the architecture of MEDINA. The components conforming the MEDINA framework are detailed, as well as the workflows. It also describes the interfaces implemented between the components, and the data model used.
- Section 5 details the CI/CD strategy followed for the continuous integration of the MEDINA framework. Also, the DevOps infrastructure used in the project is defined and described, including the set of tools and services needed to support the DevOps approach, and the different environments defined to support the development, integration, and deployment phases.
- Section 6 summarizes and briefly comments on the reported results.

The Appendixes sections contain sections of the document that are largely unchanged with respect to the previous version (D5.1):

- Methodology for requirements elicitation in MEDINA (*Appendix A. Requirements Management in MEDINA*)
- Description of the MEDINA use cases (*Appendix B. Use Cases Definition*)
- Full list of MEDINA Framework requirements (*Appendix C. List of Requirements*) and
- CI/CD Strategy to follow for the continuous integration of the MEDINA framework (*Appendix D. CI/CD Strategy*)

1.3 Update from D5.1

For simpler tracking of progress and updates with regards to the previous deliverable version (D5.1), Table 1 shows a brief overview of the changes and additions to each of the document sections.

Table 1. Overview of deliverable updates with respect to D5.1

Section	Change
2	Use Case 1 and Use Case 2 have updated their integration approach, and have included their testbed.
3	This section includes those requirements that have been added, discarded, or modified in their description. The requirements analysis includes updated tables to show the alignment between requirements and KRs, and the status of requirements. A new mapping table between functional requirements and use case requirements has been included. A summary dashboard has also been developed.
4	The MEDINA framework architecture and the Data Model diagrams have been updated to reflect changes in the last release of the components. The Component cards have also been updated. Two new components, named *Automated SSI-based certificates management (SSI)* and *Integrated User Interface (IUI)*, have been added.
5	The implemented CI/CD pipeline is described. The sub-section on the infrastructure for the MEDINA framework has been updated to reflect the actual situation in terms of supporting tools, development, test, and validation environments.
Appendix A	Includes the requirements management methodology, unchanged since v1.
Appendix B	The Use Cases definition remains basically the same. The list of use case requirement corresponds to the latest version of the WP6 documents, extracted from D6.3 [2].
Appendix C	Contains the complete list of requirements. It includes those that remain unchanged since v1 and those that have been discarded, added o modified.
Appendix D	Includes the CI/CD strategy, unchanged since v1.

2 MEDINA Use Cases

This section provides an update related to the implementation of the MEDINA use cases which have been previously described in D5.1 [1]. On one hand we refer to the use case 1 (UC1) provided by the partner Bosch, which focuses on the EUCS certification scheme for high assurance [4], based on a multicloud deployment leveraging three public cloud service providers. On the other hand, the second use case (UC2) is provided by the partner Fabasoft and builds around the real compliance activities regarding the Fabasoft Business Process Cloud.

It is worth to notice that the definition of the use cases has not changed with respect to the description found in D5.1. This part has been moved to *Appendix B. Use Cases Definition*. However, its degree of implementation has progressed due to the ongoing validation activities.

The rest of this section summarizes the implementation approach and associated progress for both UC1 and UC2. More details can be found in D6.3 [2].

2.1 UC1: European Certification of Multi-cloud backends for IoT solutions

2.1.1 Integration Approach

In D5.3 [3] a set of seven generic workflows was introduced, which form the basis for instantiating real-world scenarios with the MEDINA framework. The proposed workflows play two very important roles from a use case validation perspective, by:

- Allowing the project to evaluate if all framework's components are interacting among them, and
- enabling MEDINA's early adopters to use the framework in different cloud security certification contexts.

The implementation approach followed by Bosch instantiates each one of the seven documented generic workflows (see D6.3 [2], Appendix C) in the deployed testbed which is presented below. For the sake of completeness, D6.3 also shows the basic sequence of actions needed to implement the referred workflows in the Bosch testbed[2]. The integration approach of UC1 demonstrates how each workflow is correlated to the user stories from Bosch to indirectly guarantee full coverage of the elicited components' requirements.

As required background for this report, the following section presents the deployed Bosch testbed (more details can be found in D6.3 [2]).

2.1.2 Testbed

For the purposes of the initial validation milestone, Bosch setup a testbed in its corporate Microsoft Azure tenant (*bosch.onmicrosoft.com*), consisting of a Subscription[3] containing the 28 cloud resources including:

- Virtual Machines,
- Network security groups,
- Software defined networks,
- Virtual disks,
- Public IP addresses, and

[2] Detailed explanation of the presented workflow steps can be found in D6.3 [2]
[3] In Microsoft's Azure terminology, a Subscription is a virtual container of cloud resources. From a MEDINA perspective, a Subscription is considered synonym of "Cloud Service" to certify.

- Raw storage (virtualized).

In order to validate the capabilities of the deployed evidence collectors, and the configured technical metrics, some resources in the testbed were misconfigured (e.g., reducing the retention time in some of the virtual storage services, or disabling data-at-rest encryption on the virtual machine's disks). Despite the final MEDINA framework aims to support multiple targets of certification (ToC), the deployed testbed provides only one during this initial implementation effort. In any case, the implemented certification logic allows the MEDINA framework to aggregate the compliance status of the underlying cloud services/resources to manage a unique EUCS certificate for the ToC.

To complement the implementation of the testbed validating the technical metrics, this task also benefited from MEDINA framework's support to the assessment of organizational metrics. For that purpose, and as part of the implementation efforts, the deployed testbed also included a PDF version of Bosch's security concept of its IoT Cloud (BIC). The BIC security concept leverages the ISO/IEC 27001 structure to document cloud service aspects like shared responsibility, access control, asset management, cryptography, operations security, and communication security. Due to copyright and confidentially reasons it is not possible to include a verbatim copy of the BIC security concept in this deliverable, however further details related to the automated assessment of both technical and organizational measures can be found in D6.3 [2].

2.2 UC2: European Cloud Service Provider SaaS public & private cloud

2.2.1 Integration Approach and Expected Benefits (after MEDINA)

Fabasoft expects MEDINA to offer a strong increase in efficiency combined with a significant cost reduction in the long run, especially when it comes to multi-audits for different compliance frameworks. To achieve that and to test the MEDINA functionality, Fabasoft will set up a Demo-System in a Virtual Development Environment (VDE) which is a perfect virtual clone of the Fabasoft Cloud production environment. By doing this, both scenarios can be achieved: the application of MEDINA in a private and a public cloud SaaS solution.[4]

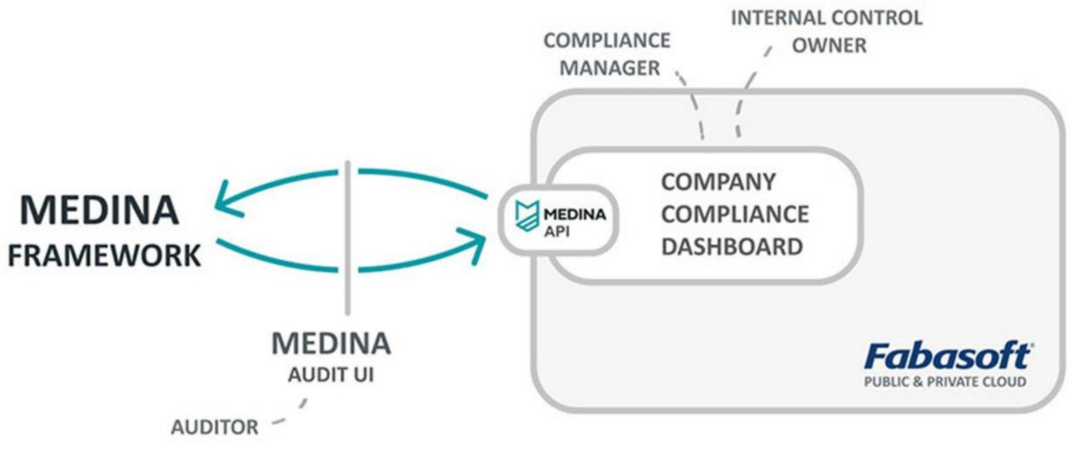


Figure 1. Illustration of UC2 Demo-System

Currently, there is no scenario where a CSP would simply install a MEDINA software framework into its own datacentres and let it "just work". So, to make use of the MEDINA approach, UC2 will tackle it as a framework that can be accessed via an audit API, illustrated as the green lines

[4] The VDE is applicable for the public Fabasoft Cloud and the on-premises installation, the private cloud.

in Figure 1, which will enable a Compliance Manager at Fabasoft to communicate with the several components offered by MEDINA. MEDINA will not have direct access to the Fabasoft Cloud at all levels and each stack, but will "ask" for correct return values according to the configuration of an audit that the Compliance Manager will specify in the Fabasoft Certification App. The Audit API is also the point of connection for the auditor, who will have to validate the Fabasoft implementation for all Security Controls and the different implementations of the configured return values to the MEDINA framework. Figure 2 shows a screenshot of the current development of the UC2 demonstrator system, the CCD. A live demo is available as of October 2022.

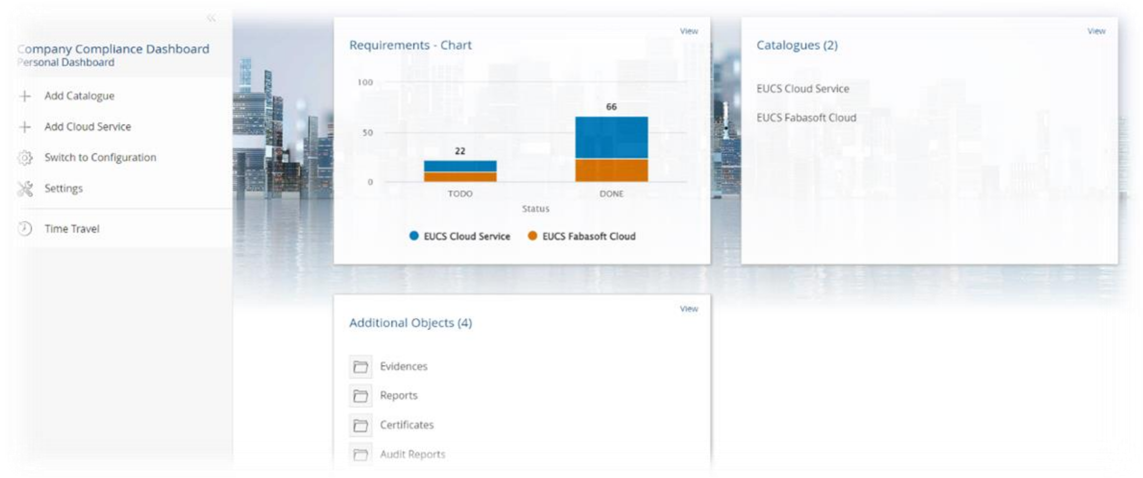


Figure 2. Screenshot of the actual UC2 demo-system implementation

Note that the auditor does not directly interact with the Fabasoft system. We will design it this way, because we expect auditors to use a specific MEDINA UI to fulfil their tasks within an audit process. We strongly believe that if auditors would have to connect to each individual implementation of a MEDINA audit functionality, the number of different systems an auditor has to deal with will create a huge overhead for them and thus is not feasible.

2.2.2 Testbed

For UC2, the testing approach is twofold. The main approach is by utilizing the Company Compliance Dashboard (CCD, see D6.2 [5] and D6.3 [2]). Here it is possible to cover a good set of EUCS requirements – some automatically, some manually – with the actual Fabasoft approach, used for instance in BSI C5 audits. The CCD will communicate via the implemented APIs with MEDINA and transport results and checks in the recommended form of an "Assessment Rule" to the orchestrating component *Clouditor*[5]. The CCD is currently in development by the Fabasoft MEDINA team, please refer to D6.3 [2] for more details.

The second approach is similar to the methodology of UC1: for the purposes of the additional validation milestones, Fabasoft will set up a testbed that hosts a micro-service deployment in OpenStack. This testbed will be able to communicate continuously with *Clouditor* once it addresses OpenStack. If OpenStack is not going to be an option in the scope of MEDINA, Fabasoft will address the issue by deploying a testbed in AWS, which *Clouditor* is already familiar with. The deployed service will be either a document-transformation-service or an OCR-service.

In order to validate the capabilities of the deployed evidence collectors, and the configured technical metrics, some resources in the testbed will be misconfigured (e.g., reducing the

[5] More details can be found in D3.5 [14].

retention time in some of the virtual storage services, or disabling data-at-rest encryption on the virtual machine's disks). Despite the final MEDINA framework aims to support multiple targets of certification (ToC), the deployed testbed provides only one during this initial implementation effort. In any case, the implemented certification logic allows the MEDINA framework to aggregate the compliance status of the underlying cloud services/resources to manage a unique EUCS certificate for the ToC.

To complement the implementation of the testbed validating the technical metrics, this task also benefited from MEDINA framework's support to the assessment of organizational metrics. For that purpose, and as part of the implementation efforts, Fabasoft created a MEDINA example policy document. Further details related to the automated assessment of both technical and organizational measures can be found in D6.3 [2].

3 MEDINA Framework Requirements

The methodology that has been used for the elicitation of requirements in MEDINA is explained in *Appendix A. Requirements Management in MEDINA*. It combines a top-down and a bottom-up approach to implement the requirements gathering process: generic functionalities of the MEDINA framework to offer its value propositions from one side and what Use Cases expect from MEDINA components from the other side.

In MEDINA, requirements are mainly defined to provide an understanding of what will the MEDINA framework will offer, i.e., its functionalities. Requirements are grouped by typology [6]:

- ***Functional requirements*** are presented as lists of features or services that the system has to provide according to the assigned priority. They also describe the behaviour of the system in the face of particular inputs and how it should react in certain situations.
- ***Non-Functional requirements*** represent system-related constraints and properties, such as time constraints, constraints on the development process and on the standards to be adopted. Non-functional requirements may constrain the process, or the elements used to develop the system (e.g., performance, usability).

These requirements cover all the components to be implemented in the context of the MEDINA technical Work Packages, namely WP2, WP3 and WP4.

As this is the second release of the MEDINA requirements, it is an evolution of the first version presented in D5.1 [1]. As a result, most of requirements remain unchanged[6], while some new ones have been added, some have been discarded and some have changed their meaning significatively.

For the sake of brevity, in Section 3.1 we list only the new, modified, and discarded functional requirements. The list of non-functional requirements remains unchanged. The complete list of requirements, also including the unchanged requirements, can be found in *Appendix C. List of Requirements*. More compact views of the list of requirements can also be obtained from the tables presented in the requirements analysis (see Section 3.2).

To make it easier for the reader to understand the changes, Table 2 shows the colour code that has been followed in the requirement tables.

Table 2. Colour coding followed in the requirements tables

A white table means the requirement has not changed.[7] It remains the same as in the previous version of the document.
An orange table means the requirement has significantly changed its definition, which affects the meaning, provides more clarity, or modifies the scope.
A red table means the requirement has been definitively rejected. The reason of the rejection is provided along with the status.
A green table means the requirement is new in this second version, so a new functionality is defined for the component.

[6] "Unchanged" is used here in the sense that the essential meaning of the requirement is unaffected. We do not refer to the status of the requirement, that could have changed with respect to the previous version of the document.

[7] In this section, the unmodified requirements are not shown, so no white table appears. For the full list of requirements, see *Appendix C. List of Requirements.*

The possible status of a requirement is one of this: proposed, discarded, partially implemented, or fully implemented.

3.1 Functional requirements

This section shows the modified, elicited or discarded functional requirements related to the components of the MEDINA framework.

3.1.1 Catalogue of control and metrics

The modified, new, and discarded requirements of the *Catalogue of controls and metrics* component are shown below. The complete list of requirements can be found in *Appendix C. List of Requirements*.

Requirement id	**RCME.02**
Short title	Metrics and TOMs in the repository
Description (*)	The repository should include realizable metrics for at least for the 70% of the TOMs referenced in EUCS-High assurance requiring “continuous (automated)” monitoring
Status	Partially implemented
Priority	Must
Related KR	KR1
Reference	DoA part B Annex 1 page 7 [7]

(*) This requirement has been modified due to the reformulation of KPI 1.1 in July 2022 based on the new scope of the MEDINA technical metrics, which focus on the high-level requirements of the ENISA EUCS Cloud Security Certification Scheme.

Requirement id	**RCME.07**
Short title	Interface to risk assurance
Description	When the certification scheme changes in some way (partial changes, requirements, new versions), the risk assurance component has to be notified, or be able to know that something has changed.
Status	Partially implemented
Priority	Should
Related KR	KR1
Reference	WP2/WP4 Technical discussions. The risk assurance component needs to be aware of the changes on the certification schemes, which are in the Catalogue.

Requirement id	**RCME.08**
Short title	Catalogue GUI
Description	The Catalogue has a GUI to search and show the different content it stores. This GUI is going to be part of the MEDINA Integrated-UI. Enhancements and adaptations due to changes in the data model are foreseen until the final version of the Catalogue.
Status	Partially implemented
Priority	Must
Related KR	KR1
Reference	WP2/WP4 Technical discussions. The introduction of the Integrated UI (IUI) component requires this interaction.

Requirement id	**RCME.09**
Short title	Questionnaire for self-assessment
Description	The Catalogue shall contain a questionnaire that helps a Cloud Service Provider to make a self-assessment of the fulfilment degree of the EUCS standard. This questionnaire will have the following features: 1) Allow the user to select the assurance level for the assessment 2) Provide one or more questions to check the fulfilment of every requirement, of each control in each EUCS category 3) Provide an easy-to-use scale of support for the questions (fully/partially/not supported) 4) Allow the user to enter comments related to a question 5) Allow the user to include textual references to locate the evidence to support the response given to a specific question 6) Provide a dashboard that summarizes the result of the assessment, and provides quantitative values to reflect the degree of fulfilment
Status	Partially implemented
Priority	Could
Related KR	KR1
Reference	WP2/WP3 Technical discussions. In the first year, the questionnaire was worked out in task T3.1 and shaped as an excel file. It will be integrated in the Catalogue v2.

Requirement id	**RCME.10**
Short title	Questionnaire for auditors
Description	The Catalogue shall contain a questionnaire that can be used by an auditor to assist him/her in the audit process. For that purpose, the tool can provide some extra functionalities such as: 1) Allow to enter non-conformities regarding a question 2) Provide a dashboard that summarizes the result of the audit, including the related comments/non-conformities for each requirement, as well as quantitative values to reflect the degree of fulfilment for each control
Status	Partially implemented
Priority	Could
Related KR	KR 1
Reference	WP2/WP3 Technical discussions. In the first year, the questionnaire was worked out in task T3.1 and shaped as an excel file. It will be integrated in the Catalogue v2.

3.1.2 Certification Language

The modified, new, and discarded requirements of the *Certification Language* components are shown below. The complete list of requirements can be found in *Appendix C. List of Requirements*.

3.1.2.1 NL2CNL Translator

Requirement id	**NL2CNL.01**
Short title	Translation from natural language to controlled natural language
Description (*)	The tool shall be able to translate in a semi-automatic way the requirements selected from a security certification scheme – originally

	expressed in natural language (English), into a set of obligations expressed in a controlled natural language. The output of the tool will be checked manually to verify if the obligations generated by the tool are correctly linked to the selected requirement.
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description [7]

(*) The description has been polished and completed. About the input, it now refers to translate “requirements” and not “most relevant aspects” of a security scheme. About the output, it says “into a set of obligations expressed in a CNL” instead of “into a controlled natural language”. It has been added a sentence about the output checking.

Requirement id	**NL2CNL.03**
Short title	Translation of organizational measures and technical measures
Description (*)	NL2CNL translator will be able to translate some of the organizational measures specified in the chosen EU cloud certification scheme, and some of the technical measures.
Status	Partially implemented
Priority	Should
Related KR	KR3
Reference	DoA, KR3 description [7]

(*) The scope has been moderated. It now talks about translate “some”, not “all the organizational measures”.

Requirement id	**NL2CNL.05**
Short title	XML compliant
Description	The controlled natural language output of NL2CNL translator will be compliant with the XML based format supported by the CNL Editor.
Status	**DISCARDED**: duplicates the requirement **NL2CNL.04**
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description [7]

3.1.2.2 CNL Editor

Requirement id	**CNLE.02**
Short title	CNL Editor policies authoring
Description	The CNL Editor will allow creating statements for security controls.
Status	**DISCARDED**: the workflow changed during project discussions respect to the initial idea, as a consequence the CNL Editor must not create Obligations.
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description [7]

3.1.2.3 DSL Mapper

Requirement id	**DSLM.02**
Short title	Mapping elements

Description	The mapping process will consider relevant elements of the target certification framework, including (some) technical and organizational measures, quantitative/qualitative security metrics, complex compliance conditions, and cloud supply chain elements. The mapping process will prioritize the translation of those requirements in CNL that can automatically be enforced by WP4 and that are considered highly relevant by the EU authorities at stage.
Status	**DISCARDED**: Already contained in the rest of requirements
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description [7]

Requirement id	**DSLM.03**
Short title	DSL output compliancy
Description	The tool will output REGO rules, compliant with the input required by the Orchestrator.
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	This requirement was introduced to allow a straightforward interoperability between the DSL mapper and the Orchestrator, which is the main component it will need to interface with.

3.1.3 Risk assessment and optimisation framework

The discarded requirement of the *Risk assessment and optimisation framework* component is shown below. The complete list of requirements can be found in *Appendix C. List of Requirements*.

Requirement id	**RBSCF.04**
Short title	Interface to the auditor
Description	Auditor follows a risk-based approach which provides flexibility to the certification process: since an ever-changing threat landscape often requires timely reaction from the security team provoking changes in the security configurations. These could be efficient from the risk treatment point of view, but will affect the previously obtained certificate, in the worst case, invalidating it.
Status	**DISCARDED**: The component provides the possibility to access the input parameters and results of the assessment to a Compliance Manager (role). An Auditor will have access to the component using the same functionality. In other words, there is no need to develop a separate interface for an auditor, as it will use the same interface that a Compliance Manager uses. In short, the requirement is automatically fulfilled by granting the auditor the rights of the Compliance Manager.
Priority	Must
Related KR	KR6
Reference	DoA. Page 9 [7]

3.1.4 Evidence gathering tools

The modified, new, and discarded requirements of the *Evidence gathering tools* are shown below. The complete list of requirements can be found in *Appendix C. List of Requirements*.

3.1.4.1 *Evidence Orchestrator*

Requirement id	ECO.04
Short title	Transmission of evidence checksums
Description	The evidence orchestrator should integrate a Ledger client that stores checksums of evidence in a DLT.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9 [7]

3.1.4.2 *MEDINA Evidence Trustworthiness Management System*

Requirement id	ETM.06
Short title	Compliance with existing standards
Description	The design and implementation of the DAT should comply with the requirements of existing standards regarding the certification chain (ISO-based approach, ISAE3402 and evidence-based).
Status	**DISCARDED**: Certification standards are not directly applicable to the *MEDINA Evidence Trustworthiness Management System* as it is not involved in the certification process. It is just a component that provides extra security features.
Priority	Should
Related KR	KR5
Reference	DoA part A Annex 1 pages 22 [7]

3.1.4.3 *Technical evidence gathering tools: Clouditor, Codyze/CPG, Automated vulnerability monitoring / detection*

3.1.4.3.1 Common requirements for all the tools

Requirement id	TEGT.C.02
Short title	Provision to defined interfaces
Description (*)	The developed tools must provide collected evidence to the central evidence Orchestrator via its offered APIs.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9 [7]

(*) More specific definition. The destination to which to send the collected evidence has changed from “a security assessment tool” to “the central evidence orchestrator”.

3.1.4.3.2 Specific tool requirements

Gathering evidence from CSP-native services

Requirement id	TEGT.S.09
Short title	Collect evidence from CSP-native services
Description	The developed tool should be able to query findings from CSP-native services, like Azure Policy, to integrate them in MEDINA by querying the respective cloud API.
Status	Proposed
Priority	Could

Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9 [7]

Gathering evidence from application source code

Requirement id	**TEGT.S.10**
Short title	Connect infrastructure- and application-level security analyses
Description	The developed tool should be able to bridge the gap between infrastructure- and application-level security analysis by extending graph-based code analysis to the cloud resources, allowing to identify data flows across cloud resources.
Status	Fully implemented
Priority	Could
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9 [7]

3.1.4.4 Organizational evidence gathering tools: AMOE

Requirement id	**OEGM.05**
Short title	Evidence Assessment results
Description	The assessment results of evidence assessments must be submitted to the evidence orchestrator via the API it provides.
Status	Fully implemented
Priority	Must
Related KR	DoA KR4 [7]

3.1.5 Evidence Assessment tool

The modified, new, and discarded requirements of the *Evidence Assessment tool* are shown below. The complete list of requirements can be found in *Appendix C. List of Requirements*.

Requirement id	**EAT.04**
Short title	Assess CSP-native evidence
Description	The developed tool should be able to assess the CSP-native evidence or translate CSP-native assessment results to the MEDINA data model.
Status	Proposed
Priority	Could
Related KR	KR5
Reference	DoA part A Annex 1 pages 8-9 [7]

3.1.6 Continuous Evaluation and Certification Life-Cycle

The modified, new, and discarded requirements of the *Continuous Evaluation and Certification Life-Cycle* components are shown below. The complete list of requirements can be found in *Appendix C. List of Requirements*.

3.1.6.1 Automation of the Cloud Security Certification Life-Cycle

Requirement id	**ACLM.08**
Short title	Secure lifecycle management
Description	The lifecycle management component can be implemented in a smart contract to ensure a tamper-proof execution.

Status	**DISCARDED**: based on the evaluation of smart contracts for the automatic management of certificates[8], it was considered that they introduce too many risks compared to the potential benefits.
Priority	Could
Related KR	KR6
Reference	DoA part A Annex 1 pages 9 [7]

3.1.6.2 SSI Framework

Requirement id	**SSI.01**
Short title	Cloud security certificate issuance
Description	The system should provide a way for appropriate entities (CAB) to issue and sign security certifications for the cloud providers as indicated by the automated certificate Life-Cycle Manager.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	D4.2 [8] & D5.4 [9]

Requirement id	**SSI.02**
Short title	Cloud security certificate update
Description	The system should provide a way for appropriate entities (CAB) to update security certifications for the cloud providers as indicated by the Life-Cycle Manager.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	D4.2 [8] & D5.4 [9]

Requirement id	**SSI.03**
Short title	Cloud security certificate revocation
Description	The system should provide a way for appropriate entities (CAB) to revoke security certifications for the cloud providers as indicated by the Life-Cycle Manager.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	D4.2 [8] & D5.4 [9]

Requirement id	**SSI.04**
Short title	Cloud security certificates listing
Description	The system must list the historical cloud security certificates issued, updated, and revoked.
Status	Fully implemented
Priority	Must
Related KR	KR6
Reference	D4.2 [8] & D5.4 [9]

[8] For more details about the study, see deliverable D4.2 [10]

Requirement id	SSI.05
Short title	Cloud security certificate verifiable public proofs generation
Description	The system must generate verifiable proofs of the security certificate state on request.
Status	Fully implemented
Priority	Must
Related KR	KR6
Reference	D4.2 [8] & D5.4 [9]

Requirement id	SSI.06
Short title	Cloud security certificate confidential proofs generation
Description	The system should generate verifiable confidential proofs of the security certificate private parameters on request.
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	D4.2 [8] & D5.4 [9]

Requirement id	SSI.07
Short title	Cloud security certificate proofs request and verification
Description	The system should provide a way for appropriate entities (potential clients) to request and verify proofs of the security certificates to the cloud service providers.
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	D4.2 [8] & D5.4 [9]

3.1.7 Integrated User Interface

The modified, new, and discarded requirements of the *Integrated User Interface* component are shown below. The complete list of requirements can be found in *Appendix C. List of Requirements*.

Requirement id	IUI.01
Short title	Authentication integration via Keycloak Adapter
Description	Every component must implement an adapter that allows it to authenticate with the Catalogue's Keycloak authentication service in order to prevent unauthenticated users to access its resources.
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

Requirement id	IUI.02
Short title	Authorization integration via Keycloak
Description	Every component that has resources that should only be accessed by specific user roles must enforce authorization on its internal logic (e.g., in a REST API, define at controller level that a specific endpoint can be accessed only with the Product Engineer role). This can be obtained by

Requirement id	**IUI.02**
	defining appropriate configuration on the Catalogue's Keycloak (Role Mapping).
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	WP5/WP6 Technical discussions

Requirement id	**IUI.03**
Short title	Allow frame embedding into Integrated UI
Description	Every component UI that needs to be embedded in an iframe inside the Integrated UI must define a header "X-Frame-Options: ALLOW-FROM integrated-ui-url" in order to allow it.
Status	**DISCARDED**: we are currently sticking to the micro frontend strategy with iframes only.
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

Requirement id	**IUI.04**
Short title	Allow CORS for Integrated UI
Description	Every component backend that needs to be programmatically REST called via Integrated UI frontend must define a header "Access-Control-Allow-Origin: <integrated-ui-url>" in order to allow it.
Status	**DISCARDED**: At the moment, no REST API integration with the IUI is planned. With this approach CORS is not needed.
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

Requirement id	**IUI.05**
Short title	External Identity Provider Configuration
Description	Users should be able to authenticate using their existing enterprise identity provider once it has been configured to do so. Ideally, MEDINA Generic Roles should be inherited from existing claims / roles.
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	WP5/WP6 Technical discussions

Requirement id	**IUI.06**
Short title	Homogeneous look and feel
Description	Each component micro-frontend embedded into IUI should abide to a set of graphical constraints and rules that the MEDINA consortium agreed on in order to homogenize look and feel.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

3.2 Analysis of Requirements

This section provides an analysis of the requirements of the MEDINA framework. It includes three main tables: a) a table showing the current alignment between requirements and KRs (see Section 3.2.1); b) a table showing the mapping between functional requirements and UC requirements (see Section 3.2.2); and c) and a table showing the status of requirements (see Section 3.2.3). Finally, a summary dashboard reflecting the overall status and progress of the requirements is included (see Section 3.2.4).

3.2.1 Mapping of Requirements to KRs

Table 3 shows the mapping between the functional requirements and Key Results (KR1-KR6). Please note that the whole list can be consulted in *Appendix C. List of Requirements*

The Key Results of the MEDINA project, defined in the DoA [7], are the following:

- KR1: Repository of metrics and measures
- KR2: Risk-based selection of controls to reach the certification assurance
- KR3: Certification language
- KR4: Continuous evidence management tools
- KR5: Cloud certificate Evaluator
- KR6: Risk-based Auditor Tool
- KR7: Use Cases
- KR8: Standardization roadmap
- KR9: Training and awareness activities

For each row in Table 3, a 'X' in a cell specifies the Key Result to which that requirement refers. A requirement can refer to several KRs, although this is not a common case. As a result of this alignment, it can be concluded that all elicited functional requirements are related to at least one specific KR.

This the colour code that has been followed in Table 3:

- 🟩 Green new requirement with respect to the previous version of this deliverable (D5.1 [1])
- 🟨 Yellow the scope or functionality of the requirement has changed significantly
- 🟥 Red the requirement has been discarded
- ⬜ White the requirement remains unchanged.

Table 3. Functional requirements and KRs alignment

#	Req. ID	Description	KR1	KR2	KR3	KR4	KR5	KR6
1	RCME.01	Catalogue of metrics, controls and TOMs	X					
2	🟨 RCME.02	Metrics and TOMs in the repository	X					
3	RCME.03	Metrics and TOMs for different assurance levels	X					
4	RCME.04	Technology agnostic security controls	X					
5	RCME.05	Interfaces to the continuous auditing tools	X					
6	RCME.06	Homogenization of the certification schemes	X					
7	🟩 RCME.07	Interface to risk assurance	X					
8	🟩 RCME.08	Catalogue GUI	X					
9	🟩 RCME.09	Questionnaire for self-assessment	X					
10	🟩 RCME.10	Questionnaire for auditors	X					
11	🟨 NL2CNL.01	Translation from NL to controlled NL			X			
12	NL2CNL.02	Based on NLP and ontologies			X			
13	🟨 NL2CNL.03	Translation of org. and technical measures			X			
14	NL2CNL.04	Compliant with the CNL editor language			X			

#	Req. ID	Description	KR1	KR2	KR3	KR4	KR5	KR6
	NL2CNL.05	XML compliant			X			
15	CNLE.01	CNL Editor GUI			X			
	CNLE.02	CNL Editor policies authoring			X			
16	CNLE.03	CNL Editor input format			X			
17	CNLE.04	CNL Editor policies changing			X			
18	CNLE.05	CNL Editor vocabulary			X			
19	CNLE.06	CNL Editor output format			X			
20	DSLM.01	Translation to selected DSLs			X			
	DSLM.02	Mapping elements			X			
21	DSLM.03	DSL output compliancy			X			
22	RBSCF.01	Risk assessment tool		X				
23	RBSCF.02	Risk assessment tool and TOMs		X				
24	RBSCF.03	Implementation selection functionality		X				
	RBSCF.04	Interface to the auditor		X				X
25	ECO.01	Provision of Interfaces				X		
26	ECO.02	Conformity to selected assurance level				X		
27	ECO.03	Secure Transmission to evidence storage				X		
28	ECO.04	Transmission of evidence checksums				X		
29	ETM.01	Trustworthiness of evidence				X		
30	ETM.02	Transmission of evidence checksums				X		
31	ETM.03	Trustworthiness guaranteeing capabilities				X		
32	ETM.04	Tamper-Resistance for evidence				X		
33	ETM.05	Tamper-Resistance for audit information				X	X	
	ETM.06	Compliance with existing standards				X	X	
34	TEGT.C.01	Continuous collection				X		
35	TEGT.C.02	Provision to defined interfaces				X		
36	TEGT.S.01	Collect evidence from cloud interfaces				X		
37	TEGT.S.02	Collect evidence from source code via CPG				X		
38	TEGT.S.03	Implement information and data flow analysis				X		
39	TEGT.S.04	Support expression of security requirements				X		
40	TEGT.S.05	Verify security requirements				X		
41	TEGT.S.06	Retrieve source code of cloud applications				X		
42	TEGT.S.07	Support for common programming languages, libraries, CS				X		
43	TEGT.S.08	Provision of malware, intrusion & vulnerability detection tools				X		
44	TEGT.S.09	Collect evidence from CSP-native services				X		
45	TEGT.S.10	Connect infrastructure- and application-level security analyses				X		
46	OEGM.01	Continuous collection of organizational evidence				X		
47	OEGM.02	Provision to defined interfaces				X		
48	OEGM.03	Usability for auditors				X		
49	OEGM.04	Minimum evidence storage				X		
50	OEGM.05	Evidence Assessment results				X		
51	EAT.01	Evidence assessment target				X		
52	EAT.02	Continuous evidence assessment				X		
53	EAT.03	Evidence assessment results				X		
54	EAT.04	Assess CSP-native evidence				X		
55	CCCE.01	Continuous Evaluation of Assessment Results					X	
56	CCCE.02	Evaluate the fulfilment degree per TOM					X	
57	CCCE.03	Configuration of needed metrics for requirements					X	
58	CCCE.04	Fulfilment degree per control, group & entire certification					X	
59	CCCE.05	Temporal fulfilment degree per TOM					X	
60	CCCE.06	Evaluate the time-to-fix indicator per TOM					X	
61	CCCE.07	APIs of the Continuous Certification Evaluation Component					X	
62	ACLM.01	Cloud security certification issuance					X	
63	ACLM.02	Automatic cloud security certification update					X	

#	Req. ID	Description	KR1	KR2	KR3	KR4	KR5	KR6
64	ACLM.03	Cloud security certification revocation					X	
65	ACLM.04	Continuous update of the certificate state					X	
66	ACLM.06	Compliance with EUCS assurance levels and certificate states					X	
67	ACLM.07	Interface for a public registry					X	
	ACLM.08	Secure lifecycle management (smart contract)					X	
68	SSI.01	Cloud security certificate issuance					X	
69	SSI.02	Cloud security certificate update					X	
70	SSI.03	Cloud security certificate revocation					X	
71	SSI.04	Cloud security certificates listing					X	
72	SSI.05	Cloud security certificate verifiable public proofs generation					X	
73	SSI.06	Cloud security certificate confidential proofs generation					X	
74	SSI.07	Cloud security certificate proofs request and verification					X	
75	RBCA.01	Dynamic risk assessment						X
76	RBCA.02	Interface to the continuous evidence management tools						X
77	IUI.01	Authentication integration via Keycloak Adapter						X
78	IUI.02	Authorization integration via Keycloak						X
	IUI.03	Allow frame embedding into Integrated UI						X
	IUI.04	Allow CORS for Integrated UI						X
79	IUI.05	External Identity Provider Configuration						X
80	IUI.06	Homogeneous look and feel						X

3.2.2 Mapping of WP5 requirements to WP6 requirements

This section shows the alignment between the functional requirements related to the MEDINA components, developed in WP5, and the Use Cases requirements (also known as “user stories”) gathered in WP6. Please note that the detailed list of WP5 and WP6 requirements can be consulted in *Appendix C. List of Requirements* and *Appendix B. Use Cases Definition*, respectively.

Table 4 shows for each functional requirement (row), which user stories will be used to test it. This mapping shows that each of the requirements defined in WP5 is related to one or more user stories. This way, a final user can check the module responsible for implementing a requirement and track the validation and coverage of the requirements along the time. The requirements in red colour are those that have been discarded.

Table 4 serves to align both the bottom-up perspective and the top-down approach followed in MEDINA for the elicitation of the requirements (see *Appendix A. Requirements Management in MEDINA*). As a result of this alignment, it can be concluded that every elicited functional requirement is related to at least one specific UC requirement.

Table 4. Mapping of Functional requirements to UC requirements

#	Req. ID	UC00 requirements	UC01 requirements	UC02 requirements
1	**RCME.01**	**UC00**: 02, 09, 17, 19, 27	**UC01**: 18, 31	
2	**RCME.02**	**UC00**: 02, 27	**UC01**: 18	
3	**RCME.03**	**UC00**: 02, 17		
4	**RCME.04**	**UC00**: 02		
5	**RCME.05**	**UC00**: 02		
6	**RCME.06**	**UC00**: 02, 19	**UC01**: 26	
7	**RCME.07**	**UC00**: 02		
8	**RCME.08**	**UC00**: 29	**UC01**: 06	**UC02**: 10
9	**RCME.09**	**UC00**: 02, 17, 19, 27	**UC01**: 18, 31	
10	**RCME.10**	**UC00**: 02, 09, 17, 19, 27	**UC01**: 18, 31	

#	Req. ID	UC00 requirements	UC01 requirements	UC02 requirements
11	**NL2CNL.01**	**UC00**: 01, 06, 09, 17, 18, 19, 21, 24, 27, 29	**UC01**: 01, 02, 03, 04, 05, 06, 07, 08, 09, 13, 14, 15, 16, 19, 20, 21, 23, 24, 28, 30, 31	**UC02**: 01, 10
12	**NL2CNL.02**	**UC00**: 01, 06, 09, 17, 18, 19, 21, 24, 27, 29	**UC01**: 01, 02, 03, 04, 05, 06, 07, 08, 09, 13, 14, 15, 16, 19, 20, 21, 23, 24, 28, 30, 31	**UC02**: 01, 10
13	**NL2CNL.03**	**UC00**: 01, 06, 09, 17, 18, 19, 21, 24, 27, 29	**UC01**: 01, 02, 03, 04, 05, 06, 07, 08, 09, 13, 14, 15, 16, 19, 20, 21, 23, 24, 28, 30, 31	**UC02**: 01, 10
14	**NL2CNL.04**	**UC00**: 01, 06, 09, 17, 18, 19, 21, 24, 27, 29	**UC01**: 01, 02, 03, 04, 05, 06, 07, 08, 09, 13, 14, 15, 16, 19, 20, 21, 23, 24, 28, 30, 31	**UC02**: 01, 10
	NL2CNL.05			
15	**CNLE.01**		**UC01**: 02, 03, 04, 06	**UC02**: 03
	CNLE.02			
16	**CNLE.03**		**UC01**: 02, 03, 04	
17	**CNLE.04**	**UC00**: 07	**UC01**: 02, 03, 04	
18	**CNLE.05**		**UC01**: 02, 03, 04, 22	**UC02**: 03, 09, 14
19	**CNLE.06**	**UC00**: 01	**UC01**: 02, 03, 04,	
20	**DSLM.01**	**UC00**: 01, 06, 09, 17, 18, 19, 21	**UC01**: 01, 02, 03, 04, 05, 06, 07, 08, 09, 13, 14, 15, 16, 19, 20, 21, 23, 24, 28, 30, 31	**UC02**: 01, 10
	DSLM.02			
21	**DSLM.03**	**UC00**: 01, 06, 09, 17, 18, 19, 21, 24, 29	**UC01**: 02, 04, 09, 23, 28, 30, 01, 03, 05, 06, 07, 08, 13, 14, 15, 16, 19, 20, 21, 24, 31	**UC02**: 01, 10
22	**RBSCF.01**	**UC00**: 19	**UC01**: 02	**UC02**: 05
23	**RBSCF.02**		**UC01**: 02	
24	**RBSCF.03**		**UC01**: 02	
	RBSCF.04			
25	**ECO.01**	**UC00**: 01, 02, 04, 07	**UC01**: 02, 04, 09, 18, 22, 23, 25, 28, 30, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
26	**ECO.02**	**UC00**: 01, 07	**UC01**: 02, 04, 09, 18, 20, 22, 23, 25, 28, 30, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
27	**ECO.03**	**UC00**: 07, 04	**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
28	**ECO.04**	**UC00**: 07, 04	**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
29	**ETM.01**	**UC00**: 31		
30	**ETM.02**	**UC00**: 31		
31	**ETM.03**	**UC00**: 31	**UC01**: 02, 04, 09	
32	**ETM.04**	**UC00**: 31	**UC01**: 02, 04, 09	
33	**ETM.05**	**UC00**: 31	**UC01**: 02, 04, 09	
	ETM.06			
34	**TEGT.C.01**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10,	

#	Req. ID	UC00 requirements	UC01 requirements	UC02 requirements
			11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
35	**TEGT.C.02**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
36	**TEGT.S.01**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
37	**TEGT.S.02**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
38	**TEGT.S.03**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
39	**TEGT.S.04**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
40	**TEGT.S.05**		**UC01**: 02, 23, 31	
41	**TEGT.S.06**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
42	**TEGT.S.07**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
43	**TEGT.S.08**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24	
44	**TEGT.S.09**		**UC01**: 29	
45	**TEGT.S.10**		**UC01**: 02, 04, 09, 18, 23, 25, 28, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
46	**OEGM.01**	**UC00**: 01, 05, 13, 18, 21, 22, 24, 25, 26, 27, 29	**UC01**: 02, 04, 09, 28, 30, 01, 05, 06, 08, 14, 16, 17, 20, 24, 26, 27, 31	**UC02**:14, 01
47	**OEGM.02**	**UC00**: 01, 05, 13, 18, 21, 22, 24, 25, 26, 27, 29	**UC01**: 02, 04, 09, 28, 30, 01, 05, 06, 08, 14, 16, 17, 20, 24, 26, 27, 31	**UC02**:14, 01
48	**OEGM.03**	**UC00**: 01, 05, 13, 18, 21, 22, 24, 25, 26, 27, 29	**UC01**: 02, 04, 09, 28, 30, 01, 05, 06, 08, 14, 16, 17, 20, 24, 26, 27, 31	**UC02**:14, 01
49	**OEGM.04**	**UC00**: 01, 05, 13, 18, 21, 22, 24, 25, 26, 27, 29	**UC01**: 02, 04, 09, 28, 30, 01, 05, 06, 08, 14, 16, 17, 20, 24, 26, 27, 31	**UC02**:14, 01
50	**OEGM.05**	**UC00**: 01, 05, 13, 18, 21, 22, 24, 25, 26, 27, 29	**UC01**: 02, 04, 09, 28, 30, 01, 05, 06, 08, 14, 16, 17, 20, 24, 26, 27, 31	**UC02**:14, 01

#	Req. ID	UC00 requirements	UC01 requirements	UC02 requirements
51	**EAT.01**	**UC00**: 07	**UC01**: 02, 04, 09, 18, 23, 25, 28, 30, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
52	**EAT.02**	**UC00**: 07	**UC01**: 02, 04, 09, 18, 23, 25, 28, 30, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
53	**EAT.03**	**UC00**: 07	**UC01**: 02, 04, 09, 18, 23, 25, 28, 30, 01, 03, 05, 06, 07, 08, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 24, 26, 31	
54	**EAT.04**		**UC01**: 29	
55	**CCCE.01**	**UC00**: 13, 14, 19, 28	**UC01**: 04, 03	**UC02**: 14, 07, 09
56	**CCCE.02**	**UC00**: 13, 14, 19, 28	**UC01**: 04, 03	**UC02**: 14, 07, 09
57	**CCCE.03**	**UC00**: 13, 14, 19, 28	**UC01**: 04, 03	**UC02**: 14, 07, 09
58	**CCCE.04**		**UC01**: 04, 03	
59	**CCCE.05**		**UC01**: 04, 03	
60	**CCCE.06**		**UC01**: 04, 03	
61	**CCCE.07**	**UC00**: 13, 14, 28		**UC02**: 14, 09
62	**ACLM.01**		**UC01**: 09, 23, 28, 12, 14, 20, 21, 24, 26	
63	**ACLM.02**		**UC01**: 09, 23, 28, 12, 14, 20, 21, 24, 26	
64	**ACLM.03**		**UC01**: 09, 23, 28, 12, 14, 20, 21, 24, 26	
65	**ACLM.04**		**UC01**: 09, 23, 28, 12, 14, 20, 21, 24, 26	**UC02**:09
66	**ACLM.06**	**UC00**: 17	**UC01**: 09, 23, 28, 12, 14, 20, 21, 24, 26	**UC02**: 07, 08
67	**ACLM.07**		**UC01**: 09, 23, 28, 12, 14, 20, 21, 24, 26	
	ACLM.08			
68	**SSI.01**	**UC00**: 20, 24, 26, 28	**UC01**: 06, 09, 14, 20, 21	
69	**SSI.02**	**UC00**: 20, 24, 26, 28	**UC01**: 06, 09, 14, 20, 21	
70	**SSI.03**	**UC00**: 20, 24, 26, 28	**UC01**: 06, 09, 14, 20, 21	
71	**SSI.04**	**UC00**: 20, 24, 26, 28	**UC01**: 06, 09, 14, 20, 21	
72	**SSI.05**	**UC00**: 20, 24, 26, 28	**UC01**: 06, 09, 14, 20, 21	
73	**SSI.06**	**UC00**: 20, 24, 26, 28	**UC01**: 06, 09, 14, 20, 21	
74	**SSI.07**	**UC00**: 20, 24, 26, 28	**UC01**: 06, 09, 14, 20, 21	
75	**RBCA.01**	**UC00**: 19	**UC01**: 02	**UC02**: 04
76	**RBCA.02**		**UC01**: 02	
77	**IUI.01**	**UC00**: 29	**UC01**: 06	**UC02**: 10
78	**IUI.02**	**UC00**: 29	**UC01**: 06	**UC02**: 10
	IUI.03			
	IUI.04			
79	**IUI.05**	**UC00**: 29	**UC01**: 06	**UC02**: 10
80	**IUI.06**	**UC00**: 29	**UC01**: 06	**UC02**: 10

3.2.3 Prioritization and status of requirements

This section provides an overview of the status of the functional and non-functional requirements elicited during the first two years of the project. For each requirement we indicate its priority, the status of the current implementation (month 24), and the expected implementation status in month 33. This information will change as the project progresses and is foreseen that more and more requirements will be implemented until the end of the project in month 36.

Regarding the implementation status of a requirement, we distinguish in the degree of fulfilment among "**Partially** implemented" (P) or "**Fully** implemented" (Fully). In the context of WP5, "**Fully implemented"** means that no more development or test is needed to implement the requirement. In other words, it is ready to be validated by the users in WP6. This status is different from the concept of "**Done**", which is used in WP6 and refers to the mentioned validation task.

Columns M15, M24 and M33 in Table 5 refer to the month where the status of the requirements has been measured: **M15** corresponds to the first version of the requirements, i.e. the requirements elicited in D5.1 [1]; **M24** refers to the current status, i.e. the requirements listed in *Appendix C. List of Requirements*; and **M33** is the foreseen status of the requirements in month 33, previous to the second integration of the MEDINA Framework.

The colour is also used to easily view the status of a requirement: green means "Fully implemented"; orange means "Partially implemented"; and blank means "not started" (-).

Table 5. Requirement prioritization and status

KR	Req. Id	Short title	Priority	M15	M24	M33
KR1 Repository of metrics and measures	RCME.01	Catalogue of metrics, controls and TOMs	MUST	P	Fully	Fully
	RCME.02	Metrics and TOMs in the repository	MUST	P	P	Fully
	RCME.03	Metrics and TOMs for different assurance levels	MUST	P	Fully	Fully
	RCME.04	Technology agnostic security controls	MUST	-	Fully	Fully
	RCME.05	Interfaces to the continuous auditing tools	MUST	P	Fully	Fully
	RCME.06	Homogenization of the certification schemes	MUST	-	P	P
	RCME.07	Interface to risk assurance	Should		-	Fully
	RCME.08	Catalogue GUI	MUST		P	Fully
	RCME.09	Questionnaire for self-assessment	Could		P	Fully
	RCME.10	Questionnaire for auditors	Could		P	Fully
KR3 Certification Language	NL2CNL.01	Translation from NL to controlled NL	MUST	P	P	Fully
	NL2CNL.02	Based on NLP and ontologies	MUST	-	P	Fully
	NL2CNL.03	Translation of org. and technical measures	Should	P	P	Fully
	NL2CNL.04	Compliant with the CNL editor language	MUST	P	P	Fully
	NL2CNL.05	XML compliant				
	CNLE.01	CNL Editor GUI	MUST	-	Fully	Fully
	CNLE.02	CNL Editor policies authoring	MUST			
	CNLE.03	CNL Editor input format	MUST	-	Fully	Fully
	CNLE.04	CNL Editor policies changing	MUST	P	Fully	Fully
	CNLE.05	CNL Editor vocabulary	MUST	-	P	Fully
	CNLE.06	CNL Editor output format	MUST	P	Fully	Fully
	DSLM.01	Translation to selected DSLs	MUST	-	P	Fully
	DSLM.02	Mapping elements				
	DSLM.03	DSL output compliancy	MUST		P	Fully
	RBSCF.01	Risk assessment tool	MUST	-	P	Fully

KR	Req. Id	Short title	Priority	M15	M24	M33
KR2 Risk based selection of controls	RBSCF.02	Risk assessment tool and TOMs	MUST	-	Fully	Fully
	RBSCF.03	Implementation selection functionality	MUST	-	P	Fully
	RBSCF.04	Interface to the auditor				
KR4 Continuous evidence manageme nt tools	ECO.01	Provision of Interfaces	MUST	-	Fully	Fully
	ECO.02	Conformity to selected assurance level	MUST	P	P	Fully
	ECO.03	Secure Transmission to evidence storage	MUST	-	Fully	Fully
	ECO.04	Transmission of evidence checksums	MUST	-	Fully	Fully
	ETM.01	Trustworthiness of evidence	MUST	P	Fully	Fully
	ETM.02	Transmission of evidence checksums	Should	P	Fully	Fully
	ETM.03	Trustworthiness guaranteeing capabilities	MUST	P	P	Fully
	ETM.04	Tamper-Resistance for evidence	MUST	P	Fully	Fully
	ETM.05	Tamper-Resistance for audit information	MUST	P	Fully	Fully
	ETM.06	Compliance with existing standards				
	TEGT.C.01	Continuous collection	MUST	P	P	Fully
	TEGT.C.02	Provision to defined interfaces	MUST	Fully	Fully	Fully
	TEGT.S.01	Collect evidence from cloud interfaces	MUST	P	Fully	Fully
	TEGT.S.02	Collect evidence from source code via CPG	MUST	-	Fully	Fully
	TEGT.S.03	Implement information and data flow analysis	MUST	P	Fully	Fully
	TEGT.S.04	Support expression of security requirements	MUST	P	P	Fully
	TEGT.S.05	Verify security requirements	MUST	P	P	Fully
	TEGT.S.06	Retrieve source code of cloud applications	Should	-	P	Fully
	TEGT.S.07	Support for common programming languages, libraries, CS	Should	P	P	Fully
	TEGT.S.08	Provision of malware, intrusion & vulnerability detection tools	MUST	-	P	Fully
	TEGT.S.09	Collect evidence from CSP-native services	Could	-	-	Fully
	TEGT.S.10	Connect infrastructure- and application-level security analyses	Could	-	Fully	Fully
	OEGM.01	Continuous collection of organizational evidence	MUST	-	Fully	Fully
	OEGM.02	Provision to defined interfaces	MUST	-	Fully	Fully
	OEGM.03	Usability for auditors	Should	-	Fully	Fully
	OEGM.04	Minimum evidence storage	MUST	-	Fully	Fully
	OEGM.05	Evidence Assessment results	MUST		Fully	Fully
KR5 Continuous certification evaluation	EAT.01	Evidence assessment target	MUST	-	Fully	Fully
	EAT.02	Continuous evidence assessment	MUST	P	Fully	Fully
	EAT.03	Evidence assessment results	MUST	P	Fully	Fully
	EAT.04	Assess CSP-native evidence	Could	-	-	Fully
	CCCE.01	Continuous Evaluation of Assessment Results	MUST	P	Fully	Fully
	CCCE.02	Evaluate the fulfilment degree per TOM	MUST	-	Fully	Fully
	CCCE.03	Configuration of needed metrics for requirements	MUST	P	Fully	Fully
	CCCE.04	Fulfilment degree per control, group & entire certification	MUST	-	Fully	Fully
	CCCE.05	Temporal fulfilment degree per TOM	Should	P	Fully	Fully
	CCCE.06	Evaluate the time-to-fix indicator per TOM	Should	-	Fully	Fully

KR	Req. Id	Short title	Priority	M15	M24	M33
	CCCE.07	APIs of the Continuous Certification Evaluation Component	MUST	P	Fully	Fully
	ACLM.01	Cloud security certification issuance	MUST	-	P	Fully
	ACLM.02	Automatic cloud security certification update	MUST	-	P	Fully
	ACLM.03	Cloud security certification revocation	MUST	-	P	Fully
	ACLM.04	Continuous update of the certificate state	MUST	P	P	Fully
	ACLM.06	Compliance with EUCS assurance levels and certificate states	MUST	P	P	Fully
	ACLM.07	Interface for a public registry	MUST	P	P	Fully
	ACLM.08	Secure lifecycle management (smart contract)				
	SSI.01	Cloud security certificate issuance	Should		P	Fully
	SSI.02	Cloud security certificate update	Should		P	Fully
	SSI.03	Cloud security certificate revocation	Should		P	Fully
	SSI.04	Cloud security certificates listing	MUST		Fully	Fully
	SSI.05	Cloud security certificate verifiable public proofs generation	MUST		Fully	Fully
	SSI.06	Cloud security certificate confidential proofs generation	Should		Fully	Fully
	SSI.07	Cloud security certificate proofs request and verification	Should		Fully	Fully
KR6 Risk-Based auditor tool	RBCA.01	Dynamic risk assessment	MUST	-	P	Fully
	RBCA.02	Interface to the continuous evidence management tools	MUST	-	P	Fully
	IUI.01	Authentication integration via Keycloak Adapter	Should	-	Fully	Fully
	IUI.02	Authorization integration via Keycloak	Should	-	P	Fully
	IUI.03	Allow frame embedding into Integrated UI				
	IUI.04	Allow CORS for Integrated UI				
	IUI.05	External Identity Provider Configuration	Should	-	P	Fully
	IUI.06	Homogeneous look and feel	Should	-	P	Fully
Non-Functional Req.	CICD.01	Code repository	MUST	Fully	Fully	Fully
	CICD.02	Automate software build	MUST	P	Fully	Fully
	CICD.03	Automate test suite	Should	-	Fully	Fully
	CICD.04	Software bugs tracking	Should	-	Fully	Fully
	CICD.05	Deploy automation	Should	P	Fully	Fully
	CICD.06	Free tools	MUST	P	Fully	Fully
	CICD.07	Commercially friendliness tools	Should	P	P	P
	CICD.08	Java support	MUST	Fully	Fully	Fully
	CICD.09	Python support	MUST	Fully	Fully	Fully
	CICD.10	C language support	MUST	Fully	Fully	Fully
	CICD.11	GO Lang support	MUST	Fully	Fully	Fully
	CICD.12	JavaScript support	MUST	Fully	Fully	Fully

3.2.4 Requirements Summary Dashboard

Table 6 summarizes how the functional requirements are distributed among the MEDINA components and Figure 3 shows the same data in visual form.

In total, 88 functional requirements have been worked out up to M24, with 8 of them being discarded. We can see that KR4 (Continuous evidence management tools) and KR5 (Continuous certification evaluator) have defined the most requirements, which is logical because they comprise five and four tools respectively.

Table 6. Summary table of requirements status at M24 (by KR)

KR	Discarded	Not started	Partially Implemented	Fully implemented	TOTAL
KR1	0	1	5	4	**10**
KR2	1	0	2	1	**4**
KR3	3	0	7	4	**14**
KR4	1	1	8	17	**27**
KR5	1	1	9	14	**25**
KR6	2	0	5	1	**8**
TOTAL	**8**	**3**	**36**	**41**	**88**

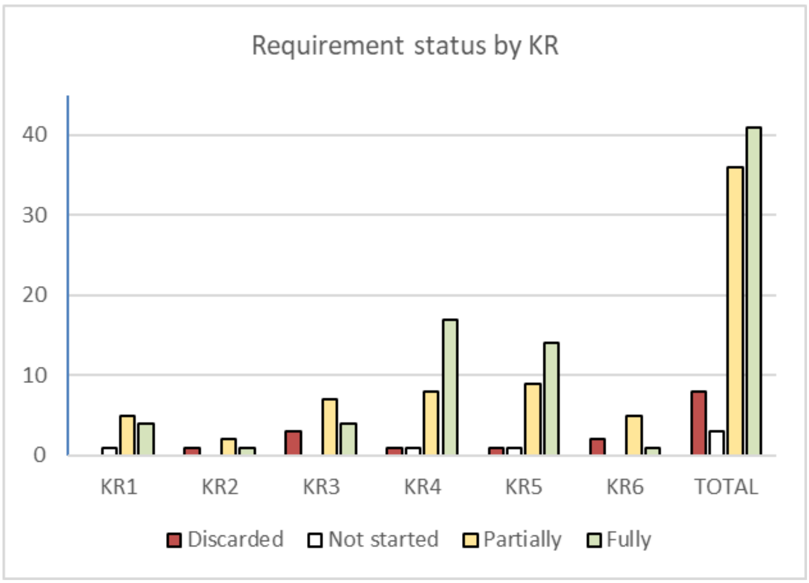


Figure 3. Requirement status by KR at M24

Some statistical conclusions about the implementation status and the evolution of the MEDINA framework can be extracted from Table 6:

- In M15, 40% of the requirements were at least partially implemented.
- In M24, 96% of the not discarded requirements have been at least partially implemented (45% partially implemented and 51% fully implemented).
- KR4 has the highest rate of fully implemented requirements (65%), followed by KR5 (58%) and KR1 (40%).
- KR6 has the lowest rate of fully implemented requirements (17%).
- The level of achievement of the different KRs (measured as the percent of the requirements implemented at least partially) range between 90% and 100%.

Table 7 and Figure 4 show a detail of the evolution of the status, differentiating among partial (P) and complete (Full) implementation of the requirements for each KR in the first year of the project (Y1) and in the second year (Y2). The last two columns show the number of requirements that have become, during this second year, Partially implemented (->P) and Fully implemented (->F).

Table 7. Requirement progress summary (by KR)

KR	Y1		Y2			
	P	Full	P	Full	->P	->Full
KR1	4	0	5	4	4	4
KR2	0	0	2	1	3	1
KR3	5	0	7	4	4	4
KR4	12	1	8	17	2	16
KR5	9	0	9	14	6	14
KR6	0	0	5	1	5	1
TOTAL	30	1	36	41	24	40

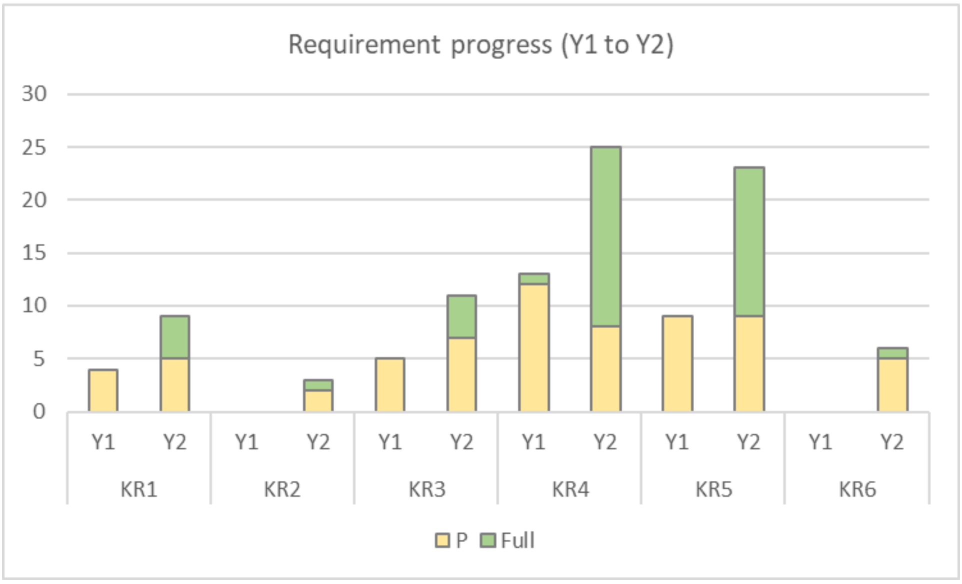


Figure 4. Requirement progress (by KR)

Leaving aside the discarded requirements, some general conclusions that can be drawn from Table 7 are the following:

- During Y2, 24 requirements have progressed to “Partially implemented”
- During Y2, 40 requirements have progressed to “Fully implemented”
- 45% of the requirements are partially implemented at this stage
- 51% of the requirements are fully implemented at this stage
- 96% of the requirements are already partially or fully implemented at this stage

4 MEDINA Framework Architecture

This section presents the second version of the MEDINA architecture. To create this version of the architecture, the process depicted in Figure 5was followed.

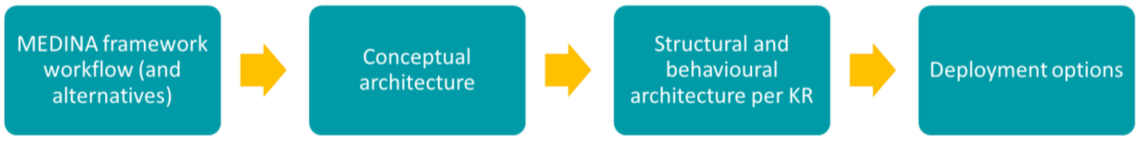

MEDINA framework workflow (and alternatives)
Conceptual architecture
Structural and behavioural architecture per KR
Deployment options

Figure 5. Process followed in MEDINA to develop the MEDINA architecture

This process comprises the following activities:

- First, an analysis of the MEDINA workflows and alternatives has been carried out (see Section 4.1).
- In parallel to the workflow definition, the overall architecture with all components has been designed (see Sections 4.2 and 4.3).
- Then, the structural and behavioural description of the components conforming the MEDINA framework has been detailed (see Section 4.4).
- Finally, as part of the architecture definition, the deployment options for the components and the MEDINA framework itself have been discussed and analysed (see Section 4.5).

4.1 MEDINA workflows

MEDINA workflows were first introduced in D5.3 [3]. They consist of the seven different scenarios/interactions, as shown in Table 8. These workflows cover different data flow paths of the architecture, each of which uses different components of the MEDINA framework. The MEDINA workflows are used in WP6 to instantiate user stories, and also to develop test steps for the user-centric evaluation.

Table 8. MEDINA Workflows

Workflow	Short Description	MEDINA Components
WF1 Preparation of ToC	Setup, configure and deploy the cloud service to certify (ToC) on top of the chosen hyperscaler(s). This process includes configuring the underlying PaaS/IaaS.	CSP testbed
WF2 Preparation of MEDINA Components	Setup, configure and deploy the MEDINA components. Only related to those components under the responsibility of the CSP.	Evidence Collectors, Integrated UI
WF3 EUCS Deployment on ToC	Setup, configure and deploy the corresponding EUCS framework (for the chosen assurance level basic/substantial/high) on the ToC.	Catalogue, NL2CNL Translator, CNL Editor, DSL Mapper
WF4 EUCS Preparedness - ToC Self-Assessment	Self-assess preparedness for EUCS certification based on the chosen assurance level. This is a risk-based approach.	SATRA
WF5 EUCS - Compliance Assessment	Performs a point-in-time (discrete) EUCS compliance assessment for the ToC. When such discrete assessment is periodically executed, then we achieve the MEDINA notion of “continuous”.	AMOE, Orchestrator, Trustworthiness System, Evidence Collectors

Workflow	Short Description	MEDINA Components
WF6 EUCS - Maintenance of ToC certificate	Start certificate maintenance life cycle for the ToC. Based on current EUCS, the maintenance process comprises the following stages: issuance, renewal, continuation, update, re-issuance (new certificate), withdrawal, and suspension.	RAOF (Dynamic), CCE, ACLM, SSI
WF7 EUCS - Report on ToC certificate	Report on EUCS certificate status for a ToC. The report can be obtained by the CAB or by the CSP, in which case the level of provided details might vary.	Integrated UI, RAOF (Dynamic), CCE, ACLM, SSI

4.2 MEDINA framework

The architectural framework proposed by MEDINA can be abstracted in the eight building blocks shown in Figure 6. Each building block corresponds to a well differentiated functionality of the proposed architecture and is instantiated by the use cases in WP6.

1. Catalogue
2. Certification language
3. Risk assessment and optimisation framework
4. Continuous Evaluation and Certification Life-Cycle
5. Organizational Evidence Gathering and Processing
6. Orchestrator and Databases
7. Evidence Collection and Security Assessment
8. Integrated UI

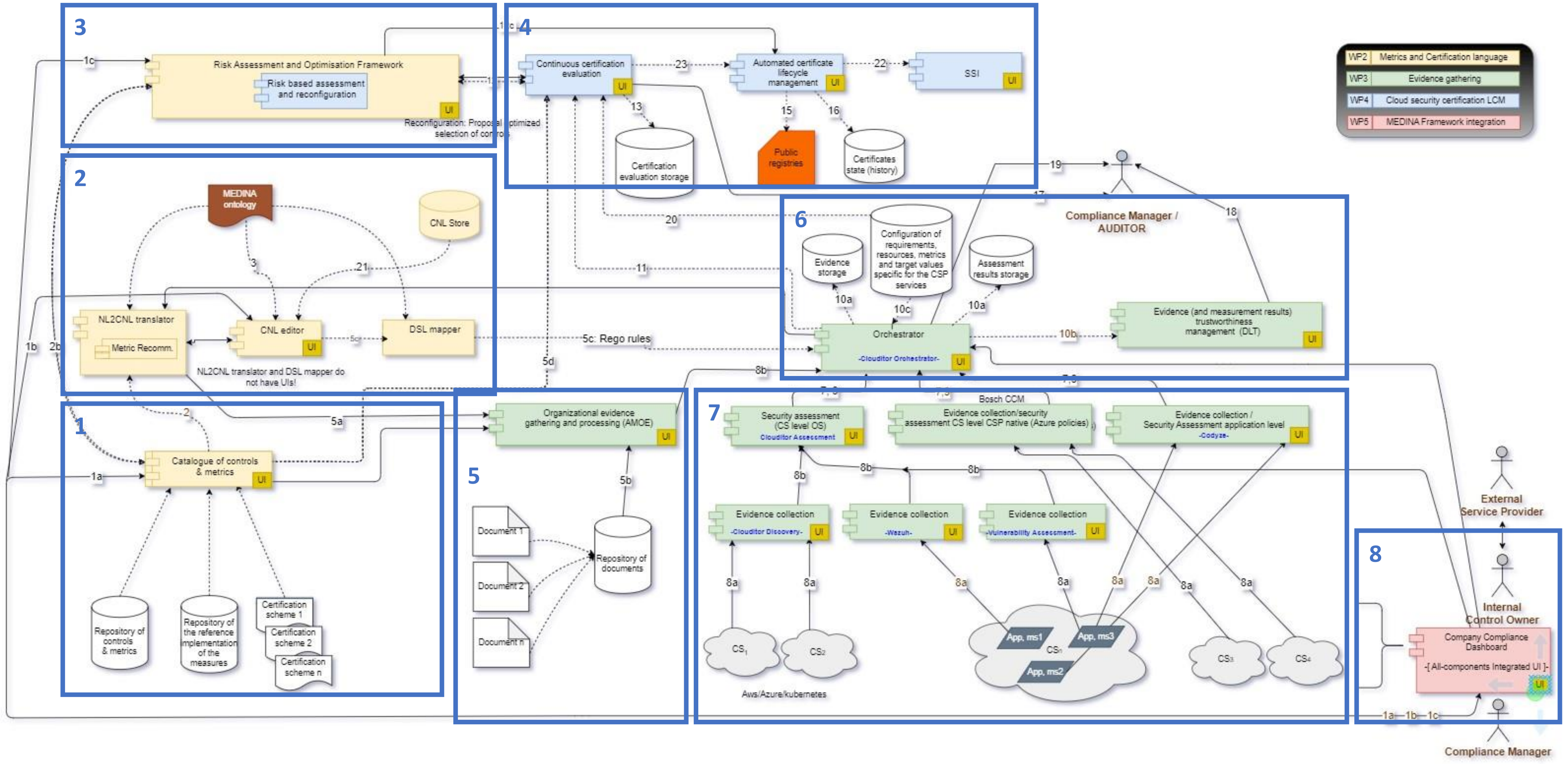

3
Risk Assessment and Optimisation Framework
Risk based assessment and reconfiguration
UI
Reconfiguration: Proposal optimized selection of controls
4
Continuous certification evaluation
Automated certificate lifecycle management
SSI
Certification evaluation storage
Public registries
Certificates state (history)
23
22
13
15
16
20
11
2
MEDINA ontology
CNL Store
NL2CNL translator
Metric Recomm.
CNL editor
DSL mapper
NL2CNL translator and DSL mapper do not have UIs!
5c: Rego rules
5d
21
1b
2b
1c
1a
1
Catalogue of controls & metrics
Repository of controls & metrics
Repository of the reference implementation of the measures
Certification scheme 1
Certification scheme 2
Certification scheme n
5a
5
Organizational evidence gathering and processing (AMOE)
Document 1
Document 2
Document n
Repository of documents
5b
6
Evidence storage
Configuration of requirements, resources, metrics and target values specific for the CSP services
Assessment results storage
Orchestrator
-Clouditor Orchestrator-
Evidence (and measurement results) trustworthiness management (DLT)
10a
10b
10c
8b
Compliance Manager / AUDITOR
19
17
18
7
Security assessment (CS level OS)
Clouditor Assessment
Bosch CCM
Evidence collection/security assessment CS level CSP native (Azure policies)
Evidence collection / Security Assessment application level
-Codyze-
Evidence collection
-Clouditor Discovery-
-Wazuh-
-Vulnerability Assessment-
8a
CS1
CS2
Aws/Azure/kubernetes
App, ms1
App, ms2
App, ms3
CSn
CS3
CS4
8
Company Compliance Dashboard
-[All-components Integrated UI]-
External Service Provider
Internal Control Owner
Compliance Manager
WP2 Metrics and Certification language
WP3 Evidence gathering
WP4 Cloud security certification LCM
WP5 MEDINA Framework integration

Figure 6. Architecture diagram of the MEDINA framework

The first building block, named *Catalogue,* relates to the set of catalogues being required by the different roles and personas interacting with MEDINA (see the deliverable D6.2 [5]), which comprises the different elements of a control's framework such as EUCS [4]. We refer to the catalogue of controls, metrics, TOMs' reference implementations, and target values being suggested to CSPs (see the deliverable D2.1 [10]). This building block also leverages self-assessment functionalities targeting specific roles e.g., CABs and compliance managers.

Then we have a second block, named *Certification Language,* which implements the NLP techniques proposed by MEDINA to guarantee that requirements from EUCS, or other catalogues, are related to the metrics in the Catalogue building block referenced in the previous paragraph .This building block leverages a novel ontology to guarantee that requirements written in natural language are automatically associated to CSP-specific resources (see the deliverable D2.4 [11]). Our goal is to automatize the current (manual) process performed by compliance managers and auditors to interpret, map, implement, and assess requirements in their own organizations.

The risk assessment functionalities provided by MEDINA, both static and dynamic, can be seen in the third building block of the architecture, named *Risk Assessment and Optimisation framework*. These functionalities take as input the Catalogue of controls & metrics and, and based on the CSP's risk appetite and assessment results can control the certification lifecycle in block n.4 (see the deliverable D2.7 [12]).

A core building block in our framework comprises the components that manage the lifecycle of the certification, which depends on the rules established by the corresponding scheme (EUCS in the case of MEDINA). This is the fourth building block in our framework, named *Continuous Evaluation and Certification Life-Cycle,* where National Certification Bodies (NCB), or even pan-European entities like ENISA, can benefit from becoming public registers of issued certificates. Our goal is to provide fully automated lifecycle management, which depends on dynamic risk assessment techniques and the automation of the security assessment processes implemented by MEDINA (see the deliverables D4.2 [8] and D4.4 [13]).

A state of practice challenge for providing automation of auditing/certification relates to the processing of organizational measures, where related documentation of the CSP (e.g., security concepts, operation manuals) is assessed for conformance with the certification scheme's requirements. Building block five in MEDINA's framework, named *Organizational Evidence Gathering and Processing,* implements both a repository for organizational evidence, and the NLP-based techniques for their processing (see the deliverable D3.5 [14]).

Complementary to this functionality in MEDINA, is building block seven, named *Evidence Collection and Security Assessment*, which provides the assessment of technical measures by integrating a variety of tools, including native CSP functionalities (see the deliverable D3.5 [14]). This building block seven targets the multi-layer assessment of the target-of-certification cloud service i.e., the related IaaS, PaaS, and SaaS stack.

All gathered assessment results, either from organizational (block n.5) or technical measures (block n.7), are holistically stored and processed by the components shown in building block 6, named *Orchestrator and Databases*. Both an orchestrator (in charge if managing the collected evidence and assessment results), and a DLT-enabled evidence manager (to guarantee tamper proof storage of evidence) are core components of this building block (see the deliverable D3.5 [14]).

Finally, building block 8, named *Integrated UI*, provides a MEDINA user interface to facilitate human interaction with the components in building block 7. Take for example a corporate compliance manager who visualizes the near real-time status of issued EUCS certificates, or an external CAB reviewing the evidence used for a certification process (see the deliverable D5.3 [3]).

4.3 MEDINA data model

This section presents the current version of the MEDINA data model. This data model describes the different entities that are used and shared by the components in the MEDINA framework, as well as their attributes (see Figure 7). The entities have been categorized into different groups depending on the building block/WP they belong to, and are coloured differently for clarity:

- Blue entities: Catalogue of controls and metrics
- Grey entities: Risk Assessment and optimisation framework
- Orange entities: Evidence gathering and MEDINA ontology
- Red entities: Evidence assessment
- Green entities: Evidence gathering and assessment
- Purple entities: Evidence and assessment result trustworthiness
- Dark orange entities: Cloud security certification

This version of the data model is an evolution of first version that was described in deliverable D5.1 [1]. The main differences lie in the extension of the list of data attributes in general, and also in the introduction of new entities to face the needs of the latest version of the components. Specifically, the new entities are:

- Target of evaluation
- Risk assessment result
- Certificate
- Security metric configuration
- User

For a more detailed description of the entities, see the deliverables describing the components in the technical WPs (WP2, WP3 and WP4).

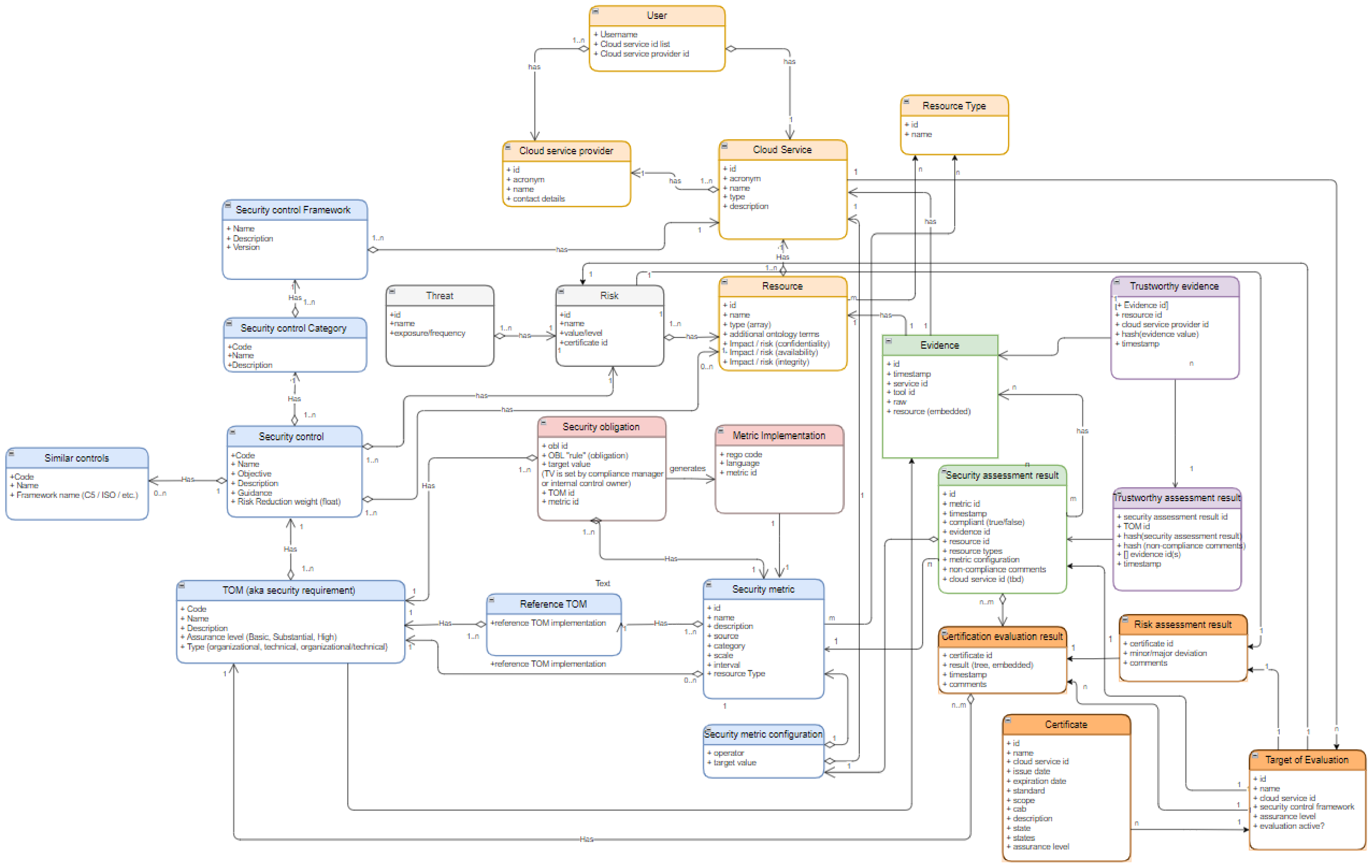


Figure 7. MEDINA framework data model

4.4 MEDINA components structural and behavioural description

This section presents the different components that are part of the MEDINA framework, grouped into the main building blocks Identified in Section 4.2.

Components are described by means of the "component card" template, which includes main functionalities, subcomponents, sequence diagrams, interfaces, etc., providing the structural and behavioural description of the components.

4.4.1 Catalogue

The *Catalogue of controls and metrics* (a.k.a. *Catalogue*) is an IT tool for the storage and management of controls, requirements, metrics, and their relationships[9].

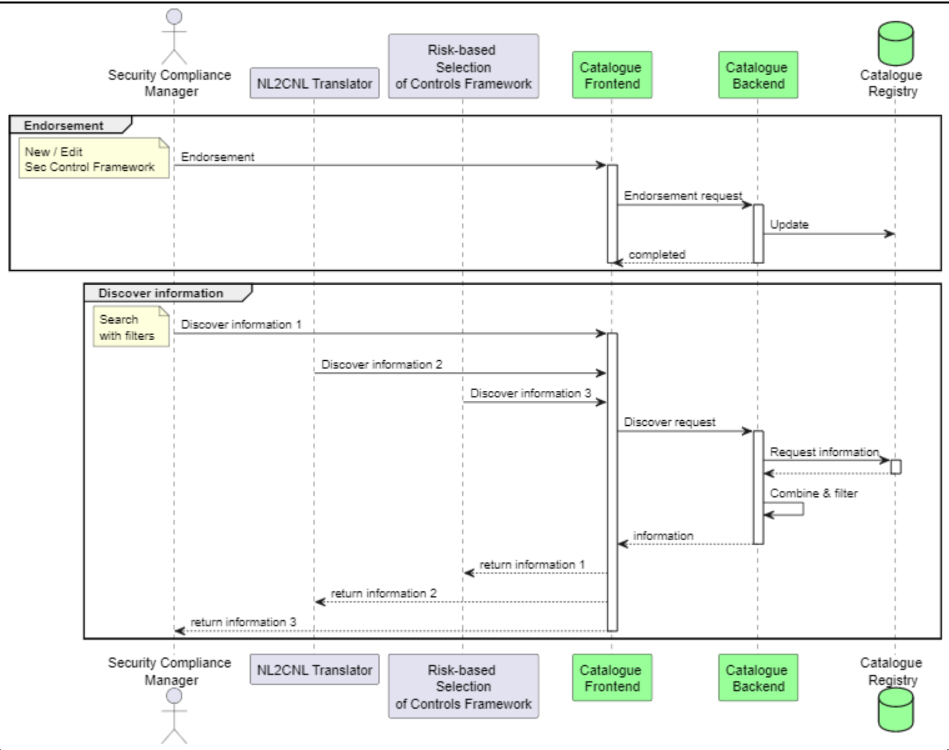
Component Name	Catalogue of controls and metrics
Main functionalities	The component provides the following functionalities: • Endorsement of Security Control Frameworks and related attributes: Security requirements, categories, controls, reference TOMs, metrics, evidence, and assurance levels. • Provision of guidance for the (self-)assessment of the requirements. • Filtering of the information based on some values for the attributes ○ Selection of requirements of a certain assurance level ○ Selection of requirements from a certain framework ○ Selection of metrics related to reference TOM ○ Etc. • Homogenization of the certification schemes: Provision of information about related requirements from different frameworks especially referenced to the EUCS.
Sub-components Description	**Registry**: The registry will store the available list of frameworks and the related info for a specific CSP. This subcomponent will also include the corresponding databases. **Back-end**: The backend is the core sub-component of the Catalogue. It will perform the actual discovery of the requirements, evidence, etc. from the registry, considering the set of filters established by the user through the UI/ API. **Frontend**: This sub-component is the graphical user interface of the Catalogue. This frontend will allow the user to indicate the requirements to filter and select a set of information related to the existing frameworks, i.e., requirements of a certain assurance level, requirements from a certain framework, metrics related to a reference TOM, references TOMs, guidance, etc.
Main logical Interfaces	see table below

Interface name	Description	Interface technology
Catalogue UI	Graphical user interface of the Catalogue	Angular and Bootstrap
Discover requirements	Select a set of requirements (and related attributes) for a given CS	Rest API

[9] The interested reader is referred to the Catalogue technical specifications in the deliverable D2.1 [11]. A second version of the Catalogue will be realised in M27 and described in the deliverable D2.2.

Requirements Mapping	List of requirements covered by this component: RCME.01, RCME.02, RCME.03, RCME.04, RCME.05, RCME.06, RCME.07, RCME.08, RCME.09, RCME.10
Interaction with other components	

Interfacing Component	Interface Description
NL2CNL Translator	NL2CNL Translator will request to the Catalogue the requirements and related information for a certain user
Risk assessment and optimisation framework	RBSCF will request to the Catalogue the requirements list and related information

Relevant sequence diagram/s	
Current TRL	Based on exiting tools (ACSmI-Tecnalia)
Programming language	jHipster framework based on microservices architecture: • Java stack on the server side with Spring Boot • Frontend with Angular and Bootstrap
License	Apache license v2.0
WP and task	WP2 - Task 2.1, Task 2.2 WP3 – Task 3.1
Workflow	WF3

4.4.2 Certification language

The *Certification Language* is responsible of converting the security requirements of the chosen certification schema, which are expressed in Natural Language (NL), into a language that can be automatically "executed" by a machine.[10]

[10] More details can be found in D2.4 [12].

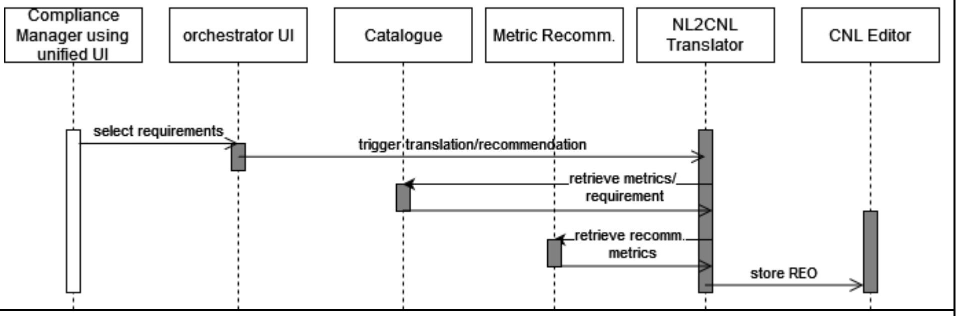
4.4.2.1 NL2CNL Translator

Component Name	NL2CNL Translator
Main functionalities	The component provides the following functionalities: • Translates the natural language text (English) of Security Requirements (TOMs) to the CNL obligations by recommending/predicting a set of metrics and integrating them into the CNL
Sub-components Description	**Recommender system**: Associates a set of metrics to a requirement. **Obligation builder:** Takes the associated metric, the predefined target value, the operator, and the definition of the resource (probably from the Catalogue). Obl = op(MI, TV) -> returns Boolean Obl = op(MV, TV) -> returns Boolean **Optional component: Database.** Stores the obligations and associated metadata like requirement-ID etc. We would prefer to include this in a sub-database in the repository of controls. Currently, obligations and associated metadata are stored in the CNL Store, provided by the CNL Editor.
Main logical Interfaces	(see table below)

Interface name	Description	Interface technology
NL2CNL Translator API	API to access NL2CNL functionalities	REST API

Requirements Mapping	List of requirements covered by this component: NL2CNL.01, NL2CNL.02, NL2CNL.03, NL2CNL.04
Interaction with other components	(see table below)

Interfacing Component	Interface Description
Catalogue of controls and metrics	NL2CNL Translator reads TOMs and metrics from it
CNL Editor API	NL2CNL Translator exploits CNL Editor API to access the CNL Store functionalities, i.e., to store the requirements and obligations information in XML format

Relevant sequence diagram/s	 Compliance Manager using unified UI orchestrator UI Catalogue Metric Recomm. NL2CNL Translator CNL Editor select requirements trigger translation/recommendation retrieve metrics/ requirement retrieve recomm. metrics store REO
Current TRL	To be developed from scratch
Programming language	Python 3.x
License	Apache 2.0
WP and task	WP2 Task 2.3
Workflow	WF3

4.4.2.2 CNL editor

Component Name	**CNL Editor**
Main functionalities	The component provides the following functionalities: • show a CNL document, i.e., a requirement description (metadata) with a list of associated metrics in the form of a list of Obligations • edit obligations parameters (operator, target value) based on the Editor ontology • delete metrics from an already filled CNL document • map a requirement invoking DSL Mapper to convert CNL in Rego code
Sub-components Description	**CNL Editor UI**: Web GUI Interface for users, with authentication **OWL vocabulary**: stores the Ontology used by the CNL Editor **Editor API**: used to access CNL documents from external clients/components **Back Store Interface**: to access internally the CNL Store **CNL Store**: document-oriented storage

Main logical Interfaces

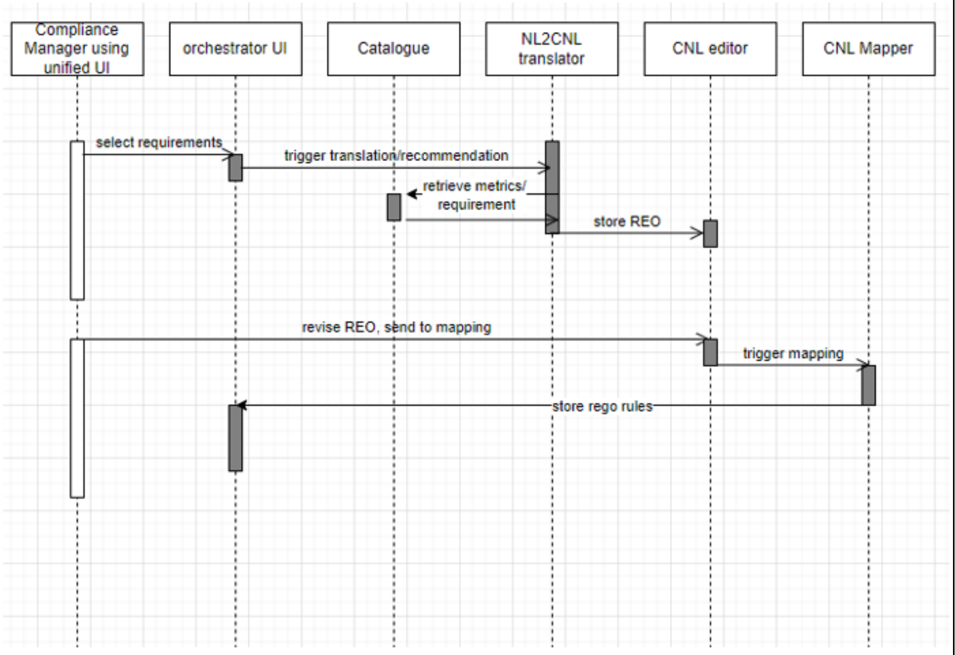
Interface name	Description	Interface technology
CNL Editor UI	CNL Editor Web GUI	HTTP (browser)
Editor API	API to access CNL documents	REST API

Requirements Mapping	List of requirements covered by this component CNLE.01, CNLE.03, CNLE.04, CNLE.05, CNLE.06

Interaction with other components

Interfacing Component	Interface Description
NL2CNL Translator	CNL Editor reads CNL documents in XML format as prepared by CNL Translator
DSL Mapper	CNL Editor provides to DSL Mapper the finalised CNL documents to be mapped

Relevant sequence diagram/s


Compliance Manager using unified UI
orchestrator UI
Catalogue
NL2CNL translator
CNL editor
CNL Mapper
select requirements
trigger translation/recommendation
retrieve metrics/ requirement
store REO
revise REO, send to mapping
trigger mapping
store rego rules

Current TRL	Based on exiting tools/components (HPE)
Programming language	Java, Springboot, GWT

License	Apache 2.0
WP and task	WP2 Task 2.4
Workflow	WF3

Component Name	**CNL Editor Ontology**
Main functionalities	The component provides the following functionalities: • Ontology allows to write or change Metrics associated to a Requirements respecting ontology rules and Vocabulary defined
Sub-components Description	**OWL vocabulary**: stores Ontology used by Editor
Main logical Interfaces	OWL vocabulary is a file in RDF/XML format that is used on reading internally from CNL Editor Interface name: Protégé[11] Description: Protégé Desktop is a feature rich ontology editing environment with full support for the OWL 2 Web Ontology Language and is W3C Standard Compliant Interface technology: Windows Desktop
Requirements Mapping	List of requirements covered by this component: CNLE.02, CNLE.04, CNLE.05
Interaction with other components	Interfacing Component: CNL Editor Interface Description: CNL Editor reads vocabulary (OWL file)
Relevant sequence diagram/s	
Current TRL	Based on exiting tools/components (HPE)
Programming language	n/a
License	n/a
WP and task	WP2 Task 2.4
Workflow	WF3

Main logical Interfaces (nested table):

Interface name	Description	Interface technology
Protégé[11]	Protégé Desktop is a feature rich ontology editing environment with full support for the OWL 2 Web Ontology Language and is W3C Standard Compliant	Windows Desktop

Interaction with other components (nested table):

Interfacing Component	Interface Description
CNL Editor	CNL Editor reads vocabulary (OWL file)

4.4.2.3 DSL Mapper

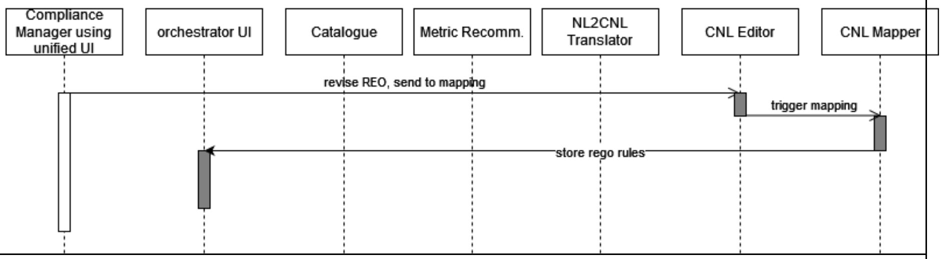
Component Name	**DSL Mapper**
Main functionalities	The component provides the following functionalities: • Mapping of the CNL obligations + metadata output to a DSL (e.g., Rego)
Sub-components Description	To be defined/ there are no sub-components

[11] https://protege.stanford.edu/

Main logical Interfaces	Interface name	Description	Interface technology
	DSL Mapper API	API to access DSL Mapper functionalities	REST API

Requirements Mapping	List of requirements covered by this component: DSLM.01, DSLM.03

Interaction with other components	Interfacing Component	Interface Description
	CNL Editor	The DSL Mapper is called from the CNL Editor, which passes, as a parameter, an object in XML format, including all the necessary requirement metadata, metrics information, CNL obligations
	Orchestrator	The DSL Mapper maps the selected obligations + metadata into a DSL (Rego) and pushes the output to the Orchestrator by exploiting its API

Relevant sequence diagram/s	 Compliance Manager using unified UI orchestrator UI Catalogue Metric Recomm. NL2CNL Translator CNL Editor CNL Mapper revise REO, send to mapping trigger mapping store rego rules
Current TRL	To be developed from scratch
Programming language	Python 3.x
License	Apache 2.0
WP and task	WP2 Task 2.5
Workflow	WF3

4.4.3 Risk assessment and optimisation framework

This block is used as a decision-making instrument for the analysis of non-conformities of a cloud service with a selected certification scheme[12].

Component Name	**Risk Assessment and Optimisation Framework (aka Risk-based selection of controls Framework, SATRA)**
Main functionalities	The component provides the following functionalities: Risk Assessment – a questionnaire-based risk assessment facility to evaluate CSP-specific risk levels for predefined threats. • Cost-Effective TOMs optimisation – selection the most cost-effective requirements/TOMs (to optimise investment) in case Certification Framework allows this (in contrast to rigid Frameworks). • Risk-based analysis of deviations – risk-based evaluation of non-conformity from the framework to determine if the deviation is major or minor.

[12] The interested reader is referred to Risk assessment technical specifications in the deliverable D2.7 [13].

Sub-components Description	**Risk Assessment Engine** – computes risk levels using the pre-established relations between asset types, threats, and requirements. Requires the list of assets and implemented requirements as input. **Risk Assessment GUI** – is the user-friendly front-end part of the Framework which guides a user (compliance manager) through the steps for identification of main input parameters and displays results of the analysis**.** **Risk Assessment API –** is a set of APIs which collect the main input parameters and provide the results of the analysis in a machine-readable format. In case all interactions with MEDINA are performed through the Compliance Manager Dashboard only, only API is relevant. **Risk optimiser Engine** – selects the most cost-relevant TOMs to optimise the expected expenditure (risk + cost) given the budget or to ensure compliance with the selected Certification Framework (with, at most, minor non-conformity). **Risk-based decision support** - compares two risk assessment results (basic and actual ones) and decides if the deviation is major or minor. **Risk storage** – the storage of the current risk practices settings.

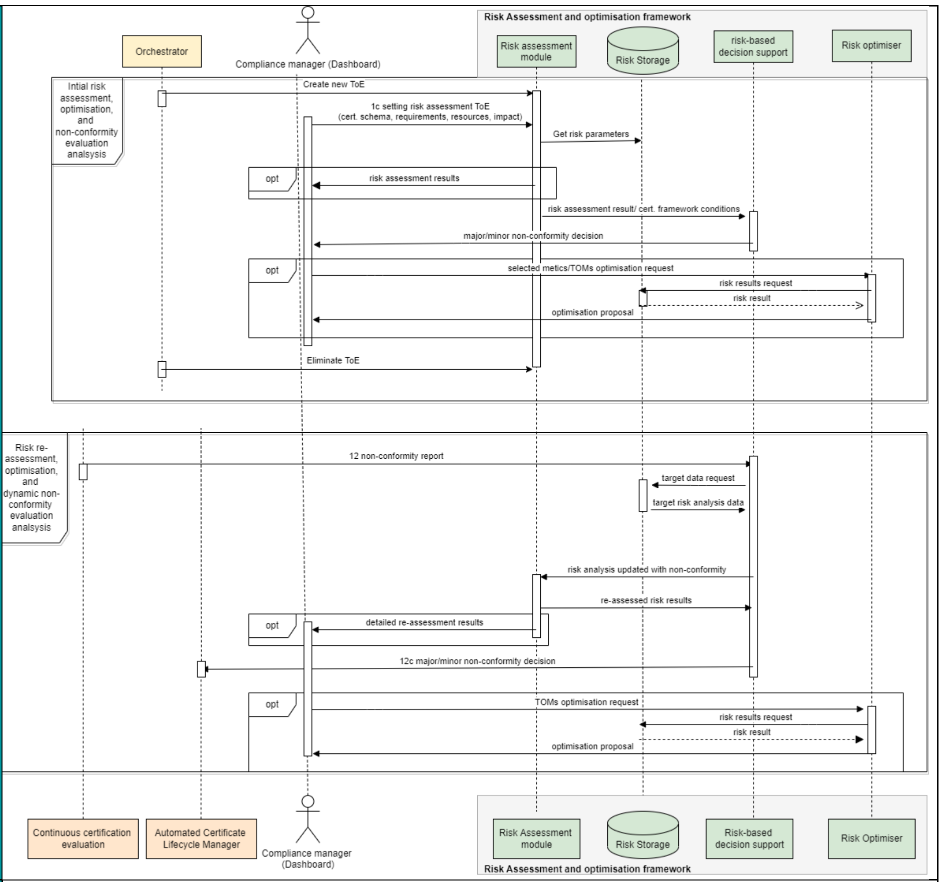
Main logical Interfaces

Interface name	Description	Interface technology
Risk Assessment GUI	Graphical user interface of risk assessment	GUI
Risk Assessment APIs	A set of machine-readable APIs for risk assessment	Rest API
Non-conformity reporting API	The API used for analysis and reporting a detected non-conformity.	Rest API

Requirements Mapping	List of requirements covered by this component: RBSCF.01, RBSCF.02, RBSCF.03, RBSCF.04, RBCA.01, RBCA.02

Interaction with other components

Interfacing Component	Interface Description
Compliance manager (Dashboard)	Invokes Risk Assessment and Optimisation Framework for the selection of suggested requirements to implement, analysis of (goal) security configuration (e.g., for deviation from the target security configuration set by a certification framework), setting up resources and possible impact.
Continuous certification evaluation	Invokes Risk Assessment and Optimisation Framework for the evaluation of the detected non-conformity
Automated certificate lifecycle management	Consumes the result of the risk-based non-conformity evaluation.
Orchestrator (Clouditor)	Notifies about creation/deletion of a Target of Evaluation.

Relevant sequence diagram/s	 Risk Assessment and optimisation framework Orchestrator Compliance manager (Dashboard) Risk assessment module Risk Storage risk-based decision support Risk optimiser Intial risk assessment, optimisation, and non-conformity evaluation analsysis Create new ToE 1c setting risk assessment ToE (cert. schema, requirements, resources, impact) Get risk parameters opt risk assessment results risk assessment result/ cert. framework conditions major/minor non-conformity decision opt selected metics/TOMs optimisation request risk results request risk result optimisation proposal Eliminate ToE Risk re-assessment, optimisation, and dynamic non-conformity evaluation analsysis 12 non-conformity report target data request target risk analysis data risk analysis updated with non-conformity re-assessed risk results opt detailed re-assessment results 12c major/minor non-conformity decision opt TOMs optimisation request risk results request risk result optimisation proposal Continuous certification evaluation Automated Certificate Lifecycle Manager Compliance manager (Dashboard) Risk Assessment module Risk Storage Risk-based decision support Risk Optimiser Risk Assessment and optimisation framework
Current TRL	Based on exiting tools/components (TRL 4/5), but the tool should be tuned for MEDINA's needs
Programming language	Java, Python
License	Apache 2.0
WP and task	WP2 (Task 2.6) and WP4 (Task 4.4)
Workflow	WF4, WF6, WP7

4.4.4 Continuous Evaluation and Certification Life-Cycle

This block is responsible for the continuous evaluation of security assessments of cloud services, including an approach for continuously aggregating assessment results, as well as deriving a decision about the certificate state[13].

4.4.4.1 Continuous certification evaluation

Component Name	**Continuous certification evaluation**
Main functionalities	Evaluates the compliance level on all levels of the certification hierarchy (resources, requirements, controls, control groups, standard) based on the aggregation of assessment results and configuration (weights of individual tree nodes).

[13] More details can be found in D4.2 [10] and D4.4 [14].

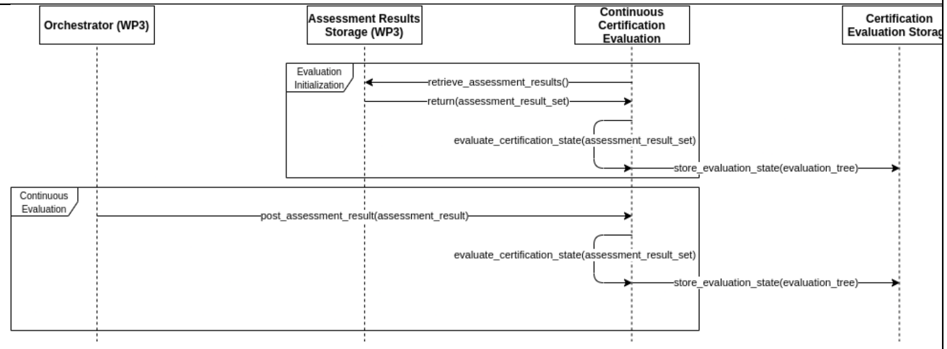
Sub-components Description	The component consists of: • **CCE**: the main back-end component which manages all the compliance level calculations and interfaces with other components, • **CCE-frontend**: a web UI interacting with the back-end to display information to users, • **Mongo-DB document database**: storing past states of evaluation trees.
Main logical Interfaces	

Interface name	Description	Interface technology
Security assessment input	Receiving security assessments from the Orchestrator.	gRPC
Web UI	UI to display evaluation results in a graphical way	HTTP
HTTP (REST) API	Offering data about current and past evaluation results to other components	HTTP

Requirements Mapping	CCCE.01, CCCE.02, CCCE.03, CCCE.04, CCCE.05, CCCE.06, CCCE.07
Interaction with other components	

Interfacing Component	Interface Description
Evidence orchestrator	CCE continuously receives assessment results from the orchestrator via gRPC. It also obtains the configuration data about targets of evaluation, their certification schemas, and metrics.
Automated certificate lifecycle manager	Lifecycle manager obtains the details about current and past evaluation results from the CCE by querying its API.
Risk assessment and optimisation framework	CCE sends the calculated evaluation data to RAOF when any significant change to the evaluation occurs. An HTTP interface exposed by RAOF is used.
Catalogue of controls and security schemas	CCE obtains certification schema information from the Catalogue.

Relevant sequence diagram/s


Orchestrator (WP3)
Assessment Results Storage (WP3)
Continuous Certification Evaluation
Certification Evaluation Storag
Evaluation Initialization
retrieve_assessment_results()
return(assessment_result_set)
evaluate_certification_state(assessment_result_set)
store_evaluation_state(evaluation_tree)
Continuous Evaluation
post_assessment_result(assessment_result)
evaluate_certification_state(assessment_result_set)
store_evaluation_state(evaluation_tree)

Current TRL	TRL 3-4
Programming language	Java, JavaScript
License	Apache License v2.0

WP and task	T4.1
Workflow	WP6, WP7

4.4.4.2 Automation of the Cloud Security Certification Life-Cycle

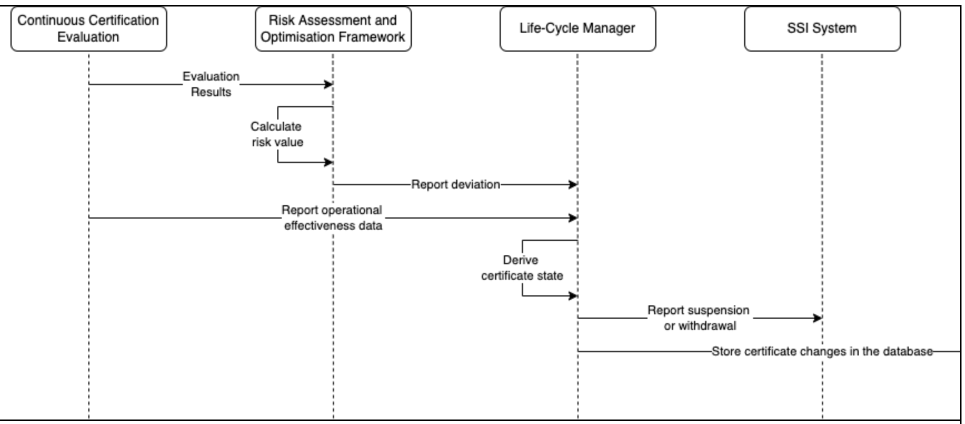
Component Name	**Life-Cycle Manager (LCM)**
Main functionalities	The component provides the following functionalities: • Consume information about minor or major deviations present in the cloud system, as well as consume information about operational effectiveness • Translate above information into a certificate state • Report changes to the SSI Framework • Store changes in the Orchestrator database and display them in the Orchestrator UI
Sub-components Description	No subcomponents exist in the LCM.
Main logical Interfaces	(see table below)
Requirements Mapping	List of requirements covered by this component ACLM.01, ACLM.02, ACLM.03, ACLM.04, ACLM.06, ACLM.07, ACLM.08
Interaction with other components	(see table below)

Main logical Interfaces:

Interface name	Description	Interface technology
Certificate Maintenance	Allows the creation, update, and deletion of certificates	REST
Deviation report	Allows the report minor or major of deviations	REST
UI	User interface to see the state and state history, as well as certificate information (integrated in the Orchestrator UI)	gRPC, Typescript / Svelte

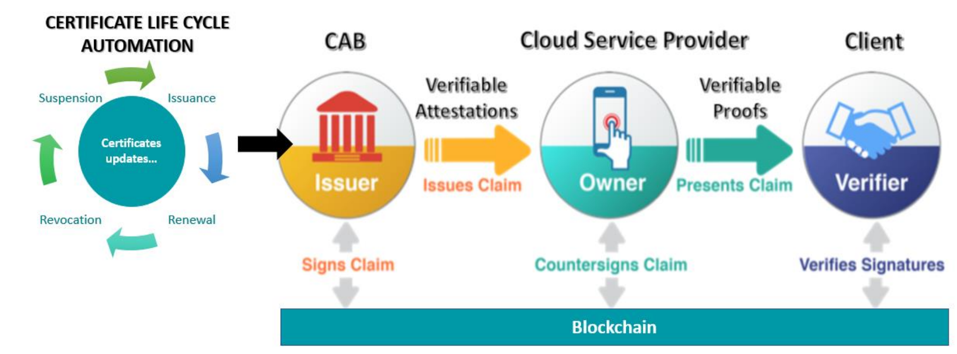
Interaction with other components:

Interfacing Component	Interface Description
Continuous Cloud Security Certification evaluation	The CCE provides operational effectiveness data to the LCM.
SSI Framework	The LCM sends certificate maintenance reports to the SSI Framework to allow auditors to review the decisions.
Orchestrator	The LCM stores any certificate data in the Orchestrator's database.

Relevant sequence diagram/s	 Continuous Certification Evaluation Risk Assessment and Optimisation Framework Life-Cycle Manager SSI System Evaluation Results Calculate risk value Report deviation Report operational effectiveness data Derive certificate state Report suspension or withdrawal Store certificate changes in the database
Current TRL	TRL3
Programming language	Go
License	Apache 2.0
WP and task	W–4 - T4.3
Workflow	WP6, WP7

4.4.4.3 SSI Framework

Component Name	**Self-Sovereign Identity Framework**
Main functionalities	The component provides the following functionalities: • Provides a tool for appropriate entities (CAB) to issue/update/revoke and sign security certifications for the cloud providers based on the updated certificate state received from the Certificate Lifecycle Automation component. • Provides a tool for appropriate entities (CAB) to publish the certificate state in a public registry. • Provides a tool for appropriate entities (for example, cloud providers clients) to ask for proofs about the state of different certifications of the cloud providers. • Provides a tool for cloud providers to see/list received certifications and their associated state. • Provides a tool for cloud providers to send proofs about the certificate state to their clients.
Sub-components Description	The SSI Framework is composed of five main components. • Public service for the CAB to receive certificates updates. • Certificate signing application for the CAB to issue, update, or revoke security certificates to a CSP as well as to save the signed security certificates in a public registry. • Application for CSP clients to request and verify proofs of security certificates. • Application for the CSPs to save the signed security certificates as well as to generate verifiable proofs based on the signed security certificates. • A blockchain network to record the different actors' signatures.



Main logical Interfaces

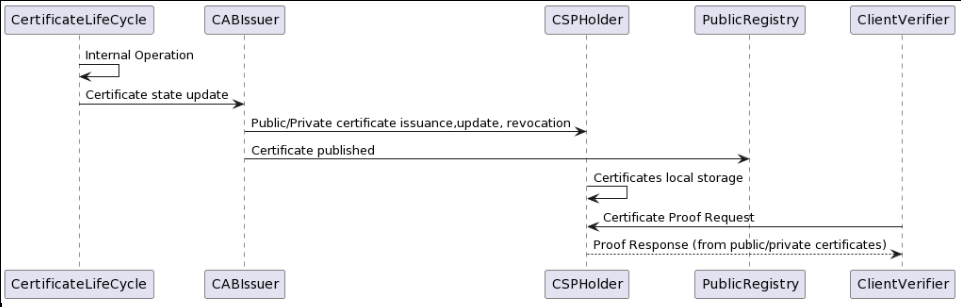
Interface name	Description	Interface technology
Certificate Life cycle automation	Provides the security certificate state update.	REST API
CAB	Sign and publicly publish security certifications	Web (Provided aaS)
CSP	List and proof generation of security certifications	Web (Provided aaS)
CSP client	Proof request and verification of security certifications.	Web (Provided aaS)

Requirements Mapping	List of requirements covered by this component SSI.01 SSI.02, SSI.03, SSI.04, SSI.05, SSI.06, SSI.07

Interaction with other components

Interfacing Component	Interface Description
Certificate Life cycle automation	It will provide the security certificate state update (and other certificate features if needed).

Relevant sequence diagram/s


CertificateLifeCycle
CABIssuer
CSPHolder
PublicRegistry
ClientVerifier
Internal Operation
Certificate state update
Public/Private certificate issuance,update, revocation
Certificate published
Certificates local storage
Certificate Proof Request
Proof Response (from public/private certificates)

Current TRL	Based on exiting tools (Identity Builder-TECNALIA)
Programming language	JavaScript (ReactJS)
License	Proprietary. Copyright by TECNALIA.
WP and task	WP4 – Task 4.3
Workflow	WP6, WP7

4.4.5 Organizational evidence gathering and processing

This block is responsible for the assessment and management of organisational evidence that is extracted from policy documents[14].

Component Name	Assessment and Management of Organizational Evidence (AMOE)
Main functionalities	The component provides the following functionalities: • Gathering and processing organizational evidence • Providing evidence to the Clouditor for assessment
Sub-components Description	Organizational evidence is collected by applying NLP and organisational metric to an uploaded document. The processing part transforms this evidence in the form of technical evidence. This transformed evidence then are provided to the security assessment of the Clouditor which can handle such technical evidence.
Main logical Interfaces	(see table below)
Requirements Mapping	OEGM.01, OEGM.02, OEGM.03, OEGM.04, OEGM.05
Interaction with other components	(see table below)

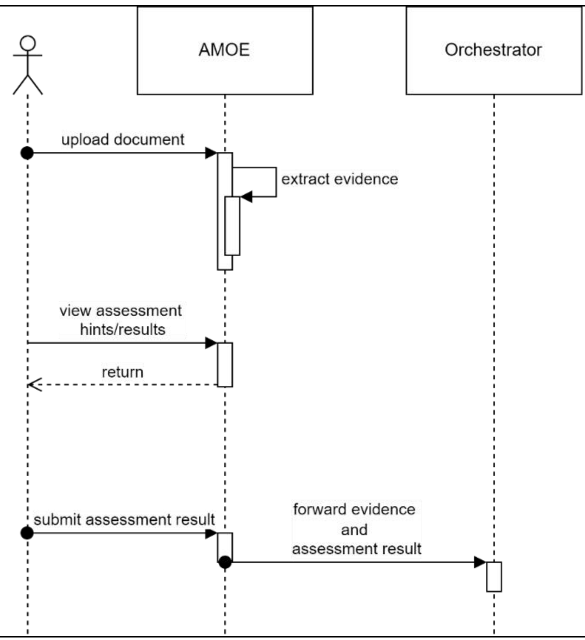
Main logical Interfaces:

Interface name	Description	Interface technology
UI	GUI to • Upload documents • Retrieve evidence • Set assessment results • Submit/forward assessment results	webservice
API	• Upload documents • Retrieve evidence • Set assessment results • Submit/forward assessment results	REST

Interaction with other components:

Interfacing Component	Interface Description
Orchestrator	Send collected evidence

[14] More details can be found in D3.5 [14].

Relevant sequence diagram/s	 AMOE Orchestrator upload document extract evidence view assessment hints/results return submit assessment result forward evidence and assessment result
Current TRL	TRL3-TRL4
Programming language	Python
License	Open source
WP and task	WP3: Task 3.4
Workflow	WP2, WP5

4.4.6 Orchestrator and databases

The Orchestrator is the central management component of MEDINA which manages database access, cloud services and a user interface[15].

Component Name	**Orchestrator**
Main functionalities	The component provides the following functionalities: • Store evidence and assessment results and provide an API to the databases • Forward assessment results to the certificate evaluation • Forward assessment result hashes to the trustworthiness system • Inform other components about new/modified cloud services and targets of evaluation
Sub-components Description	The Orchestrator mainly provides APIs to various components (see below). A dedicated subcomponent is the ledger client which transforms evidence and assessment results into the format required by the trustworthiness system and stores them on the ledger.

[15] More details can be found in D3.5 [14].

Main logical Interfaces

Interface name	Description	Interface technology
Assessment results storage	An interface to provide assessment results which are then stored in the relevant database, and forwarded to the relevant components	REST / gRPC
Database access	An interface that provides access to stored evidence and assessment results	REST / gRPC
DLT storage	An interface to the DLT through which evidence and assessment result checksums are stored to the trustworthiness system.	REST
Configure metrics and target values	An interface that provides access to metrics and target values	REST / gRPC
UI	A graphical interface that presents information about assessment results, cloud services, etc.	Typescript / Svelte

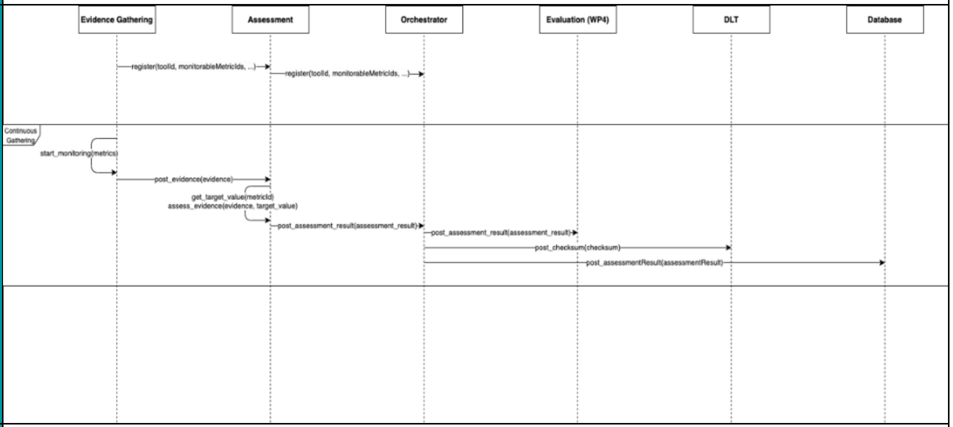
Requirements Mapping

List of requirements covered by this component:
ECO.01, ECO.02, ECO.03

Interaction with other components

Interfacing Component	Interface Description
Assessment tools	Receives assessment results from assessment tools
Databases	Stores and retrieves evidence/assessment results from the relevant databases
Trustworthiness system	Sends assessment result hashes to the trustworthiness system
Metrics and target values repository	Retrieves metrics and target values for the assessment components and offers an API to modify them
Continuous Certification Evaluation (CCE)	Forwards assessment results to the CCE component

Relevant sequence diagram/s


Evidence Gathering
Assessment
Orchestrator
Evaluation (WP4)
DLT
Database
register(toolId, monitorableMetricIds, ...)
register(toolId, monitorableMetricIds, ...)
Continuous Gathering
start_monitoring(metrics)
post_evidence(evidence)
get_target_value(metricId)
assess_evidence(evidence, target_value)
post_assessment_result(assessment_result)
post_assessment_result(assessment_result)
post_checksum(checksum)
post_assessmentResult(assessmentResult)

Current TRL	TRL4
Programming language	Go
License	Apache 2.0
WP and task	WP3: T3.1

Workflow	WF5

4.4.7 Evidence Collection and Security Assessment

These components are responsible for gathering evidence of CSP's fulfilment of technical measures, perform initial processing of the evidence, and pass it on to other MEDINA components[16].

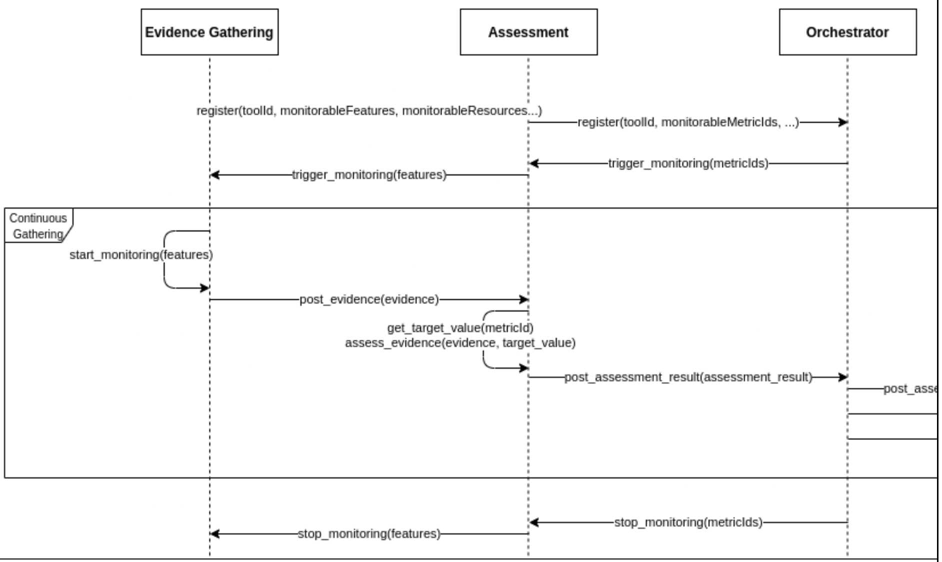
4.4.7.1 Evidence gathering tools

4.4.7.1.1 Wazuh

Component Name	Wazuh
Main functionalities	In general, Wazuh is a HIDS solution that provides the following functionalities: • Malware and intrusion detection • Log data analysis • File integrity monitoring • Vulnerability detection • Configuration assessment • (Limited) monitoring of data about AWS & Azure infrastructure with simple compliance assessment In MEDINA, Wazuh will be offered to the users as a tool to help CSPs satisfy compliance with certain EUCS controls as well as an evidence gathering tool.
Sub-components Description	It is composed of a Wazuh server and Wazuh agents. The agents are deployed on the individual monitored machines and communicate information about the detected anomalies to the server. The server includes the Wazuh manager component along with the ELK (ElasticSearch, Logstash, Kibana) stack for gathering, storing, and display of data. Custom integrations are possible to send alerts from Wazuh to any external component. Agents communicate with the server using Rsyslog. Wazuh is plugged into MEDINA with the Wazuh & VAT evidence collector component, which is responsible for extracting the data, relevant for MEDINA metrics, and transforming it into evidence, compatible with the security assessment component. It also includes two-way communication with the security assessment component (Clouditor).
Main logical Interfaces	Interface name / Description / Interface technology: Wazuh WUI / Main web UI / Web, based on Kibana ElasticSearch / / ElasticSearch HTTP API (REST)
Requirements Mapping	TEGT.C.01, TEGT.C.02 TEGT.S.08

[16] More details can be found in D3.5 [14].

Interaction with other components	Interfacing Component | Interface Description Security assessment (Clouditor) | Wazuh & VAT evidence collector component forwards every generated evidence to Clouditor through a gRPC interface.

Relevant sequence diagram/s	 Evidence Gathering
	Assessment
	Orchestrator
	register(toolId, monitorableFeatures, monitorableResources...)
	register(toolId, monitorableMetricIds, ...)
	trigger_monitoring(metricIds)
	trigger_monitoring(features)
	Continuous Gathering
	start_monitoring(features)
	post_evidence(evidence)
	get_target_value(metricId)
	assess_evidence(evidence, target_value)
	post_assessment_result(assessment_result)
	post_asse
	stop_monitoring(metricIds)
	stop_monitoring(features)

Current TRL	Based on existing open source Wazuh platform: TRL 9. Connector for integration with MEDINA (Wazuh & VAT evidence collector) is at TRL 4.
Programming language	C, Python, C++, Javascript
License	Open source: GNU GPL v2, Apache License v2.0.
WP and task	Task 3.2
Workflow	WP2, WP5

4.4.7.1.2 VAT

Component Name	**Vulnerability Assessment Tool (VAT)**
Main functionalities	The component provides the following functionalities: • Detection of web vulnerabilities by running integrated vulnerability scanners to scan web applications (OWASP ZAP[17], w3af[18]) • Network reconnaissance (running hosts, open ports – exposed services) using integrated Nmap[19] • Detection of vulnerable software (known vulnerable service versions) • Running custom scripts for detection of specific vulnerabilities or monitoring specific security metrics • Scheduling repeating tasks (vulnerability scans, monitoring, etc.) In MEDINA, VAT is offered to the users as a tool to help CSPs satisfy compliance with certain EUCS controls and as an evidence gathering tool.

[17] https://owasp.org/www-project-zap/
[18] http://w3af.org/
[19] https://nmap.org/

Sub-components Description	**Scheduler**: responsible for triggering scanning tasks according to the configured schedules **Docker interface**: a component managing the connection with the Docker runtime, executing the tasks by running appropriate docker images and obtaining their results **Frontend**: web UI management interface **RabbitMQ**: connection between the subcomponents **VAT-genscan**: integrating and orchestrating some vulnerability scanning tools and combining their results into a common report (based on Faraday CSCAN[20]) **Wazuh & VAT evidence collector**: a component responsible for extracting the data, relevant for MEDINA metrics, and transforming it into evidence, compatible with the security assessment component. It also includes two-way communication with the security assessment component (Clouditor) to send evidence and exchange configuration data.

Main logical Interfaces

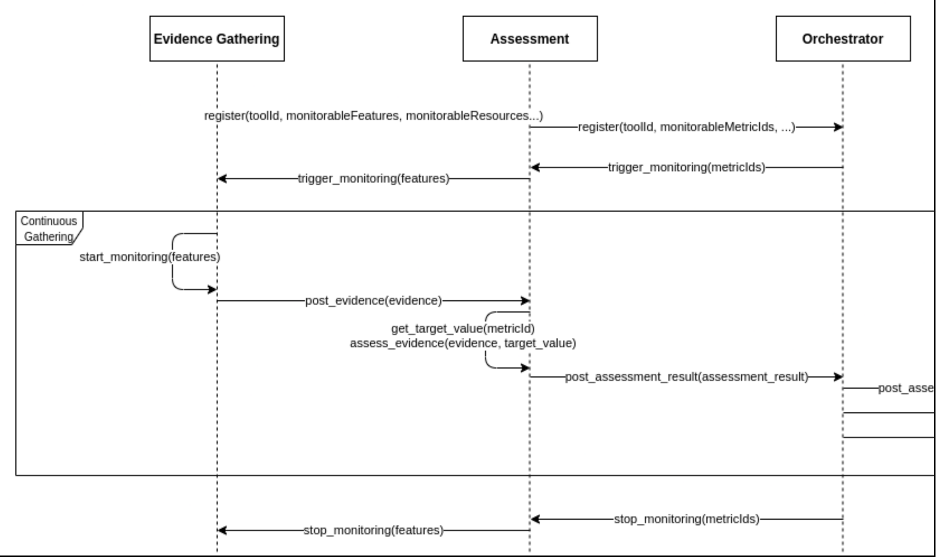
Interface name	Description	Interface technology
Scan reports output	Pushing the results of scan tasks (vulnerability reports)	RabbitMQ (AMQP), JSON
Management UI	Web UI to manage the scanning tasks and review their results	Web

Requirements Mapping	TEGT.C.01, TEGT.C.02 TEGT.S.08

Interaction with other components

Interfacing Component	Interface Description
Security assessment (Clouditor)	Wazuh & VAT evidence collector component forwards every generated evidence to Clouditor through a gRPC interface.

Relevant sequence diagram/s


Evidence Gathering
Assessment
Orchestrator
register(toolId, monitorableFeatures, monitorableResources...)
register(toolId, monitorableMetricIds, ...)
trigger_monitoring(metricIds)
trigger_monitoring(features)
Continuous Gathering
start_monitoring(features)
post_evidence(evidence)
get_target_value(metricId)
assess_evidence(evidence, target_value)
post_assessment_result(assessment_result)
post_asse
stop_monitoring(metricIds)
stop_monitoring(features)

[20] https://github.com/infobyte/faraday/tree/master/scripts/cscan

Current TRL	Based on existing Vulnerability Assessment Tool component developed in the scope of H2020 CYBERWISER. To be extended, adapted, and integrated in the MEDINA workflow. Current TRL: 4. Integrated vulnerability scanners used are separately developed components by their respective owners. Their TRLs are higher (8-9).
Programming language	Go, node.js, Javascript, Python, Bash.
License	The VAT platform is proprietary, closed-source (developed by XLAB). The VAT-genscan core component that integrates third-party vulnerability scanners and combines their results is released as open-source with Apache License v2.0. The scripts for deployment of the demo solution are also released under Apache License v2.0. Some sub-components and integrated vulnerability scanning tools are open source: • OWASP ZAP: Apache License • W3af: GNU GPL v2 • Nmap: Nmap Public Source License based on GNU GPL v2 • CSCAN framework to orchestrate scanners (part of Faraday): GNU GPL v3
WP and task	Task 3.2
Workflow	WP2, WP5

4.4.7.1.3 Cloud Evidence Collector

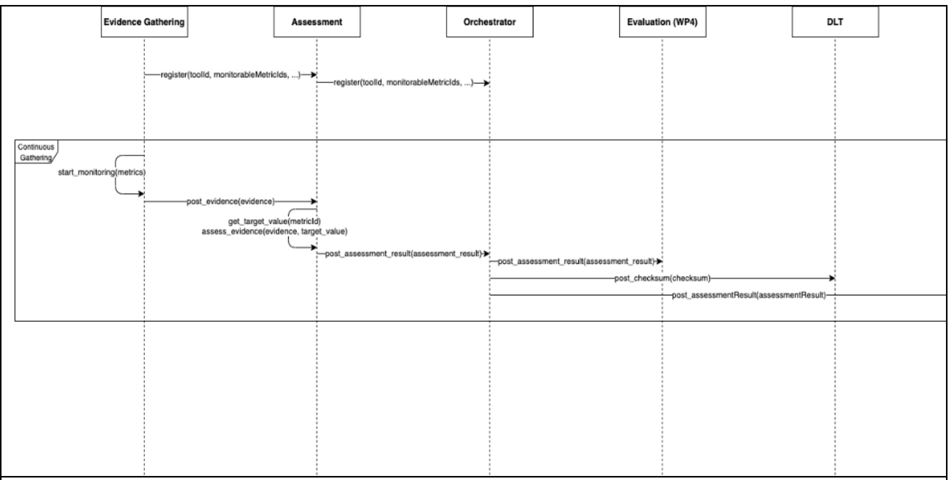
Component Name	**Cloud Evidence Collector**
Main functionalities	The component provides evidence gathering for cloud resources, like virtual machines, etc.
Sub-components Description	The evidence gathering discovers resources in cloud systems, like Azure and AWS, via their standard APIs and forwards this information to the assessment.
Main logical Interfaces	(see table below)
Requirements Mapping	TEGT.C.01, TEGT.C.02 TEGT.S.01, TEGT.S.02, TEGT.S.03, TEGT.S.04, TEGT.S.05, TEGT.S.09
Interaction with other components	(see table below)

Main logical Interfaces:

Interface name	Description	Interface technology
Assessment interface	An interface for providing evidence to be assessed against suitable metrics	gRPC
UI	A graphical user interface, e.g., for triggering discovery of resources (integrated with the Orchestrator)	Typescript / Svelte

Interaction with other components:

Interfacing Component	Interface Description
Orchestrator	Send assessment results

Relevant sequence diagram/s	 Evidence Gathering
	Assessment
	Orchestrator
	Evaluation (WP4)
	DLT
	register(toolId, monitorableMetricIds, ...)
	register(toolId, monitorableMetricIds, ...)
	Continuous Gathering
	start_monitoring(metrics)
	post_evidence(evidence)
	get_target_value(metricId)
	assess_evidence(evidence, target_value)
	post_assessment_result(assessment_result)
	post_assessment_result(assessment_result)
	post_checksum(checksum)
	post_assessmentResult(assessmentResult)

Current TRL	TRL4
Programming language	Go
License	Apache 2.0
WP and task	WP3: T3.1, T3.2
Workflow	WP2, WP5

4.4.7.1.4 Codyze

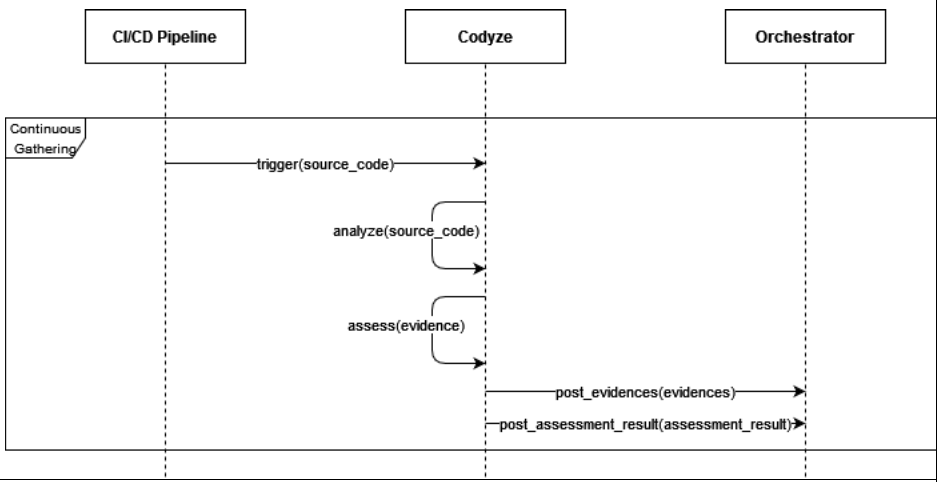
Component Name	**Codyze**
Main functionalities	The component provides the following functionalities: • Static code analysis • Validation of compliance to EUCS requirements in source code Thereby, Codyze maps findings from the source code to EUCS requirements. The resulting assessment results and evidence specified in the MEDINA data model are submit for storage and further interpretation to the Orchestrator.
Sub-components Description	**MARK** is a domain specific language to specify verifiable properties that source code must adhere to. It can, for example, restrict possible data values and their flow, or specify interactions between objects. A corresponding software library build on top of Xtext[21] provides the language grammar and parser functionality. In addition, a generated Eclipse plugin provides MARK specific editing support in Eclipse IDE. ***CPG*** is a library implementing a code representation based on the concept of a code property graph [15]. It's responsible for parsing source code and providing a graph-based code representation suitable for querying code properties. **Codyze library** provides the analysis engine for Codyze. It uses the CPG to parse source code. In addition, it uses the MARK library to parse MARK files. The Codyze library implements the analysis steps to interpret MARK rules and identify rule violations in source code. Assessed rules generate a finding that either certifies compliance or documents a rule violation.

[21] https://www.eclipse.org/Xtext/

	Interface name	Description	Interface technology
Main logical Interfaces	CLI	Codyze provides a command line interface. It can be used to call Codyze to analyse a set of files and produce results. It is suitable for example for a CI/CD pipeline. It generates reports in the SARIF format.	stdin/std out; file; format: SARIF (JSON)
	REST	Implementation of the OpenAPI REST API to communicate with the Orchestrator.	HTTP REST

Requirements Mapping	TEGT.C.01, TEGT.C.02 TEGT.S.06, TEGT.S.07, TEGT.S.08

	Interfacing Component	Interface Description
Interaction with other components	Orchestrator	Send evidence
	Orchestrator	Send assessment results

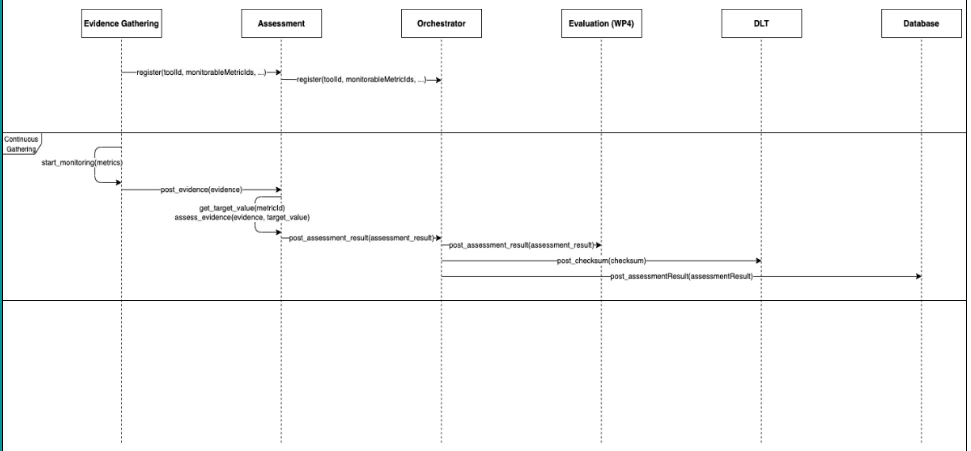
Relevant sequence diagram/s	 CI/CD Pipeline
	Codyze
	Orchestrator
	Continuous Gathering
	trigger(source_code)
	analyze(source_code)
	assess(evidence)
	post_evidences(evidences)
	post_assessment_result(assessment_result)

Current TRL	TRL4
Programming language	Kotlin, Java, Xtext
License	Apache 2.0
WP and task	WP3: T.3.3
Workflow	WP2, WP5

4.4.7.2 Security Assessment

Component Name	**Security Assessment**
Main functionalities	The component provides an API for evidence collectors to send evidence to, and assesses them according to pre-defined metrics and target values.
Sub-components Description	The evidence gathering discovers resources in cloud systems, like Azure and AWS, via their standard APIs and forwards this information to the assessment. The assessment compares the received evidence against pre-defined metrics and their target values and forwards the results to the orchestrator.

Main logical Interfaces	<table><tr><th>Interface name</th><th>Description</th><th>Interface technology</th></tr><tr><td>Assessment interface</td><td>An interface for providing evidence to be assessed against suitable metrics</td><td>gRPC</td></tr><tr><td>UI</td><td>A graphical user interface, e.g., for triggering discovery of resources</td><td>Typescript / Svelte</td></tr></table>
Requirements Mapping	EAT.01, EAT.02, EAT.03
Interaction with other components	<table><tr><th>Interfacing Component</th><th>Interface Description</th></tr><tr><td>Orchestrator</td><td>Send assessment results</td></tr></table>

Relevant sequence diagram/s	 Evidence Gathering
	Assessment
	Orchestrator
	Evaluation (WP4)
	DLT
	Database
	register(toolId, monitorableMetricIds, ...)
	register(toolId, monitorableMetricIds, ...)
	Continuous Gathering
	start_monitoring(metrics)
	post_evidence(evidence)
	get_target_value(metricId)
	assess_evidence(evidence, target_value)
	post_assessment_result(assessment_result)
	post_assessment_result(assessment_result)
	post_checksum(checksum)
	post_assessmentResult(assessmentResult)

Current TRL	TRL4
Programming language	Go
License	Apache 2.0
WP and task	WP3: T3.1, T3.2
Workflow	WP4

4.4.7.3 Evidence trustworthiness management

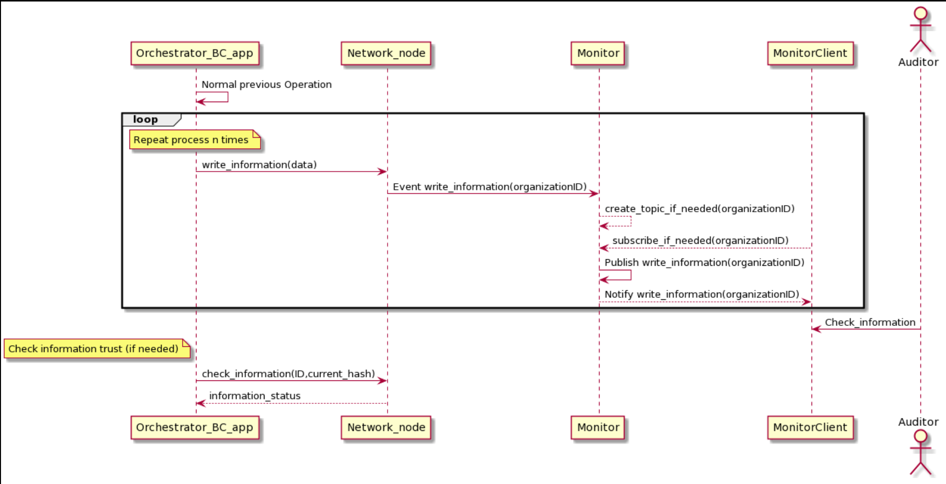
Component Name	**Evidence trustworthiness management system**
Main functionalities	The component provides the following functionalities: • Maintain an improved audit trail of evidence and assessment results. • Provide a record of information on a verifiable way (verification). • Provide a record of information on a permanent way (traceability). • Guarantee resistance to modification of stored data (integrity).
Sub-components Description	**Blockchain dApp** to be executed on the orchestrators for providing the information (evidence/assessment results) to be saved on the Blockchain. **Smart contract** deployed on Blockchain nodes for information (evidence/assessment results) writing and reading operations as well as events generation indicating the provision of new information. **Monitor tool** for subscription to the Blockchain based events and notification to the different monitors clients.

	Graphical monitor client for gathering all the information saved on the Blockchain (and be able to check it, without needed any interaction with the Blockchain).
Main logical Interfaces	

Interface name	Description	Interface technology
Blockchain dApp	Provides the required information to be saved on the Blockchain. Provides a way to check the information sabed on the Blockchain	API REST
Graphical Monitor	Provides a graphical interface to check information saved on the Blockchain	WEB

Requirements Mapping	List of requirements covered by this component: ETM.01, ETM.02, ETM.03, ETM.04, ETM.05
Interaction with other components	

Interfacing Component	Interface Description
Orchestrator	The Orchestrator will provide (and check, if needed) the information (evidence/assessment results) to be saved on the Blockchain by means of the Blockchain dApp interface.
Auditors	The auditors will check the information saved on the Blockchain by means of the graphical monitor interface.

Relevant sequence diagram/s	 Orchestrator_BC_app Network_node Monitor MonitorClient Auditor Normal previous Operation loop Repeat process n times write_information(data) Event write_information(organizationID) create_topic_if_needed(organizationID) subscribe_if_needed(organizationID) Publish write_information(organizationID) Notify write_information(organizationID) Check_information Check information trust (if needed) check_information(ID,current_hash) information_status
Current TRL	Based on exiting tools (Brokel-Tecnalia)
Programming language	Solidity, NodeJS
License	Proprietary. Copyright by Tecnalia.
WP and task	WP3 – Task 3.5 WP4 – Task 4.2
Workflow	WP5

4.4.8 Integrated User Interface

The goal of this component is to provide a primary point of access for MEDINA Framework: it integrates with existing authentication and guides users based on their authorization level to specific components UIs[22].

Component Name	Integrated User Interface (IUI)
Main functionalities	The component provides the following functionalities: • Provides a primary point of access for MEDINA framework • Provides the integration with the existing authentication • Provides the integration of all the separated components GUI into a single point of access • To guide the users based on their authorization level to specific components UIs
Sub-components Description	No subcomponents exist in the IUI
Main logical Interfaces	Interface name: IUI — Description: Main point of access to the framework, integrates all the other micro frontends — Interface technology: HTTPS (browser)
Requirements Mapping	List of requirements covered by this component IUI.01, IUI.02, IUI.05, IUI.06
Interaction with other components	(see table below)
Relevant sequence diagram/s	
Current TRL	TRL7
Programming language	AngularJS
License	Proprietary. Copyright by HPE
WP and task	WP5 – Task 5.3
Workflow	WP2, WP7

Main logical Interfaces:

Interface name	Description	Interface technology
IUI	Main point of access to the framework, integrates all the other micro frontends	HTTPS (browser)

Interaction with other components:

Interfacing Component	Interface Description
Catalogue of controls and metrics	Integrates the Catalogue of controls and metrics UI
CNL Editor	Integrates the CNL Editor UI
Continuous Certification Evaluation	Integrates the Continuous Certification Evaluation UI
Organizational evidence gathering and processing	Integrates the Organizational evidence gathering and processing UI
Orchestrator	Integrates the Orchestrator UI
Static Risk Assessment and Optimisation Framework	Integrates the Static Risk Assessment and Optimisation Framework UI

[22] More details can be found in D5.3 [3].

4.5 MEDINA Deployment Models

This section briefly classifies the MEDINA tools attending to the technological framework required for its execution and deployment. It presents a classification of the KRs based on the envisioned deployment models at this stage of the project. This analysis completes the presented in previous version of this deliverable from the technical perspective and in deliverable D7.6 [16] from the business model impact perspective. The selection of the deployment model has been influenced by the requirements of the certification stakeholders (i.e., auditors, CABs and NCCAs).

4.5.1.1 Web tools (SaaS)

These are tools accessible through any compatible browser. In MEDINA some of the tools will be offered as service, which are invoked for performing different functionalities. Internally these tools can be deployed following the multi-cloud approach. It is envisioned that the following MEDINA Key Results, tools and components will be offered as Web Tools (SaaS):

- Catalogue of controls and metrics (KR1)
- Risk based selection of controls (KR2)
- Certification language (KR3)
- Cloud certificate evaluator (KR5)
- Risk-based Auditor Tool (KR6)
- Self-Sovereign Identity-based certificates management (KR6)
- The integrated MEDINA framework

4.5.1.2 Containerized tools

Containers are lightweight software components that bundle the application, its dependencies, and its configuration in a single image, running in isolated user environments on a traditional operating system on a traditional server or in a virtualized environment [17]. Containerization of an application has several advantages as, for example:

- Portability between different platforms and clouds: write once, run anywhere. An application in a container behaves the same regardless the environment where it is deployed, avoiding issues with operating system versions.
- Efficiency through using far fewer resources than VMs and delivering higher utilization of compute resources.
- Improved security by isolating applications from the host system and from each other.
- Faster app start-up and easier and cost-effective scaling.
- Flexibility to work on virtualized infrastructures or on bare metal servers.
- Easier management since install, upgrade, and rollback processes can be built into the Kubernetes platform.
- It allows developers to integrate with their existing DevOps environment (more about DevOps infrastructure can be read in Section 5.

In this case the web application can be installed locally. The selection between this case and the SaaS model depends mainly on the exploitation strategy decided for each Key Result or component, and also on the needs of the users with respect to the usage of the component (e.g., the Evidence storage tool shall be internal (local) to the CSP)). The following tools fit into this category:

- Catalogue of controls and metrics (KR1)
- Risk based selection of controls (KR2)

- Certification language tools (KR3)
- Cloud certificate evaluator (KR5)
- Risk-based Auditor Tool (KR6)
- Continuous evidence management tool (KR4)

5 MEDINA DevOps Infrastructure and CI/CD and Verification Strategy

This section provides updates related to the design of the MEDINA infrastructure and the definition of the CI/CD and verification strategy previously described in D5.1 [1].

We have added details for the design of the Jenkins pipelines to be used in the MEDINA framework, with focus on the technologies adopted such as the containerization with Docker and the use of the Kubernetes container orchestrator. Moreover, few changes in the CI/CD supporting tools have been produced and we provided updates for the three environments indicated in D5.1 concerning the resources and the status.

The unchanged parts, those that define the CI/CD strategy, quality and assurance methods, and the containerization deployment model, have been moved to *Appendix D. CI/CD Strategy*.

5.1 Implemented CI/CD pipeline

This section describes the CI/CD pipeline focusing on the strategies adopted to make each partner independent to create its own pipeline (Seed Jobs) and the adoption of the containerization technology for the release of part of the CI/CD tools and environments.

The implemented pipelines are three: the **Build** pipeline, the **Deploy** pipeline and the **Security** pipeline. As described in *Appendix D. CI/CD Strategy*, we make use of Jenkins as CI/CD orchestration tool. This tool contains a particular plugin called *Seed Job* which aids to automate the creation of the three ad-hoc pipelines designed for the MEDINA framework.

The process consists of filling in a form with parameters such as:

- Work Packages/Task folder where the Jenkins Jobs will be created
- Job basename, that typically is the component name
- GitLab URL, retrieved from the TECNALIA GitLab web interface
- Build template, chosen from a preconfigured template or customizing it manually
- Dockerfile, the name of the dockerfile to build the container image
- Image, the name of the container image pushed to the private Artifactory registry
- Kubernetes manifests, used for deployment into Kubernetes cluster

Once these details are provided, the Seed Job automatically creates the three standardized pipelines for build, deploy and security. *Figure* 8 shows how these pipelines work.

Build pipeline

In the **Build** pipeline, the code is checked out from GitLab and a docker container is setup to execute the other build stages. Then there is the compile, testing and package stages and partners can customize them depending on the build tools used. The next three stages are referred to the Docker image building and pushing to the Artifactory repository. By default, the image is pushed with the "latest" tag but there is an optional phase to tag it in a different manner. At last, if no errors occur the Deploy Job is automatically called.

Deploy pipeline

The **Deploy** pipeline deals with the release of the components in the Kubernetes cluster. As described in the D5.3 (Section 2.1) [3], the Kubernetes cluster is divided in two isolated and virtual environments, "dev" and "test". Jenkins is configured to access to the Kubernetes cluster with exchanged credentials to enable the application of Kubernetes manifests to release the configuration to the environment. By default, the Deploy pipeline releases the component on the “dev” environment. Partners can also use this pipeline to manually release the component on the "test" environment changing it with one click on the Jenkins platform. The Security pipeline is automatically triggered upon a successful Build and Deploy.

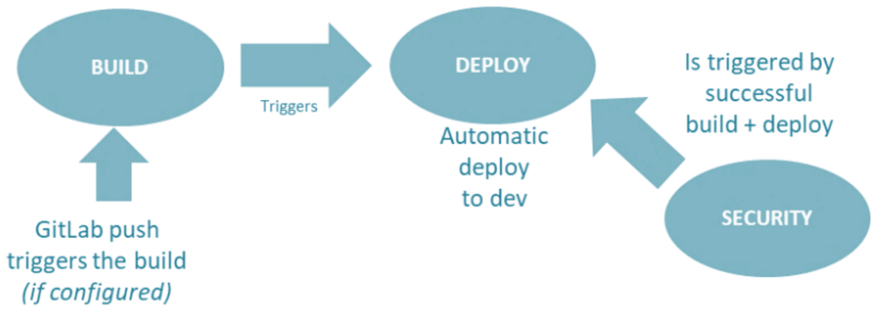

BUILD
Triggers
DEPLOY
Is triggered by successful build + deploy
Automatic deploy to dev
SECURITY
GitLab push triggers the build (if configured)

Figure 8. CI/CD Pipelines in MEDINA (source D5.3 [3])

Security pipeline

For assessing Quality & Assurance in the entire toolchain, the **Security** pipeline includes security analysis of the software artefacts at different levels: Static Code analysis for checking the source code, Container security for scanning vulnerabilities into the container packages and Software Composition Analysis (SCA) for spotting security issues in third party libraries. Each type of analysis is running by a specific tool. Concerning the first two types, the tools (respectively Semgrep [18] and Anchore [19]) are running into containers called into the security pipeline. After the analysis is done, these containers, in which the tools are installed, are destroyed but the output file of the analysis persists. In this way, it can be easy and fast to update the tool to the latest version, forcing the download of the latest tag of the container images. Regarding the SCA, the tool is OWASP Dependency Check, installed via command line. The latest stage of this implemented Security pipeline has foreseen a further step to make possible to see all the analysis results of the security controls in a unique view thanks to the use of a vulnerability report aggregator tool called DefectDojo [20], which will be described better in Section 5.2.1.

5.2 Infrastructure in MEDINA framework

This section describes the updates about the CI/CD tools chosen and the current state of the Development, Test and Production environments.

5.2.1 CI/CD supporting tools

Table 9 presents a new updated toolset with respect to the list provided in D5.1 [1].

Table 9. Software development tools

MEDINA – Software Development Tools	
Category	**Tool**
Collaborative Code & Version Control	GitLab
Build Automation	Maven and Gradle
Artefact Repository	Artifactory
Continuous Integration	Jenkins
Testing	JUnit and REST Assured
Bug Tracking	GitLab issues
Q–A - static code analysis *	Semgrep
Q–A - dynamic code analysis	OWASP ZAP
Q–A - container security	Anchore
Q&A- vulnerability report aggregator **	DefectDojo

(*) The first change concerns the tool to perform Static Code Analysis. Despite the initial selection of FindBugs and SonarQube, we now turn our choice to Semgrep [18] because it supports many languages, and is easy to integrate with Jenkins. Semgrep [18] is a fast, open-source, static analysis tool for finding bugs and enforcing code standards at editor, commit, and CI time. Semgrep analyses code locally on the user's PC or in its build environment, there is no need to upload the code. It supports 20+ languages such as C#, Go, Java, Python, Ruby, etc.

(**) The second change is the addition the new vulnerability report aggregator tool named DefectDojo [20]. DefectDojo is an open-source DevSecOps and vulnerability management tool. DefectDojo streamlines the application security testing process by offering features such as importing third party security findings, merging and de-duping, templating, report generation and security metrics, maintaining product and application information, and pushing findings to systems such as JIRA or Slack. Figure 9 shows an example of the DefectDojo dashboard.

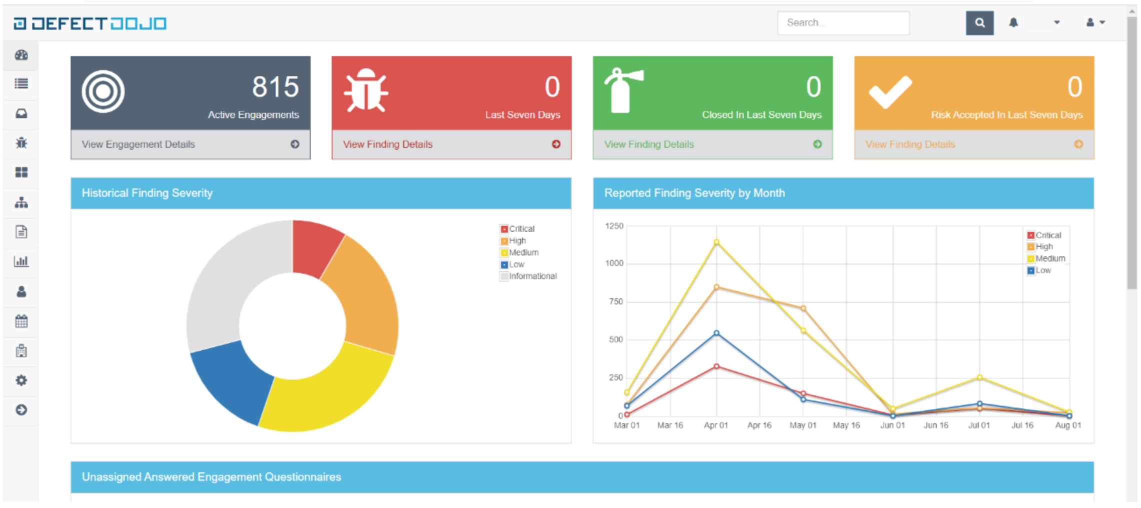


Figure 9. DefectDojo Dashboard

5.2.2 Development and Test Environment

This section describes the updates to the MEDINA Development and Test environments. These environments are implemented in a Kubernetes cluster and run on three VMs hosted by TECNALIA and based on Ubuntu 20.04.

A dedicated VM hosts the CI/CD orchestration engine, the supporting tools, as well as the Kubernetes cluster management. Its current resources status is:

- Memory: 16G
- Cores: 4
- HDD: 400G

The CI/CD is reachable at: *cicd.medina.esilab.org*.

The Development and Testing Environments are implemented on a 3-node container cluster that virtualizes both environments making them independent and isolated (see Figure 10). Such environments will run the MEDINA micro-services (containers) depending on their current maturity level: development will be highly unstable, while testing will host the reference implementation of MEDINA available for integration testing (a more stable codebase).

The resources dedicated to each machine have been increased to meet the needs of the partners' components. The current status is:

- Memory: 16G
- Cores: 8
- HDD: 200G

The 200G of storage of each node are organized as a distributed filesystem for data persistent layer and managed by Rook/Ceph [21]: the Kubernetes cluster offers 200G of storage and the data are duplicated among the three nodes as described in Section 2.1.1 of D5.3 with more details [3].

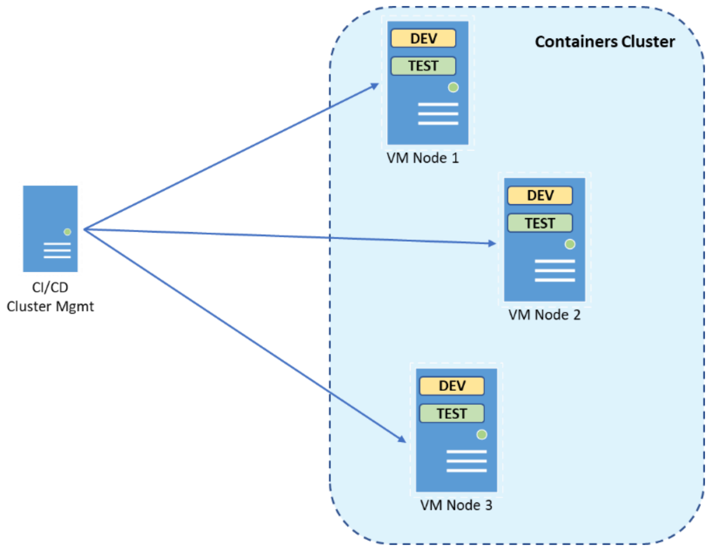


Figure 10. Development and Testing Environments

5.2.3 Validation Environment

The components released in the Kubernetes "Test" environment are validated by the Fabasoft and Bosch environments (see Sections 2.1.2 and 2.2.2). For this reason, the "Production Environment", as mentioned in *Appendix D. CI/CD Strategy,* will be replaced by a validation environment that relies on the MEDINA framework deployed in "Test".

6 Conclusions

In this deliverable -D5.2- we have described the general MEDINA framework in month 24 of the project. The document builds on the first version of the deliverable, D5.1 [1], produced a year ago. The document maintains the same structure, but modifying and extending the content where necessary.

The document presents the functional and non-functional requirements for each MEDINA component. A total of 88 functional and 12 non-functional requirements are presented, corresponding to 15 different components, being two of them new components defined in the second year of the project. The evolution of the requirement status has been presented. An overall of 21 new requirements have been defined in this period, for a total of 88 requirements. On the other side, 6 of them have been discarded. Regarding the status of the implementation, 4% of the requirements are still to be started; in other words, 96% of the requirements are already partially or fully implemented, and most of them (53%) are fully implemented.

The description of the architecture of MEDINA framework has also been updated. That includes updated descriptions of the components, their structural and behavioural description, through the use of the "component card" templates. Additionally, the latest version of the data model of MEDINA has been presented, along with the general architecture, grouping the components in eight groups or "building blocks" depending on the features and participation in the workflow.

Finally, the document describes the development and integration methodology followed in the project since its inception, and the infrastructure employed to construct and demonstrate the solution.

This is the second and final version of the requirements documentation in MEDINA. With the project being at the end of its second year, not many changes are expected in the coming months regarding components definition, general architecture, or communication interfaces. The next steps will be dedicated to finalising the implementation of the requirements, to have a complete set of components developed, and to integrate and test them in the MEDINA infrastructure.

7 References

[1] MEDINA Consortium, "D5.1 MEDINA Requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy," 2021.

[2] MEDINA Consortium, "D6.3 Use cases development and validation prototypes," 2022.

[3] MEDINA Consortium, "D5.3 MEDINA integrated solution-v1," 2022.

[4] ENISA, "EUCS – Cloud Services Scheme," [Online]. Available: https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme. [Accessed October 2022].

[5] MEDINA Consortium, "D6.2 Use cases specification and evaluation methodology v2," 2021.

[6] ISO/IEC/IEEE International, "Systems and software engineering—Vocabulary," 2017.

[7] MEDINA Consortium, "MEDINA Annex 1 Part B - GA Number 952633," 2020.

[8] MEDINA Consortium, "D4.2 Tools and Techniques for the Management and Evaluation of Cloud Security Certifications," 2022.

[9] MEDINA Consortium, "D5.4 MEDINA integrated solution-v2," 2023.

[10] MEDINA Consortium;, "D2.1 Continuously certifiable technical and organizational measures and catalogue of cloud security metrics-v1," 2021.

[11] MEDINA Consortium, "D2.4 Specification of the Cloud Security Certification Language - v2," 2022.

[12] MEDINA Consortium, "D2.7 Risk-based techniques and tools for Cloud Security Certification-v2," 2022.

[13] MEDINA Consortium, "D4.4 Methodology and tools for risk-based assessment and security control reconfiguration-v1," 2022.

[14] MEDINA Consortium, "D3.5 Tools and techniques for collecting evidence of technical and organisational measures-v2," 2022.

[15] F. Yamaguchi, N. Golde, D. Arp and K. Rieck, "Modeling and Discovering Vulnerabilities with Code Property Graphs," in *2014 IEEE Symposium on Security and Privacy*.

[16] MEDINA Consortium, "D7.6 Exploitation and sustainability Report-v1," 2022.

[17] IBM, "The Benefits of Containerization and What It Means for You," 6 February 2019. [Online]. Available: https://www.ibm.com/cloud/blog/the-benefits-of-containerization-and-what-it-means-for-you. [Accessed 26 10 2021].

[18] "Semgrep," [Online]. Available: https://semgrep.dev/docs/. [Accessed October 2022].

[19] [Online]. Available: https://anchore.com/. [Accessed October 2022].

[20] “DefectDojo,” [Online]. Available: https://www.defectdojo.org/. [Accessed October 2022].

[21] “Rook/Ceph,” [Online]. Available: https://rook.io/docs/rook/v1.10/Getting-Started/intro/. [Accessed October 2022].

[22] MEDINA Consortium;, "D6.1 Use cases specification and evaluation methodology," 2021.

[23] S. Madsen, "How to Prioritize with the MoSCoW Technique," October 2017. [Online]. Available: https://www.projectmanager.com/training/prioritize-moscow-technique. [Accessed March 2018].

[24] F. Emily, in *DevOps For Dummies*, For Dummies, 2019.

[25] OWASP, “OWASP ORG,” [Online]. Available: https://owasp.org/. [Accessed October 2022].

[26] OWASP, “OWASP Top Ten Project,” [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed October 2022].

[27] Mitre corporation, “Common Weakness Enumeration,” 14 10 2021. [Online]. Available: https://cwe.mitre.org/. [Accessed October 2022].

[28] "CISQ Standard," [Online]. Available: https://www.it-cisq.org/standards/code-quality-standards/. [Accessed October 2022].

[29] "Quality Assurance in the DevOps strategy," [Online]. Available: https://virtualrealitybrisbane.com/quality-assurance-in-the-devops-strategy/. [Accessed October 2022].

[30] "Docker," [Online]. Available: https://www.docker.com. [Accessed October 2022].

[31] "Gitlab," [Online]. Available: https://gitlab.com/gitlab-org/gitlab-foss/. [Accessed October 2022].

[32] "Subversion," [Online]. Available: https://subversion.apache.org/features.html. [Accessed October 2022].

[33] "Maven vs Gradle," [Online]. Available: https://gradle.org/maven-vs-gradle/. [Accessed October 2022].

[34] "Difference between gradle and maven," [Online]. Available: https://www.geeksforgeeks.org/difference-between-gradle-and-maven/. [Accessed October 2022].

[35] "Nexus Repository," [Online]. Available: https://www.sonatype.com/nexus/repository-oss. [Accessed October 2022].

[36] "JFrog Artifactory," [Online]. Available: https://jfrog.com/artifactory/. [Accessed October 2022].

[37] "Jenkins CI/CD server," [Online]. Available: https://jenkins.io. [Accessed October 2022].

[38] "Tekton," [Online]. Available: https://tekton.dev/. [Accessed October 2022].

[39] "Kubernetes," [Online]. Available: https://kubernetes.io/it/. [Accessed October 2022].

[40] "Argo-cd vs Tekton vs Jenkins X," [Online]. Available: https://platform9.com/blog/argo-cd-vs-tekton-vs-jenkins-x-finding-the-right-gitops-tooling/. [Accessed October 2022].

[41] "Jenkins vs Gitlab CI/CD," [Online]. Available: https://www.lambdatest.com/blog/jenkins-vs-gitlab-ci-battle-of-ci-cd-tools/. [Accessed October 2022].

[42] "JUnit," [Online]. Available: https://junit.org/junit5/. [Accessed October 2022].

[43] "TestNG," [Online]. Available: https://testng.org/doc/. [Accessed October 2022].

[44] "Rest Assured," [Online]. Available: https://rest-assured.io/. [Accessed October 2022].

[45] "Jira," [Online]. Available: https://www.atlassian.com/it/software/jira. [Accessed October 2022].

[46] "Track," [Online]. Available: https://trac.edgewall.org/. [Accessed October 2022].

[47] "Bugzilla," [Online]. Available: https://www.bugzilla.org/about/. [Accessed October 2022].

[48] "SpotBugs," [Online]. Available: https://spotbugs.github.io/. [Accessed October 2022].

[49] "Find Security Bugs," [Online]. Available: https://find-sec-bugs.github.io/. [Accessed October 2022].

[50] "SonarQube," [Online]. Available: https://www.sonarqube.org/. [Accessed October 2022].

[51] OWASP, "OWASP Dependency Check," [Online]. Available: https://owasp.org/www-project-dependency-check/. [Accessed October 2022].

[52] OWASP, "OWASP ZAP," [Online]. Available: https://www.zaproxy.org/. [Accessed October 2022].

[53] [Online]. Available: https://www.aquasec.com/products/trivy/. [Accessed October 2022].

8 Appendix A. Requirements Management in MEDINA

This section is devoted to explaining the process followed to collect, manage, prioritise, and document requirements in MEDINA. It remains unchanged with respect to the previous version in D5.1 [1]. Hence, it has been moved to an Appendix.

8.1 Methodology for requirements elicitation

In MEDINA a combined top-down and bottom-up approach is followed to implement the requirements gathering process (see Figure 11). Therefore, the requirements are elicited in parallel in two strands:

i) Generic functionalities of the MEDINA framework. That is, the functionalities MEDINA aims to offer as its value propositions
ii) Use Cases (UC) requirements for MEDINA. That is, what UCs expect from MEDINA components. Eventually, these two strands must merge.

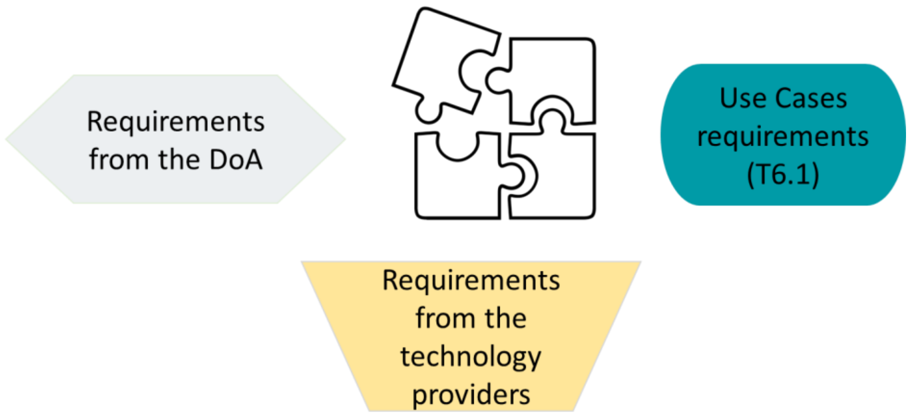

Requirements from the DoA
Use Cases requirements (T6.1)
Requirements from the technology providers

Figure 11. Different requirements sources in MEDINA

The elicitation of requirements for the MEDINA framework will be fed through several sources:

- **Requirements coming from the MEDINA action specification**: The first version of the requirements are elicited from the Description of Action (DoA [7]) by each responsible technical partner, and detailed based on their knowledge and technical discussions held during the requirements gathering process.
- **Requirements coming from the Use Cases**: The Use Cases propose functionalities for the MEDINA framework, so that the offered features can cover their needs. This is done in the context of WP6.
- **Requirements from the technical providers**. The technical providers may provide new functional requirements based on the decisions made during the development of the design of the different tools, the definition of the workflows in MEDINA and the set-up of the MEDINA data model.

8.2 Requirements gathering and prioritization process

The process followed in MEDINA for the elicitation of requirements can be seen in Figure 12. Legend for the figure is as follows:

- Activities performed in this work package (WP5) are marked in grey
- Activities marked in green are performed in WP6 (use cases validation).

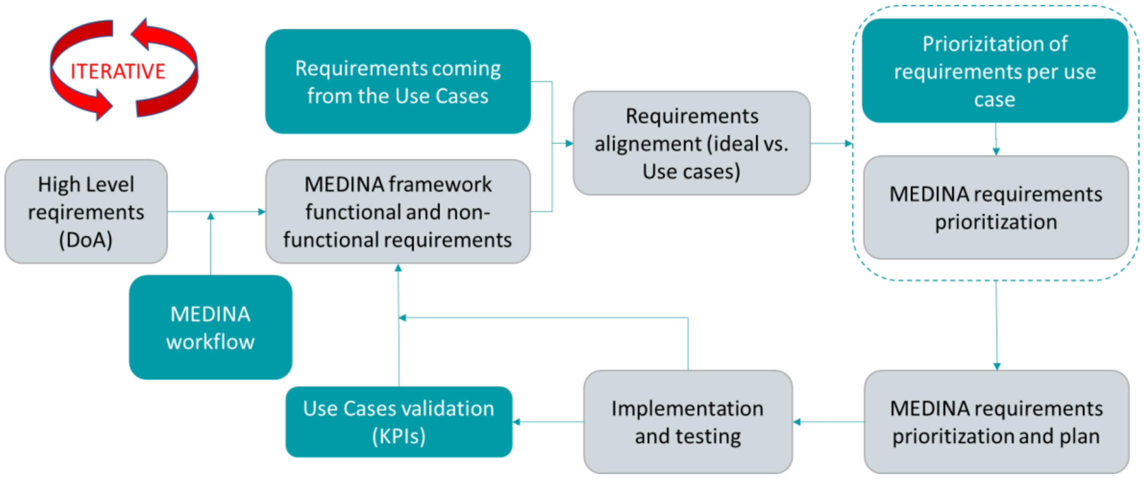

ITERATIVE
Requirements coming from the Use Cases
Requirements alignement (ideal vs. Use cases)
Priorizitation of requirements per use case
MEDINA requirements prioritization
High Level reqirements (DoA)
MEDINA framework functional and non-functional requirements
MEDINA workflow
Use Cases validation (KPIs)
Implementation and testing
MEDINA requirements prioritization and plan

Figure 12. Process followed in MEDINA for requirements gathering and prioritization

The process followed in MEDINA for the elicitation of requirements is described as follows:

- High-level requirements, that is, what MEDINA aims to offer, are elicited from what it is written in the DoA [7]. These requirements involve both functional and non-functional aspects that the MEDINA framework should provide.
- These initial list of requirements is enriched with the understanding on how the workflow of information happens among the different components through the description of the MEDINA workflow (for details of the workflow consult deliverable D6.3 [22]) which ends up with the definition of the MEDINA data model (see details in section 4.3).
- The result of the activities described beforehand have been decomposed into functional and non-functional requirements.
- In parallel the Use Cases elicit their requirements for the MEDINA components. This is done in the context of WP6.
- The technical partners then align the generic requirements elicited in previous steps with the requirements gathered by the use cases, and reformulate them, when needed, to end up with a consensus version.
- Based on the final set of requirements (agreed in the previous step), both use cases and technical partners will prioritize them, taking into consideration not only use cases needs but also baseline requirements that affect other requirements, and in the event, it was not implemented, the related functionality would not be delivered in a successful manner.
- The outcome of this activity results in a prioritization matrix (see Section 3.2.3), that indicates for each release which requirements will be implemented and will therefore be able to be validated by the use cases.
- Requirements will then be implemented into functionalities. The implementation includes the architectural design, the coding, testing (unit and integration) and deployment. During this phase, and especially during the testing activities, new requirements or improved requirements may arise. These new requirements are carefully analysed by the technical partners in order to avoid scope creep, before deciding if they can or cannot be accepted. If they are accepted, the functional requirements list will be updated.
- Use cases validate the functionalities following the evaluation plan defined in D6.1 [22]. The evaluation can result in new requirements, as well as in updated versions of the current requirements. As in step 8, these new requirements are carefully analysed by

the technical partners in order to avoid scope creep, before deciding if they will or will not be accepted.

Furthermore, several traceability matrixes are maintained, to keep all the relationships affecting the requirements up to date. These matrixes are included in the Section 3.2.

8.3 Requirements documentation

Documenting requirements is a key issue in every software project. In the case of MEDINA, the requirements are defined to provide an understanding of what will MEDINA do, i.e., the functionalities.

For the prioritization of the requirements, the MoSCoW method [23] has been followed. The MoSCoW method allows to define clear priority levels while also determining which functionalities will be developed in each of the project iterations. The prioritization levels of MoSCoW can be defined as follows:

- **M (Must)**: mandatory requirements. These requirements will be included definitely in the release.
- **S (Should)**: requirements that should be included in the release or in the final version. The inclusion of these type of requirements must not affect the 'must' requirements and they will only be included in the case there is additional time of capacity.
- **C (Could)**: requirements that could be included, because they provide nice-to-have functionalities. These shall only be implemented when the M and S have been successfully implemented.
- **W (Won't have)**: requirements that will not be included but they could be delivered some time as additional or extended functionalities.

Must and Should requirements are prioritized for the first versions of the components, while Could requirements are normally added towards the final versions of the components.

In MEDINA, requirements are reported in the requirements document, and are described as follows:

- **Requirement id**: unique identifier of the requirement
- **Short title**: short description of the requirement
- **Description**: more detailed description of the requirement. This is especially relevant for the creation of the test cases.
- **Status**: Proposed / Accepted / Rejected / Work in Progress / Fully implemented
- **Priority**: Must have / Could have / Should have / Won't have
- **Related KR**: which MEDINA result is affected by this requirement
- **Reference**: Source of the requirement

9 Appendix B. Use Cases Definition

9.1 UC1: European Certification of Multi-cloud backends for IoT solutions

Bosch's validation use case (European Certification of Multi-cloud backends for IoT solutions) applies the MEDINA's framework to the multi-cloud architecture shown in Figure 13. Currently, UC1 has been rolled out leveraging the approach and testbed presented in Sections 2.1.1 and 2.1.2 respectively.

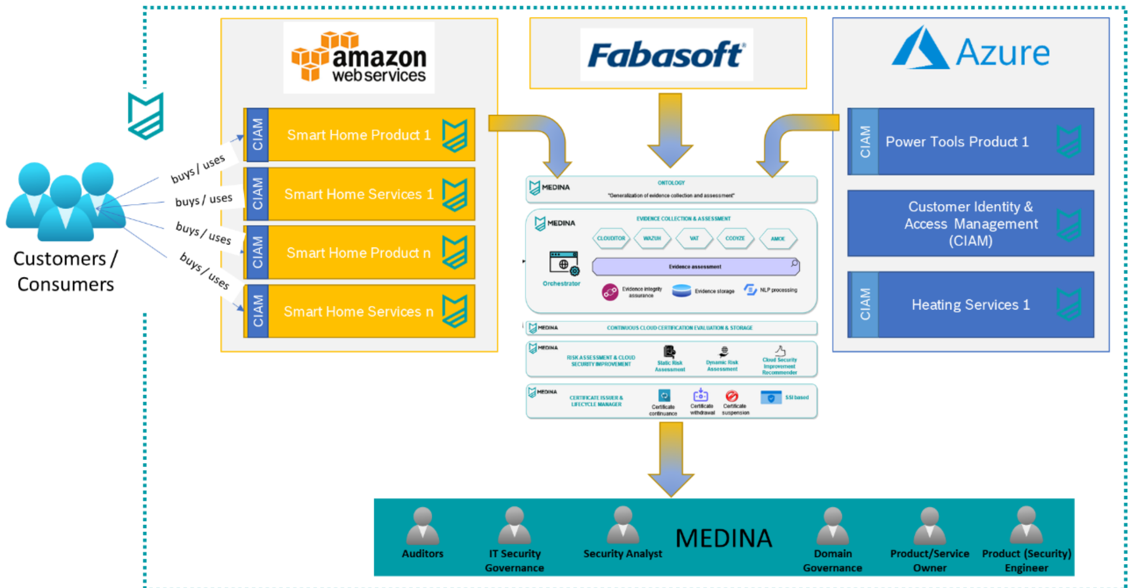


Figure 13. High-level view of Use Case 1 deployment.

9.2 UC2: European Cloud Service Provider SaaS public & private cloud

Fabasoft is a European software manufacturer and cloud provider. The software products and cloud services from Fabasoft ensure the consistent capture, sorting, process-oriented handling, secure storage, and context-sensitive finding of all digital business documents. These functions are used in both on-premises installations, as well as in Software as a Service (SaaS) cloud solutions. Beyond that, the Fabasoft appliance concept offers a direct way to provide customers with standardized complete systems (hardware and software) for use in their own data processing centres.

Fabasoft is already compliant with the following relevant standards or certifications: ISO 9001; ISO 20000-1; ISO 27001 including ISO 27018 controls; BSI C5:2017 (C5:2020 audit is currently in progress); ISAE 3402 Type 2; and ISAE 3000 SOC 2 Type 1.

The motivation of Fabasoft to participate in MEDINA and to provide this use case is the business driver of cost efficiency behind the idea of a successful automated, continuous audit approach. Currently, an audit for a BSIC5:2020 attestation at Fabasoft follows the traditional conventions and splits into three phases: set-up, compliance-evaluation, and re-evaluation.

1. **Set-up-Phase**: This phase is a one-time activity for each new compliance framework that Fabasoft applies for. The Compliance Manager—responsible for organizing the compliance process—has to select the applicable categories of the BSI C5:2020 framework. Together with a system description, this comprises the Statement of Applicability (SoA).

In a next step, the Compliance Manager delegates all Security Controls collected in the SoA internally to key-experts and -employees who are able to provide compliance-solutions for a sub-set Security Controls to meet their requirements and implement the evidence collection. These implementations are called Internal Controls and thus the responsible person for such a control is called Internal Control Owner (ICO).

The combination of SoA and implementation of Internal Controls is forwarded to the Auditor and assessed for first feedback and starting point of the next phase.

2. **Compliance-Evaluation-Phase**: This phase is defined to be a one-time activity for each new Security Control Framework that Fabasoft applies for.

   For a BSI C5:2020 attestation this phase follows an annual standard and evidence is collected over the course of 12 months for each Internal Control and then audited over the course of approximately 3 – 4 weeks between the Auditors, the Compliance Manager and all responsible ICOs to verify the correctness of the controls and the management of incidences if they occurred.

3. **Re-Evaluation-Phase**: This phase is defined to be a yearly repetition of the Compliance-Evaluation-Phase. The difference is that it also involves verification and updates of controls in place and checks the correctness of responsibilities of ICOs. It is a plan-do-check-act cycle and culminates in a 3 – 4 weeks audit with all responsible persons every 12 months.

Problem Statement: By looking at the three described phases, it becomes obvious that this is not only a costly undertaking with respect to the service costs of the auditing instance but also very resource intensive for a company like Fabasoft. For comparison, at early 2021, Fabasoft has had about 300 employees and involved nearly 12 people in a 4-week audit for BSI C5: 2020. These figures indicate a recurring high-cost involvement for a CSP when it comes to this framework. Currently there is no reason to believe that this will be any different for the upcoming EUCS.

The ultimate goal for Fabasoft in this project is to achieve a framework that allows for an (almost) automated, continuous audit process when applying a scheme like BSI C5 or EUCS. To achieve that we expect MEDINA to offer methods and tools to analyse information either directly – via tools like Clouditor – or indirectly by being able to process output from their internal monitoring tools or currently active scripts we implemented for the traditional BSI C5 evidence collection. We also expect MEDINA to offer us tools and methods to fetch information about Security Controls from some kind of repository that are translated into a rule-based language so that an internal expert can implement the security measures necessary to comply to the rules stated in that Security Control. This is, offer a framework that is less prone to different interpretation for Security Controls and offers more clear instruction on what to report for each Security Controls to achieve compliance to a framework like EUCS.

9.3 List of Use Cases requirements

This section offers the list of the Use Cases requirements collected in WP6. As mentioned before, they are extracted from deliverable D6.3 [2], where a more detailed description of each of these requirements can be found. The table includes actual requirements only. For clarity, the discarded ones with respect to previous versions (D6.1 [22] and D6.2 [5]) have been removed, and the new requirements have been added. The possible changes in the list itself or in the status of the requirements will be reflected in future versions of the WP6 deliverables.

The requirements are divided into "Common Requirements" i.e., those not directly related to one of the use cases or that might occur in both, and "Use Case Specific Requirements" i.e., those only related to one use case and do not directly or indirectly apply to the other use case.

Table 10. List of Use Case requirements

Type	Req. ID	Short Title
Common	UC00.01	MEDINA Audit API
	UC00.02	Repository of Security Controls
	UC00.04	Secure Evidence Storage
	UC00.05	Evidence Mapping
	UC00.06	Security Controls Translator
	UC00.07	Define Measurement Targets
	UC00.08	Add Traditional Audit results to MEDINA results
	UC00.09	Request for Change on Assessment Rules
	UC00.13	Auditor access to Assessment Results
	UC00.14	Compliance Status notification
	UC00.16	MEDINA Measurement Target run-time change
	UC00.17	MEDINA complies to EUCS
	UC00.18	MEDINA tool modularity
	UC00.19	Mapping of Frameworks
	UC00.20	Scalable Framework
	UC00.21	Interoperability API
	UC00.22	Data collection extent
	UC00.23	Retention of evidence
	UC00.24	Compliance Certification Status
	UC00.25	Efficiency Improvements (Automation)
	UC00.26	Efficiency Improvements
	UC00.27	Monitoring of Organizational Measures
	UC00.28	Report Generation
	UC00.29	Unified Graphical User Interface
	UC00.30	Internal Regulation Checks Support
	UC00.31	Trustworthiness of Evidence
UC1 (BOSCH)	UC01.1	Compliance Status Aggregation on Corporate Level
	UC01.2	Compliance Dashboard on Corporate Level
	UC01.3	Misconfiguration Monitoring on Corporate Level
	UC01.4	Non-compliance Monitoring on Corporate Level
	UC01.5	Aggregate Attribution of Non-Compliances
	UC01.6	Graphical User Interface
	UC01.7	Misconfiguration Tracking
	UC01.9	Compliance Certification Status on Corporate Level
	UC01.10	Compliance Status Aggregation on Domain Level
	UC01.11	Compliance Status Aggregation on Domain Level
	UC01.12	Compliance Status Aggregation on Domain Level
	UC01.14	Compliance Certification Status on Domain Level
	UC01.15	Misconfiguration Monitoring on Product Level
	UC01.16	Non-compliance Monitoring on Product Level
	UC01.17	Feedback on Remediation Actions
	UC01.18	Provision of Remediation Guidance
	UC01.19	Configuration
	UC01.20	Compliance Certification Status on Product Level
	UC01.21	Compliance and Certification Status in Product Composition
	UC01.22	Integration with Asset Management
	UC01.23	Framework Tools Setup and Configuration Automation
	UC01.24	User Interface Single Pane of Glass

Type	Req. ID	Short Title
	UC01.25	User Interface Time and Product Scope Adjustability
	UC01.26	Selectable Certification Schemes and Security Frameworks
	UC01.27	Selectable Controls in Compliance Dashboard
	UC01.28	Identification of Compliance Issues in Composed Products / Services
	UC01.29	Integration with Cloud-native Security Posture Management
	UC01.30	Interfaces for the Provision of Organizational Evidence
	UC01.31	Product-specific Customization of Certification Scheme
UC2 (FABASOFT)	UC02.01	Dashboard for the Compliance Manager
	UC02.02	Delegation of compliance tasks from Compliance Manager to Internal Control Owner
	UC02.03	Set up of Internal Control by Compliance Manager
	UC02.04	Mapping of Internal Controls to Security Controls by Compliance Manager or ICO
	UC02.05	Application to the same Security Control Framework to different Subsidiaries or Projects of the Corporation
	UC02.06	Tracking of Internal Controls by Compliance Manager
	UC02.07	Comprisal of Implementation Report by Compliance Manager for Auditor
	UC02.08	Comprisal of Report by Compliance Manager
	UC02.09	Elimination of risk for non-compliance
	UC02.10	Sleek UI
	UC02.11	Verify correspondence of Internal Controls and Persona Ref. TOMS to Internal Control System Policy
	UC02.12	Accept or decline ownership of Security Control
	UC02.13	Change status of Internal Control
	UC02.14	Assessment Results return value
	UC02.15	Attribution of Security Controls

10 Appendix C. List of Requirements

As explained earlier in the document, in this Appendix we include the full list or requirements, while in Section 3 we only include those that are new, have changed or have been discarded. To make it easier for the reader to understand the changes, the following colour coding has been used in the requirement tables.

A white table means the requirement has not changed. It remains the same as in the previous version of the document.
An orange table means the requirement has significantly changed its definition, which affects the meaning, provides more clarity, or modifies the scope.
A red table means the requirement has been definitively rejected. The reason of the rejection is provided along with the status.
A green table means the requirement is new in this second version, so a new functionality is defined for the component.

10.1 Functional requirements

10.1.1 Catalogue of controls and metrics

Requirement id	**RCME.01**
Short title	Catalogue of metrics, controls and TOMs
Description	The repository shall contain a catalogue of elements (categories, TOMs and reference implementations, controls and controls objectives, assurance levels) associated to the security control frameworks (see RCME.02 for the list of frameworks to be included), including: 1) clear definition of the categories of each security control framework included in the catalogue 2) clear definition of the security controls inside each category 3) clear definition of the TOMs (aka security requirements) for each control relevant for cloud service providers if relevant[23]. This definition shall contain guidance in the techniques and tools to be used for the evidence 4) clear definition of the assurance level corresponding to each security requirement if relevant 5) clear identification of evidence that would comply with the security controls and requirements 6) clear definition of the reference implementations of TOMs. These references implementations shall be selected by the CSP for each cloud service. 7) corresponding quantitative and qualitative metrics for each TOM
Status	Partially implemented
Priority	Must
Related KR	KR 1
Reference	DoA part B Annex 1 page 6 DoA part B Annex 1 page 7 DoA part B Annex 1 page 10 For point 4) technology provider

[23] Not all security certification schemes have requirements defined. Some, such as BSI C5 and ISO 27001 remain at the level of security control.

Requirement id	RCME.02
Short title	Metrics and TOMs in the repository
Description (*)	The repository should include realizable metrics for at least for the 70% of the TOMs referenced in EUCS-High assurance requiring “continuous (automated)” monitoring
Status	Partially implemented
Priority	Must
Related KR	KR 1
Reference	DoA part B Annex 1 page 7

(*) This requirement has been modified due to the reformulation of KPI 1.1 based on the new scope of the MEDINA technical metrics, which focus on the high-level requirements of the ENISA Cloud Security Certification Scheme.

Requirement id	RCME.03
Short title	Metrics and TOMs for different assurance levels
Description	The repository should include metrics for TOMs for basic (Y3), substantial (Y2) and high assurance levels(Y1)
Status	Fully implemented
Priority	Must
Related KR	KR 1
Reference	DoA part B Annex 1 page 7

Requirement id	RCME.04
Short title	Technology agnostic security controls
Description	The definition of the security controls in the repository should be technology agnostic, that is, they must be valid for a number of different implementations and cannot be technology specific.
Status	Fully implemented
Priority	Must
Related KR	KR 1
Reference	DoA part B Annex 1 page 15

Requirement id	RCME.05
Short title	Interfaces to the continuous auditing tools
Description	The repository should be accessible by the continuous evaluation tools.
Status	Fully implemented
Priority	Must
Related KR	KR 1
Reference	DoA part B Annex 1 page 15

Requirement id	RCME.06
Short title	Homogenization of the certification schemes
Description	The repository as part of the MEDINA framework should support the homogenization of certification schemes, by aligning to the EUCS. Thus, the repository must include information about the coverage of the different similar controls[24] in the different (national) schemes.

[24] In a preliminary comparative analysis of the security control frameworks, it has been confirmed that the minimum level to compare the frameworks is the controls. See D2.1 for more detail [11].

Status	Partially implemented
Priority	Must
Related KR	KR 1
Reference	DoA part B Annex 1 page 31

Requirement id	**RCME.07**
Short title	Interface to risk assurance
Description	When the certification scheme changes in some way (partial changes, requirements, new versions), the risk assurance component has to be notified, or be able to know that something has changed.
Status	Partially implemented
Priority	Should
Related KR	KR1 +
Reference	New in V2

Requirement id	**RCME.08**
Short title	Catalogue GUI
Description	The Catalogue has a GUI to search and show the different content it stores. This GUI is going to be part of the MEDINA Integrated-UI. Enhancements and adaptation to changes in data model are foreseen until the final version of the catalogue.
Status	Partially implemented
Priority	Must
Related KR	KR 1
Reference	New in V2

Requirement id	**RCME.09**
Short title	Questionnaire for self-assessment
Description	The Catalogue shall contain a questionnaire that helps a Cloud Service Provider to make a self-assessment of the fulfilment degree of the EUCS standard. This questionnaire will have the following features: 1) Allow the user to select the assurance level for the assessment 2) Include one or more questions for every requirement, of each control in each EUCS category 3) Provide an easy-to-use scale of support (fully/partially/not supported) 4) Allow to enter comments related to a question 5) Allow the user to include textual references for locating the evidence that support the response given to a specific question 6) Provide a dashboard that summarizes the result of the assessment, and provides quantitative values to reflect the degree of fulfilment
Status	Partially implemented
Priority	Could
Related KR	KR 1
Reference	New in V2

Requirement id	**RCME.10**
Short title	Questionnaire for auditors

Description	The questionnaire can be used by an auditor to help him in the audit process. For that purpose, the tool can provide some extra functionalities like: 1) Allow to enter non-conformities regarding a question 2) Provide a dashboard that summarizes the result of the audit, including the related comments/non-conformities for each question, as well as quantitative values to reflect the degree of fulfilment
Status	Partially implemented
Priority	Could
Related KR	KR 1
Reference	New in V2

10.1.2 Certification Language

10.1.2.1 NL2CNL Translator

Requirement id	**NL2CNL.01**
Short title	Translation from natural language to controlled natural language
Description (*)	The tool shall be able to translate in a semi-automatic way the requirements selected from a security certification scheme – originally expressed in natural language (English), into a set of obligations expressed in a controlled natural language. The output of the tool will be checked manually to verify if the obligations generated by the tool are correctly linked to the selected requirement.
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

(*) The description has been polished and completed. About the input, it now refers to translate “requirements” and not “most relevant aspects” of a security scheme. About the output, it says “into a set of obligations expressed in a CNL” instead of “into a controlled natural language”. It has been added a sentence about the output checking.

Requirement id	**NL2CNL.02**
Short title	Based on NLP and ontologies
Description	Given natural sentences taken from the cloud certification schema, the tool will rely on NLP techniques to link these sentences to a list of recommended metrics.
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	**NL2CNL.03**
Short title	Translation of organizational measures and technical measures
Description (*)	NL2CNL translator will be able to translate some of the organizational measures specified in the chosen EU cloud certification schemas, and some of the technical measures.
Status	Partially implemented
Priority	Should

Related KR	KR3
Reference	DoA, KR3 description

(*) Scope has been moderated. It now talks about translate "some", not "all the organizational measures".

Requirement id	**NL2CNL.04**
Short title	Compliant with the CNL editor language
Description	The controlled natural language output of NL2CNL translator will be compliant with the format used by the CNL Editor to represent the obligations
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	**NL2CNL.05**
Short title	XML compliant
Description	The controlled natural language output of NL2CNL translator will be compliant with the XML based format supported by the CNL Editor.
Status	**DISCARDED**: duplicates the requirement **NL2CNL.04**
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

10.1.2.2 CNL Editor

Requirement id	**CNLE.01**
Short title	CNL Editor GUI
Description	The controlled natural language Editor will have an interface accessible by web browser.
Status	Fully implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	**CNLE.02**
Short title	CNL Editor policies authoring
Description	The CNL Editor will allow creating statements for security controls.
Status	**DISCARDED**: workflow changed during project discussions respect to the initial idea, as a consequence the CNL Editor must not create Obligations.
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	**CNLE.03**
Short title	CNL Editor input format
Description	The CNL Editor will accept as input NL2CNL translator format (XML based).
Status	Fully implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	CNLE.04
Short title	CNL Editor policies changing
Description	The CNL Editor will allow changing input (policies) from NL2CNL translator.
Status	Fully implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	CNLE.05
Short title	CNL Editor vocabulary
Description	The CNL Editor will use an ontology-based vocabulary to model security controls. Ontology will be the same used by NL2CNL translator and based on W3C Web Ontology Language (OWL) standard format.
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	CNLE.06
Short title	CNL Editor output format
Description	The CNL Editor will generate security controls with an XML format suitable for DSL mapper.
Status	Fully implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

10.1.2.3 DSL Mapper

Requirement id	DSLM.01
Short title	Translation to selected DSLs
Description	The controlled natural language output of NL2CNL translator and further edited— when needed— with the CNL editor, will be semi-automatically mapped (meaning, with little human intervention) to the enforceable languages (aka, Domain Specific Languages, DSLs) inputs to tools such as Clouditor, or whatever will be the chosen DSL in MEDINA.
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	DSLM.02
Short title	Mapping elements
Description	The mapping process will consider relevant elements of the target certification framework, including (some) technical and organizational measures, quantitative/qualitative security metrics, complex compliance conditions, and cloud supply chain elements. The mapping process will prioritize the translation of those requirements in CNL that can automatically be enforced by WP4 and that are considered highly relevant by the EU authorities at stage.

Status	**DISCARDED**: Already contained in the rest of requirements
Priority	Must
Related KR	KR3
Reference	DoA, KR3 description

Requirement id	**DSLM.03**
Short title	DSL output compliancy
Description	The tool will output REGO rules, compliant with the input required by the orchestrator.
Status	Partially implemented
Priority	Must
Related KR	KR3
Reference	

10.1.3 Risk based selection of controls framework

Requirement id	**RBSCF.01**
Short title	Risk assessment tool
Description	The tool shall be based on a risk-assessment methodology and in order to help CSP, as well as an auditor, to identify the key assets, threats, and existing weaknesses of the cloud system.
Status	Partially implemented
Priority	Must
Related KR	KR2
Reference	DoA. Page 8

Requirement id	**RBSCF.02**
Short title	Risk assessment tool and TOMs
Description	Identification of key assets, threats and existing weaknesses should support stakeholders in reflecting their chosen TOMs in accordance with their risk strategy, along with risk treatment options.
Status	Fully implemented
Priority	Must
Related KR	KR2
Reference	DoA. Page 8

Requirement id	**RBSCF.03**
Short title	Implementation selection functionality
Description	Provide a tool-supported methodology for risk-based proposition of TOMs to ensure at most minor non-conformity with the selected certification schema within the target budget.
Status	Partially implemented
Priority	Must
Related KR	KR2
Reference	DoA. Page 8

Requirement id	**RBSCF.04**
Short title	Interface to the auditor
Description	Auditor follows a risk-based approach which provides flexibility to the certification process: since an ever-changing threat landscape often requires timely reaction from the security team provoking changes in the

	security configurations. These could be efficient from the risk treatment point of view, but will affect the previously obtained certificate, in the worst case, invalidating it.
Status	**DISCARDED**: The component provides the possibility to access the input parameters and results of the assessment to a Compliance Manager (role). An Auditor will have access to the component using the same functionality. In other words, there is no need to develop a separate interface for an auditor, as it will use the same interface that a Compliance Manager uses. In short, the requirement is automatically fulfilled by granting the auditor the rights of the compliance manager.
Priority	Must
Related KR	KR6
Reference	DoA. Page 9

10.1.4 Evidence gathering tools

10.1.4.1 Evidence Orchestrator

Requirement id	**ECO.01**
Short title	Provision of Interfaces
Description	The evidence orchestrator must provide standard interfaces for the evidence collection and assessment tools (T3.2-T3.4) to securely store their results.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	**ECO.02**
Short title	Conformity to selected assurance level
Description	The evidence orchestrator must ensure that the evidence collection (T3.2-T3.4) is performed according to the selected assurance level, i.e., it must trigger the evidence collection of the respective tools.
Status	Partially implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	**ECO.03**
Short title	Secure Transmission to evidence storage
Description	The evidence orchestrator must securely transmit evidence to the evidence storage.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	**ECO.04**
Short title	Transmission of evidence checksums
Description	The evidence orchestrator should integrate a Ledger client that stores checksums of evidence in a DLT.

Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

10.1.4.2 MEDINA Evidence Trustworthiness management

Requirement id	**ETM.01**
Short title	Trustworthiness of evidence
Description	The evidence orchestrator must integrate reasonable safeguards for guaranteeing the trustworthiness of collected evidence.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 18-19

Requirement id	**ETM.02**
Short title	Transmission of evidence checksums
Description	The evidence orchestrator should integrate a Ledger client that stores checksums of evidence in a DLT.
Status	Fully implemented
Priority	Should
Related KR	KR4
Reference	DoA part A Annex 1 pages 18-19

Requirement id	**ETM.03**
Short title	Trustworthiness guaranteeing capabilities
Description	Enable trustworthiness guaranteeing capabilities by extracting checksums from DLT and comparing with current checksums to detect modifications.
Status	Partially implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 18-19

Requirement id	**ETM.04**
Short title	Tamper-Resistance
Description	The developed tool must provide a tamper-proof way of storing evidence in the considered attacker model.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 18-19

Requirement id	**ETM.05**
Short title	Tamper-Resistance
Description	The DAT must provide a tamper-proof way of storing audit information in the considered attacker model.
Status	Fully implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 pages 22

Requirement id	**ETM.06**
Short title	Compliance with existing standards
Description	The design and implementation of the DAT should comply with the requirements of existing standards regarding the certification chain (ISO-based approach, ISAE3402 and evidence-based).
Status	**DISCARDED**: Certification standards are not directly applicable to the *MEDINA Evidence Trustworthiness Management System* as it is not involved in the certification process. It is just a component that provides extra security features.
Priority	Should
Related KR	KR5
Reference	DoA part A Annex 1 page 22

10.1.4.3 Technical evidence gathering tools: Clouditor, Codyze/CPG, Automated vulnerability monitoring / detection

10.1.4.3.1 Common requirements for all the tools

Requirement id	**TEGT.C.01**
Short title	Continuous collection
Description	The developed tools must be able to collect evidence continuously, i.e., in (high)-frequency intervals.
Status	Partially implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	**TEGT.C.02**
Short title	Provision to defined interfaces
Description (*)	The developed tools must provide collected evidence to the central evidence orchestrator via its offered APIs.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

(*) More specific definition. The destination to which to send the collected evidence has changed from “a security assessment tool” to “the central evidence orchestrator”.

10.1.4.3.2 Specific tool requirements

Gathering evidence from cloud interfaces

Requirement id	**TEGT.S.01**
Short title	Collect evidence from cloud interfaces
Description	The developed tool must be able to collect evidence of cloud workloads, e.g., virtual machines, containers, and serverless functions.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Gathering evidence from application source code

Requirement id	TEGT.S.02
Short title	Collect evidence from source code via CPG
Description	The developed tool must be able to parse the source code of cloud applications written in different programming languages and transform into the agnostic representation of the CPG, and derive evidence from its analysis.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	TEGT.S.03
Short title	Implement information and data flow analysis
Description	The developed tool must be able to perform information and data flow analysis on a cloud application.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	TEGT.S.04
Short title	Support expression of security requirements
Description	The developed tool must be able to support the expression of security requirements to be checked on application code. Requirements come for example from WP2.
Status	Partially implemented
Priority	Must
Related KR	KR1, KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	TEGT.S.05
Short title	Verify security requirements
Description	The developed tool must be able to verify security requirements and raise warnings/errors with respect to secure coding practices and secure information and data flows.
Status	Partially implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	TEGT.S.06
Short title	Retrieve source code of cloud applications
Description	The developed tool should be able to retrieve (semi-)automatically the source code of cloud applications requiring analysis.
Status	Partially implemented
Priority	Should
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	TEGT.S.07
Short title	Support for common programming languages, libraries, cloud services
Description	The developed tool should support common programming languages, libraries, and cloud services. Support for all programming languages, libraries and cloud services is infeasible.
Status	Partially implemented
Priority	Should
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Requirement id	TEGT.S.10
Short title	Connect infrastructure- and application-level security analyses
Description	The developed tool should be able to bridge the gap between infrastructure- and application-level security analysis by extending graph-based code analysis to the cloud resources, allowing to identify data flows across cloud resources.
Status	Fully implemented
Priority	Can
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Gathering evidence from computing resources (VMs, containers, software)

Requirement id	TEGT.S.08
Short title	Provision of malware, intrusion, and vulnerability detection tools
Description	Tools for malware detection, intrusion detection, and vulnerability scanning must be provided to assist CSPs with satisfying related requirements of security standards or to verify the compliance with such requirements.
Status	Partially implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

Gathering evidence from CSP-native services

Requirement id	TEGT.S.09
Short title	Collect evidence from CSP-native services
Description	The developed tool should be able to query findings from CSP-native services, like Azure Policy, to integrate them in MEDINA by querying the respective cloud API.
Status	Proposed
Priority	Could
Related KR	KR4
Reference	DoA part A Annex 1 pages 8-9

10.1.4.4 Organizational evidence gathering tools: AMOE

Requirement id	OEGM.01
Short title	Continuous collection of organizational evidence
Description	The developed tool using NLP must be able to collect evidence.

Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 page 18-19

Requirement id	**OEGM.02**
Short title	Provision to defined interfaces
Description	The developed tool using NLP must provide collected evidence to the central evidence collection component (T3.1) via its offered APIs.
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 page 18-19

Requirement id	**OEGM.03**
Short title	Usability for auditors
Description	The evidence management component should provide easy-to-use functionalities for auditors to search through relevant evidence. The assessment is handled manually though the UI. The assessment can be adjusted via API (should be checked/verified by a human beforehand).
Status	Fully implemented
Priority	Should
Related KR	KR4
Reference	DoA part A Annex 1 page 18-19

Requirement id	**OEGM.04**
Short title	Minimum evidence storage
Description	The evidence management component must be able to store and provide evidence at least back to the last assessment (if needed).
Status	Fully implemented
Priority	Must
Related KR	KR4
Reference	DoA part A Annex 1 page 18-19

Requirement id	**OEGM.05**
Short title	Evidence Assessment results
Description	The assessment results of evidence assessments must be submitted to the evidence orchestrator via the API it provides.
Status	Fully implemented
Priority	Must
Related KR	KR4

10.1.5 Evidence Assessment tool

Requirement id	**EAT.01**
Short title	Evidence assessment target
Description	The target values for the evidence assessment must be retrieved from a central repository of target values (WP2).
Status	Fully implemented
Priority	Must
Related KR	KR5

Reference	DoA part A Annex 1 pages 8-9

Requirement id	**EAT.02**
Short title	Continuous evidence assessment
Description	All evidence collection tools must forward evidence and measurement results (according to the data format defined in MEDINA) to the respective assessment components.
Status	Fully implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 pages 8-9

Requirement id	**EAT.03**
Short title	Evidence assessment results
Description	The assessment results of evidence assessments must be submitted to the evidence orchestrator via the API it provides.
Status	Fully implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 pages 8-9

Requirement id	**EAT.04**
Short title	Assess CSP-native evidence
Description	The developed tool should be able to assess the CSP-native evidence or translate CSP-native assessment results to the MEDINA data model.
Status	Proposed
Priority	Could
Related KR	KR5
Reference	DoA part A Annex 1 pages 8-9

10.1.6 Continuous Evaluation and Certification Life-Cycle

10.1.6.1 Continuous certification evaluation

Requirement id	**CCCE.01**
Short title	Continuous Evaluation of Assessment Results
Description	The evaluation component must be able to continuously evaluate incoming assessment results and integrate them into the overall certification evaluation.
Status	Fully implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 22

Requirement id	**CCCE.02**
Short title	Evaluate the fulfilment degree per TOM
Description	The evaluation component must be able to evaluate continuously generated assessment results according to previously defined TOMs to calculate the degree of fulfilment per individual audited resource and for the TOM in general.
Status	Fully implemented

Requirement id	CCCE.02
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 22

Requirement id	CCCE.03
Short title	Configuration of needed metrics for requirements
Description	The evaluation component must be able to receive a selection of metrics needed to be satisfied for a particular requirement (as selected by the CSP) and consider it in the evaluation of requirements' fulfilment values.
Status	Fully implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 22

Requirement id	CCCE.04
Short title	Evaluate the fulfilment degree per control, control group, and entire certification
Description	The evaluation component must be able to aggregate the TOMs' fulfilment degrees to calculate the degree of fulfilment for controls, control groups, and the entire certification scheme.
Status	Fully implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 22

Requirement id	CCCE.05
Short title	Evaluate the temporal fulfilment degree per TOM
Description	The evaluation component should be able to evaluate continuously generated assessment results according to previously defined TOMs to calculate a degree of fulfilment over time.
Status	Fully implemented
Priority	Should
Related KR	KR5
Reference	DoA part A Annex 1 page 22

Requirement id	CCCE.06
Short title	Evaluate the time-to-fix indicator per TOM
Description	The evaluation component should be able to evaluate continuously generated assessment results to calculate a time-to-fix indicator.
Status	Fully implemented
Priority	Should
Related KR	KR5
Reference	DoA part A Annex 1 page 22

Requirement id	CCCE.07
Short title	APIs of the Continuous Certification Evaluation Component
Description	The evaluation component must provide APIs to the relevant components (security assessment tools) to receive assessment results, as well as to the

	digital audit trail and the certificate lifecycle management component to exchange relevant data.
Status	Fully implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 22

10.1.6.2 Automation of the Cloud Security Certification Life-Cycle

Requirement id	**ACLM.01**
Short title	Cloud security certification issuance
Description	Based on the quality evaluation results, the system will push appropriate entities (CAB) to issue and sign security certifications for the cloud providers.
Status	Partially implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 9

Requirement id	**ACLM.02**
Short title	Automatic cloud security certification update
Description	Based on the quality evaluation results, the system will push appropriate entities (CAB) to update the security certifications for the cloud providers.
Status	Partially implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 9

Requirement id	**ACLM.03**
Short title	Cloud security certification revocation
Description	Based on quality evaluation results, the system will push appropriate entities (CAB) to revoke the security certifications for the cloud providers.
Status	Partially implemented
Priority	Must
Related KR	KR5
Reference	DoA part A Annex 1 page 9

Requirement id	**ACLM.04**
Short title	Continuous update of the certificate state
Description	The certificate lifecycle management component must continuously, i.e., in high-frequency intervals, convert the evaluation results from the CCE to the corresponding certificate state.
Status	Partially implemented
Priority	Must
Related KR	KR6
Reference	DoA part A Annex 1 page 9

Requirement id	**ACLM.06**
Short title	Compliance with EUCS assurance levels and certificate states
Description	The certificate lifecycle management component must map the certificate states and assurance levels defined in the EUCS scheme.

Status	Partially implemented
Priority	Must
Related KR	KR6
Reference	DoA part A Annex 1 page 9

Requirement id	**ACLM.07**
Short title	Interface for a public registry
Description	The lifecycle management component must provide an interface for publishing the certificate status in a public registry by the corresponding entities (CAB).
Status	Partially implemented
Priority	Must
Related KR	KR6
Reference	DoA part A Annex 1 pages 9

Requirement id	**ACLM.08**
Short title	Secure lifecycle management
Description	The lifecycle management component can be implemented in a smart contract to ensure a tamper-proof execution.
Status	DISCARDED: based on the evaluation of smart contracts for the automatic management of certificates[25], it was considered that they introduce too many risks compared to the potential benefits.
Priority	Could
Related KR	KR6
Reference	DoA part A Annex 1 pages 9

10.1.6.1 SSI Framework

Requirement id	**SSI.01**
Short title	Cloud security certificate issuance
Description	The system should provide a way for appropriate entities (CAB) to issue and sign security certifications for the cloud providers as indicated by the automated certificate Life-Cycle Manager.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	D4.2 & D5.4

Requirement id	**SSI.02**
Short title	Cloud security certificate update
Description	The system should provide a way for appropriate entities (CAB) to update security certifications for the cloud providers as indicated by the Life-Cycle Manager.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	D4.2 & D5.4

[25] For more details, see deliverable D4.2 [11]

Requirement id	**SSI.03**
Short title	Cloud security certificate revocation
Description	The system should provide a way for appropriate entities (CAB) to revoke security certifications for the cloud providers as indicated by the Life-Cycle Manager.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	D4.2 & D5.4

Requirement id	**SSI.04**
Short title	Cloud security certificates listing
Description	The system must list the historical cloud security certificates issued, updated, and revoked.
Status	Fully implemented
Priority	Must
Related KR	KR6
Reference	D4.2 & D5.4

Requirement id	**SSI.05**
Short title	Cloud security certificate verifiable public proofs generation
Description	The system must generate verifiable proofs of the security certificate state on request.
Status	Fully implemented
Priority	Must
Related KR	KR6
Reference	D4.2 & D5.4

Requirement id	**SSI.06**
Short title	Cloud security certificate confidential proofs generation
Description	The system should generate verifiable confidential proofs of the security certificate private parameters on request.
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	D4.2 & D5.4

Requirement **id**	**SSI.07**
Short title	Cloud security certificate proofs request and verification
Description	The system should provide a way for appropriate entities (potential clients) to request and verify proofs of the security certificates to the cloud service providers.
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	D4.2 & D5.4

10.1.6.2 Risk-Based continuous assessment

Requirement id	**RBCA.01**
Short title	Dynamic risk assessment
Description	Timely adjust the CSP's risk profile and re-evaluate efficiency of security configuration
Status	Partially implemented
Priority	Must
Related KR	KR6
Reference	DoA. Page 9

Requirement id	**RBCA.02**
Short title	Interface to the continuous evidence management tools
Description	Requires consuming the current status of the system configuration to re-adjust risk profile.
Status	Partially implemented
Priority	Must
Related KR	KR6
Reference	DoA. Page 9

10.1.7 Integrated User Interface

Requirement id	**IUI.01**
Short title	Authentication integration via Keycloak Adapter
Description	Every component must implement an adapter that allows it to authenticate with Catalogue's Keycloak Authentication Service in order to prevent unauthenticated users to access its resources
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

Requirement id	**IUI.02**
Short title	Authorization integration via Keycloak
Description	Every component that has resources that should only be accessed by specific user roles must enforce authorization on its internal logic (e.g., in a REST API, define at controller level that a specific endpoint can be accessed only with Product Engineer Role). This can be obtained by defining appropriate configuration on Catalogue's Keycloak (Role Mapping).
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	WP5/WP6 Technical discussions

Requirement id	**IUI.03**
Short title	Allow frame embedding into Integrated UI
Description	Every component UI that needs to be embedded in an iframe inside the Integrated UI must define a header "X-Frame-Options: ALLOW-FROM integrated-ui-url" in order to allow it.

Requirement id	IUI.03
Status	**DISCARDED**: we are currently sticking to the micro frontend strategy with iframes only.
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

Requirement id	IUI.04
Short title	Allow CORS for Integrated UI
Description	Every component backend that needs to be programmatically REST called via Integrated UI frontend must define a header “Access-Control-Allow-Origin: <integrated-ui-url>” in order to allow it.
Status	**DISCARDED**: At the moment, no REST API integration with the IUI is planned. With this approach CORS is not needed.
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

Requirement id	IUI.05
Short title	External Identity Provider Configuration
Description	Users should be able to authenticate using their existing enterprise identity provider once it has been configured to do so. Ideally, MEDINA Generic Roles should be inherited from existing claims / roles.
Status	Fully implemented
Priority	Should
Related KR	KR6
Reference	WP5/WP6 Technical discussions

Requirement id	IUI.06
Short title	Homogeneous look and feel
Description	Each component micro-frontend embedded into IUI should abide to a set of graphical constraints and rules that the consortium agreed on in order to homogenize look and feel.
Status	Partially implemented
Priority	Should
Related KR	KR6
Reference	WP5 Technical discussions

10.2 Non-functional requirements

10.2.1 NF Requirements for the development of CI/CD tools

The NFRs refer about the features that the CI/CD tools should/must have in order to effectively meet the needs of the CI/CD strategy for the MEDINA project.

These requirements have been discussed with the partners and all have agreed. Therefore, we intend to follow this methodology based on this NFRs to make decision for the CI/CD tools/products. The assessment resulted in the requirements listed in the following tables.

Requirement id	CICD.01
Short title	Code repository

Description	The development environment has a Revision Control System for storing the source code of the project's components
Status	Fully implemented
Priority	Must
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.02**
Short title	Automate software build
Description	Speed up the build process by automating the steps necessary to produce executable and deployable software artefacts
Status	Fully implemented
Priority	Must
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.03**
Short title	Automate test suite
Description	Make the build process self-testing in order to spot early software defects, both at component/unit and at integration level
Status	Fully implemented
Priority	Should
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.04**
Short title	Software bugs tracking
Description	Facilitate the collection and monitoring of software issues in order to fix them with discipline
Status	Fully implemented
Priority	Should
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.05**
Short title	Deploy automation
Description	Automate the process of delivering the software in installable form in a bug-per-bug reproducible way
Status	Fully implemented
Priority	Should
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.06**
Short title	Free tools
Description	Selected tools are free or open-source software, in order to allow the easy setup of the self-hosted development environments
Status	Fully implemented
Priority	Must
Related KR	Supporting Integration of KR1-KR5

Reference	DoA

Requirement id	**CICD.07**
Short title	Commercially friendliness tools
Description	The license is commercially friendly (i.e., Apache) and not copy-left, such in a way it does not impact on the developed software artefacts commercialization
Status	Partially implemented
Priority	Should
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.08**
Short title	Java support
Description	Support Java programming language for MEDINA framework
Status	Fully implemented
Priority	Done
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.09**
Short title	Python support
Description	Support Python programming language for MEDINA framework
Status	Fully implemented
Priority	Done
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.10**
Short title	C language support
Description	Support C/C++ programming language for MEDINA framework
Status	Fully implemented
Priority	Must
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.11**
Short title	GO Lang support
Description	Support GO programming language for MEDINA framework
Status	Fully implemented
Priority	Must
Related KR	Supporting Integration of KR1-KR5
Reference	DoA

Requirement id	**CICD.12**
Short title	JavaScript support
Description	Support JavaScript/Typescript programming language for MEDINA framework
Status	Fully implemented
Priority	Must

Related KR	Supporting Integration of KR1-KR5
Reference	DoA

11 Appendix D. CI/CD Strategy

DevOps (short for Development and Operations) is an approach based on lean and agile principles in which business owners and the development, operations, and quality assurance departments collaborate to deliver software in a continuous manner that enables the business to seize market opportunities more quickly and reduce the time to include customer feedback [24].

“Develop and test against production-like systems” is one of the principles that stems from the DevOps concept *shift left*. The shift left concept moves operations earlier in the development life cycle. The goal of this principle is that it can be seen how the application behaves and performs well before it is ready for deployment, anticipating issues that are easier and less expansive to address earlier than in more advanced stages.

A required element to support the DevOps is to automate. Automation is essential to create processes that are iterative, frequent, repeatable, and reliable. Iterative process means that it can be represented by a well-defined series of steps; this enables the implementation of a repeatable process which is reproducible, consistent, and finally reliable. Only a reliable automated process can be used very frequently, with the assurance that it will not break. To apply these principles, organizations must create a so called “delivery pipeline” that allows for continuous, automated deployment and testing of reliable software artefacts. This is the “Continuous Integration/Continuous Delivery pipeline” or “CI/CD pipeline” concept.

At an abstract level, a delivery pipeline is an automated process for releasing software from the version control system (e.g., GitLab) to the end users. Every change to the software goes through a multi-stage process on its way to being released. That process starts from building the software and is followed by the progress of these builds through multiple stages of testing and deployment. Continuous integration and continuous delivery allow to see and control the progress of each change as it moves, from version control through various sets of tests and deployments, to release to users.

Continuous Integration (CI) is the phase in the software development lifecycle where code from different developers is integrated together. This usually involves merging code (integration phase), building the application (build), and carrying out basic tests (test phase), all within an ephemeral environment. In the test phase, every element of the entire application is examined, from classes to functions. If a conflict is found between the new and existing code, the continuous integration helps to correct it.

The adoption of CI involves the introduction of some practices:

- **Use of version control**: every part of the project like source code, test cases, database definitions, build and deployment scripts, and anything else needed to create, install, run, and test the application must be checked into a single version control repository. A Version Control System (VCS) makes it possible to record the different revisions of the project artefacts, and allows to answer the question of who changed what and when, and permits, if necessary, to step back to revert unwanted changes.
- **Check-in regularly**: checking-in means submitting changes of software artefacts into the VCS. Doing this regularly brings lots of benefits: mistakes are easier to spot and correct; it is easy to revert to a recent known-good version if something goes wrong; it avoids altering too many files among versions.
- **Create a comprehensive automated test suite**: for frequent check-ins to work best, it is essential to have some level of automated testing to provide confidence that the application is actually working. There are three kinds of tests which are usually executed:

unit tests, integration tests, and acceptance tests. *Unit tests* are meant to test the behaviour of small pieces of the application (a method, a function, or the interactions between a small set of them). *Integration tests* do the same but for several components of the application together, typically developed by different groups of people. *Acceptance tests* verify if the application meets the acceptance criteria decided by the business and is a formal phase in the software development lifecycle. These three sets of tests, combined, provide a high-level of confidence in the application.

CD can take several meanings. CD as **Continuous Delivery** automates the release of validated code into a repository (software artifacts). CD as **Continuous Deployment** automates the release of the app into production without waiting for the explicit approval of the developer. The goal of continuous delivery is to have a base code that is always ready to be deployed in a production environment. Continuous deployment means that the changes made by a developer can become active within minutes, provided it passes the automated testing phase. This allows receiving and integrating the feedback of the end users easier.

The **delivery pipeline** consists of the stages an application goes through from development to production. These stages may vary from one organization to another, and the level of automation also vary. Some organizations fully automate their delivery pipelines; others maintain manual checks and gates due to company requirements.

Typical stages of the **CI/CD pipeline** are the following (see Figure 12):

- **Develop:** In which developers write the source code and scripts. This stage includes tools for source control management, collaboration, and project planning. Tools in this stage are typically cross-platform and cross-technology.
- **Build**: The build stage is where the code is compiled/processed to create executable binary programs and where unit testing is performed.
- **Package repository**: A package repository (also referred as artefact repository) is a common storage mechanism for the binaries created during the build stage. These repositories also need to store the assets associated with the binaries to facilitate their deployment, such as configuration files, and deployment scripts.
- **Test**: A test stage is where development/testing teams do the quality testing and user acceptance. These tests include integration, functional, performance, and security tests. Automated tools are available for each of these types of tests. These tools are commonly integrated within a test asset management tool, that allows also to trace test results.
- **Release**: In this phase the final application artefacts are made available on a repository.
- **Deploy**: At this stage the code is deployed from the repository to the operating environment.

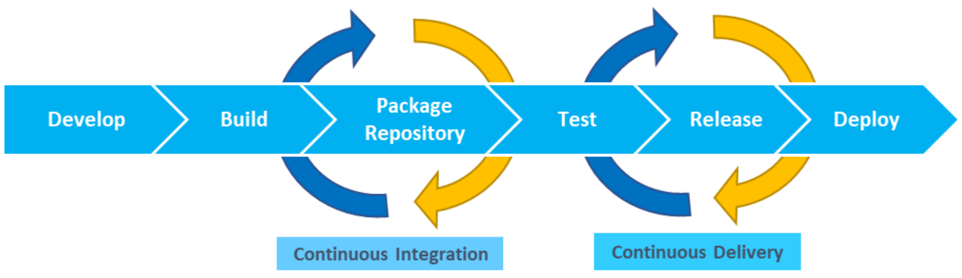


Figure 14. CI/CD pipeline

11.1 Quality & Assurance

In our project, we will extend the DevOps approach by adding quality and security checks, hence we will talk about "SecDevOps". The term reflects the fact that the role of quality and assurance (QA) in DevOps processes is increasingly important.

In traditional DevOps, a separate security team is responsible for security testing and its role is limited to the final phase. If the security team finds a security concern, the application must be corrected and go through the entire process again: this is not an agile principle and may cause a delay on the release.

Instead, SecDevOps means integrating application and infrastructure security early in the development lifecycle, and automating security control to prevent slowing down the DevOps workflow. Putting quality and assurance at the heart of the software lifecycle can actually save time, and can accelerate delivery and deployment.

In a SecDevOps approach, vulnerabilities are prevented and managed proactively. The Open Web Application Security Project (OWASP) [25] is a non-profit foundation dedicated to improving the security of software. OWASP is like a community in which everyone can participate and where all materials and information are free and available on the website. OWASP provides an online document called *OWASP Top 10* [26] where the top 10 most critical web application security risks are listed, and is updated every 2-3 years in accordance to advancements and changes in the application security market. The purpose of the document is to offer developers and web application security professionals, recommendations about the most prevalent security risks in order to adopt security practices that minimize the presence of these risks in applications. The risks are ranked and based on the frequency of discovered security defects, the severity of the vulnerabilities, and the magnitude of their potential impacts. OWASP's importance lies in the actionable information it provides; it serves as a key checklist and web application development standard for many organizations. Integrating the Top 10 into the software development lifecycle (SDLC) demonstrates a commitment to improve secure development practices. For example, let's take one of the risks from the list: *A9-Using Components with Known Vulnerabilities;* it states that applications and APIs using components with known vulnerabilities may undermine application defences and enable various attacks and impacts. This leads to use tools that verify security vulnerabilities in third parties' dependencies, like OWASP Dependency-Check [26].

Instead of testing the entire product for gaps and bugs, each developer or team is encouraged to check the newly created code for such problems. Development teams can use code quality standards to evaluate the structural quality of software ahead of each release. By applying standards earlier in the software development lifecycle, a codebase can be developed further, or open sourced with greater confidence, resulting in less complexity. These measures are designed to be automated on source code through static analysis tools.

In ana analogous way, the Consortium for Information & Software Quality (CISQ) developed Automated Source Code measurement standards for Reliability, Security, Performance, Efficiency and Maintainability, which were approved as OMG standards. Each code quality measure is comprised by a set of weaknesses in the Common Weakness Enumeration (CWE). The CWE [27] is a reference point for developers and codifies over 900 known software weaknesses. The coding rules contained in CISQ standards [28] include CWEs as SQL Injection and Buffer Overflow. In fact, CISQ considers the security principle as one of the aspects of good software quality practices.

11.2 Containerization deployment model

Nowadays, modern applications are developed and packaged by following the concept of micro-services. This is an architectural software pattern that enables clear Application Programming Interfaces (API) to foster an easy integration among different application's components that are loosely coupled. Micro-services provide their functionalities as a service (like in a service-oriented architecture) and typically are specialised to perform a very specific job or task.

Micro-services are usually packaged in software containers, as the unit of deployment on a platform. A container is a lightweight form of virtualization at operating system level, where each container is isolated from the others, bundles all its needed software and libraries, and can communicate only through well-defined channels. Among its advantages, containerization allows building a reproducible software package, and this is why it successfully matches the principles of CI/CD and fits well for micro-services. An application, packaged in a container, behaves the same regardless the environment where it is deployed; avoids issues with operating system versions and tools that could hinder the smooth execution of the software; and eases a reproducible and secure deployment.

It is thus easy to understand that the **Package Repository** stage we described earlier could be fed with containers created for micro-services applications during the execution of the CI/CD pipeline.

Infrastructure as Code (IaC) refers to the fact that the environment in which the software runs should be software itself. Containers offer one such approach. In fact, developers can create the container descriptor file, that includes all the steps to package a micro-service in the container. This descriptor (software code) can be versioned and deployed according to the defined CI/CD pipeline stages. Containers allow fast and cost-effective scaling, can be secured more easily, and can be quickly duplicated, reassembled, or replaced. A container needs to be booted up on an already existing server structure or cloud service. In addition, one instance can be developed while the other is already running [29]. Docker containers are the most common implementation available today [30].

Since containers are packaged applications that includes many third-party tools, such as OS libraries and software, it is not uncommon to face security issues on these minimal OSs that are bundled along with the developed micro-service. For example, a micro-service written in Java needs a Java run-time execution environment, which in turns requires several libraries at operating system level: through this software chain, software bugs and security vulnerabilities can be nested or can be discovered as time goes by. Hence, hardening these so-called *container images* becomes imperative. Developers needs to be aware of known CVEs that affects some components before containers are deployed, reducing the overall risk profile. Furthermore, this needs to be iterative since new vulnerabilities and bugs can be discovered over time once the container image was designed. Some vendors provide CVE scanning tools for container images, which can be integrated as a stage in the CI/CD pipeline to assess the container security at every build.

11.3 CI/CD supporting tools

In this section we describe the tools we intend to utilize with the aim to set up a continuous integration service following the CI/CD strategy outlined in Section 5.1. The tools can be grouped in the following categories according to literature:

- **Version control system**: Version control, also known as source control, source code management, or revision control, is a mechanism for keeping multiple versions of software files, so that when a developer modifies a file s/he can still access the previous versions. These tools also provide mechanisms to people involved in software delivery to collaborate.

- **Build Automation system**: its first use is to convert the source code in a machine-understandable code that can be executed. All build tools have a common core in order to support build reproducibility: to model a dependency network (e.g., software libraries). A build tool must also ensure that for a given goal, each prerequisite must be executed exactly once. If a prerequisite is missed, the result of the build process will be wrong.
- **Artefact repository**: The outputs of the Build Automation system —reports and binaries— need to be stored somewhere for reuse in later stages of the CI/CD pipeline. That is, in the repository.
- **Continuous Integration software**: It is the orchestrator of the whole build process that integrates with all the other solutions and enables automation of the cycle. The orchestration goes through all stages, from fetching the code from the Revision Control system, through compiling it with the Build Automation, until storing it on the Artefact repository and evaluating the solution for quality and security issues.
- **Testing system**: Verify code changes through testing, preferably automated testing. This system supports the automation of both unit testing and integration testing.
- **Bug Tracking system**: It is a system to trace software defects or improvements that are found for the systems components in development and/or related to the deployment environments.

11.3.1 Analysis of CI/CD Tools

In this section, we evaluate a list of tools that could be selected for the building of the CI/CD pipeline in the MEDINA project and highlight their advantages and disadvantages in order to guide the final choice(s).

First of all, we have defined the methodology to search and select these tools. On one hand we have collected the tools from the MEDINA partners, listing the tools they provided or already knew; on the other hand, we have analysed other tools in the market, attending to the requirements. All tools were grouped in classes by their role in the CI/CD pipeline**.**

The result of this survey was summarized in the NF requirements we proposed and agreed by the consortium, already described in Section 10.2.1.

In addition, to improve the effectiveness of the analysis, we make use of a free instrument called OpenHub[26], where we can find useful information about some key aspects of software tools like the license, the activity of community and the current or past vulnerabilities.

A license is typically permissive or not permissive. In the first case we can release software in an open-source manner without copy-left; in the second case we cannot use it without copy-left, so it is more restrictive. Permissive licenses allow you to copy, modify, recombine, and redistribute the work with minimal restrictions. Copy-left requires that to release any derivative works done it is needed the same copy-left license. If you release a software library under a copy-left license like the GPL, and someone else wants to write a program using both your library and a proprietary library, hey would not be allowed to do so. The purpose of GPL-like license is to force continuing the open-source nature of the piece of software and its open development. On the other hand, the permissive licenses are preferred by the industrial partners, who need to protect their intellectual property resulting from software development. Some typical examples are Apache or BSD licenses.

[26] https://www.openhub.net/tools

The activity of the community is another important parameter to decide if a tool is good. If it has a lively community, measured by the number of commits in a month and the number of contributors, we can have a fair degree of confidence in that there would always be someone to fix bugs and improve the software overtime.

Regarding vulnerabilities tracking, that a tool presented a lot of vulnerabilities in the past is not necessarily a bad sign, especially if the owners were able to solve them and the community is big. On the contrary, can be a guarantee that there is an active development team and a particular focus on security.

To conclude, we can merge this information with the requirements already identified to have a more complete view for the selection of tools and decide which are more suitable for MEDINA development. We list next list of candidate tools in each class.

11.3.1.1 Code & Version Control systems

1. GitLab CE

GitLab CE [31] is an open-source software to collaborate on source code development. GitLab offers a git version control repository management, code reviews, issue tracking, activity feeds and wikis. Git is a distributed versioning tool, not necessarily with a single centralised repository, and for this reason it tends to be more complex for beginners with respect to centralised tools (e.g., Subversion). Apart from source code repository, it can be used for tracking software defects and enhancement thanks to its build-in issue tracking mechanism.

It is strongly established, and has a mature codebase maintained by a very large development team and few high vulnerabilities reported during the years as shown on OpenHub. It uses an open-source MIT License that is commercially friendly.

2. Apache Subversion

Apache Subversion [32] is a full-featured version control system that boasts of a model, design, and interface that is said to be more advanced than other Code Versioning Systems (CVS). The open-source revision control and software versioning platform’s primary solutions include interactive conflict resolution, merge tracking, and file locking, with the most recent updates, containing features for path-based authorization, interactive resolvers, compression, and shelving.

CVS users attest to Subversion’s centralized version control capabilities, which are said to be a reliable repository of valuable data and is easy enough to be learned and used even by those with only beginner’s knowledge of software development. Subversion supports various types of users and projects, either individuals or enterprise-level organizations.

It is mostly written in C and uses the Apache License 2.0.

Analysis

Subversion adopts a centralized approach, while Git leverages on the distributed approach. This means that on Subversion there is a single repository where all developers commit their changes and retrieve updates. With Git, every repository can trace the changes and in fact every developer has a full Git repository on his or her development machine: typically, by convention, a repository hosted on a server is logically promoted in a way that developers push all their local changes which are there merged together. Git allows having multiple remote repositories, for example, a developer can push the changes to a public Git repository while pushing them also to his or her company internal Git repository.

As a result of their structure, Subversion tends to be easier to use for new developers, while Git has a steeper learning curve.

Subversion is usually used as the single tool, while Git is very frequently packaged in comprehensive environments that provides more than versioning only, like a well-structured web interface and other added-value services.

11.3.1.2 Build Automation system

1. Apache Maven

Maven is a build automation tool used primarily for Java projects. The Maven project is hosted by the Apache Software Foundation. Maven addresses two aspects of building software: how software is built, and how its dependencies are managed.

Maven provides a quite rigid model that makes customization difficult. It is based on an external DSL written in XML, which can be extended by writing code for plug-ins. Maven is, in its default configuration, self-updating and downloads its own plugins from the Internet. This can be an advantage, but also a point of attention, since as a result we can have an unwanted upgrade of its plugins and we could hinder the process of reproduceable builds. Nevertheless, it manages software dependencies (e.g., third-party libraries) by automatically downloading them from the Maven Central repository over the Internet.

2. Gradle

Gradle is a build automation tool for multi-language software development. It controls the development process in the tasks of compilation and packaging, testing, deployment, and publishing. Supported languages include Java (and other Java-based languages like Kotlin, Groovy, and Scala), C/C++, and JavaScript. It supports a light DSL (not XML-based but based on the Groovy language) and manage software dependencies by automatically downloading them from the Internet.

Gradle is distributed as open-source software under the Apache License 2.0 and was first released in 2007.

Analysis

About the user experience, a large number of users prefer to execute build operations through a command-line interface. For this, Gradle provides a modern CLI for this reason.

Both build systems provide built-in capability to resolve dependencies from configurable repositories. Both can cache dependencies locally and download them in parallel.

As a library consumer, Maven allows to override a dependency, but only by version. Gradle provides customizable dependency selection and substitution rules that can be declared once and handle unwanted dependencies project-wide. This substitution mechanism enables Gradle to build multiple source projects together to create composite builds.

Maven has few, built-in dependency scopes. There is no separation between unit and integration tests, for example. Gradle allows custom dependency scopes, which provides better-modelled and faster builds.

Maven is simpler to learn than Gradle and, being more mature, has a larger set of plugins for several needs [33] [34].

11.3.1.3 Artefact repository

1. Nexus

Nexus [35] is an open-source repository that supports many artefact formats, including Java binary artefacts and Docker images. With the Nexus tool integration, pipelines in the CI/CD can publish and retrieve versioned components and their dependencies by using a central repository.

Nexus OSS has a broad support for many tools:

- Store and distribute Maven/Java, npm, Docker and more.
- Manage components from dev through delivery: binaries, containers, assemblies, etc.
- Advanced support for the Java Virtual Machine (JVM) ecosystem, including Gradle, Ant, Maven, and Ivy.
- Compatible with popular tools like Eclipse, IntelliJ, Jenkins, Puppet, Chef, Docker, and more.

Nexus OSS provides security features to centralise user accounts (e.g., on LDAP) and provides the capability to handle authorisations of users (or group of users, even LDAP groups) to the different archived artefacts.

2. JFrog Artifactory

JFrog Artifactory [36] is a scalable, universal, binary repository manager that automatically manages artefacts and dependencies throughout the application development and delivery process. Artifactory supports Kubernetes, the de facto orchestration tool in the industry, for automating deployment, scaling, and management of micro-services and containerized applications.

With the JFrog Artifactory it is possible to:

- Achieve a high availability with active/active clustering and multi-site replication for DevOps setup to support scaling.
- Release faster and automate CI/CD pipeline via powerful REST APIs.
- Deploy Artifactory as repository manager on-prem, in the cloud, or in a hybrid model.

Analysis

Both Nexus and Artifactory support integration with external authentication systems, like LDAP, which is a much-requested enterprise feature. However, Nexus has a quite difficult authorization mechanism to protect the different repositories, while Artifactory has a cleaner interface to allow assigning permissions to users and roles for protecting the access to the available repositories and individual paths.

The auditing mechanism provided by Artifactory seems to be easier to use, because there is a dedicated UI that allows seeing logins and accesses to the repositories. Nexus provides some of this information, but only through log files.

Both products support the setup of a Docker registry with virtual registry capability, which allows combining two or more Docker registries in a single registry. The web interface for Docker registry of Artifactory supports previewing the Dockerfile descriptor, while Nexus does not, and it seems more complete than that of Nexus also in regard to searching capabilities.

11.3.1.4 Continuous Integration software

1. Gitlab CI/CD

GitLab CI/CD is a free and self-hosted Continuous Integration server. GitLab CI/CD has a community edition and provides Git repository management, issue tracking, code reviews, wikis, and activity feeds. Companies can install GitLab CI/CD on-premises and can connect it with a corporate directory server (e.g., LDAP) for secure authorization and authentication. GitLab CI/CD is written in Ruby and Go and released under an MIT license. Since GitLab CI/CD provides Git repositories, the integration of the CI/CD pipelines are quite simple and straightforward.

2. Jenkins

Jenkins [37] is a Continuous Integration server, allowing to automatically monitor source code repositories, build software, run tests and deploying software. Through the installation of plugins, Jenkins integrates with a huge set of tools in the continuous integration and continuous delivery toolchain, including GitLab CE and JFrog Artifactory. It has several dashboards for controlling the status of the unit and integration tests (e.g., JUnit compatible) and dashboards for visualising the status of the quality and security tests performed on code artefacts. Jenkins is made available via an open-source MIT License and has a strong community of developers.

3. Tekton

Tekton [38] was originally part of the Knative project and is now under the umbrella of the Continuous Delivery Foundation. Tekton was released in 2018 under the Apache 2.0 license.

Tekton takes a modular approach to cloud-native CI/CD by implementing components that form the building blocks needed to create a complete CI/CD ecosystem. Due to its nature, Tekton is extremely powerful, and it provides the ability to customize entities which can then be shared and reused as needed.

A great advantage of Tekton is its modularity, which allows for componentization, standardization, and reusability within the CI/CD pipelines. The steps are operations in the CI/CD workflow that are execute in containers, and they are organized into tasks that run as pods on a container cluster orchestration engine (i.e., Kubernetes [39]). Tasks can be assembled and ordered within pipelines in any way needed [40].

Analysis

With the aid of GitLab CI/CD, it is possible to control Git repositories with total control over branches and several other facets to keep the code safe from sudden threats. However, in the Jenkins case, it is possible to control repositories but up to a few extents only. For example, it does not allow complete control over branches, but Jenkins supports many types of source code repositories others than Git, including Subversion.

In GitLab CI/CD, every single project has a tracker that will track problems and carry out code reviews to improve efficiency. On the other hand, Jenkins has a simple procedure for installation as well as configuration, and a huge set of plugins to integrate many different developers' tools. Jenkins is also able to automate non-software build pipelines (e.g., connect to machines to perform some tasks), while GitLab CI/CD cannot [41].

Tekton is the youngest project among those analysed and it is specifically tailored to use containers as its running mechanism, which helps managing scalability. Pipelines building blocks can be factored and reused among different pipelines, fostering components reusability. Also, Jenkins provides the ability to create libraries where functionalities can be factored out and reused between different jobs.

11.3.1.5 Testing system (unit and integration)

1. JUnit

JUnit [42] is a mature unit testing framework for the Java programming language. JUnit is an open source (Eclipse Public License 1.0) framework used to write and run repeatable automated tests. It integrates with Jenkins for reporting the testing outcomes and with build tools such as Maven and Gradle for executing the tests.

2. TestNG

TestNG [43] is a testing framework for the Java programming language, inspired by JUnit and NUnit (for Microsoft .NET framework) and it is released under Apache 2.0. It is a testing framework designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers).

3. REST Assured

REST Assured [44] is a Java DSL framework for simplifying testing of REST based services. It supports all the different HTTP verbs (e.g., POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD) and can be used to validate and verify the response of these requests. Since RESTful services are used by micro-services API, this is a good candidate for testing such applications. REST Assured is released via the permissive Apache License 2.0.

Analysis

JUnit, in particular its version 4, is a very wide-spread tool, used by many Java developers. JUnit introduced a reporting testing format based on XML that has become recognized in many tools, including Jenkins, to represent the outcome of the testing activities. Such XML format has been also employed by tools other than Junit, as a testing framework for non-Java languages. TestNG also uses this XML format.

On the other hand, TestNG is not very used even if it provides some more features compared to JUnit. REST Assured, instead, is more focused on the definition of end-to-end test cases for RESTful web services: it uses a declarative way that uses the Java programming language. Since the MEDINA framework is being developed as a set of interacting micro-services, the use of the REST Assured framework is highly recommended.

11.3.1.6 Bug Tracking system

1. Gitlab Issues

Gitlab Issues is a free tool built into GitLab CE that makes it easier to track software development progress. It supports many of the same features as commercial competitors like Atlassian Jira [45], while being easier to use since it is integrated in the GitLab CE software. GitLab CE also provides an integrated Wiki platform. It uses the same license as GitLab CE.

2. Trac

Trac [46] is an enhanced wiki and issue tracking system for software development projects. Trac uses a minimalistic approach to web-based software project management. It provides an interface to Subversion and Git version control systems, an integrated Wiki and convenient reporting facilities. Trac provides an integrated Wiki system and a timeline showing all current and past project events in order, making the acquisition of an overview of the project, and tracking progress easy. It is released under a BSD-like license.

3. Bugzilla

Bugzilla [47] is a robust and mature defect-tracking system. It allows teams of developers to effectively keep track of outstanding bugs, problems, issues, enhancements, and other change requests for the application being developed. Bugzilla is free software released under a Mozilla Public License.

Analysis

GitLab Issues is the ideal choice for those using GitLab CE for code versioning since it is very well integrated. It is tied to each specific Git project, but you can report on groups of projects as well.

Trac is not a very sophisticated tool, but simple to use. However, Trac has a small community compared to GitLab CE users and developers.

Bugzilla is the oldest tool, once quite widespread, but with an old user interface and quite difficult to customize. Also, Bugzilla community does not release frequent updates and did not report many security issues over time.

11.3.2 Quality and Assurance Tools

This section provides an analysis for the tools split between static code analysers and dynamic analysers. In static code analysis, we perform an off-line verification of the source code to spot both issues that affects software quality and security; this analysis works at the programming language level and software code descriptors (e.g., build files) in a white-box fashion.

On the contrary, dynamic analysis verifies on-line the piece of software when it is running on a compute node. This type of analysis can detect issues by considering the running software as a black box. It is useful to spot, for example, security attacks at application level like input validation problems. Testing of live RESTful web services falls into dynamic analysis testing, as well.

Since MEDINA is going to be developed as a micro-services application running on containers, in addition to the analysis described, we can consider specific testing activities for assessing the security of the containers that will run MEDINA: we refer to these further testing as container security.

STATIC CODE ANALYSIS TOOLS:

1. SpotBugs

SpotBugs [48] is a program which uses static analysis to look for bugs in Java code. It is free software, distributed with a GNU Lesser GPL. SpotBugs can be used standalone and through several integrations, including Maven and Gradle. It is extensible through plugins and the most popular for our purposes are *FindSecurityBugs* and *FindBugs Contrib*.

In particular, ***FindSecurityBugs*** is a plugin for security audits of Java web applications. It can detect more than a hundred different vulnerability types including Command Injection, XML Injection, SQL Injection, and Cryptography weaknesses [49].

2. SonarQube

SonarQube [50] is an open-source tool for code quality and code security, distributed under GNU Lesser GPL. It performs source code reviews with static analysis to detect bugs, code smells, and security vulnerabilities on more than 20 programming languages, including Java, GO, and Typescript. SonarQube provides integration with Maven, Gradle and continuous integration tools like Jenkins.

3. OWASP Dependency-Check

OWASP Dependency-Check [51] is a Software Composition Analysis (SCA) tool that tries to detect well-known disclosed vulnerabilities contained within the dependencies of program. The tool extracts the Common Platform Enumeration (CPE) identifiers of the third-parties software libraries that the developed program depends on. It then uses such list to match against the Common Vulnerabilities and Exposures (CVE) public database to retrieve, when available, the correspondent security vulnerabilities. Dependency-Check integrates with Maven, Gradle, and Jenkins. It is released under the Apache Software License 2.0.

DYNAMIC CODE ANALYSIS TOOLS:

4. OWASP Zed Attack Proxy

OWASP ZAP [52] is one of the most popular open-source tool for the dynamic analysis. A very active and mature community supports the project. OWASP ZAP supports a wide range of scripting languages (e.g., JavaScript, Ruby, Python, etc.). ZAP provides several functionalities like intercepting proxy, passive scanner, forced browsing and provides a REST API to interact programmatically with the tool. Using such API, ZAP can be integrated with continuous integration servers, like Jenkins, to programmatically start a security scan against a deployed and running software components. ZAP can be used to test both web sites by URL and RESTful web services methods. ZAP is release under an Apache 2.0 License.

CONTAINER SECURITY TOOLS:

5. Anchore

Anchore [19] is a scanning tool that inspects container images to unpack and analyse everything inside. It can be easily integrated into the CICD workflow thanks to its Jenkins plugin. During the pipeline, if the scanning of the container image doesn't meet the security requirements, it fails and returns back a report or an alert via webhook notification. It allows to detect both CVEs and customizable policy rules. It is released as an open source under the Apache license.

6. Clair

Clair is a popular open-source container static vulnerability analyser. It periodically collects data and stores them into a database. If a vulnerability is found, it produces alerts, reports, or block release in production. It has an Apache license. Unfortunately, it does not provide integration with Jenkins.

7. Trivy

Trivy [53] is a tool that analyses operating system packages and application dependencies in container images. It is easy to install, suitable for CICD and integrates with Jenkins in the latest versions. It produces reports about the vulnerabilities detected, illustrating for each library the CVE id and a CVSS score assigned. It is open source under an Apache license.

11.3.3 Selection of tools

Table 11 enables to check what tool is in line with the corresponding non-functional requirements described in Section 10.2.1

Table 11. Mapping of CI/CD tools with NF requirements.

	CICD.01	CICD.02	CICD.03	CICD.04	CICD.05	CICD.06	CICD.07	CICD.08	CICD.09	CICD.10	CICD.11	CICD.12
GitLab	√					√	√	√	√	√		
Subversion	√					√	√	√	√	√		
Maven		√				√	√	√				
Gradle		√				√	√	√				
Nexus						√	√	√				
Artifactory						√	√	√				
GitLab CI/CD		√	√			√	√	√	√			
Jenkins		√	√		√	√	√	√	√			
Tekton		√	√			√	√	√	√			
JUnit			√			√	√	√				
TestNG			√			√	√	√				
RestAssured			√			√	√	√	√	√	√	√
Gitlab issues				√		√	√					
Trac				√		√	√					
Bugzilla				√		√	√					
SpotBugs						√		√				
SonarQube						√		√	√		√	√
OWASP ZAP						√	√	√	√	√	√	√
Anchore						√	√					
Clair						√	√					
Trivy						√	√					

A first selection of tools was made in the first version of this deliverable (D5.1). The final selection of tools for each class is presented in Section 5.2.1.