.

D5.2 ICOS Beta Release

Document Identification			
Status	Final	**Due Date**	30/06/2024
Version	1.0	**Submission Date**	28/06/2024

Related WP	WP5	Document Reference	D5.2
Related Delivera-ble(s)	D5.1, D2.2, D2.3, D2.4	Dissemination Level (*)	PU
Lead Participant	ENG	Lead Author	Gabriele Giammatteo
Contributors	NKUA, NCSRD, ATOS, BSC, TUBS, XLAB, SixSQ, UPC	Reviewers	Artur Jaworski (PSNC)
			Hrvoje Ratkajec (XLAB)

Keywords:
System Integration, CI/CD, Release, Testing, Assessment

Document Information

List of Contributors	
Name	Partner
Alex Barcelo	BSC
Artur Jaworski	PSNC
Francesc Lordan	BSC
Andreas Suarez Cetruolo	CeADAR
Jaydeep Samanta	CeADAR
Gabriele Giammatteo	ENG
Maria Antonietta Di Girolamo	ENG
Kalman Meth	IBM
Andreas Oikonomakis	NCSRD
Dimitris Santorinaios	NCSRD
Nikos Dimitriou	NCSRD
Menelaos Zetas	NKUA
John White	SIXSQ
Konstantin Skaburskas	SIXSQ
Fin Gentzen	TUBS
Marc Michalke	TUBS
Jordi Garcia	UPC
Montse Farreras	UPC
Hrvoje Ratkajec	XLAB

Document History			
Version	Date	Change editors	Changes
0.1	20/05/2024	ENG	ToC
0.2	01/06/2024	NKUA, NCSRD, ENG	First draft for section on testing Content for Integration process chapter
0.3	12/06/2024	ENG, NKUA	More contributions on section 4
0.4	17/06/2024	ENG	Final version ready for internal review
0.5	25/06/2024	XLAB, NKUA, ENG	Reviewed version
0.6	28/06/2024	ENG, PSNC. XLAB	Final version
1.0	28/06/2024	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Gabriele Giammatteo (ENG)	28/06/2024
Quality manager	Carmen San Román (ATOS)	28/06/2024
Project Coordinator	Francesco D'Andria (ATOS)	28/06/2024

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AI	Artificial Intelligence
API	Application Programmatic Interface
CI/CD	Continuous Integration / Continuous Deployment
CLI	Command Line Interface
CNI	Container Networking Interface
CVE	Common Vulnerability and Exposure
DNS	Domain Name System
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
ENG	Engineering S.p.A.
GUI	Graphical User Interface
HTML	HyperText Markup Language
IaC	Infrastructure As a Code
ID	Identifier
IT-x	Project's Iteration x
ML	Machine Learning
NCSRD	National Center For Scientific Research "Demokritos"
OCM	Open Cluster Management
OS	Operating System
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTx	Unit Test x
WPx	Work Package x

Executive Summary

This document describes and accompanies the second release of the ICOS software: the ICOS Beta release. The release has been delivered at project's month 22, seven months after the first release ICOS Alpha. The ICOS Beta release was developed and integrated following the system architecture and the implementation plan, and incorporating changes implemented from the feedback received from the first evaluation of ICOS provided by the project's Use Cases and the first project's review.

This release brings several improvements in the ICOS software in terms of new functionalities, in all the system areas: runtime management, orchestration, security, intelligence and data management.

The document also describes the main activities carried out and the tools used to successfully integrate the release. It focuses more on new processes and tools introduced for the ICOS Beta release like the creation of technical documentation and the publication of the source code in GitHub.

During the integration of the ICOS Beta release a more formal, coherent, and comprehensive approach to software testing was introduced. A set of goals and a methodology have been defined for executing unit testing of the ICOS components as well as quality verification of the source code and artifacts of the ICOS releases. The document outlines the main type of tests executed, the tools used and a summary of the results.

Finally, a plan for the next release (ICOS Final) is presented considering a) the new requirements elicited and prioritized for the second iteration; b) the new version of the system architecture and c) the feedback from the project's Use Cases validation.

1 Introduction

1.1 Purpose of the document

This document accompanies the delivery of the ICOS Beta release. The main goal is to provide information on the software that composes the release. The document lays out the structure of the release, the functionalities delivered, how to get the source code and the software packages, how to deploy and use the software.

The document also presents the process adopted and the tools used for the integration of the ICOS Beta release. A summary is presented for the core integration activities that were already presented in the previous WP5 deliverable "D5.1 - First ICOS Release: ICOS Alpha" [1]. Two activities are described more in detail since they have been introduced for the ICOS Beta for the first time: the creation of the technical documentation for the ICOS Meta OS and the publication of the source code in a public repository (GitHub) for the ICOS Beta release.

The document also presents the testing process adopted in ICOS to assess the quality of the ICOS releases. While for the first release, an end-to-end testing workflow was provided, for the ICOS Beta a per-component test plan was defined and executed. It started from the definition of a set of goals for the testing activities that identified which aspects and properties of the software the consortium wanted to assess. Then, a formal methodology and process for the testing of those aspects was defined. This was the input for the implementation of the test cases, the selection of the testing tools, the execution of the tests and, finally, the collection of the results. The different phases are presented in the document as well as the results from the testing activities.

Finally, the document presents a plan for the integration activities and, in particular, the release of the last and final ICOS release in project's month 30. The main functionalities that are expected for that release are listed to provide a guidance for development, integration, and testing teams and to plan the work in the WP5.

The document is mainly intended to be read by technical teams both internal or external to the ICOS project that wants either to deploy and manage an ICOS System (to understand how the software is released and where to get the artifacts and the documentation) or to contribute to the ICOS software (to understand where to access the source code and how the integration, testing and release processes work).

1.2 Relation to other project work

This deliverable presents the ICOS Beta release as results of the integration work done in Work Package 5. ICOS Beta is the second release of the ICOS Meta OS. It is created from the evolution and the improvement of the first release ICOS Alpha, which is documented in deliverable D5.1.

In addition, the ICOS software requirements and the architecture were a reference source during the integration activities. In particular, the deliverables "D2.3 - ICOS ecosystem: Technologies, requirements and state of the art (IT-2)" [2] and "D2.4 - ICOS architectural design (IT-2) [3] have been considered.

Finally, this document will be updated by the deliverable "D5.3 - Third ICOS Release: Complete ICOS version" that will be released in project's month 32 and will document the final ICOS release.

1.3 Structure of the document

This document is structured in 6 major sections:

1. “Introduction”: (this section) presents the objectives of the document and introduces its content, structure, and relationships with the other project’s deliverables.
2. “ICOS Beta Release Notes”: presents the main details of the ICOS Beta release including its structure, a changelog, where source code, binaries and documentation can be found.
3. “Release Integration”: reports the process, the tools and the work done for the integration of the release. It highlights new activities and tools introduced during the integration of ICOS Beta.
4. “Release Testing”: presents the formal testing process and the tools introduced in the ICOS Beta release. It describes the main methodology and goals for the testing activities, the type of tests defined, how they have been implemented and the results.
5. “Release Plan”: presents the plan for the integration and release of the final ICOS release.
6. “Conclusions”: summarizes the content of the document and provides final considerations on how the technical work will continue in the project.

2 ICOS Beta Release Notes

ICOS Beta is the second software release of the ICOS project delivered in project's month 22 (June 2024). This release includes several updates with regards to the first ICOS release (November 2023). This is also the first release for which the source code and the technical documentation are publicly available.

This section briefly presents the main new functionalities of the release (section 2.1) and provides links for the online resources: source code (section 2.2), artifacts (section 2.3) and documentation (section 2.4).

2.1 What's New

ICOS Beta delivers several improvements for all the layers of the ICOS Meta OS.

In the **Meta-Kernel** layer, several new functionalities have been introduced to allow a better orchestration of more complex applications. The main improvements are:

- Added support for the orchestration of multi-component applications.
- Added support for additional orchestration requirements like minimal security level requirement, node hardware architecture (e.g., amd64, arm), peripheral availability.
- Added full support for nodes orchestrated by Nuvla (supporting both Docker and Kubernetes runtimes).
- Added monitoring of energy efficiency metrics produced by Scaphandre.
- Added support for collecting, storing, and visualizing system and applications logs produced in the nodes.
- Improved discovering of nodes peripherals.
- Introduced the Policy Manager component supporting node and application performance policies monitoring with pre-defined application scale-up/down remediation actions.

In the **Security** layer, additions have been done to enlarge the security assessment capabilities and improve the security in the ICOS Controller and Agents. In particular:

- Added the Wazuh security vulnerabilities scanning and the SCA score for ICOS nodes.
- Added LOMOS logs analysis.
- Added integrated audit mechanism using Tetragon tool to monitor kernel security related events in the Kubernetes cluster.
- Added encryption of communications between ICOS services using TLS encryption mechanism.
- Implemented OAuth2 authentication in ICOS services to secure service-to-service calls.

In the **Intelligence** layer, an improvement in the management of the models along with a deeper integration with the ICOS Meta-Kernel have been achieved. In particular:

- Introduced prediction of telemetry metrics. This mechanism, integrated with the ICOS Telemetry database, allows for having a predictive monitoring data available in the ICOS Meta-Kernel.
- Added drift detection for model monitoring.
- Model explainability XAI in MLOps.

The **Data Management** services have been improved releasing a new version (DataClay version 4.0) that includes ICOS specific features. In particular:

- Added asynchronous support on Data Management services for more parallelism and throughput.
- Data access proxy services available, which can be integrated with ICOS IAM and provide fine-grain ACL.
- More support for structured data models.

Finally, a new web-based ICOS **Shell** GUI has been introduced for connecting to an ICOS Controller, check the available resources and the application deployed.

2.2 Source Code

The open-source code of the ICOS Beta release is publicly available in a GitHub organization at: https://github.com/icos-project.

Within the GitHub organization, the code is organised in multiple repositories:

- repositories related to the **single components** (e.g., Job Manager, Match Maker). In general, there is one repository per component, but some components are divided into multiple repositories.
- the **"ICOS Meta OS"** repository that collects all components together. This repository exploits the *git submodules* concept to have a snapshot of the entire ICOS Beta release in a single repository. It is located at https://github.com/icos-project/ICOS-Meta-OS.
- the "**ICOS Suites"** repository that contains the source code for the ICOS Suites (see section 2.3) that are the main artifacts used for the distribution and deployment of the ICOS Meta OS. It is located at https://github.com/icos-project/ICOS-Suites.
- the "**ICOS Shell"** repository contains the source code for the user-level tools used to interact with the ICOS System. It is located at https://github.com/icos-project/ICOS-Shell.

The public repositories only contain the released code (ICOS Beta version), while development version of the components and the non open-source code are managed internally in the ICOS project in a private **GitLab instance** (see section 3.1). Access to this instance can be given upon request.

2.3 ICOS Suites

The ICOS Meta OS software is distributed as three different **ICOS Suites.** They represent three different distributions to be used to deploy ICOS depending on the type of node:

- **ICOS Controller Suite**: used to deploy a new ICOS Controller. It is distributed as a Helm Chart that includes all ICOS Controller services.
- **ICOS Agent Suite**: used to deploy a new ICOS Agent. It is distributed as a Helm Chart that includes all ICOS Agent services.
- **ICOS Client Suite**: this is the suite that contains the tools to interact with an ICOS System. It is intended to be installed by ICOS users.

Table 1 provides a mapping of which services are included in each ICOS Suites for the ICOS Beta release. A complete and detailed description of the ICOS Suite is provided in the deliverable (D5.1, section 2).

Table 1: ICOS Suites composition in ICOS Beta

Suites composition			
Component	Client	Controller	Agent
Shell CLI	X		
Shell BackEnd		X	
Job Manager		X	
Aggregator		X	
OCM Deployment Manager			X
Nuvla Deployment Manager			X
DataClay		X	X
LOMOS APIs		X	
Intelligence API		X	
Telemetry Controller		X	
Dynamic Policy Manager		X	
Security Layer Coordination Module API		X	

There are additional components released as part of ICOS Beta that are not part of any suite and need to be installed separately. They are:

- **Telemetry Agent**: this is a set of ICOS and third-party components that are installed on the ICOS nodes to collect metrics and logs. It is distributed as Helm Chart and Docker Compose.
- **ICOS Lighthouse** and the **ICOS Identity and Management** services: they have not been integrated yet in any suite and need to be managed separately.
- **Third-party** components: software not developed by ICOS and not listed as part of the ICOS Beta. However, they are requested for having a fully functional ICOS System. For instance, Open Cluster Management[1] (OCM), Nuvla[2] and Wazuh Server and Agent[3].

All the artifacts, mentioned above, released as part of the ICOS Beta release, are publicly available in the repository https://harbor.res.engit/icos.

This repository contains both the Helm Charts and the Docker Images used in the deployment of the ICOS nodes and can be directly referenced in the installation commands. For instance, the command to install an ICOS Controller can be similar to:

```
helm install --namespace icos-system icos-controller \
  oci://harbor.res.eng.it/icos/helm/icos-controller \
  --set global.icos.controllerId=ctrl123 \
  --set global.external.host=10.160.3.236
```

For a guide on how to install the different artifacts of the ICOS Beta release, please refer to the online Administrator Guide (see section 2.4).

[1] https://open-cluster-management.io/
[2] https://nuvla.io/
[3] https://wazuh.com/

2.4 Documentation

The technical documentation of the ICOS Meta OS can be found online at the following URL: https://www.icos-project.eu/docs/.

The website hosts the four different ICOS technical guides: i) Concepts Guide, ii) Administration Guide, iii) User Guide and iv) Developer Guide. The details on each guide, the process and the work that led to the creation of this documentation are reported in detail in section 3.2.

An additional source of documentation is the ICOS project website at https://www.icos-project.eu/. The website contains technical and non-technical publications, information on the project's Use Cases and the project's deliverables.

Finally, all scientific publications supported by the ICOS project that presents the research at the basis of the ICOS software are collected at https://zenodo.org/communities/icosproject.

3 Release Integration

The integration of the ICOS Beta release required a coordinated and collaborative effort by all project's technical partners. This section briefly summarizes the work done for the integration of the ICOS Beta Release (section 3.1), the realization of the technical documentation (section 3.2) and the publication of the source code (section 3.3). The testing activities carried out during the integration of the release are extensively discussed in section 4.

3.1 Release Integration Process and Infrastructure

Since the beginning of the project, a well-defined and rigorous Continuous Integration and Continuous Deployment (CI/CD) process has been defined supported by multiple tools set-up and maintained by the project's partners. This allowed to establish a continuous development, integration, release, and validation cycle that allows to run development activities in parallel with integration and validation activities, as well as reduce the time for newly developed features to be available to Use Cases for validation.

The process and the tools are extensively described in deliverable D5.1. For convenience, the diagram of the main integration tools (Figure 1) and the integration process (Figure 2) are reported below.

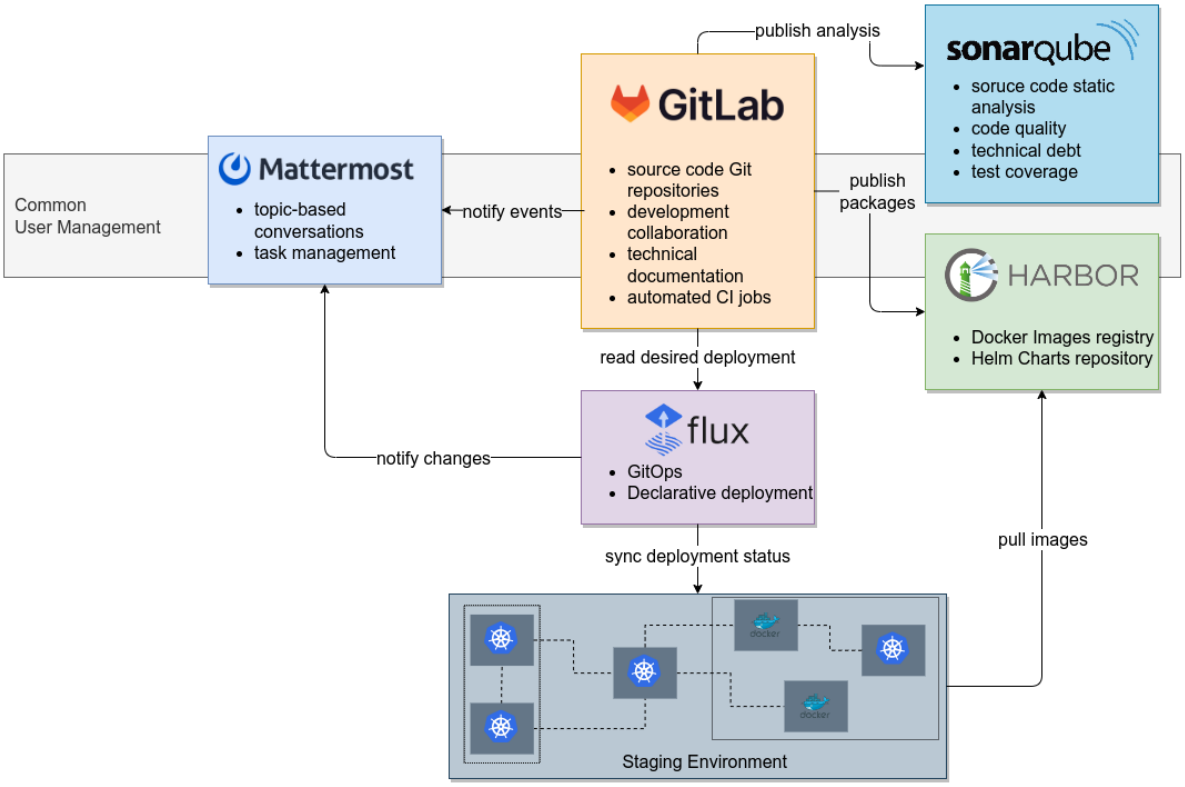

Figure 1: Integration Infrastructure

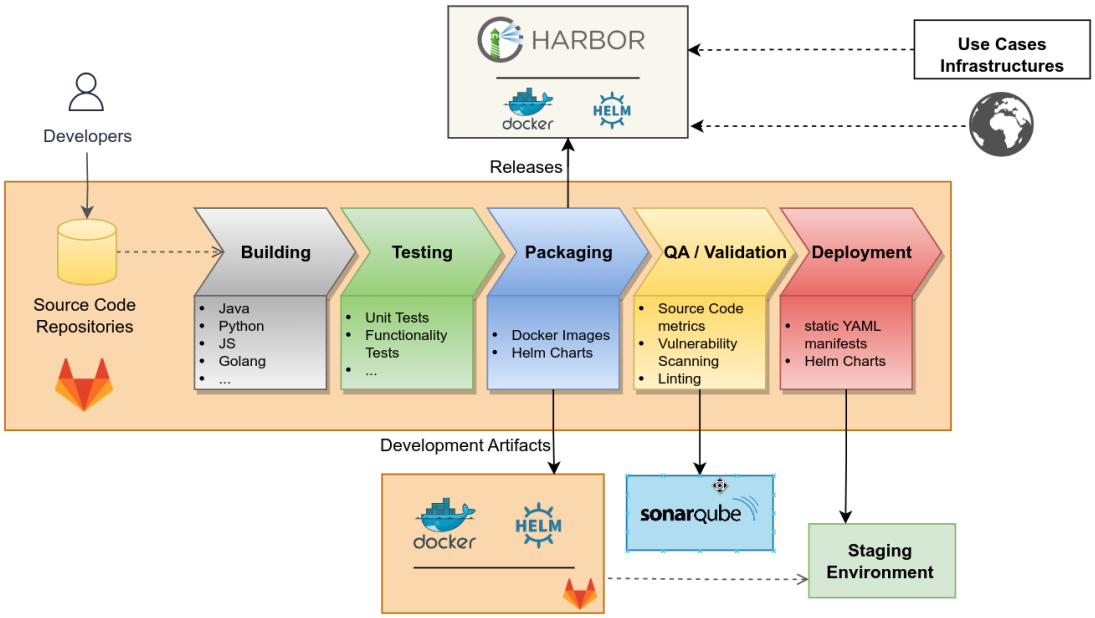

Figure 2: Integration Process

The integration of the ICOS Beta release exploited the same tools and procedures set-up for the CI/CD of the ICOS components. For each component under release, the automated GitLab jobs allowed to build the component packages, check their quality, and automatically deploy them in the staging environment. The source code of the ICOS Suites has been also uploaded in GitLab and followed a similar pipeline allowing to test their integration and the deployment in the staging environment in an effortless and automated way.

3.2 Technical Documentation

An important activity that accompanied the integration and the release of the ICOS software in IT-2 was the creation of a proper technical documentation for the ICOS Meta OS. The benefits of having such a documentation are multiple. Internally, it constitutes a common knowledge base for all the people working in the project and helps in the day-by-day development, integration, and validation activities. Externally, it is useful for people that want to learn and use the ICOS Meta OS (e.g., the Open Call partners). It also greatly supports the dissemination, exploitability, and sustainability of the software.

It has been decided to create four different guides for the ICOS Meta OS:

- **Concepts Guide**: provides a theoretical introduction to the ICOS Meta OS introducing the architecture, the main functionalities, and the main components. This guide is intended for persons that want to be introduced to ICOS, what it is and how it works.
- **Administration Guide**: provides tutorials and instructions on how to create and manage an ICOS System. This guide is intended for the administrators of an entire ICOS System or a part of it (e.g., an ICOS Agent).
- **User Guide**: provides tutorials and instructions on how to use ICOS Meta OS for running user's applications. This guide is intended for application developers and integrators that want to use ICOS for orchestrating their applications.

- **Developer Guide**: provides tutorials and instructions on how to use and configure single ICOS components and how to programmatically integrate and/or extend them. It includes a reference for all commands and APIs exposed by ICOS. This guide is intended for expert technical persons who want to learn how ICOS works internally or want to develop integrations or extensions for ICOS components.

The creation of content for the four guides was carried out in WP5 with a structured and collaborative effort. After the table of content was agreed, an editor and a reviewer were appointed for each section. The status and the progresses were discussed weekly in WP5 calls. Content for the "Concepts Guide" section has been largely adapted from already existing content published in ICOS deliverables and papers. In addition, some sections (e.g., API references) of the "Developer Guide" have been auto generated with specific tools (e.g., Swagger[4]). This was a convenient way to have an updated and accurate documentation with a little effort.

Concerning the **format** of the guides, it has been decided to create them as an **HTML Website** instead of Word or PDF documents. The benefits are multiple:

- easier distribution and access: the guides are hosted along with the ICOS Website at https://www.icos-project.eu/ and are easily and publicly accessible over Internet.
- easier maintenance and update: the content can be updated without the need of downloading new versions of the document.
- easier interaction with the content: hyperlinks, collapsible sections and video/animation offer an easier and richer reading experience.

Another benefits of using HTML pages for the documentation is the automation. The guides are written as **Markdown documents** and stored in a shared Git repository. At every change pushed to the documentation repository, an automated job generates the HTML website (using MkDocs[5]) that is uploaded on the hosting server. This automation minimizes the effort for creating and publishing the guides and allow editors to focus more on the content (the Markdown files) while the formatting and the styling is automatically applied during the automatic generation of pages in a consistent way across all the guides. It also makes updating and distributing the documentation effortless.

3.3 GitHub publication

The ICOS source code is hosted in a private GitLab instance set-up and maintained by the ICOS consortium. This decision has been taken at the beginning of the project to be able to share and collaborate on source code within the consortium, but, at the same time, avoid making source code with an undecided license and not enough maturity public.

However, it has been decided to upload the open-source code of the ICOS Beta release in a public repository. The decision was taken considering that the source code of this release is mature, tested and documented enough to be published. Multiple benefits can be obtained from the publication of the source code, such as:

- easier **collaboration** with developers external to the ICOS consortium (e.g., Open Call partners, other EU research projects' developers).
- make the code findable and analysable by the open-source community increasing the **awareness** and the **trust** for the ICOS Meta OS.
- start to build a **community** of people around the ICOS Meta OS that can test, use and extend ICOS leading to the creation of an **ecosystem** of tools and applications based on ICOS.
- support the **exploitation** strategy of the ICOS components.

[4] https://swagger.io/
[5] https://squidfunk.github.io/mkdocs-material/

As a repository to host the code, **GitHub**[6] was selected as being the biggest and most popular developers' community. The code for ICOS is hosted in a GitHub organization named "ICOS Project" at the URL: https://github.com/icos-project.

The publication procedure followed two main phases:

1. selection and preparation of the source code.
2. adaptation and upload in GitHub.

For the **first phase**, an internal **checklist** has been created and shared with the developers in ICOS to determine whether a given repository was eligible to being published or not:

- the source code is correctly licensed under an open-source license and follows the rules of that license (e.g., source code files start with a header with a reference to the license);
- presence of README and LICENSE files.
- the README file reports the essential information to understand what the code is about, its structure, how to build and how to use it.
- software quality and security scanning have been performed on the source code and reported no major issues. This should be verified by the execution of SonarQube[7] scanning configured as part of the building pipeline for each component.
- the source code is accompanied by a proper documentation that can be located, depending on the component, in the README file, in a dedicated folder (e.g., "docs/") and/or in the source code files.
- for components that expose a REST API, the repository should contain a schema file following the OpenAPI specification[8].
- the source code is accompanied by automated Unit Tests. At this stage, a coverage threshold is not set, but it might be set for the next release.
- a Git tag corresponding to released version of the component and corresponding release notes should be available.

The **second phase** consisted in uploading all eligible repositories in GitHub. Unlike internally used GitLab[9] where the code is grouped in multiple groups and subgroups, GitHub does not have the concept of grouping, so multiple code repositories have been merged in few top-level GitHub repositories using the concept of *git submodules*[10]. The following repositories have been created in GitHub:

- ICOS-Meta-OS (https://github.com/icos-project/ICOS-Meta-OS): to group all the source code of the ICOS System in a single codebase.
- Shell (https://github.com/icos-project/Shell): to host the source code for the ICOS Shell tools.
- Suites (https://github.com/icos-project/Suites): to host the source code for the ICOS Suites used to deploy ICOS.
- Documentation (https://github.com/icos-project/Documentation): to host the documentation for the ICOS software (see section 3.2).

First, each source code repository in GitLab has been uploaded in GitHub. Then the grouping hierarchy has been recreated in the top-level repositories (listed above) using folders and submodules. This was done with a semi-automated procedure to be able to easily replicate it for multiple repositories and for the next releases.

[6] https://github.com/
[7] https://www.sonarsource.com/products/sonarqube/
[8] https://swagger.io/specification/
[9] https://production.eng.it/gitlab
[10] https://www.git-scm.com/book/en/v2/Git-Tools-Submodules

4 Release Testing

The goal of this section is to provide a detailed overview of the testing strategies and methodologies employed in the ICOS project. By delving into the testing goals, methodologies, and specific test suites implemented, it aims to elucidate how each component of the ICOS software system is rigorously validated to ensure its robustness, reliability, and efficiency. This section outlines the comprehensive testing framework currently in place, setting the stage for the future expansion of testing efforts in subsequent deliverable.

To date, a comprehensive Unit Test methodology has been meticulously established to validate the functionality and reliability of individual components within the ICOS software ecosystem. This foundational framework ensures that each unit, from the smallest function to larger modules, operates correctly in isolation, minimizing dependencies and enabling precise detection of issues early in the development cycle. By rigorously applying this methodology, high standards of accuracy, reliability, and maintainability are upheld across all ICOS applications.

The next crucial step involves expanding the testing efforts to encompass additional components, thereby enhancing the breadth and depth of the Unit Tests. This extension will ensure that every part of the system is thoroughly vetted, reinforcing the commitment to delivering robust and high-quality software solutions. Moreover, comprehensive end-to-end tests, including integration tests, will be integrated to validate the seamless execution and interaction of all ICOS components within the full deployment environment. These tests are vital for confirming that the entire ICOS shell and its components function cohesively, meeting all operational and performance expectations in real-world scenarios.

The structure of this section is organized into several subsections, each focusing on a specific aspect of the testing approach:

1. **Component Unit Testing and Helm Testing**: This section provides an overview of the comprehensive testing framework implemented in the ICOS project. By detailing specific test suites and execution results, it demonstrates how each component of the ICOS software system is validated to ensure robustness, reliability, and efficiency. each test suite, covering components like the Aggregator, Deployment Manager, Intelligence Layer, Job Manager, Policy Manager, Scaphandre[11] and Security Layer.
2. **Source Code Quality**: This part discusses the importance of code quality and the methodologies employed to measure and ensure it throughout the development cycle. Tools like SonarQube[12] are highlighted for their role in assessing multiple non-functional properties of the source code.
3. **Container Security Vulnerability Scanning**: This section focuses on the methodologies and results of scanning Docker images for security vulnerabilities, ensuring the reliability and trustworthiness of the software.
4. **Staging Testbed**: Finally, this subsection provides an overview of the enhanced infrastructure environment, detailing hardware updates, Kubernetes cluster configurations, and the transition to more efficient and scalable solutions.

Additional details on testing are provided in the annexes at the end of the document:

1. **Annex I - Testing Goals**: (at Annex) This subsection outlines the various types of tests conducted to cover critical areas of the application lifecycle, including component

[11] https://github.com/hubblo-org/scaphandre
[12] https://www.sonarsource.com/products/sonarqube/

connectivity, functionality verification, configuration accuracy, integration readiness, and operational behaviour.

2. **Annex II - Unit Testing Methodology**: This part delves into the systematic approach taken for unit testing, detailing the key elements such as test case ID, test description, preconditions, test steps, expected results, actual results, and status. This methodology ensures that each unit of the application is tested in isolation to confirm its intended performance.

By structuring the section in this manner, the aim is to provide a comprehensive and coherent narrative of the testing strategies and methodologies employed in the ICOS project, highlighting the commitment to delivering high-quality and reliable software solutions.

4.1 Component Unit Testing and Helm Testing

Methodology and Goals

Component unit testing and Helm testing are integral to ensuring that each piece of the ICOS software performs as expected both independently and within the Kubernetes environment. This section will discuss the execution and results of these tests, focusing on their impact on software quality and operational reliability.

1. Component Unit Testing: Each component within ICOS is tested to validate its functionality and interaction within the system. Tests for components like the Aggregator, Deployment Manager, and Intelligence Layer ensure that each unit performs its role effectively, facilitating seamless system operations.
2. Helm Tests: Helm tests validate the integration and functionality of Kubernetes resources managed through Helm charts. These tests ensure that configurations are applied correctly and that components function cohesively within the deployment environment. Key results from these tests, demonstrate successful deployments and operational functionality.

Implementation

1. **Aggregator**: The component tests for the Aggregator application are designed to verify its deployment on the Kubernetes testbed and ensure it successfully retrieves all required metrics from Thanos. These metrics pertain to the OCM and Nuvla clusters and nodes. The tests include:

Table 2: Aggregator Component Unit Test

Definition of the Unit Tests	Verify the Aggregator application component deployed on Kubernetes testbed to check it gets all the required metrics from Thanos[13]. These metrics are related to the OCM and Nuvla clusters and nodes.
Definition of the Interfaces to be tested	Kubernetes API to check the correct deployment of the application, and Aggregator API for metrics reporting.
Unit Test Cases descriptions	**UT1**: Aggregator is correctly deployed on the Kubernetes testbed. **UT2**: Thanos metrics are correctly collected and formatted by the Aggregator.

[13] https://thanos.io/

2. **Deployment Manager:** The component tests for the Deployment Manager are designed to ensure it can pull all executable jobs and execute them to successfully create a deployment. The tests include:

Table 3: Deployment Manager Component Unit Test

Definition of the Unit Tests	Make sure the Deployment Manager can pull all executable jobs and execute them to successfully create a deployment.
Definition of the Interfaces to be tested	Deployment Manager API.
Unit Test Cases descriptions	**UT1**: Successful Job Deployment on Deployment Manager.

3. **Intelligence Layer**: The component tests for the Intelligence Layer are crafted to validate the successful deployment of the associated API, its interaction with various ICOS components, and the availability of the features it offers. These tests are summarized as follows:

Table 4: Intelligence Coordinator Component Unit Test

Definition of the Unit Tests	Verify the deployment of the Intelligence co-ordination API and ensure if all its components are working as expected.
Definition of the Interfaces to be tested	Intelligence co-ordination API, model training and prediction API, Lomos[14] API, Mlflow[15] UI for tracking models, Jupyter[16] Hub and Jupyter Lab instances for running Jupyter notebooks
Unit Test Cases descriptions	**UT1**: Successfully launching and running the Docker instance for the intelligence co-ordination API. **UT2**: Access the Intelligence layer co-ordination UI on any browser. **UT3**: Perform model training with the specified dataset and model training parameters. **UT4**: Offload model training workload to DataClay. **UT5**: Perform model inference with the specified trained model to generate predictions. **UT6**: Access the MLFlow Tracking UI to visualise and manage multiple trained models. **UT7**: Start Jupyter notebook sessions for performing ML operations utilising the Docker environment. **UT8**: Get anomalies from the Lomos API service.

[14] https://xlab.si/solutions/artificial-intelligence-and-machine-learning/

[15] https://mlflow.org/

[16] https://jupyter.org/

4. **Job Manager:** The component tests for the Job Manager are designed to verify its deployment on the Kubernetes Management cluster and ensure it can create an application deployment from the application description file, allocate resources via the Matchmaking service, and prepare the application for execution. The tests include:

Table 5: Job Manager Component Unit Test

Definition of the Unit Tests	Verify the deployment of Job Manager Helm chart on Kubernetes Management cluster and ensure its able to create an application deployment from the application description file and prepare it for execution.
Definition of the Interfaces to be tested	Helm Chart parameters, Job Manager API for deployment, Matchmaking API for allocation of the deployment.
Unit Test Cases descriptions	**UT1:** Successful deployment of Job Manager on a Kubernetes management cluster. **UT2:** Successful creation of an application deployment (in form of jobs) from application description file. **UT3**: Request target node for each application's component to Matchmaking service. **UT4:** Application deployment becomes executable (all components are ready to be executed).

5. **Policy Manager**: The component tests for the Policy Manager are designed to ensure its successful deployment, verify the proper functioning of all its features, and confirm that the UI is accessible via a web browser. These tests include:

Table 6: Policy Manager Component Unit Tests

Definition of the Unit Tests	Verify the deployment of the Policy Manager, ensure all things working as well and UI can be accessed by the browser.
Definition of the Interfaces to be tested	Policy Manager UI
Unit Test Cases descriptions	**UT1**: Successfully launching and running the helm instance for the Policy Manager. **UT2**: Run python unit tests. **UT3**: Access the Policy Manager UI. **UT4**: List of all Policies created is showed from the Policy Manager UI. **UT5**: History of the policy selected is showed from the Policy Manager UI.

6. **Scaphandre**: Helm tests for the Scaphandre component are specifically designed to verify its power consumption metrics collection capabilities within a Kubernetes environment. The tests summarized below:

Table 7: Scaphandre Helm Test Suite

Definition of the Unit Tests	Verify the deployment of Scaphandre Helm chart on Kubernetes nodes and ensure it monitors energy consumption accurately.
Definition of the Interfaces to be tested	Helm chart parameters, Kubernetes API for deployment, Scaphandre APIs for energy data reporting.
Unit Test Cases descriptions	**UT1**: Successful deployment of Scaphandre on a Kubernetes node (test health of pod and response status code of a simple GET operation to Scaphandre to be 200). **UT2**: Attempted deployment of Scaphandre on an ARM Kubernetes node (Expect failure as Scaphandre in not compatible with ARM architecture). **UT3**: Energy consumption data collection from a node running Scaphandre (validate that the response from calling Scaphandre service contains specific eco efficiency metrics).

7. **Security Layer**: The component tests for the Security Layer aim to ensure the successful deployment of the component, verify access to the Security Layer Coordination API, and confirm that its primary functionalities are operating as intended. These tests are outlined as follows:

Table 8: Security Layer Test Suite

Definition of the Unit Tests	Verify the deployment of the Security Layer Coordination API and ensure all its components are working as expected and the UI can be accessed by the browser.
Definition of the Interfaces to be tested	Security Layer Coordination API
Unit Test Cases descriptions	**UT1**: Successfully launching and running the helm instance for the Security Layer Coordination API **UT2**: Access the Security Layer Coordination API **UT3**: List all Security Scan agents in the ICOS topology **UT4**: Run Security Scan for vulnerabilities **UT5**: Get Security Scan results for vulnerabilities in the Telemetry metrics format

Execution and Results

Component tests of all components, including the Intelligence Layer, Security Layer, Policy Manager, and more, has consistently demonstrated successful deployments and functionality within the CI/CD pipeline and on the testbed. Each component has been verified to perform its intended functions smoothly, with no disruptions during deployment or updates. This ensures that all systems operate reliably, maintaining expected performance and stability throughout various deployment scenarios.

Helm tests for the component mentioned are also executed as part of the CI/CD pipeline whenever changes are pushed to the repository. Also, these tests are run for each component when it is deployed to a node. These tests can also be manually triggered pre-installation, post-upgrade, and before the release is fully integrated into the production environment. This ensures that any deployments or updates meet the expected criteria before they affect the production systems.

The aggregated results of the Helm tests have consistently demonstrated successful deployments and functionality within the Kubernetes environment of nodes. The tests have verified that components deployments and functionalities are working as expected. Additionally, these tests have ensured that configuration updates (changing memory or disk limitations) are applied seamlessly and without disruption to ongoing processes.

4.2 Source Code Quality

Methodology and Goals

Most of the software developed in ICOS will be released as open-source project and will be published in a public repository (see section 3.3). An important property of the source code to be ensured before making it public is its quality. **Code quality** is a property that is the sum of multiple different non-functional properties of source code like:

- readability: how easy it is to read and understand the code;
- maintainability: how easy it is to change and evolve the code;
- reusability: how easy it is to adapt the code to use it in a different context;
- reliability: can the code work without failures;
- testability: how easy it is to test the code;
- security: is the code immune to known security vulnerabilities (e.g., does not use vulnerable libraries/functions);
- documentation: how well is the code documented with code comments, docstrings and/or other types of documentations;
- portability: how portable is the code (i.e., can be compiled for different hardware architectures).

In the case of ICOS, having a code with a good quality is critical for the adoption and evolution of the system by external actors (exploitation) and for the sustainability of the system after the end of the ICOS project.

It has been decided to measure the quality of the code of the software developed in the ICOS project in a consistent and reliable way during the entire development cycle and not just at the end. This allows for faster feedback to developers that can take the quality of their code under control since the beginning of the development.

In additional, a check on the quality of the code is done before publishing the code, ensuring that only source code that passes the quality check can be published.

Implementation

The main tool used to measure the quality of source code used in ICOS is **SonarQube**. The main reasons that led to the selection of this tool are that: a) it includes measurement of several aspects of code quality in a single tool (while many other tools only measure one aspect), b) has customizable thresholds, c) can integrate new analysis, programming languages and tools through plugins, d) it easily integrates with CI/CD tools, and d) it is an open-source project.

An instance of SonarQube has been deployed and it is maintained for the ICOS project by Engineering S.p.A. (ENG). In GitLab, for each repository that contains source code, an automated job has been created that execute the analysis of the source code and uploads the results to SonarQube. This job is triggered and run for each commit in the repository. As depicted in the screenshots in Figure 3, the job fails in case that the quality threshold is not satisfied. The jobs also print useful information to debug the quality of the source code. The threshold is configured and can be customized directly in the SonarQube portal.

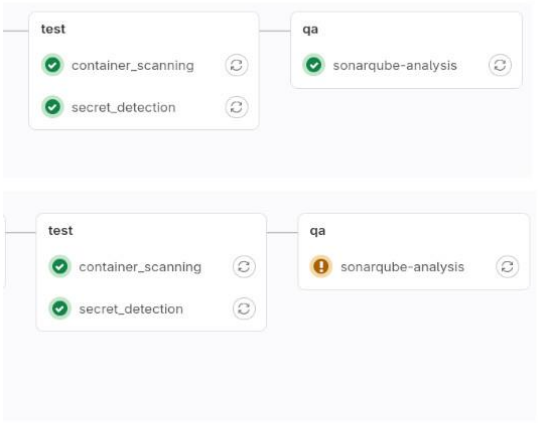

Figure 3: GitLab and SonarQube integration

In addition to SonarQube analysis, additional jobs have been added in GitLab to check the quality of the code:

- **secrets detection**: scans the files in the code repositories looking for strings that resemble passwords, secrets and private keys. This is a very effective check to enforce the good practice of not sharing secrets in code repositories;
- **Dockerfile and Helm linting**: analyses the Dockerfile and the Helm's files to find errors or bad practices.

Execution and Results

All analysis executed in the GitLab jobs are uploaded in SonarQube and can be accessed through its web portal. The portal keeps the current status of the quality of code for each component as well as the historical trends and make it is easy to aggregate results across all projects as well as navigate and review the issues for single projects.

SonarQube aggregates and compute multiple quality metrics. Figure 4 shows the quality indicators for all the analysed ICOS code repositories collected on the 28th of June 2024.

Figure 4: SonarQube quality metrics

The main conclusions on these results and guidelines for the upcoming activities are that:

- the ICOS source code has a very good Reliability and Maintainability (all analysed components score "A");
- some components have security suggestions to address (potential security issues that needs to be assessed manually);
- the coverage of unit tests needs to be improved (however, not all the unit tests are counted by SonarQube).

4.3 Container Security Vulnerability Scanning

Methodology and Goals

The main distribution format for ICOS software is Docker Images. Each ICOS component is packaged into a Docker Image that is built during the CI/CD pipeline. It is extremely important to assess the quality and the security of the generated Docker Images to make sure that the software delivered is trustable and reliable.

Implementation

There exist multiple tools that can scan Docker Images. In ICOS, Trivy[17] was selected for its capabilities and ease of integration in the existing GitLab CI/CD pipeline. Trivy is able to identify issues in Docker Images related to i) OS packages and software dependencies; ii) known Common Vulnerabilities and Exposures (CVEs); iii) Infrastructure as a Code (IaC) issues and misconfigurations; iv) sensitive information and secrets and v) software licenses.

[17] https://aquasecurity.github.io/trivy/

Execution and Results

Trivy is integrated as an automated job in the GitLab CI/CD pipelines for all source code repositories that generates a Docker Image. The job executes the Trivy scan of the newly generated Docker Images and reports the issues found in the logs.

At the moment of writing, while the scans are executed systematically for all the ICOS Docker Images, the results are only printed in the execution logs, so no aggregated data is available. This aspect will be improved in the remaining part of the project by exporting the results as metrics that can be easily analysed and aggregated (e.g., "total number of high priority vulnerabilities found").

4.4 Staging Testbed

The infrastructure environment provided by National Center For Scientific Research "Demokritos" (NCSRD) has been enhanced with several updates to improve its computing and networking capabilities. Firstly, on the hardware level, two additional servers have been integrated into the infrastructure, expanding the capacity and potential for virtualization. The Proxmox[18] cluster now consists of five dedicated physical servers as depicted in Figure 5, further improving performance.

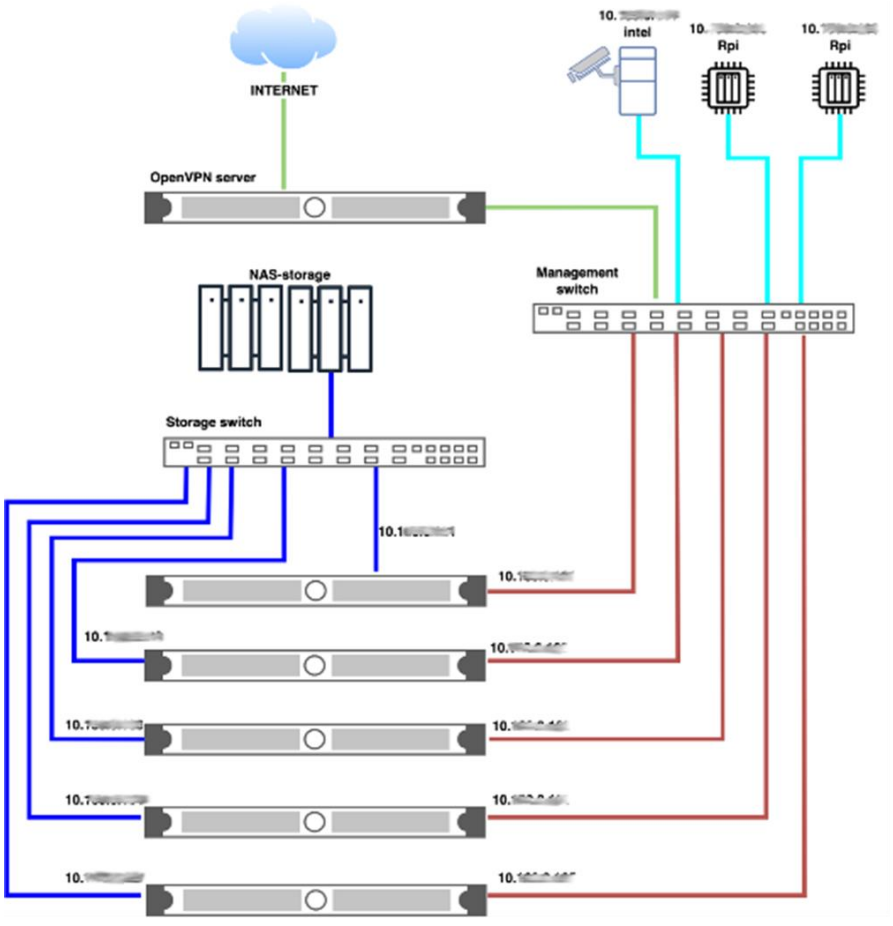

Figure 5: NCSRD Testbed

[18] https://www.proxmox.com/en/

Regarding the Raspberry Pi devices, the previous setup using Minikube[19] has been deprecated. Both Raspberry Pi 4 devices now run K3s, a lightweight Kubernetes distribution, and are configured with the Cilium Container Network Interface (CNI). The transition from Minikube to K3s was motivated by the need for a more efficient and scalable solution. K3s is particularly well-suited for lightweight and resource-constrained environments like the Raspberry Pi, offering better performance and easier management compared to Minikube. This transition makes the infrastructure more efficient and easier to maintain.

To enhance connectivity between different nodes, ClusterLink[20] has been leveraged. This tool interconnects pods running on the Intel edge device with those running on the Raspberry Pi devices, ensuring seamless communication and data exchange across the entire infrastructure. Additionally, a USB camera has been attached to the Intel node. It is used to exploit and validate the peripherals discovery and match making capabilities of ICOS. More cluster interconnections will be supported according to the needs that arise.

Furthermore, a new Kubernetes cluster identical in configuration to the existing one has been established. This additional cluster serves as a staging environment, allowing for testing and development without impacting the stability of the first environment. The original Kubernetes cluster now functions as the stable environment.

[19] https://minikube.sigs.k8s.io/
[20] https://clusterlink.net/docs/v0.2/

5 Release Plan

The ICOS Beta software release is the intermediate release of the ICOS Meta OS. The last and final release for ICOS software will be delivered in project's month 32 (March 2025).

In line with the continuous development strategy of the project, the integration of final release will be done in an incremental way starting soon after the release of ICOS Beta, exploiting the automated CI/CD process in WP5. In addition, depending on the short-term plans in WP3 and WP4, internal releases could be scheduled before the final release to deliver updated software/functionalities to the project's Use Cases and to the open call projects.

This section provides an initial plan for the functionalities that will be included in the ICOS Final release, accordingly to: a) the current status of the project; b) the plans provided in deliverable D2.3.

The main functionalities expected to be included in the ICOS Final release are:

- **Continuum Management**: The creation and the management of the Cloud-Edge continuum will be improved making it easier to on-board new infrastructure/devices into an ICOS Continuum. The management of the resources will be made more effective, efficient, and distributed.
  - **On-Boarding**: procedures for the On-boarding and management of the devices will be eased and partially automated. This will allow a faster, reliable, and more interoperable management of the continuum.
  - **Multi-controller communication**. In the final release, a variable number of ICOS Controllers will be deployed as part of an ICOS infrastructure and will coordinate through cross-communication protocols to deliver collaborative management.
  - **Networking**. Full network connectivity across ICOS sites thanks to the integration of Cluster-Link. ClusterLink simplifies the connection between application services that are located in different domains, networks, and cloud infrastructures. ClusterLink is useful when multiple parties are collaborating across administrative boundaries. With ClusterLink, information sharing policies can be defined, customized, and programmatically accessed around the world by the right people for maximum productivity while optimizing network performance and security.
- **Runtime Management:** The definition and the execution of user's applications will be improved adding more advanced orchestration capabilities and control on the application behaviour:
  - **Application Descriptor:** Definition of the syntax for describing the application topology and orchestration requirements will be completed offering more flexibility and expressivity.
  - **Intelligent application deployment.** The matchmaking process will be AI-driven in order to provide optimal deployment strategies. More open and flexible solutions will be considered to leverage the capabilities of the whole continuum and estimating the effects of local and remote data sources access. The intelligent technology will be fed with historical and run-time telemetry and monitoring data.
  - **Enhanced dynamic adaptation loop**. For the final release of ICOS, the dynamic adaptation loop will be improved in two directions. First, it will publish new metrics in the monitoring infrastructure (i.e., security and green efficiency related) to allow a better understanding of the service behaviour and its current status/needs. Second, it will leverage on AI to make decisions on the current resource allocation and propose possible adaptations.
- **Data Management**: For the final release, it will include the necessary features required for managing data across the continuum. The task offloading mechanism will allow the Intelligence Layer to perform training with high data locality without having to deal with the specific topology of ICOS nodes nor data persistency. The Data Management service on the ICOS agents will give a direct access to agent data, allowing other components to interact and react in a decentralized manner with lower latency and lower bandwidth requirements. The expertise of the Data Management partners will also be leveraged in the Use Cases and result in transparent continuum-

aware architectures that will manage data without relying solely on the cloud or on fully centralized solutions.

- **Intelligence Layer**: For the final release, the intelligence layer will incorporate a full AI trustworthiness module featuring model re-training based on model degradation, explainable AI, and federated learning. It will further enhance the meta-kernel models, support to Use Cases and will count with an online AI model repository (formerly called AI Marketplace, or AI models catalogue).
- **Security**: The Security Layer functionalities planned for the final release, enhance the existing scanning functionality to include the execution of mitigation or recovery actions for detected anomalies, security vulnerabilities and threats as well as enhance the basic management of security compliance policies with the ability to execute remediation actions for compliance policy violations. The ICOS's zero-trust approach will be completed adding authentication, authorization and encryption not only for external communications, but also for intra-node, service-to-service calls.
- **Shell:** The ICOS Shell will be completed, adding more functionalities to the Graphical User Interface (GUI) like visualization of real-time performance metrics and alerts.

The plan presented in this section will provide a guidance to both technical Work Packages for the implementation and to WP5 for planning the integration and the testing of the ICOS Final release. However, being still in a very active phase of the project, some requirements could change or be re-prioritized, leading to a change to this plan too.

6 Conclusions

This document accompanies the delivery of the ICOS Beta release, the second software release for the ICOS project.

With regards to the previous release (ICOS Alpha, November 2023), several improvements in all the ICOS layers have been delivered (section 2) of this document, provides a summary of all the released functionalities as well as pointers to online resources for this release.

In addition, two important activities have been introduced during the integration of the ICOS Beta release that improves the impact, the exploitability and the future sustainability of the ICOS Meta OS:

- the **source code** of the software developed in ICOS as part of the ICOS Meta OS has been published as open-source code at https://github.com/icos-project;
- the **technical documentation** to administer and use an ICOS System has been created and made available online at https://www.icos-project.eu/docs/;

Both activities are the results of an extensive collaborative effort among the project's partners. The main procedures and guidelines followed as well as the results are reported in the document (sections 3.2 and 3.3).

The document reports (section 4) the effort towards the introduction of a formal and comprehensive **software testing process** in the project. The document outlines the goals and the methodology for the testing process introduced and describes the current type of tests that are executed. To date, a comprehensive Unit Test methodology has been meticulously established to validate the functionality and reliability of individual components within the ICOS software ecosystem. By rigorously applying this methodology, high standards of accuracy, reliability, and maintainability are upheld across all ICOS applications. In addition, a set of automated checks and validation of ICOS source code and artifacts are executed during the CI/CD pipelines: a) SonarQube source code analysis, b) Container Vulnerability Scanning, c) Secret Detection, and d) Docker and Helm linting checks. The next crucial step involves expanding the testing efforts to encompass additional components, thereby enhancing the breadth and depth of the Unit Tests.

Finally, the document includes (in section 5) an initial **release plan** for the final ICOS release (ICOS Final) that will be delivered at the end of the project. The plan describes the main functionalities that are expected to be added/improved to the ICOS Meta OS by the end of the project. The plan might change during the project, due to the feedback received by the Use Cases and/or open call projects. However, it is intended to provide an initial guideline to the technical Work Packages to organize the activities in the remaining part of the project. The final ICOS software release will be documented in deliverable "D5.3 - Third ICOS Release: Complete ICOS version" that will be delivered in project's month 32.

7 References

[1] ICOS. *D5.1 -ICOS Alpha Release*, M. Michalke, 2023. https://www.icos-project.eu/deliverables

[2] ICOS. *D2.3 - ICOS ecosystem: Technologies, requirements and state of the art (IT-2)*, G. Xylouris, 2024. https://www.icos-project.eu/deliverables

[3] ICOS. *D2.4 - ICOS architectural design (IT-2)*, J. Garcia, 2024.

8 Annex I - Testing Goals

In the development and deployment of complex software systems, particularly those deployed on Kubernetes, testing encompasses various aspects to ensure robust, reliable, and efficient operation of developed components. The types of tests conducted are strategically chosen to cover multiple critical areas of the application lifecycle:

Component Connectivity: Testing connectivity ensures that each component can successfully communicate with others according to the defined network policies and service configurations. This is crucial in distributed systems where components often operate in different pods or even across clusters. Connectivity tests validate network configuration, DNS resolution, and service discovery mechanisms, ensuring that components can reach each other and function cohesively.

Functionality Verification: This involves detailed testing of each function within the components to ensure they perform as expected under various scenarios. Functionality tests include both positive and negative scenarios to validate the handling of correct inputs as well as error conditions. These tests help in identifying any functional shortcomings or deviations from the expected behaviour, which is critical for ensuring that the software meets its specifications.

Configuration Accuracy: Configuration tests verify that the settings applied to each component are correct and effective. This includes environmental variables, configuration files, and runtime parameters. Accurate configuration testing is essential to ensure that the application operates under right conditions, and misconfiguration, which are a common source of failure in software systems, are identified and corrected.

Integration Readiness: Before individual components are combined into a larger system, it is vital to test their readiness for integration. This type of testing checks if components can work together without issues and that interfaces between them are correctly defined and implemented. Integration readiness tests are a precursor to full integration tests and help in isolating issues that can affect the interactions between components.

Operational Behaviour: Operational tests assess the system's behaviour under normal and peak load conditions to ensure it can handle real-world use. This includes performance testing, stress testing, and resilience testing under varied conditions, such as high traffic or data load, network latency simulation, and more. Testing operational behaviour is crucial for understanding how the system behaves in production, ensuring that it remains stable and performs efficiently even when pushed to its limits.

By covering these critical aspects through targeted testing strategies, the project aims to minimize the risk of failures, ensure high quality of service, and guarantee that the system can meet the required operational standards post-deployment. Each testing phase builds on the previous, from ensuring individual component functionality to verifying that the whole system can operate cohesively and handle expected loads, thus paving the way for a successful integration and deployment in a production environment.

To further enhance the robustness and reliability of ICOS software systems across various deployment environments —development, staging, and production— testing strategy is meticulously designed to be automated and thorough. This strategic automation is essential not only for maintaining high development velocity but also for ensuring that each deployment adheres to stringent quality and performance criteria.

Automation and Continuous Integration/Continuous Deployment (CI/CD): An advanced CI/CD pipeline using GitLab is integrated, which plays a crucial role in the testing automation strategy. This setup allows for the automatic execution of tests every time changes are committed to the codebase. Automating tests in development, staging, and production environments ensures that any version of

the software being prepared for release is tested under conditions that closely mimic the real-world scenarios in which it will operate.

Helm Tests for Kubernetes Deployments: In addition to the CI/CD pipeline, Helm tests are utilized. These tests are specifically designed for Kubernetes deployments, enabling the validation of configurations and functionalities in a manner that is native to Kubernetes's orchestration environment. Helm tests are instrumental in confirming that services are deployed correctly with all dependencies and configurations in place as expected.

Component-Specific Custom Tests: Each component within the system is equipped with its own set of custom tests. These are developed to address the unique requirements and functionalities of individual components, providing a deep, focused examination of critical features and operational behaviour. These tests are vital for verifying that all parts of the system work harmoniously and meet predefined functional benchmarks.

Ensuring Deployment Criteria: The combination of GitLab CI/CD, Helm tests, and component-specific tests ensures that every deployment is rigorously vetted before it is released into production. This approach not only streamlines the identification and rectification of potential issues before they affect users but also helps maintain consistent compliance with our operational standards and security policies.

8.1 Component Unit Testing and Helm Testing

Component Unit Testing is a fundamental part of ICOS software development lifecycle aimed at validating individual parts of the application in isolation from the wider system. This method involves testing each component at the smallest testable part of the application, such as functions, methods, classes, interfaces, and modules. Focusing on the smallest unit of code ensures that each element performs as expected before it interacts with other parts of the system.

The main goals are:

1. **Accuracy and Reliability**: The primary goal of component unit testing is to verify that each component behaves correctly under various scenarios, including both expected use and edge cases. This helps to ensure the software's reliability by catching bugs and issues at the earliest possible stage.
2. **Facilitate Changes**: Component unit testing makes the codebase safer to refactor or upgrade, as changes can be validated quickly and independently. This is crucial for maintaining a robust codebase that can evolve without breaking existing functionality.
3. **Documentation**: Component unit tests serve as project documentation because they illustrate what the code is supposed to do. New developers can look at the tests to understand the component’s intended behaviour without diving deep into the implementation details.
4. **Design Feedback**: Writing tests for each component unit forces developers to consider the design of the code. If a unit is hard to test, it might indicate design problems such as high coupling or poor encapsulation, which can then be addressed to improve code quality.
5. **Development Efficiency**: With a comprehensive suite of unit tests, developers can work more confidently and efficiently. They can make changes to the code and get immediate feedback on what, if anything, broke, which reduces the debugging time drastically.

Helm Tests are specifically designed to validate the integration and functionality of Kubernetes resources managed through Helm charts. These tests are essential for ensuring that Helm releases are correctly configured and operational within a Kubernetes environment. By leveraging Kubernetes's native mechanisms, Helm tests allow for automated testing of the deployment, management, and un-installation processes of Kubernetes applications.

Goals:

1. **Integration Validation**: Verify that all Kubernetes resources defined in the Helm chart interact correctly with each other once deployed.
2. **Configuration Verification**: Ensure that configurations specified in the Helm chart are correctly applied and functional within the deployment.
3. **Operational Assurance**: Confirm that the application behaves as expected in a live Kubernetes environment, including its response to simulated real-world conditions.
4. **Upgrade and Rollback Testing**: Validate the processes of upgrading and rolling back releases via Helm, ensuring that these critical operations proceed without disrupting service availability.

One of the primary tools utilized is Helm tests, which are provided by Kubernetes. These tests are designed to verify that Helm charts are correctly deploying the applications within the Kubernetes environment, ensuring that all Kubernetes resources are configured and interact as expected. They are particularly valuable for automating the deployment and testing of Kubernetes resources. Helm tests allow to script and run diagnostics on the deployed applications, checking for proper service configuration and operational behaviour directly within the Kubernetes clusters.

9 Annex II - Unit Testing Methodology

The **methodology** involves a systematic approach where each unit of the application is tested in isolation to ensure that it performs exactly as intended. This approach minimizes dependencies and uses mock data and interfaces to test each component's functionality independently.

For defining the unit test design for each component, an Excel template was used (see Figure 6). The skeleton of the unit test design, as outlined below, includes several key elements:

- **Test Case ID**: Each test case is uniquely identified to track test execution and results systematically.
- **Test Description**: This provides a brief description of what the test is intended to verify, which can be a specific function or a broader feature within the component.
- **Preconditions**: These are the conditions that must be met before the test can be executed, ensuring that the environment and component state are correctly set up for the test.
- **Test Steps**: Detailed steps are laid out to follow during the test. These steps guide the tester through the required actions to execute the test scenario effectively.
- **Expected Results**: Clearly defined expectations for what the test should result in if the component behaves as expected. This is crucial for verifying the test's success or failure objectively.
- **Actual Results**: This is where the outcomes of the test execution are recorded, providing direct feedback on the component's behaviour under test conditions.
- **Status**: Indicates the test's pass or fail status, which is essential for assessing the overall stability and readiness of the software component.

Unit Testing Matrix (UTM)	
Definition of the Unit Tests	Describes the features to test, including a description of how they will be tested.
Definition of the Interfaces to be tested	Describe the definition of exposed interfaces and their data types and parameters.
Unit Test Cases descriptions	These are in the form of tables that demonstrate scenario cases that include interface class name, interface name, operation/function name, input and system output and evaluation of the expected and actual result.

Template of the Unit Test Matrix

Unit Test	Interface	Input	Output	Unit Test Cases
UT#	/interface_ name	Call input: data http://URL/interface_name/data	Data format	UTC#

Template of the Unit Test Case Matrix

Description	Description of the UTC for appropriate UT, this involves further documentation of the input/information model set.
Unit Test:	UT number.
Class Name:	The class name of the interface.
Interface name:	The interface name.
Operation:	Description of the operation/functionality of the interface.
Input:	Input data and information model.
Output:	Resulting output.
Expected Result:	Description of the testing results (result as expected or failure)
Actual Result:	Required results that in the case of failure are
Invalid data input:	Description of invalid input set (intentionally submission for testing reasons)
Details:	Description of the testing environment and/or software (e.g. browser).
Other remarks:	Description of notes.
Screenshot	Illustration of results.
Deployed Instance:	Endpoint of the cloud enabler. Information about the cloud provider.

Figure 6: Test Cases template

A test design was created for each Unit test or Unit test group (tests that contain many functional tests together) for each component.